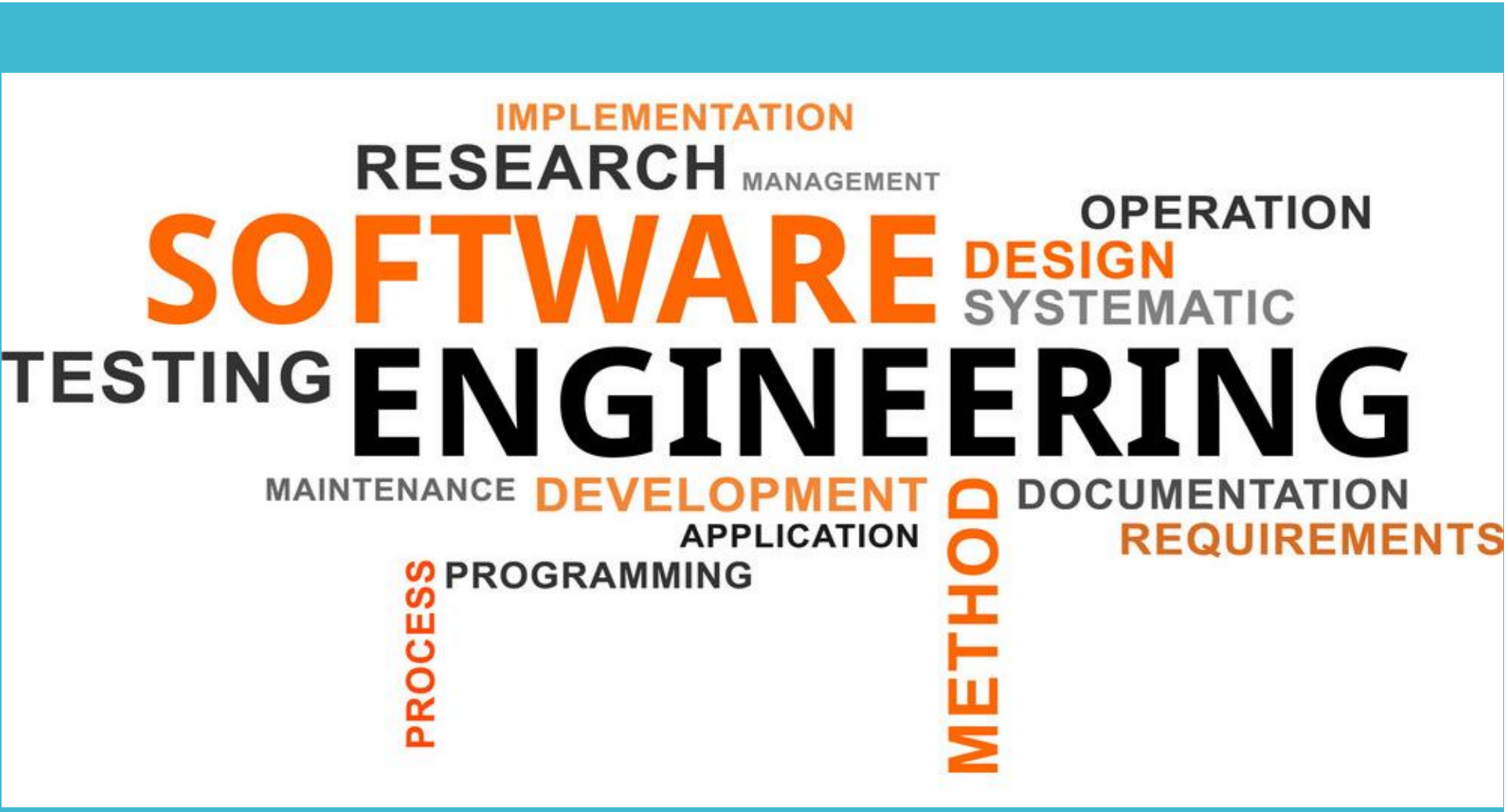


Prof. Suhag Baldaniya

Marwadi University

Department of Information and Communication Technology

Chapter 1 Introduction and Software Process Models

Subject Code: 01CT0615

Contents

- Software engineering
- Dual role of software
- Software Crisis history
- Various Myths Associated with Software
- Different Software Process Models
- The Linear Sequential Model
- The Prototyping Model
- The RAD Model
- Evolutionary Process Models
- Component-Based Development
- Process, Product and Process

Why Software Engineering ?

- As technology advances, the ability to build quality software while considering design, development, security, and maintenance is sought after amongst all kinds of companies, from finance and banking to healthcare and national security.
- Software Engineering applies the knowledge and theoretical understanding gained through computer science to building high-quality software products. As a maturing discipline, software is becoming more and more important in our everyday lives.
- According to the Bureau of Labor Statistics' Occupational Outlook Handbook, 2014-15 Edition, "software engineers are among the occupations projected to grow the fastest and add the most new jobs over the 2012-2022 decade, resulting in excellent job prospects."

- Enable to understand different phases of software development
- Enables to appreciate their role and contribution in different phases of software development
- Introduce well known methods to analyse requirement of a client and to design a system that meet these requirements
- Help to realise the need for good practices in developing software
- Introduce effective methods of testing software
- Help to understand the importance of adhering to the quality standards

What is Software?

- Software encompasses:

1. Instructions (computer programs) that when executed provide desired features, function, and performance
2. Data structures that enable the programs to store and manipulate information
3. Documentation that describes the operation and use of the programs.

Software Products

- Software products may be of two types

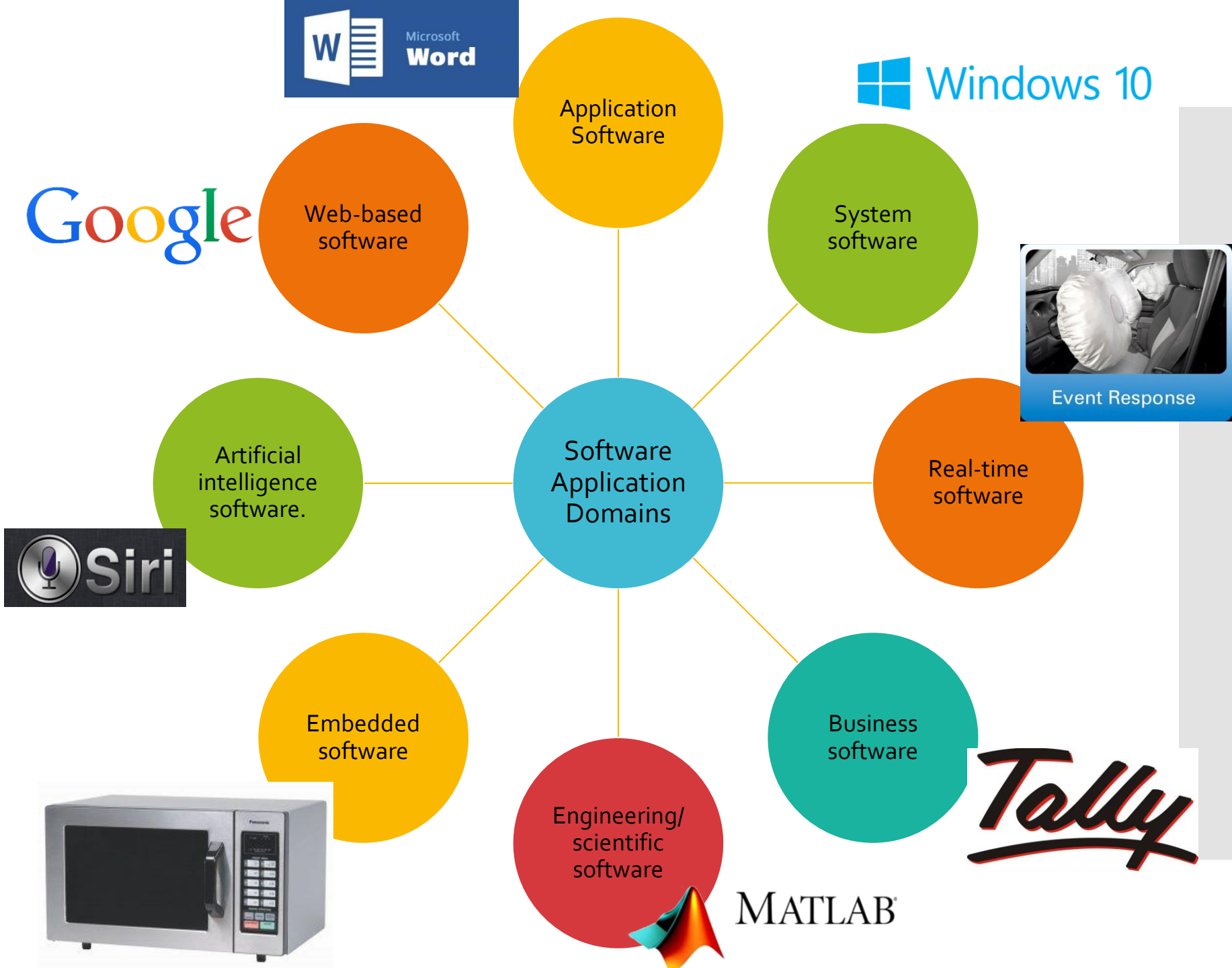
1. Generic products:-

Developed to be sold to a range of different customers

2. Customized products:-

developed for a single customer according to their specification

Software Application Domains



What is Software Engineering?

The application of a

systematic, disciplined, quantifiable

approach to the

development, operation, and

maintenance of software

What is Software Engineering?

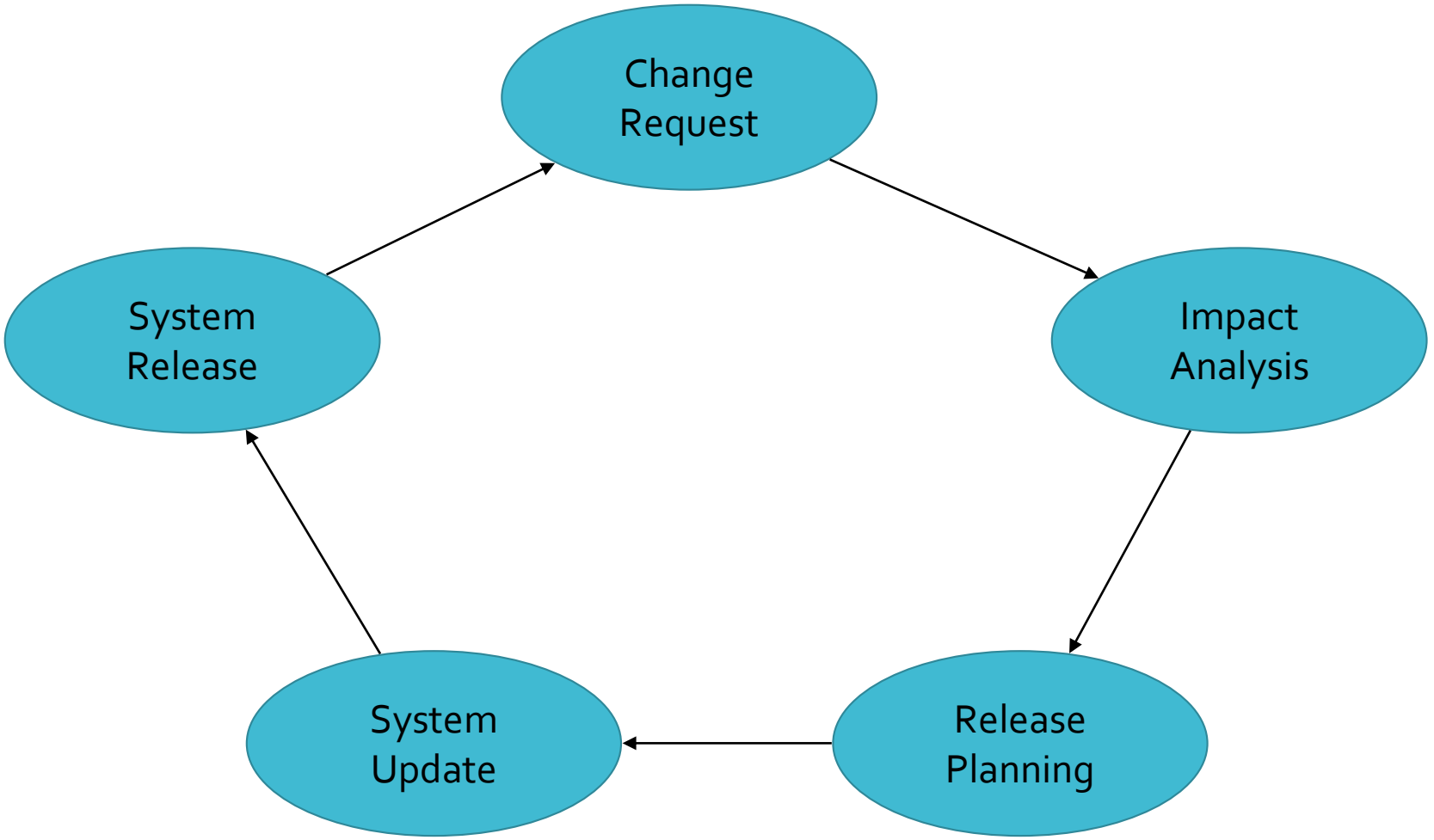
- Software Engineering is the science and art of building (designing and writing programs) a software systems that are:
  1. on time
  2. on budget
  3. with acceptable performance
  4. with correct operation

Dual role of software

Dual role of software:

1. A Product
   - Information transformer
   - Producing, managing, acquiring, modifying, displaying, or transmitting information
2. The vehicle for delivering a product
   - The control of the computer(ex. OS)
   - The communication of information(ex. Networking software)
   - Helps build other software (ex. Software Tools)

Dual role of software

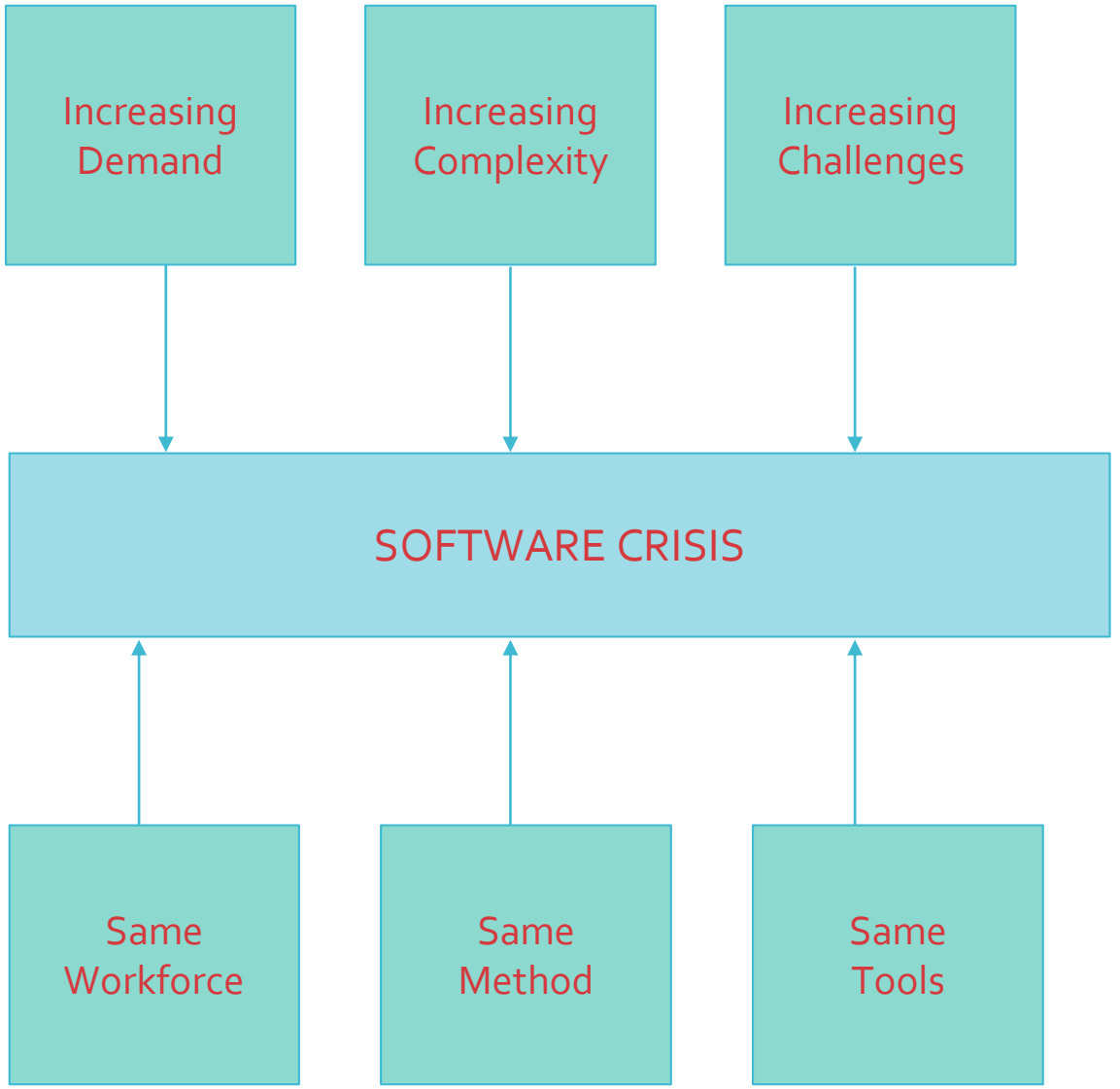


Dual role of software

When modern computer is built, it is necessary to answer the following questions:

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all the errors before we give the software to customers?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Software: A Crisis on the Horizon



Causes of Software Crisis

- Maintenance cost of software is as expensive as development cost
- Projects was running over-time
- Less efficient
- Low quality
- Software often did not meet requirements
- Software was never delivered on time

Solution of Software Crisis

SOFTWARE ENGINEERING

Guidelines

- Reduction in software Over-Budget
- The quality of software must be high.
- Less time needed for software project
- Experience team members on software project
- Software must be delivered on time.

Software Quality Attributes

1. USABILITY
2. REUSABILITY
3. FLEXIBILITY
4. MAINTAINABLITY
5. PORTABILITY
6. RELIABILITY
7. FUNCTIONALITY

Software Characteristics

1. Software is developed or engineered, it is not manufactured
   - It is not manufactured like hardware
   - The manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software
   - Both require the construction of a "product" but the approaches are different.

Software Characteristics

2. Software doesn't "wear out"

- When a hardware component wears out, it is replaced by a spare part.
- There are no software spare parts.
- Every software failure indicates an error in design or in the process through which design was translated into machine executable code.
- Software maintenance involves considerably more complexity than hardware maintenance.

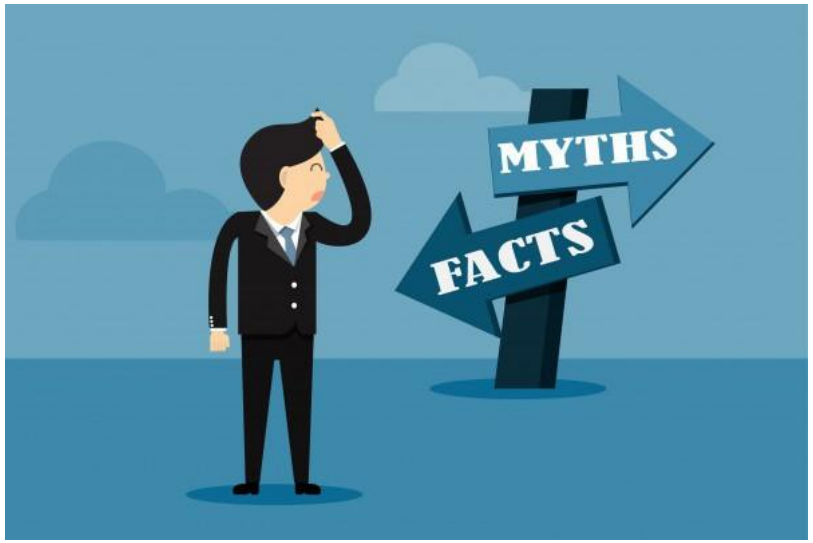
Software Characteristics

3. The industry is moving toward component-based assembly, most software continues to be custom built
   - Component reused in many different programs
   - Modern reusable components are created so that the engineers can concentrate on truly innovative elements of the design.
   - In hardware, component reuse is a natural part of the engineering process but in software world, it is has just begun on broad scale.

Various Myths Associated with Software

- Misleading attitudes that leads to serious problem called myths

1. Management Myths
2. Customer Myths
3. Practitioner's(developer) Myths



Management Myths

- Myth 1: We already have a book that's full of standards and procedures for building software which is enough.
- Reality:
  - ✓ The book of standards may very well exist, but is it used?
  - ✓ Are software practitioners aware of its existence?
  - ✓ Does it reflect modern software engineering practice?
  - ✓ Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality?

Management Myths

- Myth 2: We have the newest computers and tools
- Reality:
  - ✓ It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development
  - ✓ CASE tools are more important than hardware for achieving good quality and productivity
  - ✓ Majority of software developers still do not use tools effectively

Management Myths

- Myth 3: We can add more programmers and catch up the schedule
- Reality:
  - ✓ Software development is not a mechanistic process like manufacturing
  - ✓ As new people are added, people who were working must spend time educating the newcomers thereby reduce the amount of time spent on productive development effort
  - ✓ People can be added but only in a planned and well-coordinated manner

Management Myths

- Myth 4: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.
- Reality:
  - ✓ If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths

- Myth 1: A general statement of objectives is sufficient to begin writing programs– we can fill in the details later.
- Reality:
  - ✓ A formal and detailed description of software is essential.
  - ✓ Detailed requirements gathered with effective communication between customer and developer.

Customer myths

- Myth 2: Project requirements continually change, but change can be easily accommodated because software is flexible
- Reality:
  - ✓ Software requirements change, but the impact of change varies with the time at which it is introduced.
  - ✓ Early requests for change can be accommodated easily.

Practitioner's (Developer) myths

- Myth 1: Once we write the program and get it to work, our job is done.
- Reality:
  - ✓ The sooner you begin 'writing code', the longer it'll take you to get done.
  - ✓ Industry data indicate that between 60% to 80% of all effort expended on software will be expended after it is delivered to the customer for the first time

Practitioner's (Developer) myths

- Myth 2: I can not achieve quality until program is running
- Reality:
  - ✓ One of the most effective software quality assurance mechanisms can be applied from the beginning of a project—the formal technical review.
  - ✓ Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

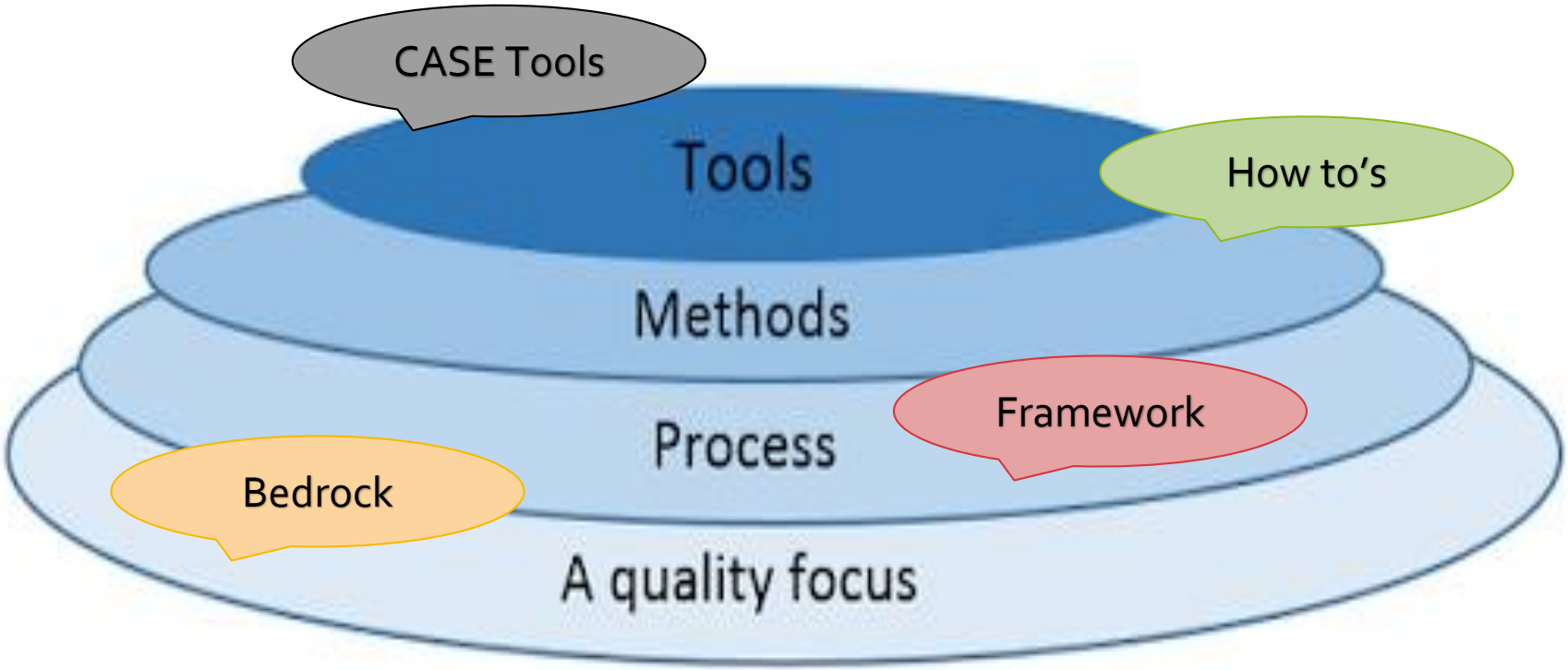
Practitioner's (Developer) myths

- Myth 3: The only deliverable work product for a successful project is the working program.
- Reality:
  - ✓ A working program is only one part of a software configuration
  - ✓ Documentation, Model and plans provides a foundation for successful engineering and, more important, guidance for software support.

Practitioner's (Developer) myths

- Myth 4: Software engineering is about creating unnecessary documentation.
- Reality:
  - ✓ Software engineering is not about creating documents. It is about creating quality.
  - ✓ Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Software Engineering: A Layered Technology



Pic Courtesy : Software Engineer - A practitioner's Approach By Roger Pressman

Software Engineering: A Layered Technology

Quality Focus:

- Any engineering approach must rest on an organizational commitment to quality.
- Total quality management fosters a continuous process improvement culture.
- The bedrock that supports software engineering is a quality focus.

Software Engineering: A Layered Technology

Process Layer:

- The foundation for software engineering is the process layer
- It covers all activities, actions and tasks required to be carried out for software development
- Process defines a framework for a set of key process areas (KPAs)
- KPA establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed

Software Engineering: A Layered Technology

Methods:

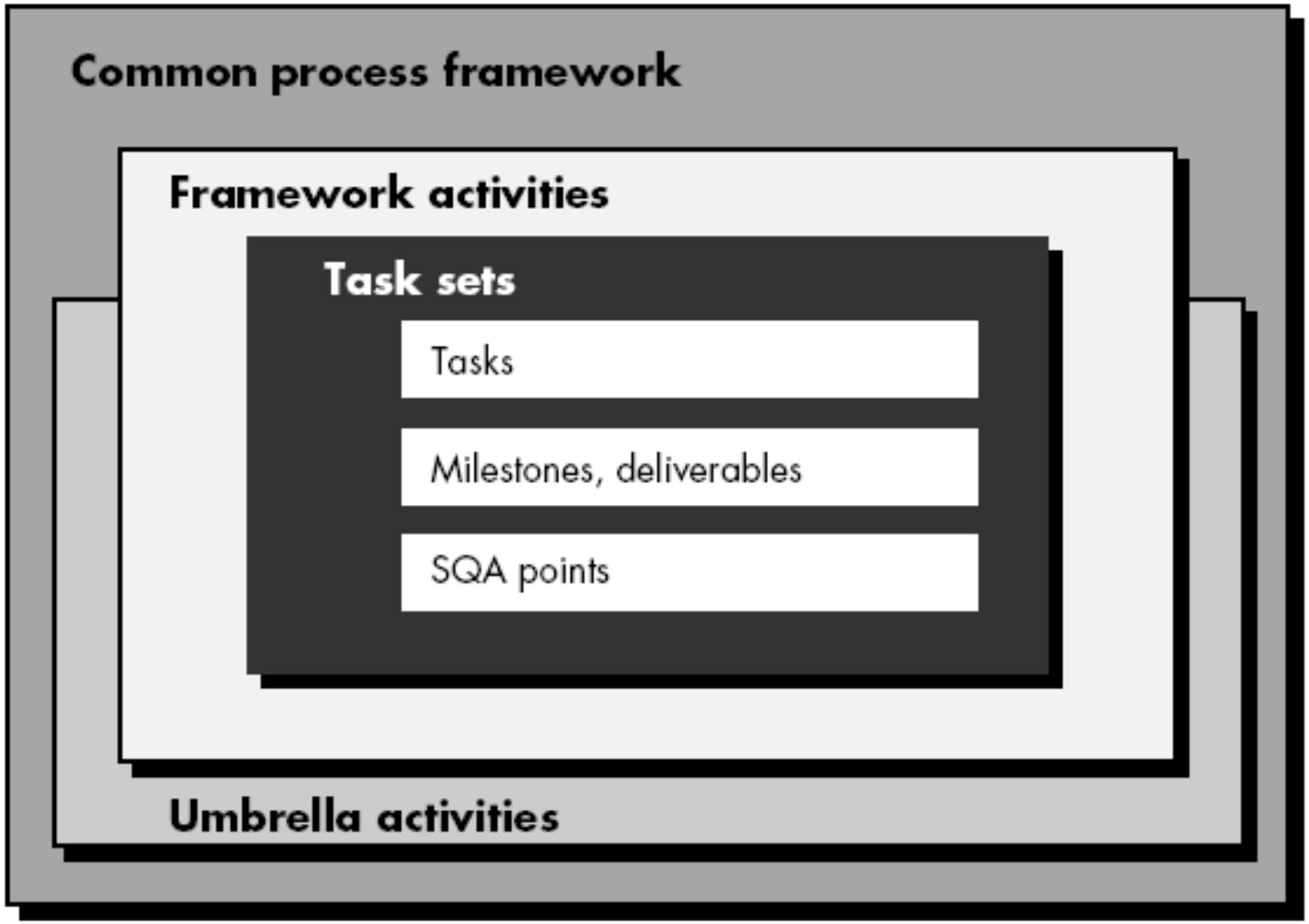
- Software engineering methods provide the technical how-to's for building software.
- It provides the technical way to implement the software.
- Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.

Software Engineering: A Layered Technology

Tools:

- Software engineering tools provide automated or semi automated support for the process and the methods.
- The tools are integrated i.e the information created by one tool can be used by the other tool.
- CASE(computer-aided software engineering) tools automate many of the activities involved in various life cycle phases.

Software Process Framework



Pic Courtesy : Software Engineer - A practitioner's Approach By Roger Pressman

Software Process Framework

- Define a small number of framework activities that are applicable to all software projects, regardless of their size or complexity
- A process is a collection of activities, actions and tasks that are performed when some work product is to be created
- Each framework activities are divided into set of task.
- Each task sets identifies work to be completed, product to be produced, quality assurance points & milestones to indicate progress

Why Software Process Framework?

- A process defines who is doing what, when and how to reach a certain goal.
- To deliver software in timely manner
- To deliver qualitative product for those who have given proposal for software development and those who will use software

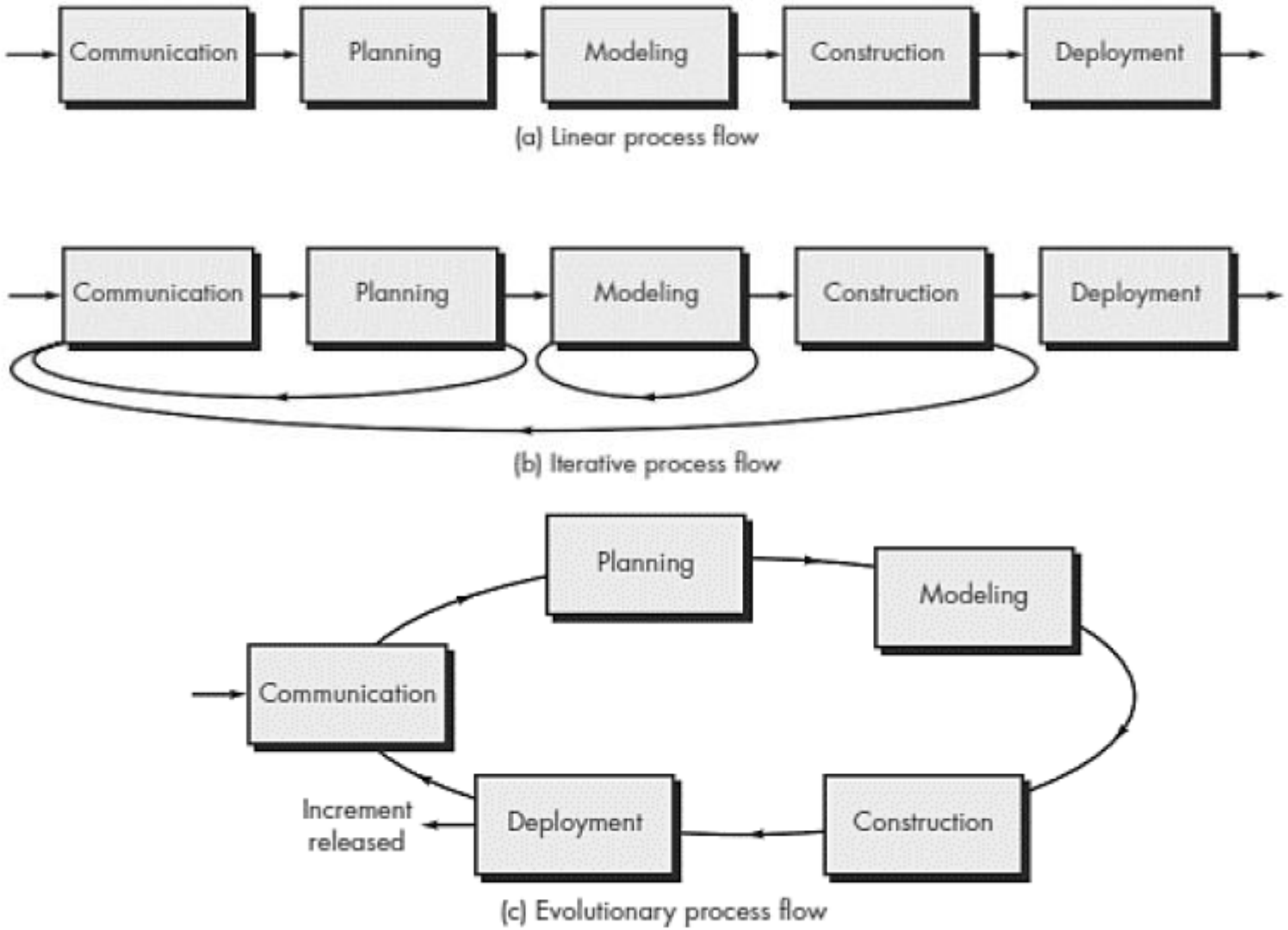
Generic Process Framework Activities

- Communication:
  - ✓ Communication with Customers / stakeholders to understand project requirements for defining software features/functionality
- Planning:
  - ✓ Software Project Plan
  - ✓ It describes technical task, risks, resources, product to be produced & work schedule

Generic Process Framework Activities

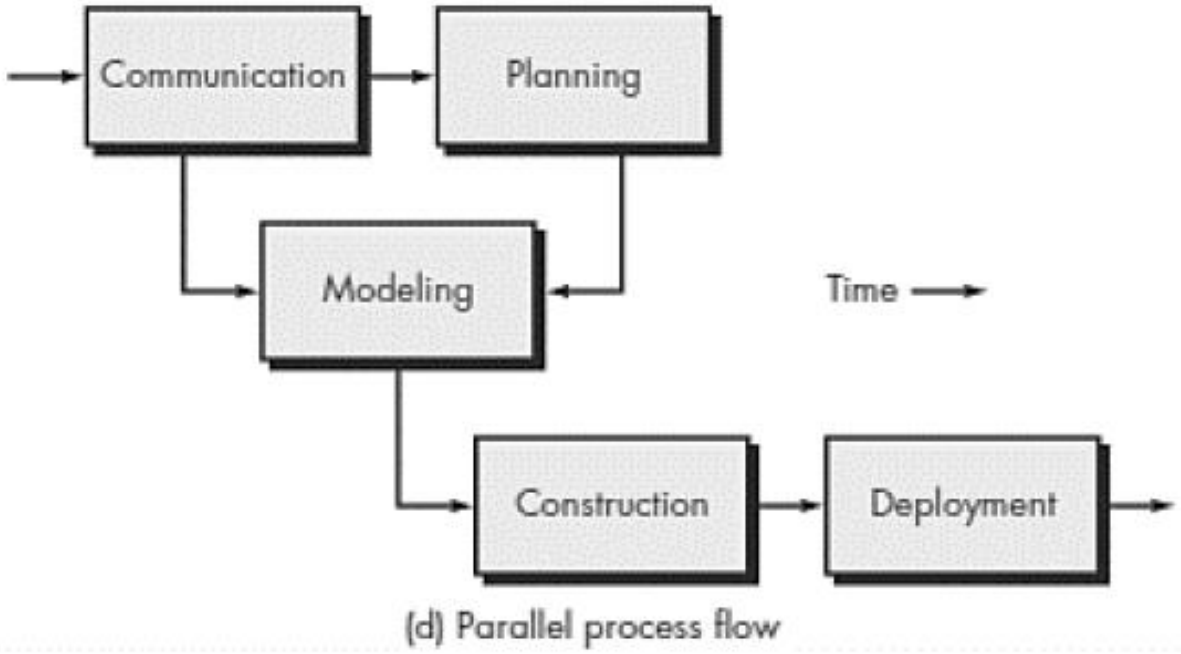
- Modeling:
  - ✓ Creating models to understand requirements and shows design of software to developers as well as customers to achieve requirements
- Construction:
  - ✓ Code generation(either manual or automated or both) and Testing(to uncover error in the code)
- Deployment:
  - ✓ Delivery to the customer for evaluation, Customer provide feedback and Software Support

Process Flow



Pic Courtesy : Software Engineer - A practitioner's Approach By Roger Pressman

Process Flow



(d) Parallel process flow

Pic Courtesy : Software Engineer - A practitioner's Approach By Roger Pressman

Umbrella Activities

Umbrella activities applied throughout the software project & help a software team to manage and control progress, quality, change & risks

- Software project tracking and control: It allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- Formal technical reviews: Assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- Software quality assurance: Defines and conducts the activities required to ensure software quality.
- Software configuration management: It manages the effects of change throughout the software process.

Umbrella Activities

- Document preparation and production: It encompasses (includes) the activities required to create work products such as models, documents, logs, forms and lists.
- Reusability management: It defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components
- Measurement: Defines and collects process, project and product measures that assist the team in delivering software that meets stakeholders' needs.
- Risk management: Evaluates risks that may affect the outcome of the project or the quality of the product.

Software Process Models

- Process models prescribe a distinct set of activities, actions, tasks, milestones, and work products required to engineer high quality software.
- Process models are not perfect, but provide roadmap for software engineering work.
- Software models provide stability, control, and organization to a process that if not managed can easily get out of control
- Software process models are adapted to meet the needs of software engineers and managers for a specific project.

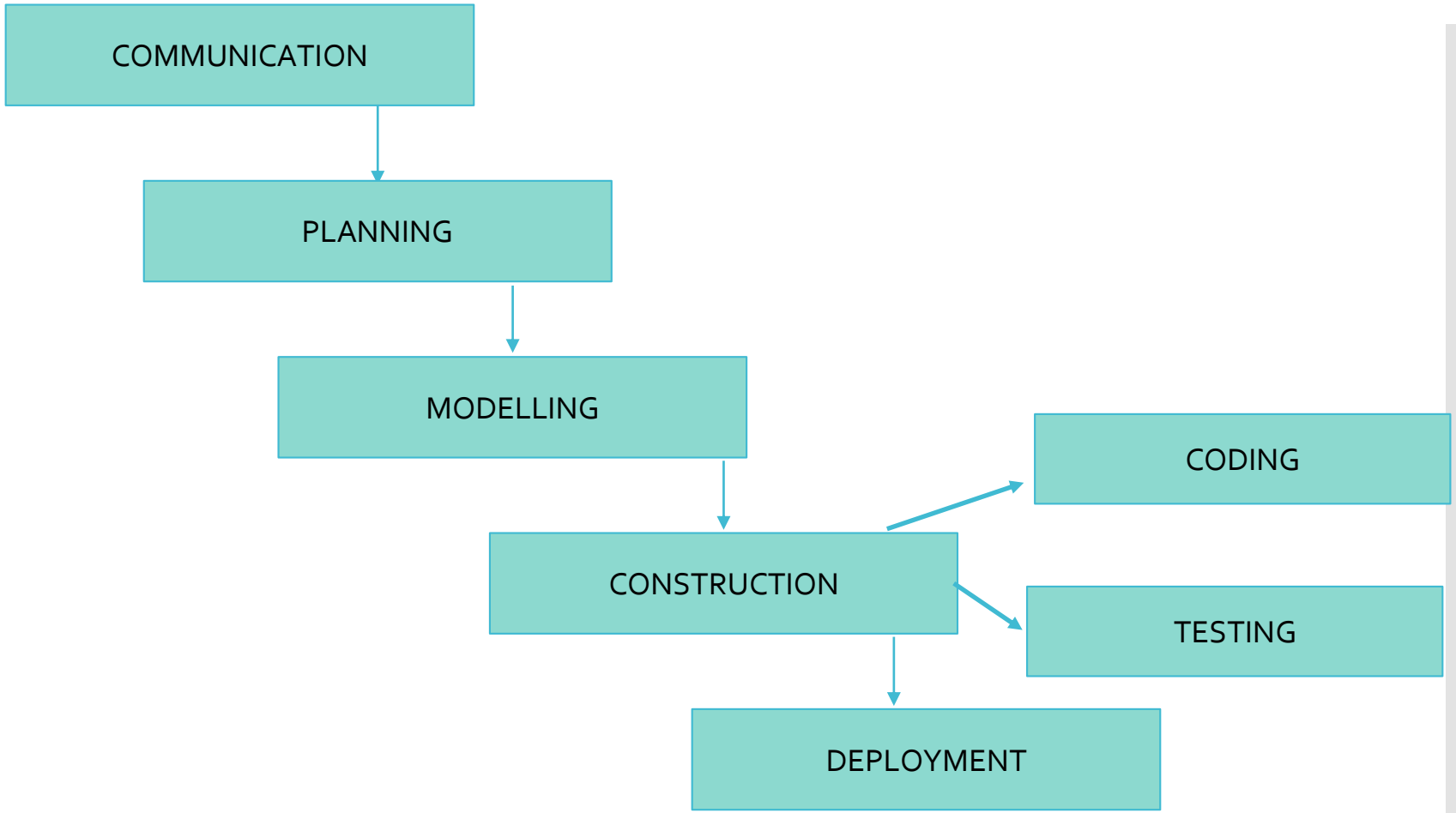
Software Process Models

- Process model is selected based on different parameters
  - ✓ Type of the project & people
  - ✓ Complexity of the project
  - ✓ Size of team
  - ✓ Expertise of people in team
  - ✓ Working environment of team
  - ✓ Software delivery deadline

Software Process Models

1. The Linear Sequential Model
2. Prototyping Model
3. The RAD Model
4. Evolutionary Process Models
5. Component-Based Development

1. The Linear Sequential Model or Waterfall Model or Classic Life Cycle



1. The Linear Sequential Model or Waterfall Model or Classic Life Cycle

- Communication:
  - ✓ Requirement gathering and analysis
  - ✓ Requirement specification(SRS)
- Planning:
  - ✓ Assessing progress against the project plan.
  - ✓ Perform required action to maintain schedule
- Modelling:
  - ✓ To transform the requirements specified in the SRS document into a structure that is suitable for implementation

1. The Linear Sequential Model or Waterfall Model or Classic Life Cycle

- Construction:
  - ✓ Coding and Testing
  - ✓ Unit testing, Integration and System testing
- Deployment:
  - ✓ Maintenance
    - Corrective Maintenance: Correct errors that were not discovered in development phase
    - Perfective Maintenance: To enhance the functionalities of the system based on the customer's request
    - Adaptive Maintenance: To port the software to work in a new environment

When to use?

- Requirements are very well known, clear and fixed
- Product definition is stable
- Technology is understood
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short

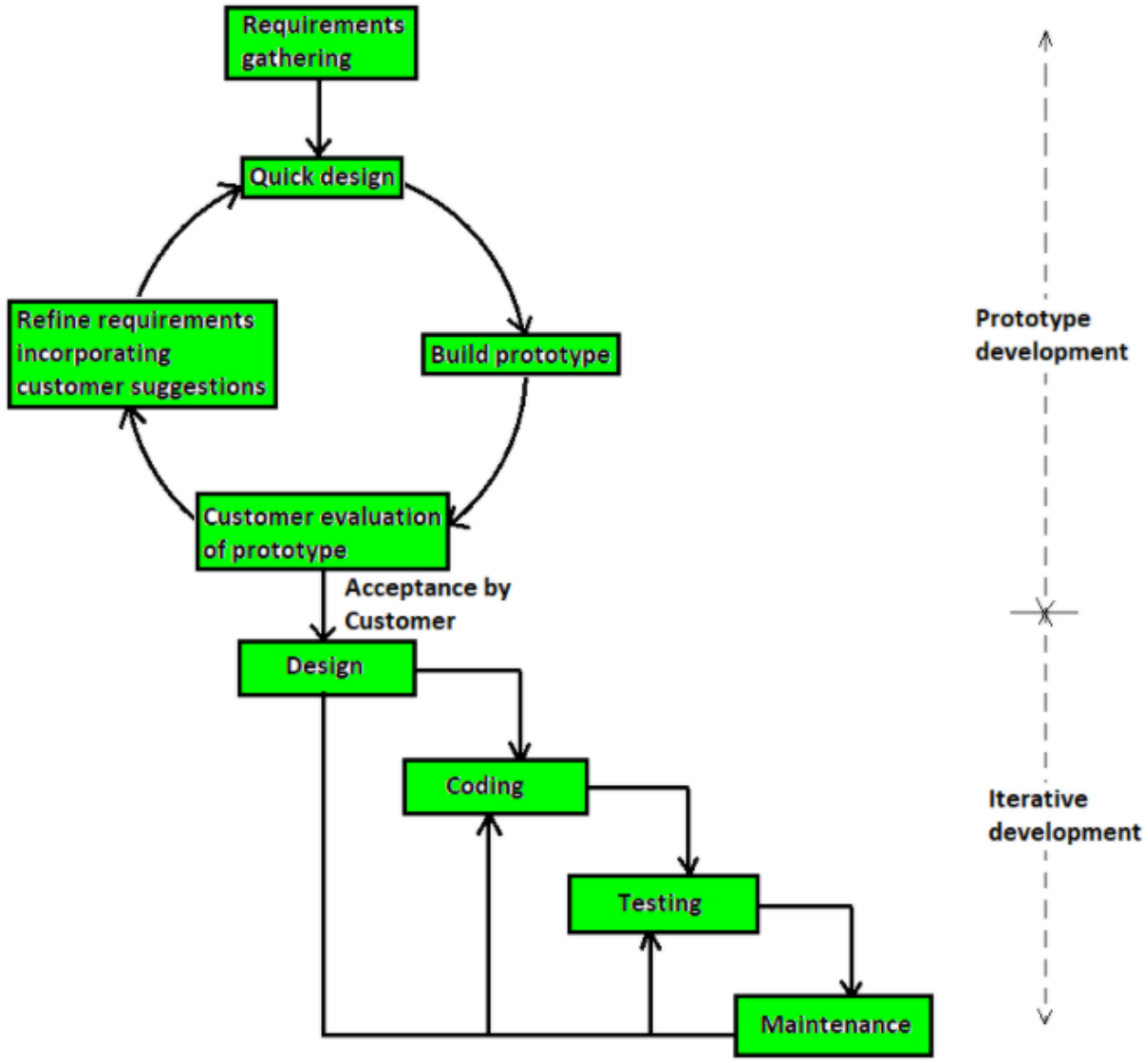
Advantages of the Linear Sequential Model

- Simple and easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well understood milestones.
- Reinforces good habits: define-before-design, design-before-code.
- This model works well for smaller projects and projects where requirements are well understood.

Disadvantages of the Linear Sequential Model

- It is often difficult for the customer to state all requirements explicitly.
- The model implies that you should attempt to complete a given stage before moving on to the next stage
- Some teams sit idle for other teams to finish
- Appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

2.The Prototyping Model



Pic Courtesy : https://www.geeksforgeeks.org2

The Prototyping Model

- Requirement Gathering
- Quick Design
- Build a Prototype
- Initial User Evaluation
- Refining Prototype
- Implement Product and Maintain

The Prototyping Model

- It can be used as standalone process model. Prototype can be serve as "the first system".
- Model assist software engineer and customer to better understand what is to be built when requirement are fuzzy.
- Generates working software quickly and early during the software life cycle.
- Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

When to use?

- A customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements.
- The developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take.

Types of Prototype

- Rapid Throwaway
- Evolutionary Prototype
- Incremental Prototype

Advantages

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost and enhance the quality
- Reusability for complex projects.
- Flexibility in design.

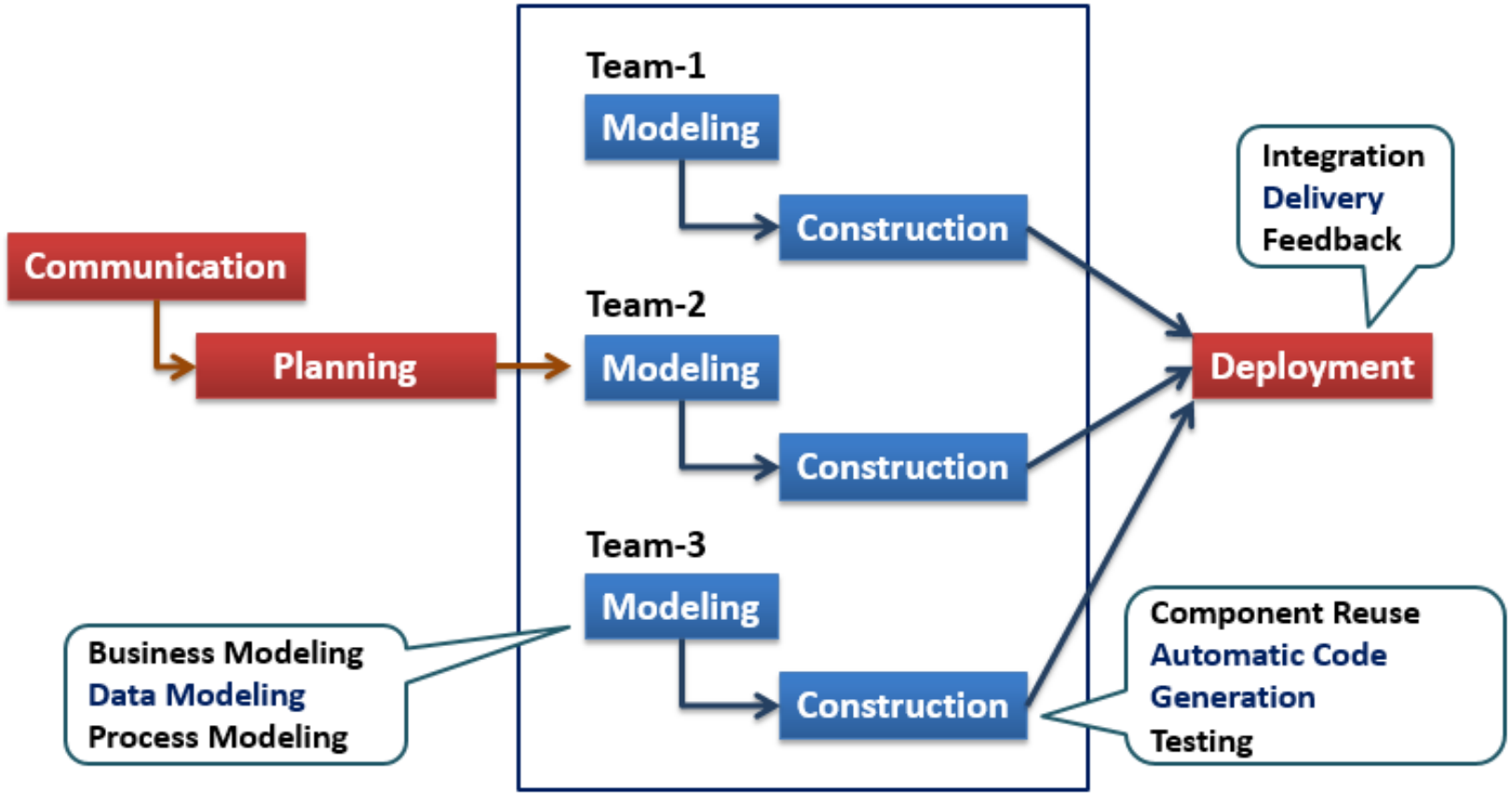
Disadvantages

- Costly
- Too much variation in requirements each time the prototype is evaluated by the customer.
- It is very difficult for the developers to accommodate all the changes demanded by the customer.
- Uncertainty in determining the number of iterations
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

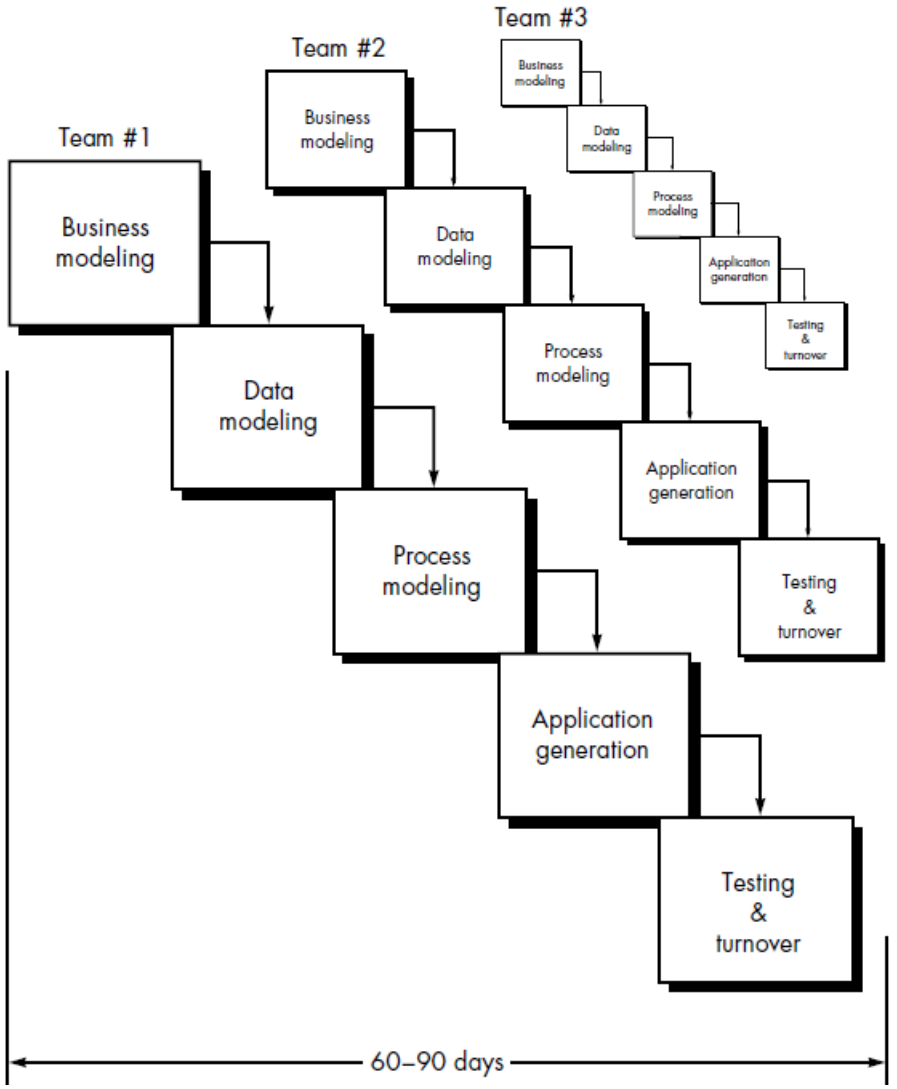
3. RAD (Rapid Application Development) Model

- It is based on prototyping and iterative development with minimal planning involved.
- The functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.
- If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days)

RAD (Rapid Application Development) Model



RAD (Rapid Application Developmen t) Model



3. RAD (Rapid Application Development) Model

- The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- The developments are time boxed, delivered and then assembled into a working prototype.
- This can quickly give the customer something to see and use and to provide feedback.

1. Requirements Planning
   - It involves the use of various techniques used in requirements elicitation like brainstorming to understand business problem
   - It also consists of the entire structured plan describing the critical data, methods to obtain it and then processing it to form final refined model.

2. User Description

- This phase consists of taking user feedback and building the prototype using developer tools.

3. Construction

- In this phase, refinement of the prototype and delivery takes place. All the required modifications and enhancements are too done in this phase.

3. RAD (Rapid Application Development) Model

4. Cutover

- All the interfaces between the independent modules developed by separate teams have to be tested properly.
- Use of powerfully automated tools
- This is followed by acceptance testing by the user.

When to Use ?

- There is a need to create a system that can be modularized in 2-3 months of time.
- High availability of designers and budget for modeling along with the cost of automated code generating tools.
- Resources with high business knowledge are available.

Features of RAD Model

1. Rapid Prototype
   - Allows customer to get quick initial views about the product.
2. Use of powerful development tool
   - Development time goes down eg CASE tool.
3. Active user involvement
   - It increases the acceptability of software.

Advantages of RAD Model

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

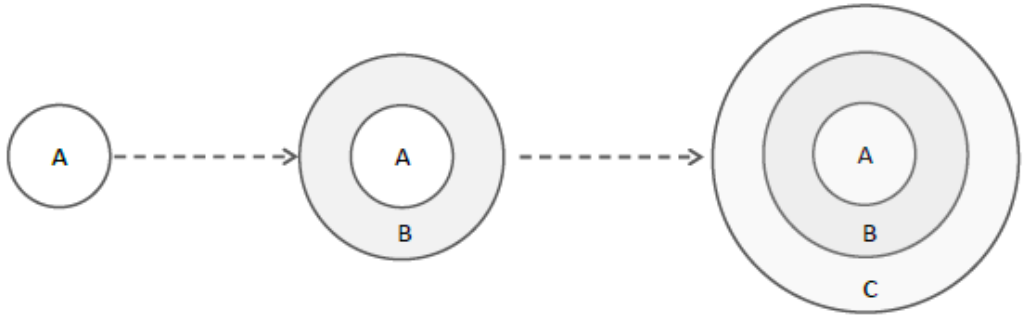
When not to use RAD Model

- Users can’t be involved continuously.
- Tools and reusable component can’t be used.
- Highly skilled and specialized developer are not available.
- Software cannot be a modularized .

4. Evolutionary Process Model

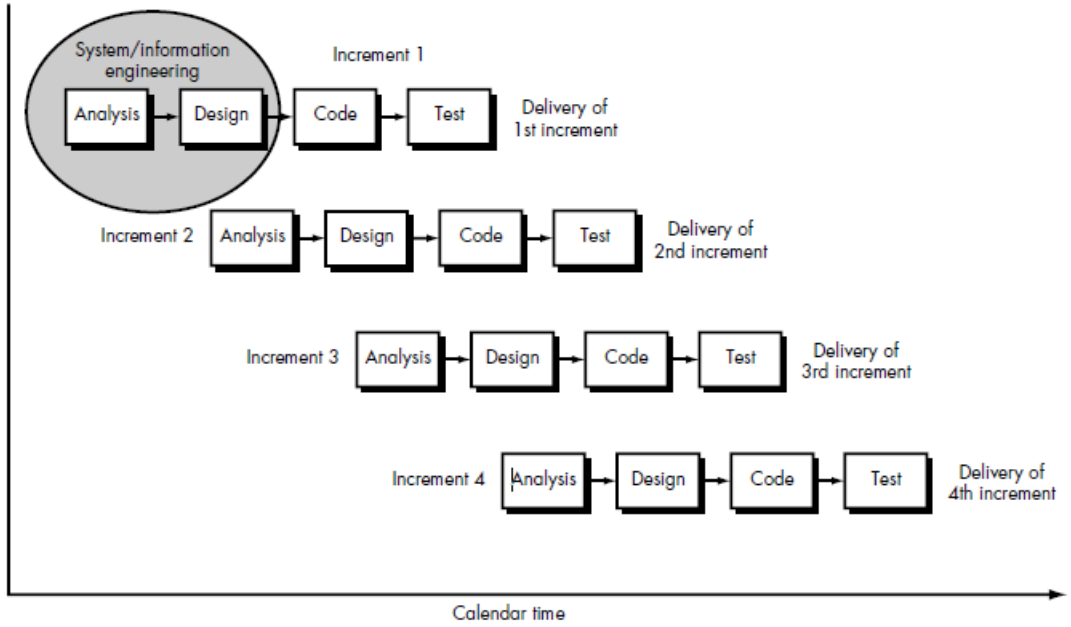
- Produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are iterative.
- Evolutionary models are:
  - Incremental Process Model
  - Spiral Model
  - Concurrent Development Model

4. Evolutionary Process Model



- **Evolutionary model** is also referred to as the **successive versions model** and sometimes as the **incremental model.**
- The development first develops the **core modules (independent)** of the system.
- Each successive version/model of the product is a fully functioning software capable of performing more work than the previous versions/model.
- These are useful for very **large products**, where it is easier to find modules for incremental implementation.

4.1 Incremental Process Model



Pic Courtesy : Software Engineer - A practitioner's Approach By Roger Pressman

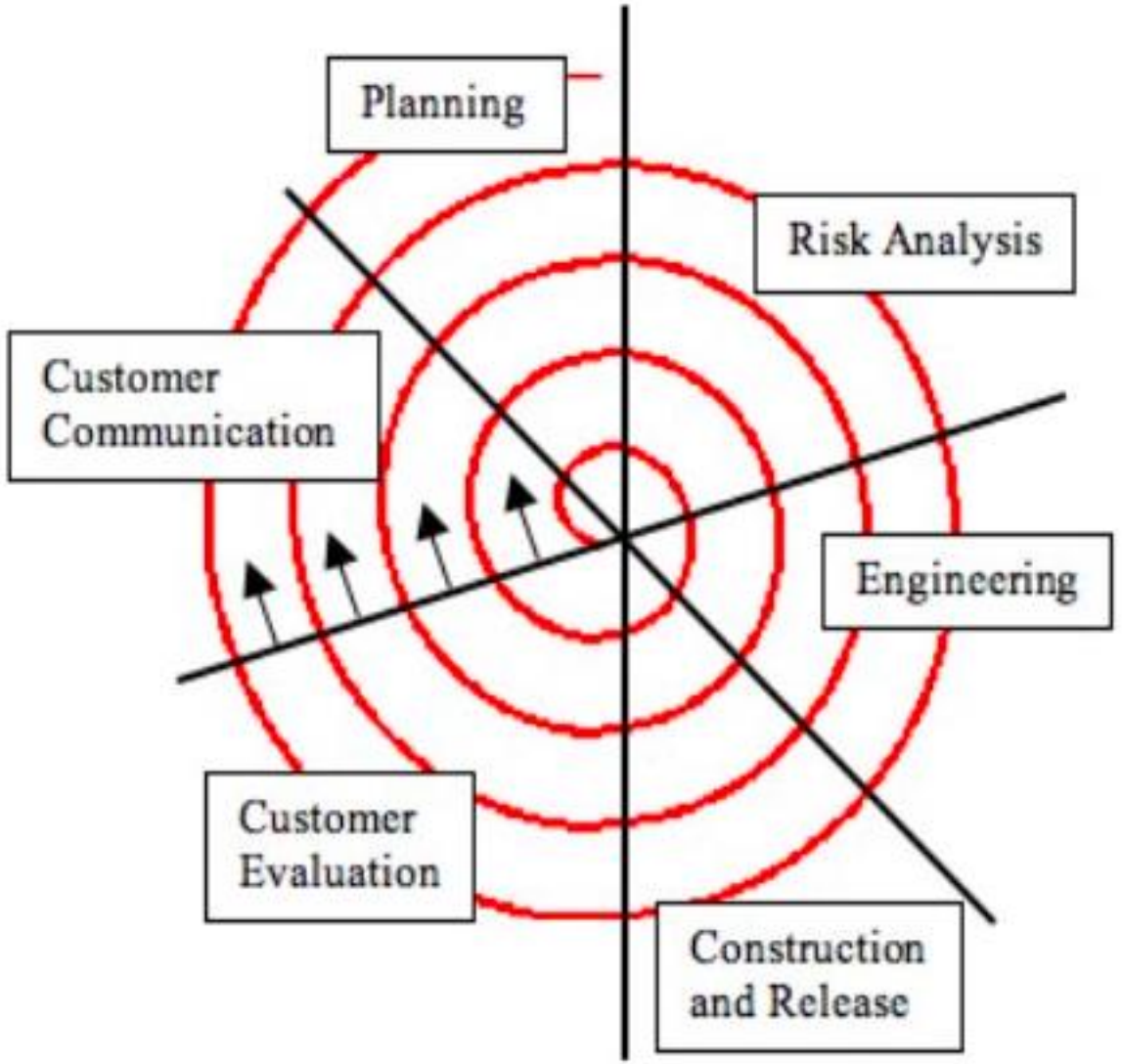
Advantages of Incremental Process Model

- Generates working software quickly and early during the software life cycle.
- It is easier to test and debug during a smaller iteration.
- Customer can respond to each built and developer can accommodate those changes.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during iterations.
- Works with small sized team.

Disadvantages of Incremental Process Model

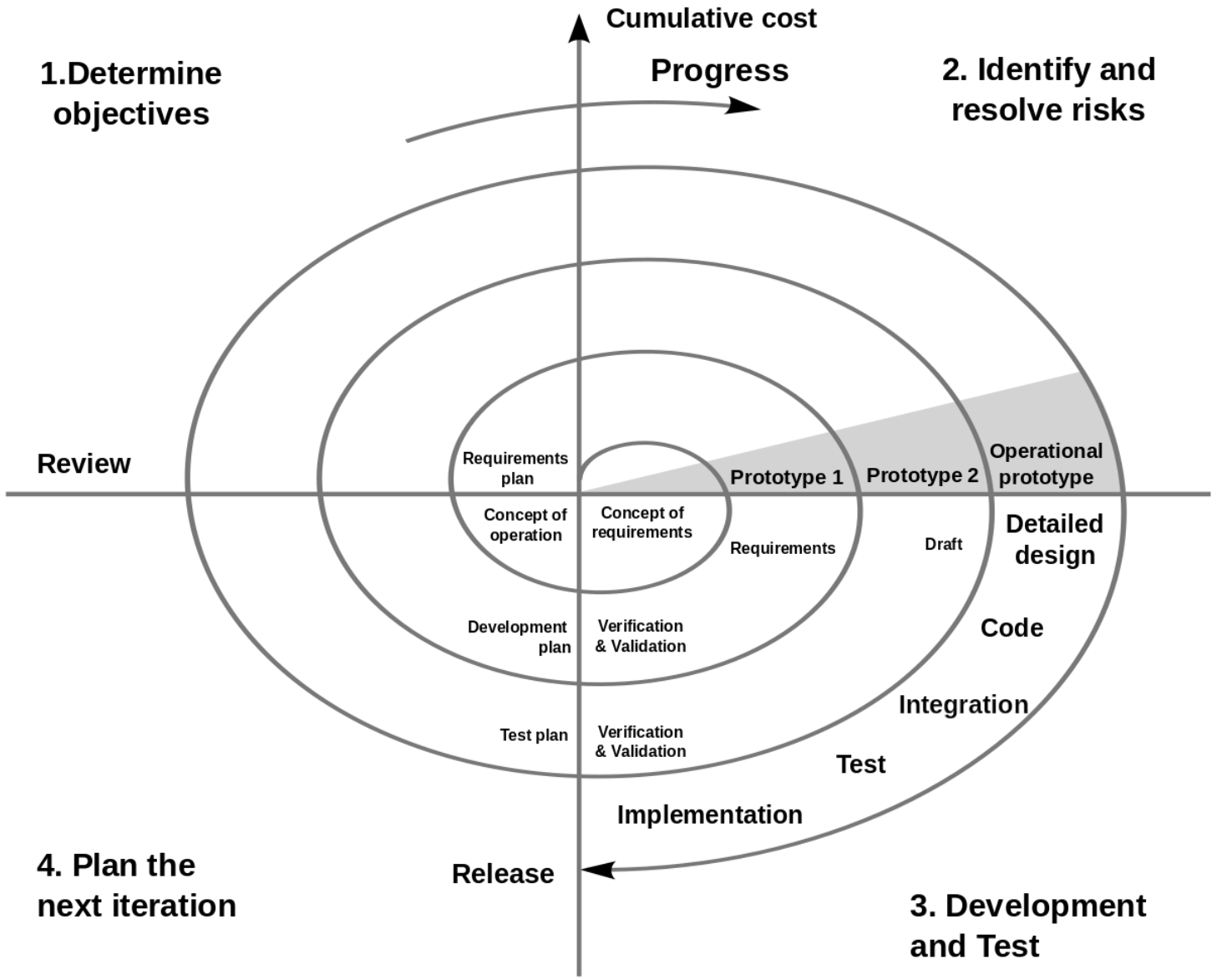
- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle

4.2 Spiral Model



Pic Courtesy :https://www.researchgate.net/publication/224086904

Spiral Model



Spiral Model

- Couples iterative nature of prototyping with the controlled and systematic aspects.
- Using spiral, software developed in a series of evolutionary release.
- Early iteration, release might be on paper or prototype.
- Later iteration, more complete version of software.
- Divided into framework activities (C,P,M,C,D). Each activity represent one segment.
- The spiral model is a realistic approach to the development of large scale system and software.
- Radius of the spiral represent the cost of the project.

Spiral Model Phases

- Identification/Planning
- Risk Analysis
- Engineering (Build + Testing)
- Progress & Customer Feedback

When to use?

- When costs and risk evaluation is important.
- For medium to high-risk projects.
- Users are unsure of their needs.
- Requirements are complex.
- Significant changes are expected.

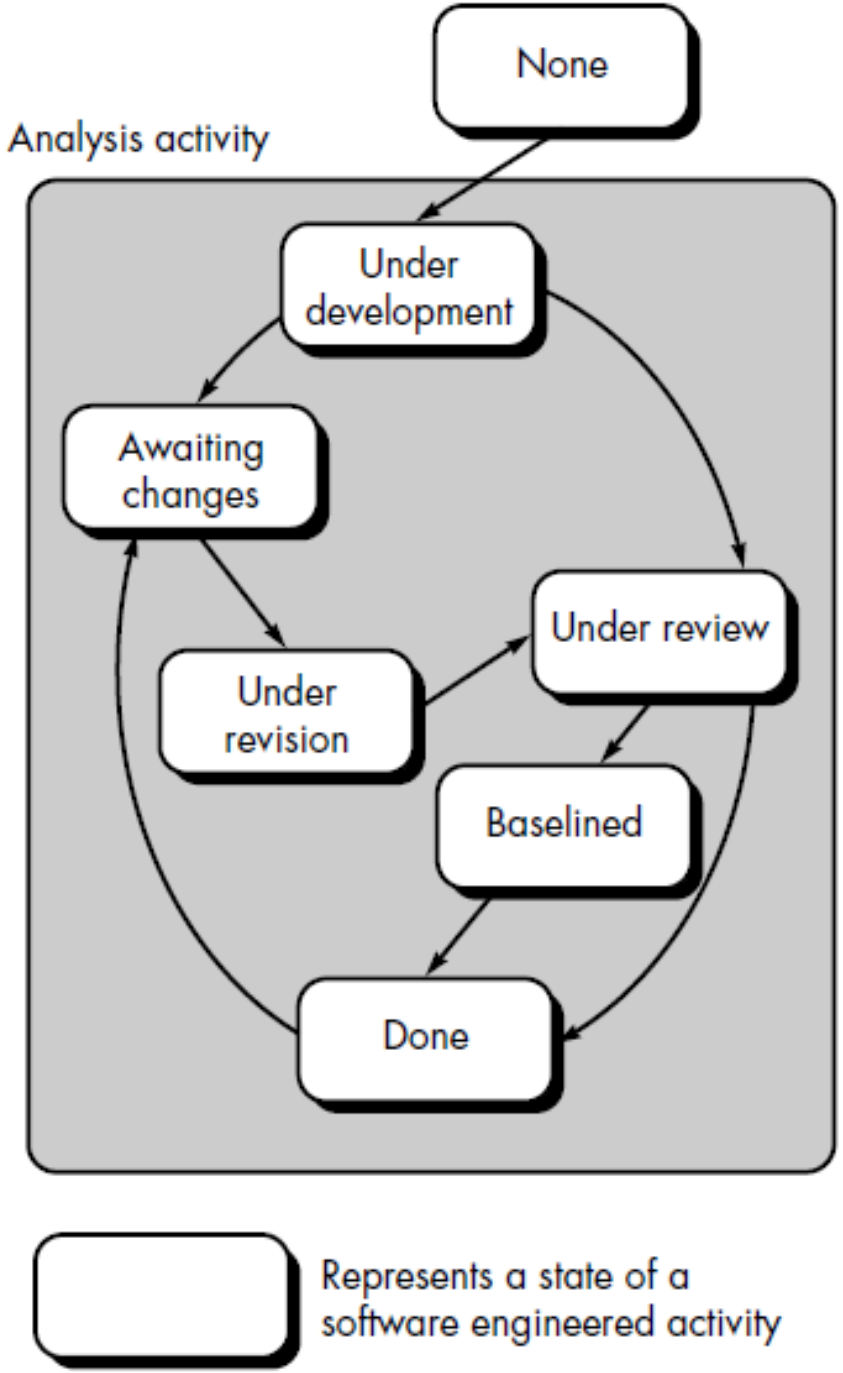
Advantages

- Risk Handling
- Good for large projects
- Flexibility in Requirements
- Customer Satisfaction
- **Prototyping Model** also support risk handling, but the risks must be identified completely before the start of the development work of the project. But in real life project risk may occur after the development work starts, in that case, we cannot use Prototyping Model

Disadvantages

- Complex as it is risk driven.
- Expensive
- Too much dependable on Risk Analysis
- Difficulty in time management
- Requires knowledgeable and experienced staff

4.3 Concurrent Development Model



Pic Courtesy: R.Pressman

Concurrent Development Model

- It represented schematically as series of major technical activities, tasks, and their associated states.
- It is often more appropriate for system engineering projects where different engineering teams are involved.
- The activity “modeling” may be in any one of the states for a given time.
- All activities exist concurrently but reside in different states.

Concurrent Development Model

- E.g. The “analysis” activity (existed in the none state while initial customer communication was completed) now makes a transition into the under development state.
- Analysis activity moves from the under development state into the awaiting changes state only if customer indicates changes in requirements.
- Series of event will trigger transition from state to state.

Advantages

- Applicable to all types of software development processes.
- Easy to understand and use.
- It provide an accurate picture of the current state of a project.

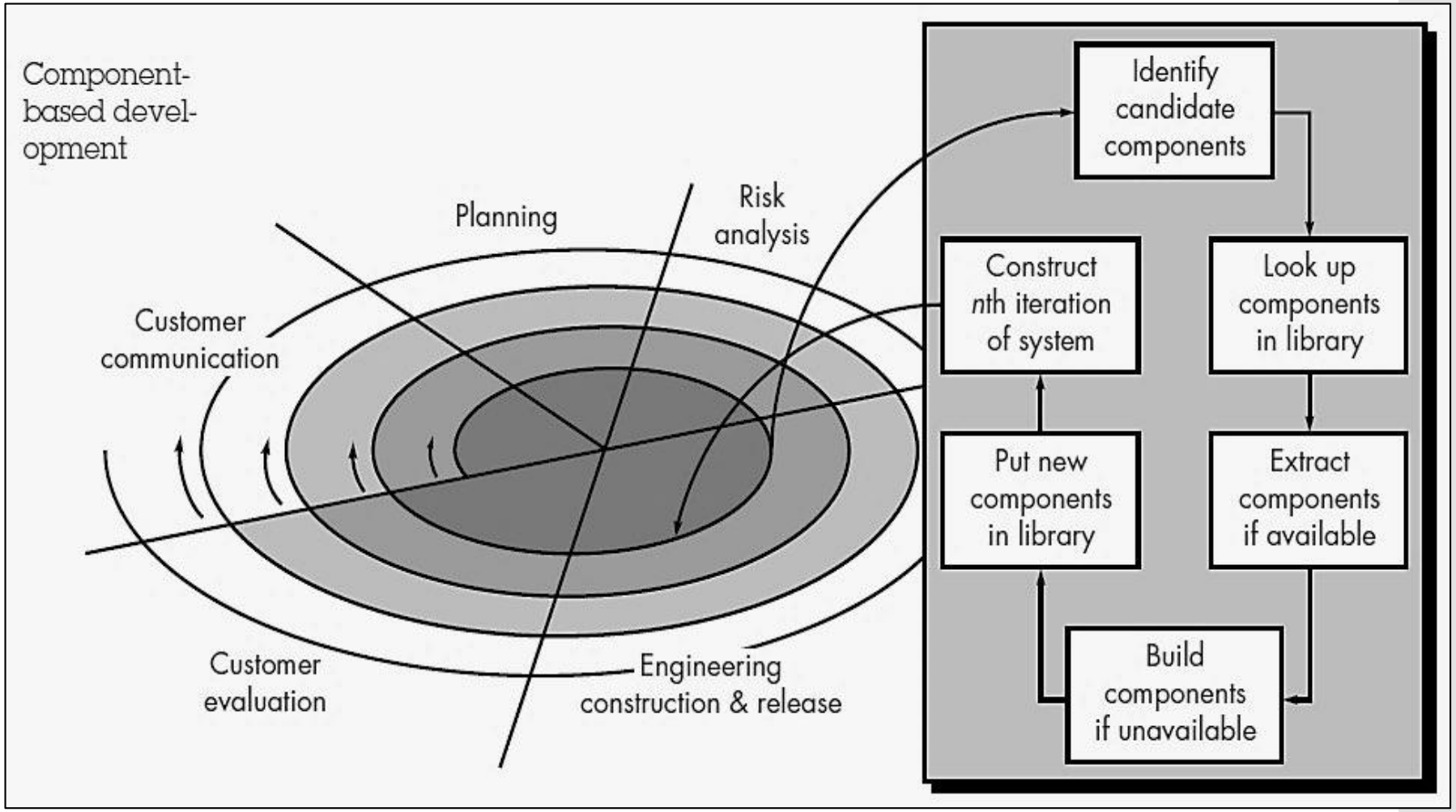
Disadvantages

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

5. Component-based development

- Component-based development (CBD) model incorporates many of the characteristics of the spiral model.
- It is evolutionary by nature and iterative approach to create software.
- CBD model creates applications from prepackaged software components (called classes).

Component-based development



Component-based development

- The engineering activities begin with identification of candidate components.
- Classes created in past software engineering projects are stored in a class library or repository.
- Once candidate classes are identified, the class library is searched to determine if these classes already exist.
- If class is already available in library extract and reuse it.
- If class is not available in library, it is engineered or developed using object-oriented methods.
- The first iteration of the application to be built is then composed, using classes extracted from the library and any new classes built to meet the unique needs of the application.

Advantages

- CBD model leads to software reusability.
- Based on studies, CBD model leads to 70 % reduction in development cycle time.
- 84% reduction in project cost.
- Productivity is very high.

Product and Process

Product

In the context of software engineering, Product includes any software manufactured based on the customer's request. This can be a problem solving software or computer based system. It can also be said that this is the result of a project.

Process

Process is a set of sequence steps that have to be followed to create a project.

The main purpose of a process is to improve the quality of the project.

Product and Process

- If the process is weak, the end product will suffer. But an over-reliance on process is also dangerous.
- People derive as much satisfaction from the creative process as they do from the end product.
  - Like an artist enjoys the brush strokes as much as the framed result.
  - A writer enjoys the search for the proper metaphor (comparison) as much as the finished book.
- As creative software professional, you should also derive as much satisfaction from the process as the end product.
- The duality of product and process is one important element in keeping creative people engaged as software engineering continues to evolve.

THANK YOU