Introduction…

- After taking this course, students will understand
  - what is software engineering;
  - why software engineering is important;
  - How to Plan, develop software and manage a software project by using the software engineering concepts, in-detail.

- At the **end** you should have the ability to plan, execute, and manage small software projects

UNIT-1

Software Engineering and Software Process

Chapter 1
Software and Software Engineering

- Dual role of software
- An IEEE definition of software
- Differences between hardware and software
- Nature of software
- Dealing with legacy software
- Software myths

What is a Software?

&

What is Software Engineering?

Software and Software Engineering

- Software is a **product,** that software professionals build and then support over the long term.

- Software engineering encompasses a **process**, a collection of **methods (practice)** and an array of **tools** that allow professionals to build high-quality computer software.

Nature Of Software

The Nature of Software- Dual Role

Software as a product

- Transforms information - produces, manages, acquires, modifies, displays, or transmits information.
- Delivers computing potential of hardware and networks.

Software as a vehicle for delivering a product

- Supports or directly provides system functionality.
- Controls other programs (operating system).
- Effective communications (networking software).
- Helps build other software (software tools & environments).

What is Software?

IEEE definition of Software:

- ***Software*** *is:* ***(1)*** ***Instructions*** (computer programs) that when executed provide desired features, function, and performance;
- ***(2)*** ***Data structures*** that enable the programs to adequately manipulate information and
- ***(3)*** ***Documentation*** that describes the operation and use of the programs.

What type of Software product developed?

- Software products may be developed for a **particular customer** or may be developed for a **general market**.

- That is, a Software products may be
  - **Generic** - Developed to be sold to a **range of different customers** e.g. PC software such as Excel or Word.
  - **Customized** - Developed for a **single customer** according to their specification.

Characteristics of Software

Differences between Software and Hardware

Hardware

- Manufactured
- wear out
- Built using components
- Relatively simple

Software

- Developed/ engineered
- deteriorate
- Custom built
- Complex

Manufacturing vs. Development

- Once a **hardware** product has been manufactured, it is difficult **or impossible to modify**.
  - In contrast, **software products** are routinely **modified** and **upgraded**.

- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.

- Unlike hardware, software costs are concentrated in design rather than production.

Component Based vs. Custom Built

- Hardware products typically employ many **standardized design components**.
- Most software continues to be custom built.
- The software industry does seem to be moving (slowly) towards component-based construction.

Software characteristics

Summary:

1) Software is **developed or engineered**,it is not manufactured.
2) Software does not “wear out” but it does **deteriorate**.
3) Software continues to be **custom built**, as industry is moving toward **component based construction**.

Characteristics of Software…

Software	Hardware
Logical system element	Physical system element
Developed/engineered	Manufactured
Doesn't ware-out but deteriorate - no spare parts	Ware-out - yes, with spare parts
Usually custom-built	Assembled from existing component

Software Application Domains

1) System software
2) Application software
3) Engineering / scientific software
4) Embedded software
5) Product line software
6) Web applications
7) Artificial intelligence software

1. System Software

- System software is a **collection of programs written to service other programs**.
- It is characterized by
  - heavy **interaction with computer hardware,**
  - heavy **usage by multiple users,**
  - concurrent operation that requires scheduling,
  - **resource sharing**, and sophisticated process management ,
  - complex data structures , and multiple external interfaces.

Ex. Compilers, operating system, drivers etc.

2. Application Software

- Application software consists of **standalone programs** that solve a **specific business need**.

- Application software is used to control the business function in real-time.

- For **Eg: Transaction processing**, real-time manufacturing process control

3. Engineering /Scientific software

- Characterized by "**number crunching**" algorithms.

- Applications range from
  - astronomy to volcanology,
  - automotive stress analysis to space shuttle orbital dynamics, and
  - molecular biology to automated manufacturing.

Ex. Computer Aided Design (CAD), System Simulation etc.

4. Embedded Software

- It **resides within a product or system** and is used to implement and control features and functions for the end user and for the system itself.

- Embedded software can perform limited and intended functions.

Ex. keypad control for a microwave oven.

5. Product line software

- Designed to **provide a specific capability for use by many different customers**, product line software can focus on a limited and intended marketplace.

Ex. Word processing, spreadsheet, Computer Graphics, multimedia, etc.

6. Web Applications

- Web apps can be little more than a set of linked hypertext files.

- It evolving into sophisticated computing environments that **not only provide standalone features, functions but also integrated with corporate database and business applications**.

- Eg: E-commerce application, Online shopping etc.

7. Artificial Intelligence software

- AI software **makes use of non-numerical algorithms** to solve complex problems that are not amenable to computation or straightforward analysis.

Ex. Robotics, expert system, game playing, etc.

Software Applications—New Categories

- **Open world computing**—pervasive, distributed computing
- **Ubiquitous computing**—wireless networks
- **Netsourcing** (pay as you use)—the Web as a computing engine
- **Open source**—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also …
  - **Data mining**
  - **Grid computing**
  - **Cognitive machines**
  - **Software for nanotechnologies**

Legacy software

- Legacy software is an **old and outdated** program that is still used to perform a task for a user, even though newer and more efficient options are available.

  Ex--factory's computer system running on an old version of Windows because there is not a need to invest in the most updated software.

Legacy Software - Characteristics

- Support core business functions.
- Have longevity and business criticality.
- Exhibit poor quality
  - Convoluted code, poor documentation, poor testing, poor change management .

Legacy Software -Why it must change?

- The software must be **adapted** to meet the needs of new computing environments or technology.
- The software must be **enhanced** to implement new business requirements.
- The software must be **extended** to make it interoperable with other more modern systems or databases.
- The software must be **re-architected** to make it viable within a network environment.

Software Engineering

Software Engineering

- **In order to build software that is ready to meet the challenges of 21st century, must recognize few realities**
- **Some realities:**
  - a concerted effort should be made to **understand the problem** before a software solution is developed.
  - **design** becomes a **pivotal activity.**
  - software should exhibit **high quality.**
  - software should be **maintainable.**

Software Engineering

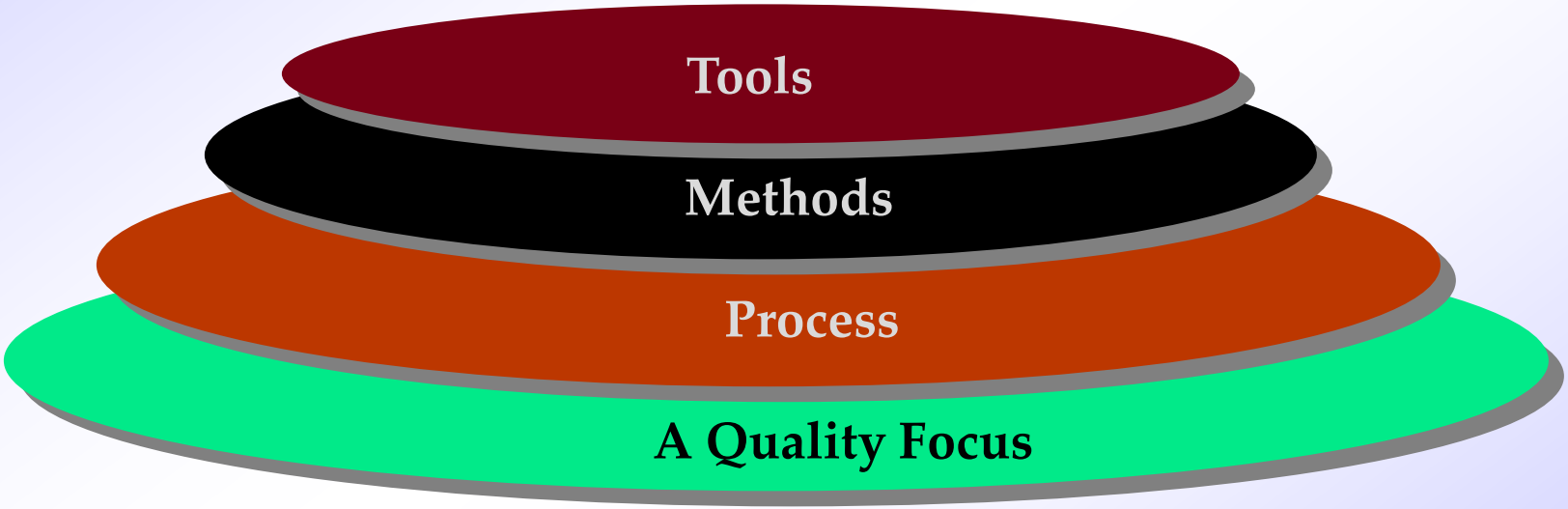
- The seminal definition:
  - Software engineering is an establishment and use *of* ***sound engineering principles*** in order to obtain ***economically*** software that is ***reliable and works efficiently*** ***on*** ***real machines*****.**

Software Engineering

- **The IEEE definition**:
  - **Software Engineering**: (1) The application of ***a systematic, disciplined, quantifiable approach*** to the ***development, operation, and maintenance*** of software; that is, the application of engineering to software.
  - (2) The study of approaches as in (1).

Software Engineering layers

- Software Engineering is a **layered Technology**.



The Software Process

- A process is a collection of **activities, actions and tasks** that are performed when some work product is to be created.

- **An *activity*** *strives(try) to* ***achieve a broad objective*** and is applied regardless of the application domain, size of the project, with which software engineering is to be applied.

e.g., communication with stakeholders.

The Software Process…

- **An *action*** *encompasses* a **set of tasks** that produce a major work product .

Ex--an architectural design model.

- **A *task*** *focuses on a small, but well-defined objective* that produces a tangible outcome.

Ex--*conducting a unit* test.

Process framework

- A ***process framework*** *establishes the foundation for a complete software engineering* process by
  - identifying a small number of ***framework activities*** that are applicable to all software projects and
  - a set of ***umbrella activities*** *that are applicable* across the entire software process.

Framework Activities

- A **generic process framework** for software engineering encompasses **five activities**:

1.Communication

2.Planning

3.Modeling

- Analysis of requirements
- Design

4.Construction

- Code generation
- Testing

5.Deployment

1. Communication

- **Before any technical work can commence,** it is important to **communicate and collaborate with the customer.**
- The intent is **to understand** stakeholders' **objectives for the project .**
- To **gather requirements** that helps to define software features and functions.

2. Planning

- **Any complicated journey can be simplified if a map exists.**
- A software project is a complicated journey, and the planning activity creates a “map” that helps to guide the team as it makes the journey.
- The map—called a ***software project plan—defines the software engineering work*** *by describing*
  - the technical tasks to be conducted,
  - the risks that are likely,
  - the resources that will be required,
  - the work products to be produced, and
  - a work schedule.

3. Modeling

- Create a “sketch” of the thing so that one can understand the big picture.
- In other sectors, a landscaper, a bridge builder, an aeronautical engineer, a carpenter, or an architect, is the one who perform this (modeling) activity.

- Here, a **software engineer** does the same thing by **creating models** to **better understand** software requirements and the design, that will achieve those requirements.

4. Construction

- This activity **combines**
  - **code generation and**
  - **the testing**

  that is required to uncover errors in the code.

5. Deployment

- The software is **delivered** to the **customer** who **evaluates** the delivered product and provides **feedback** based on the evaluation.

Umbrella Activities

- **Umbrella activities** occur throughout the software process and focus primarily on **project management, tracking, and control.**
- Typical umbrella activities are:

Umbrella Activities

1. Software project tracking and control.
2. Risk management.
3. Software quality assurance.
4. Technical reviews.
5. Measurement.
6. Software configuration management.
7. Reusability management.
8. Work product preparation and production.

1. **Software project tracking and control—** Allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
2. **Risk management—** Assesses risks that may affect the outcome of the project or the quality of the product.
3. **Software quality assurance—** Defines and conducts the activities required to ensure software quality.

4. **Technical reviews—** assesses software engineering work products in an effort to **uncover and remove errors** before they are propagated to the next activity.

5. **Measurement—** defines and collects process, project, and product measures that assist the team in delivering software that **meets stakeholders needs.**

- It can be **used** in conjunction with all other **framework and umbrella activities.**

6. **Software configuration management—** manages the **effects of change** throughout the software process.
7. **Reusability management—** defines criteria for **work product reuse** (including software components) and establishes mechanisms to achieve **reusable components.**
8. **Work product preparation and production—** encompasses the activities required to **create work products** such as **models, documents, logs, forms, and lists.**

Software Engineering Practice

The Essence of Practice

- George Polya outlined the essence of problem solving and the essence of software engineering practice:

1. ***Understand the problem*** (communication and analysis).
2. ***Plan a solution*** (modeling and software design).
3. ***Carry out the plan*** (code generation).
4. ***Examine the result for accuracy*** (testing and quality assurance).

1. Understand the Problem

- ***Who has a stake in the solution to the problem*****?** That is, who are the stakeholders?
- ***What are the unknowns****?* What data, functions, and features are required to properly solve the problem**?**
- ***Can the problem be compartmentalized****?* Is it possible to represent smaller problems that may be easier to understand?
- ***Can the problem be represented graphically****?* Can an analysis model be created?

2. Plan the Solution

- ***Have you seen similar problems before?*** Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- ***Has a similar problem been solved?*** If so, are elements of the solution reusable?
- ***Can subproblems be defined?*** If so, are solutions readily apparent for the subproblems?
- ***Can you represent a solution in a manner that leads to effective implementation?*** Can a design model be created?

3. Carry Out the Plan

- ***Does the solution conform to the plan?*** Is source code traceable to the design model?
- ***Is each component part of the solution provably correct?*** Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

4. Examine the Result

- ***Is it possible to test each component part of the solution?*** Has a reasonable testing strategy been implemented?
- ***Does the solution produce results that conform to the data, functions, and features that are required?*** Has the software been validated against all stakeholder requirements?

Software Myths

Beliefs about software and the process used to build it.

•Myths have **number of attributes** that have made them insidious (i.e. dangerous).

• **Misleading Attitudes -** caused serious problem for managers and technical people.

Management myths

Managers in most disciplines, are often under **pressure to maintain budgets, keep schedules on time, and improve quality.**

Management myths…

Myth1:

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?

Reality :

- Are software practitioners aware of existence standards?
- 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?

---Not used, not up to date, not complete, not focused on quality, time, and money.

Management myths…

Myth2: If we get behind schedule, we can add more programmers and catch up

Reality: Software development is not a mechanistic process like manufacturing. **Adding people to a late software project makes it later.**

- People can be added but only in a planned and well-coordinated manner.

Myth3: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality:---Software projects need to be controlled and managed

Customer Myths

Myth1: A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.

Reality: A poor up-front definition is the major cause of failed software efforts. **A formal and detailed description** of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is **essential**. These characteristics can be determined only after thorough communication between customer and developer.

.

Customer Myths

Myth2: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: Customer can review requirements and recommend modifications with relatively little impact on cost.

- When changes are requested during software design, the cost impact grows rapidly.

Practitioner's Myths

Myth1: *Once we write the program and get it to work, our job is done*.

Reality: "the sooner you begin 'writing code', the longer it will take you to get done".

- Actually 60% to 80% of all effort will be expended after software is delivered to the customer for the first time.
- **Myth2:** *Until I get the program "running" I have no way of assessing its quality.*
- **Reality:** the *technical review*– a software reviews are a "quality filter" that have been found more effective than testing for finding certain classes of software defects.

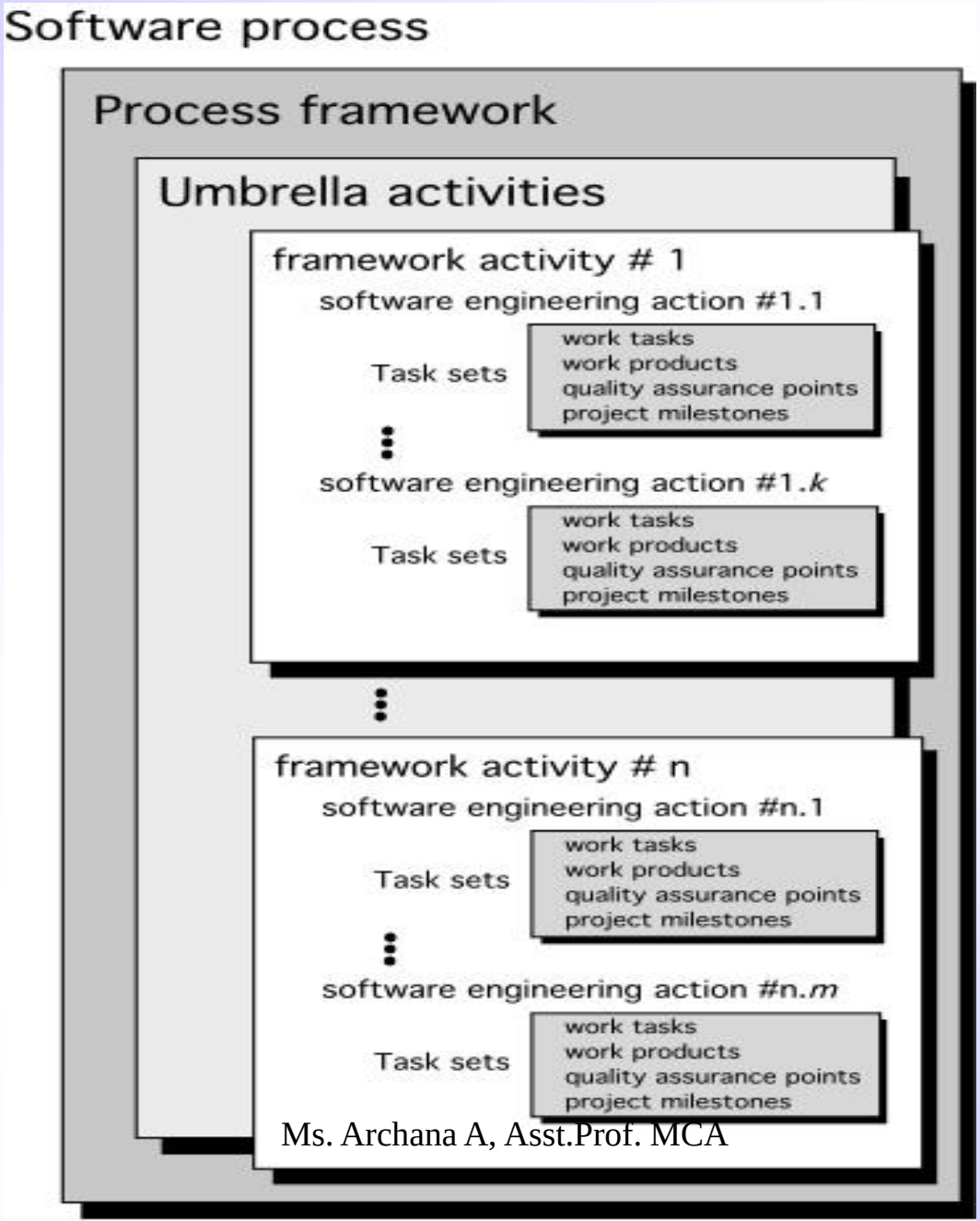
Practitioner's Myths…

- **Myth3:** *Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.*
- **Reality:** Software engineering is not about creating documents. It is about **creating a quality product**.
- Better **quality leads** to reduced rework and reduced rework results in faster delivery times.

Chapter 2

Process Models

A Generic Process Model



Generic Process Model…

- Referring to figure, Each ***framework activity*** is populated by a set of ***software engineering actions***.
- Each ***Software Engineering action*** is defined by a ***task set***, that identifies
  - the work task that are to be completed,
  - the work products that will be produced,
  - the quality assurance points that will be required and
  - the milestones that will be used to indicate progress.

Identifying a Task Set

- A task set defines the **actual work** to be done to accomplish the objectives of a software engineering action.
  - A list of the **task** to be **accomplished**
  - A list of the **work products** to be **produced**
  - A list of the **quality assurance filters** to be applied

Task Set example:

- For a **small project**, the task set (requirements gathering) might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

For a **larger**, more complex software project, the task set include:

1.Make a **list of stakeholders** for the project.

2. **Interview each stakeholder** separately to determine overall wants and needs.

3. **Build a preliminary list of functions** and **features** based on stakeholder's input.

4. Schedule a series of facilitated application specification meetings.

5. Conduct meetings.

6. Produce informal user scenarios as part of each meeting.

7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to **prioritize** requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Process Assessment & Improvement

- The existence of a software process is no guarantee that
  - Software will be delivered on time,
  - it will meet the customer's need, or
  - it will exhibit the technical characteristics that will lead to long-term quality characteristics.
- So the process need to be assessed. And the process assessment directly leads to the process improvement.

- A number of different approaches to software process assessment and improvement have been proposed over the past few decades

Process Assessment and Improvement…

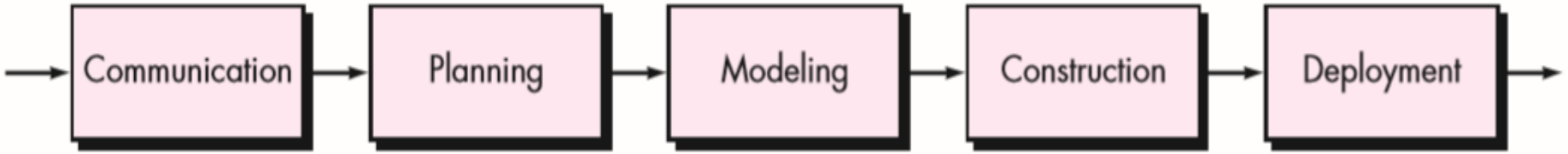
- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step **process assessment model** that incorporates five phases: initiating, diagnosing, establishing, acting and learning.
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)—provides a diagnostic technique for assessing the relative maturity of a software organization**; uses the SEI CMM as the basis for the assessment [Dun01]
- **SPICE—The SPICE (ISO/IEC15504)** standard defines a set of requirements for software process assessment. The intent of the standard is to **assist organizations in developing an objective** evaluation of the efficacy of any defined software process. [ISO08]
- **ISO 9001:2000 for Software—**a generic standard that applies to any organization that wants to **improve the overall quality of the products, systems, or services** that it provides.

Process flow

- Software Process follow the **road map(planning)** to develop the best product.
- **Process flow—**describes ***how the framework activities*** and the ***actions*** and ***tasks*** that occur within each framework activity are organized with respect to sequence and time.
- A process flow may be of **four** types
  - **Linear process flow**
  - **Iterative process flow**
  - **Evolutionary process flow**
  - **Parallel process flow**

1. Linear Process flow

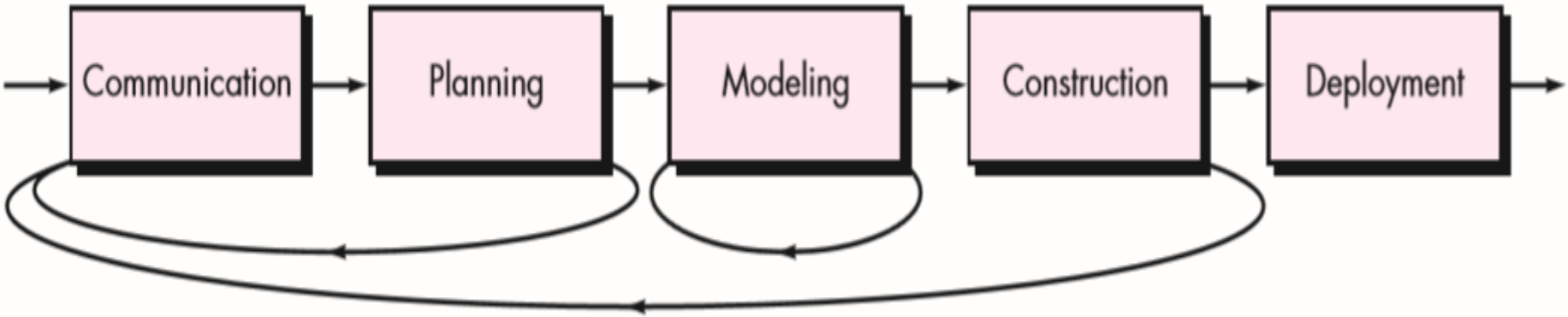
- ***A linear process flow executes each of the five framework activities in sequence***



(a) Linear process flow

2. Iterative Process Flow

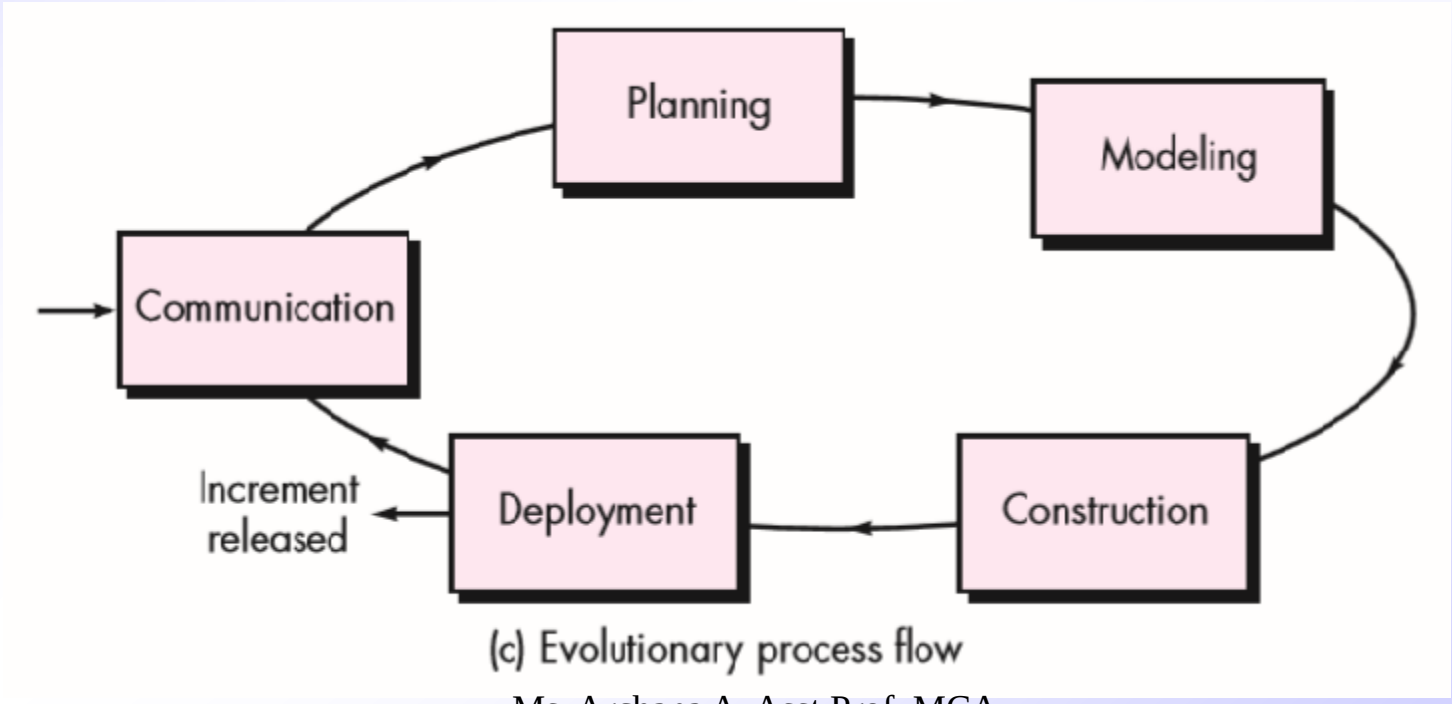
- **An *iterative process flow* *repeats* *one or more of the activities before proceeding to the* next.**



(b) Iterative process flow

3. Evolutionary Process Flow

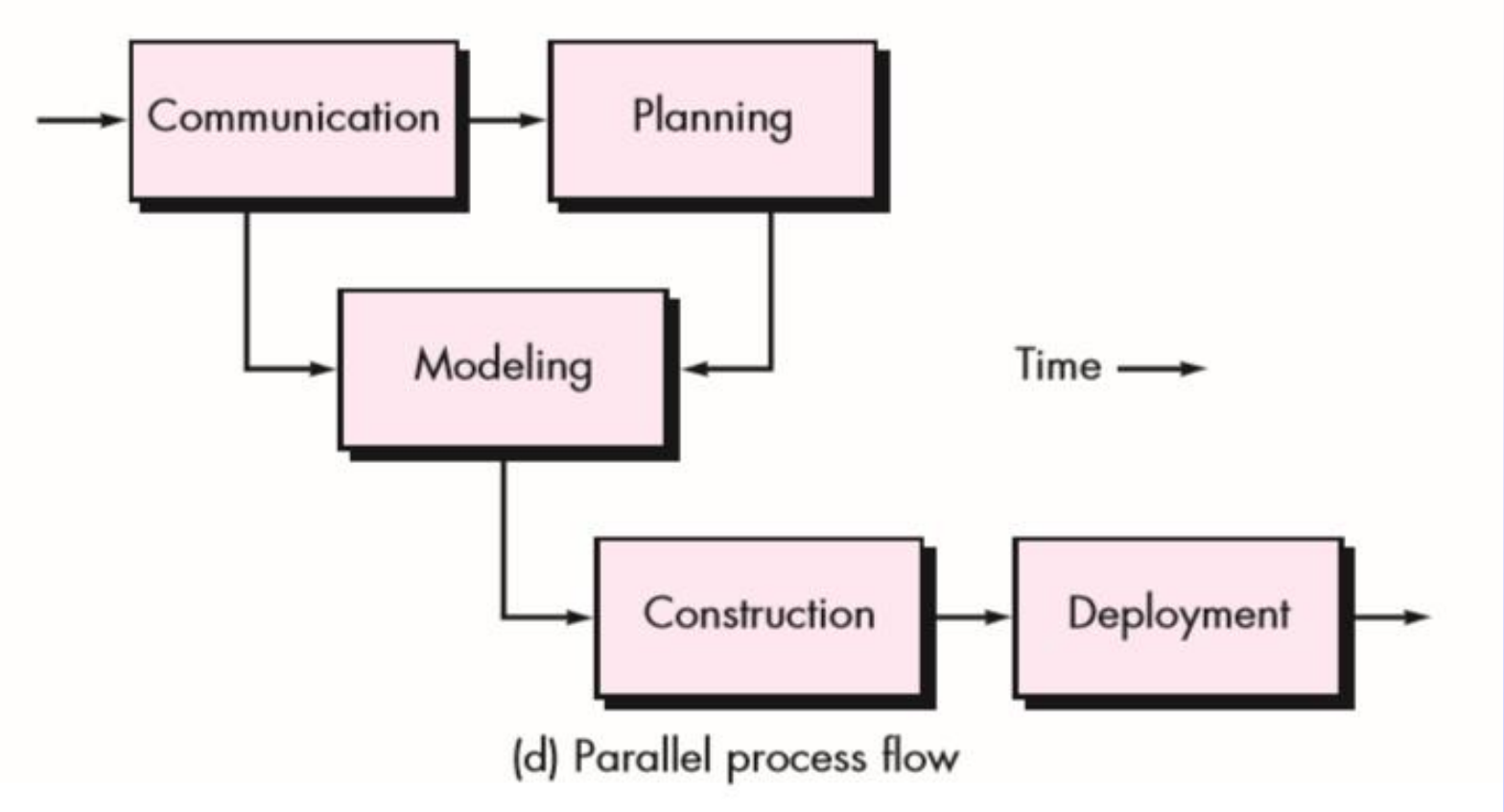
- **An *evolutionary process flow executes the activities in a "circular"* manner.**
- **Each circle pass through the five activities, which leads to a more complete version of the software.**



(c) Evolutionary process flow

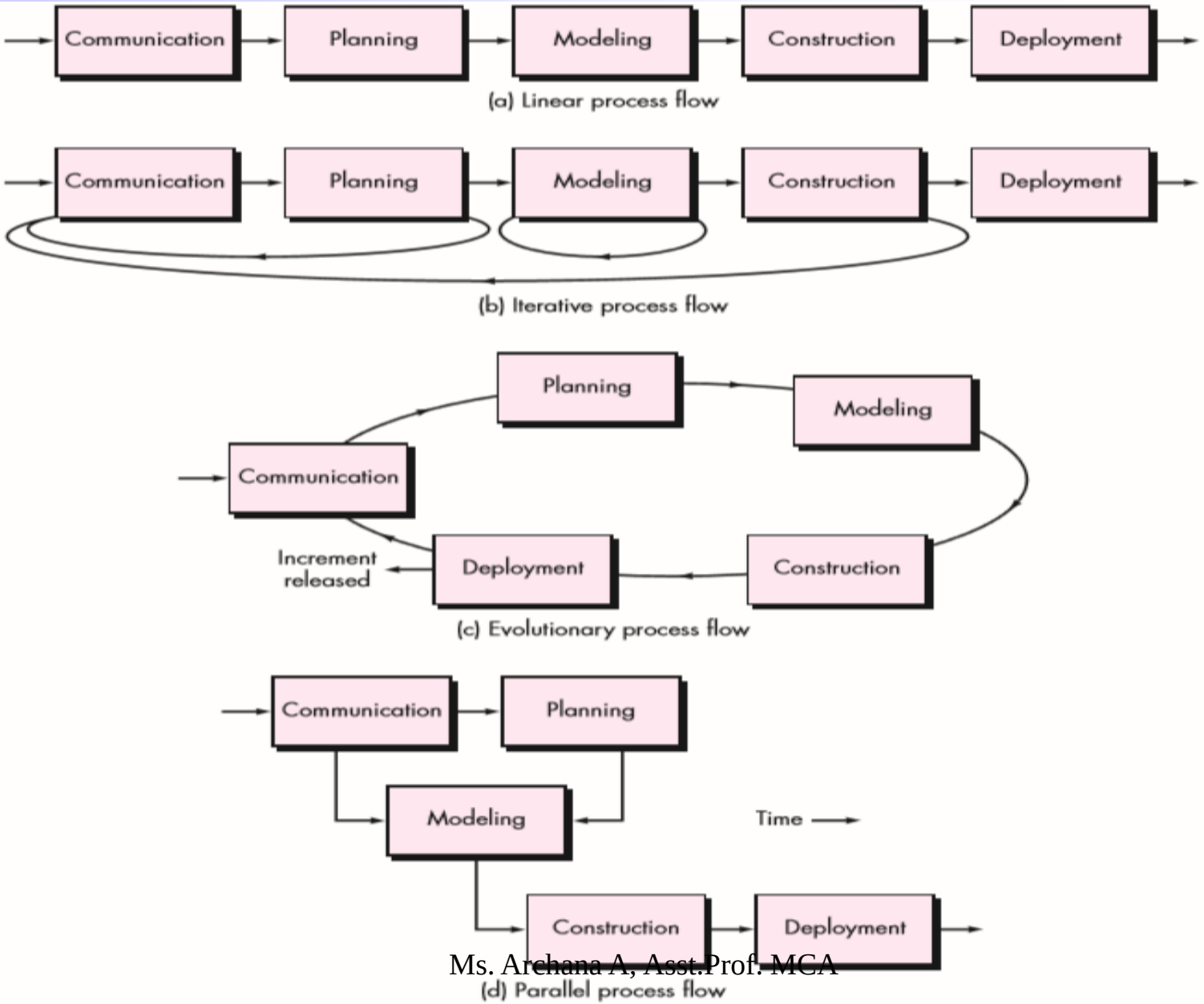
4. Parallel Process Flow

- **A *parallel process flow executes one or* more activities in parallel with other activities**



(d) Parallel process flow

Process Flow types…



Software Process Model

- **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.

1. Prescriptive Model

- **Prescriptive process models advocate an orderly approach to software engineering**
  - **Organize framework activities in a certain order.**
- Model have brought a certain amount of useful **structure to software engineering work** and have provided a reasonably effective road map for software teams.
- Software engineer choose process framework that includes activities like; **Communication, Planning, Modeling, Construction, Deployment.**

Prescriptive Model…

- Each elements are inter related to one another (called workflow).
- Prescriptive model are also known as "**Traditional models**".

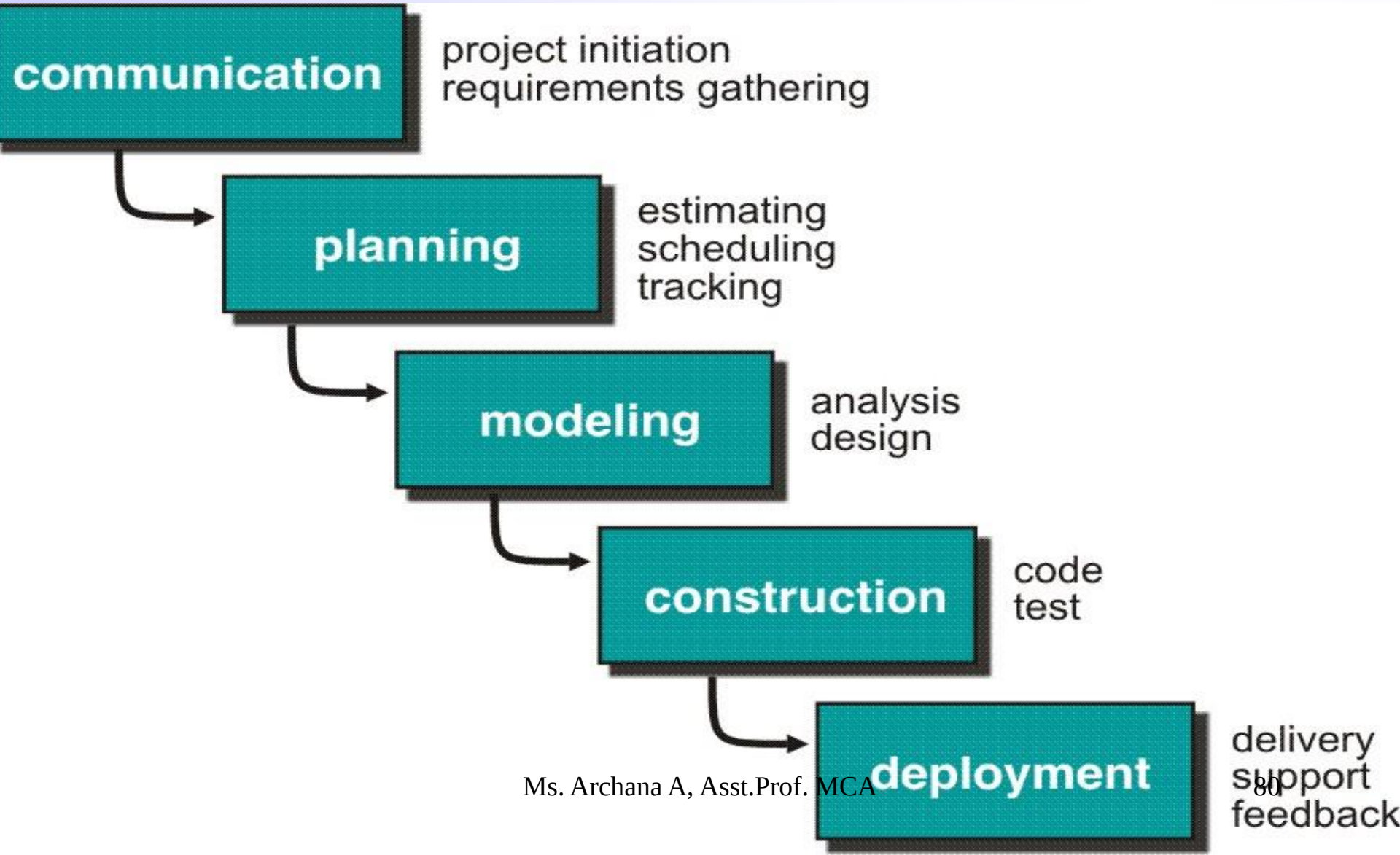
Traditional models

- **The Waterfall Model.**
- **The V-Model.**
- **Incremental Process Model.**
- **Evolutionary Process Models**
  - **Prototyping**
  - **Spiral Model**

1. The Waterfall Model

- The **waterfall model**, sometimes called the ***classic life cycle***, suggests a **systematic sequential approach to software development** that begins with **customer specification of requirements and progresses** through planning, modeling, construction, deployment.

Waterfall Model or Classic Life Cycle



Waterfall Model…

- **Requirement Analysis and Definition**: **What** - The systems services, constraints and goals are defined by customers with system users.
- **Scheduling tracking** -
  - Assessing progress against the project plan.
  - Require action to maintain schedule.
- **System and Software Design**: **How** –It establishes the overall **system architecture**. Software design involves fundamental system abstractions and their relationships.

- **Integration and system testing**: The individual program unit or **programs are integrated** and tested as a complete system to **ensure** that the **software requirements have been met**. After testing, the software system is delivered to the customer.
- **Operation and Maintenance**: longest phase of the software life cycle. The system is installed and put into **practical use**. Maintenance involves **correcting errors** which were not discovered in earlier stages of the life-cycle.

Limitations of the waterfall model

- The nature of the requirements will not change very much During development and evolution.
- It also means that customers can not use anything until the entire system is complete.
- The **model implies** that we should attempt to **complete a given stage before moving on to the next stage .**
- Some teams sit ideal for other teams to finish.
- This model is only appropriate when the **requirements are well-understood(**requirements are fixed).

Problems:

- **Real projects** rarely follow the sequential flow that the model propose.
- **Difficult** for the customer to state all the requirement **explicitly**.
- The customer must have **patience** - working version of program will not be available until programs not getting change fully.
- High amounts of risk and uncertainty.

2. The V-Model

- There is **no fundamental difference** between the waterfall model and the V-model.

- The V-model provides a way of **visualizing how verification and validation actions** are applied to earlier engineering work.

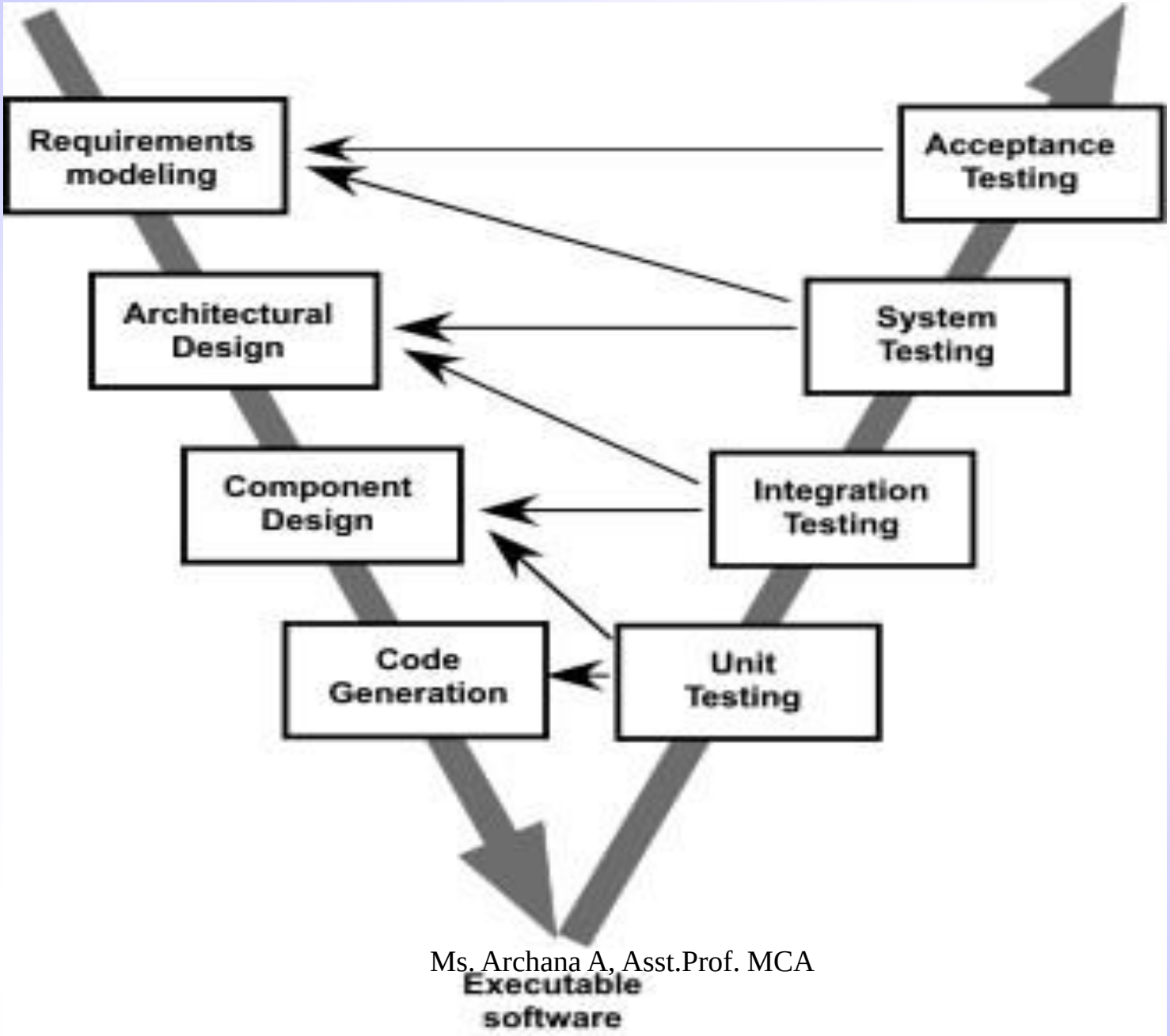
The V-Model…

- A **variation** in the representation of the **waterfall** model is called the ***V-model.***

- This model depicts the **relationship of quality assurance actions** to the **actions associated with communication, modeling,** and **early construction** activities.

- Performing a series of tests → quality assurance actions.

The V-Model


Requirements modeling
Architectural Design
Component Design
Code Generation
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Executable software

V – Model…

- As a software team moves down the left side of the **V**, basic **problem requirements** are refined into progressively more detailed form.
- **Once code has been generated** the team moves up the right side of the V, essentially performing a series of tests (Quality assurance actions) that validate each of the models created.
- V- model provides a way of visualizing how verification & validation actions are applied to earlier engineering work.

Advantages of V-Model:

- It is very easy to understand, manage and apply.
- This is a highly disciplined model and phases are completed one at a time.
- It works well for smaller projects where requirements are **very well understood**.
- This model is used in the **medical development field**, as it is strictly disciplined domain.

Disadvantages of V-Model:

- It has high risk and uncertainty.
- It is not a good model for complex and object-oriented projects.

3. The Incremental Model

- The incremental model delivers a **series of releases,** called **increments.**

The Incremental Model…

- Rather than deliver the system as a single delivery, the development and delivery is **broken down** into **increments** with each increment delivering part of the required functionality.

- First Increment is often **core product:**
  - Includes **basic requirement,**
  - **Many supplementary features** (known & unknown) remain undelivered.

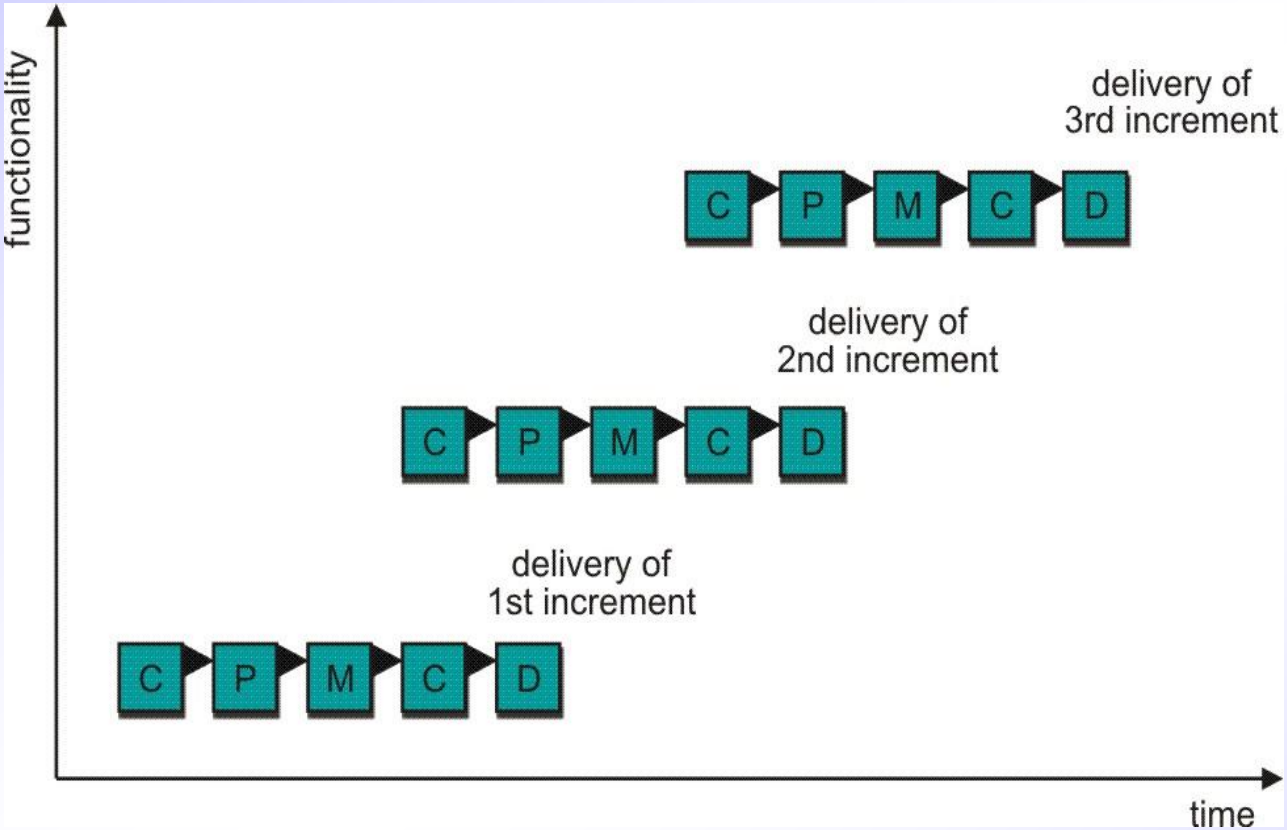
The Incremental Model…

- A plan of **next increment** is prepared
  - Modifications of the first increment,
  - Additional features of the first increment

- It is particularly useful when enough **staffing is not available** for the whole project.

- Increment can be planned to manage technical risks.
- Incremental model focus more on **delivery of operation product with each increment**.

Incremental Process Model…



C- Communication
P - Planning
M – Modeling
C - Construction
D - Deployment

Delivers software in **small but usable pieces**, each piece builds on pieces already delivered

For example: word-processing software developed---4 increments

1st increment→ Deliver basic file management, editing, and document production functions.

2nd increment→ More sophisticated editing and document production capabilities.

3rd increment→ Spelling and grammar checking.

4th increment→ Advanced page layout capability.

Advantages :

- Using this model some **working functionality** can be developed **quickly** and **early** in the life cycle.
- **Results are obtained early and periodically**. Parallel development can be planned.
- Progress can be **measured**.
- **Risks are identified** and **resolved** during **iteration**. Easier to manage risk - High risk part is done first. i,e., **Risk analysis is better**.
- It is **suited for large and mission-critical projects.**

Disadvantages:

- In this model more **resources and management attention** is required.
- System architecture or **design issues** may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- It is **not suitable for smaller projects**.
- **Highly skilled resources** are required for **risk analysis.**

4. Evolutionary Process Model

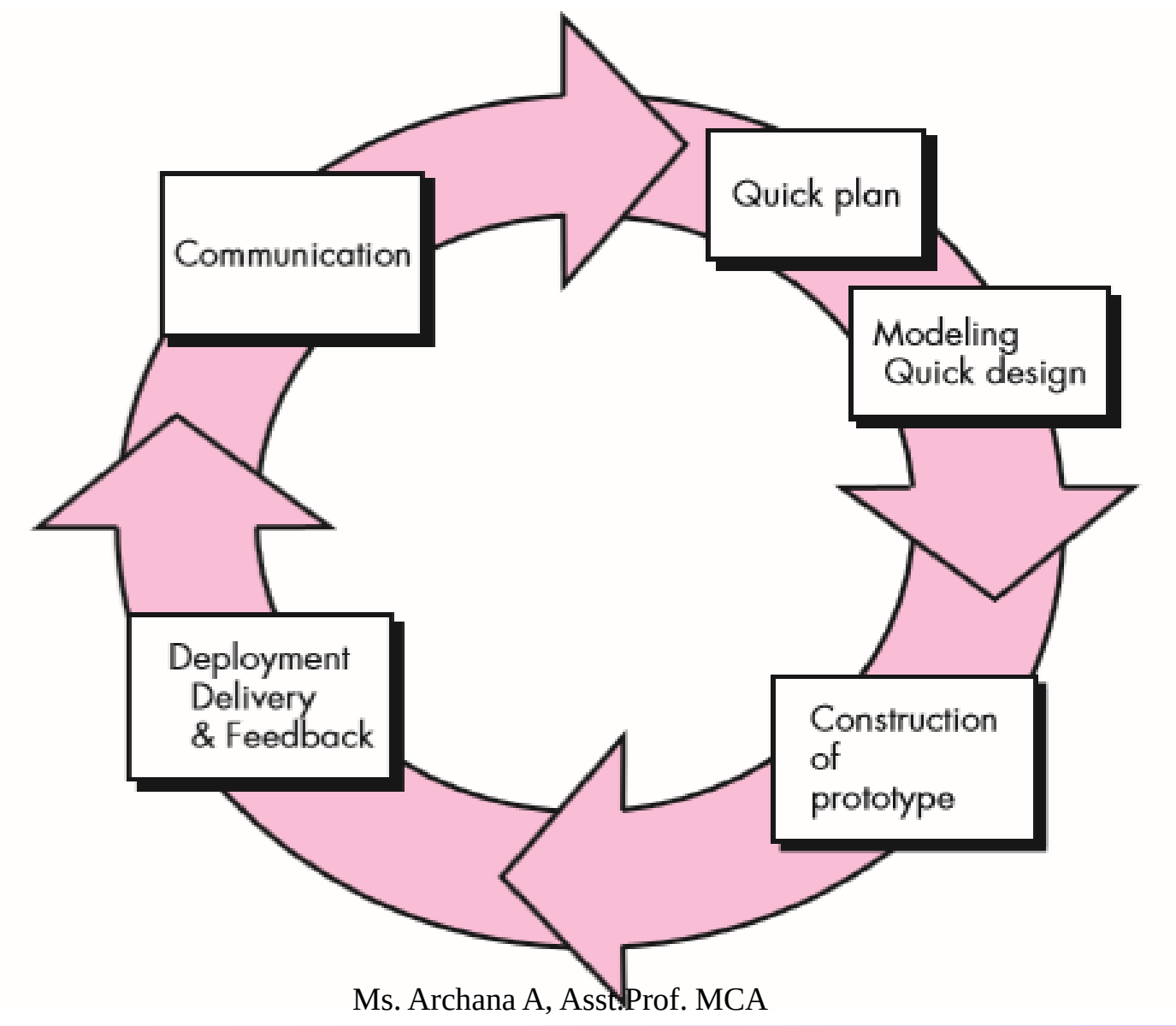
- Evolutionary Models are **iterative.**
- **Produce** an increasingly more **complete version of the software** with each iteration.
- These are a process model that has been **explicitly designed** to accommodate a product that **evolves over time**.
- Evolutionary models are:
  - **Prototyping**
  - **Spiral Model**

(i) Prototyping

- **Best approach when**:
  - **Objectives defined** by customer are general but **does not have details** like input, processing, or output requirement.
  - **Developer** may be **unsure of the efficiency of an algorithm, O.S**., or the **form** that human machine interaction should take.
- It can be used as **standalone process model**.
- **Model assist** software engineer and customer to **better understand** what is to be built when requirement are fuzzy.

- Prototyping **start with communication**, between a customer and software engineer to define **overall objective, identify requirements and make a boundary.**
- Moving ahead, **planned quickly** and modeling occurs (software layout visible to the customers/end-user).
- Quick design leads to **prototype construction**.
- **Prototype is deployed and evaluated** by the customer/user.
- **Feedback from customer**/end user will **refine requirement** and that is how iteration occurs during prototype to satisfy the needs of the customer.

The Prototyping paradigm



Prototyping…

- In most projects, the first system built is **barely usable**. It may be **too slow, too big, awkward** in use or all three.
- There is no alternative but to **start again, smarting but smarter**, and build a **redesigned** version in which these problems are solved

Prototyping…

- Prototype can be serve as "**the first system**".
- **Both customers and developers like the prototyping paradigm**.
  - Customer/end-user gets a feel for the actual system
  - Developer get to build something immediately.

Advantages and Disadvantages of prototyping

- Reduced time and costs
- Improved and increased user involvement

Disadvantages

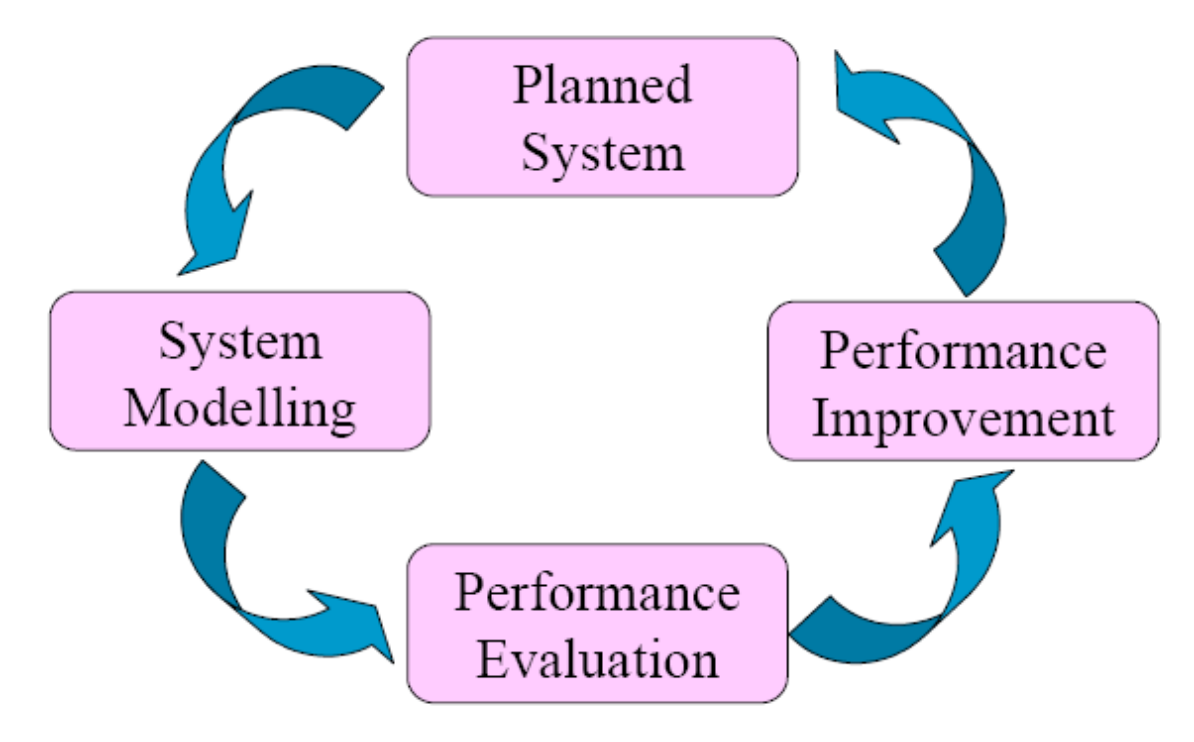
- Insufficient analysis
- User's confusion of prototype and finished system
- Developer's misunderstanding of user objectives
- Developer's attachment towards prototype
- Excessive prototype development time.
- Expense of implementing prototyping

Evolutionary Model:
(ii) Spiral Model

Main characteristics

- Iterative.
- Prototype-oriented.
- Starts with planning and ends with customer evaluation.
- Low risk.

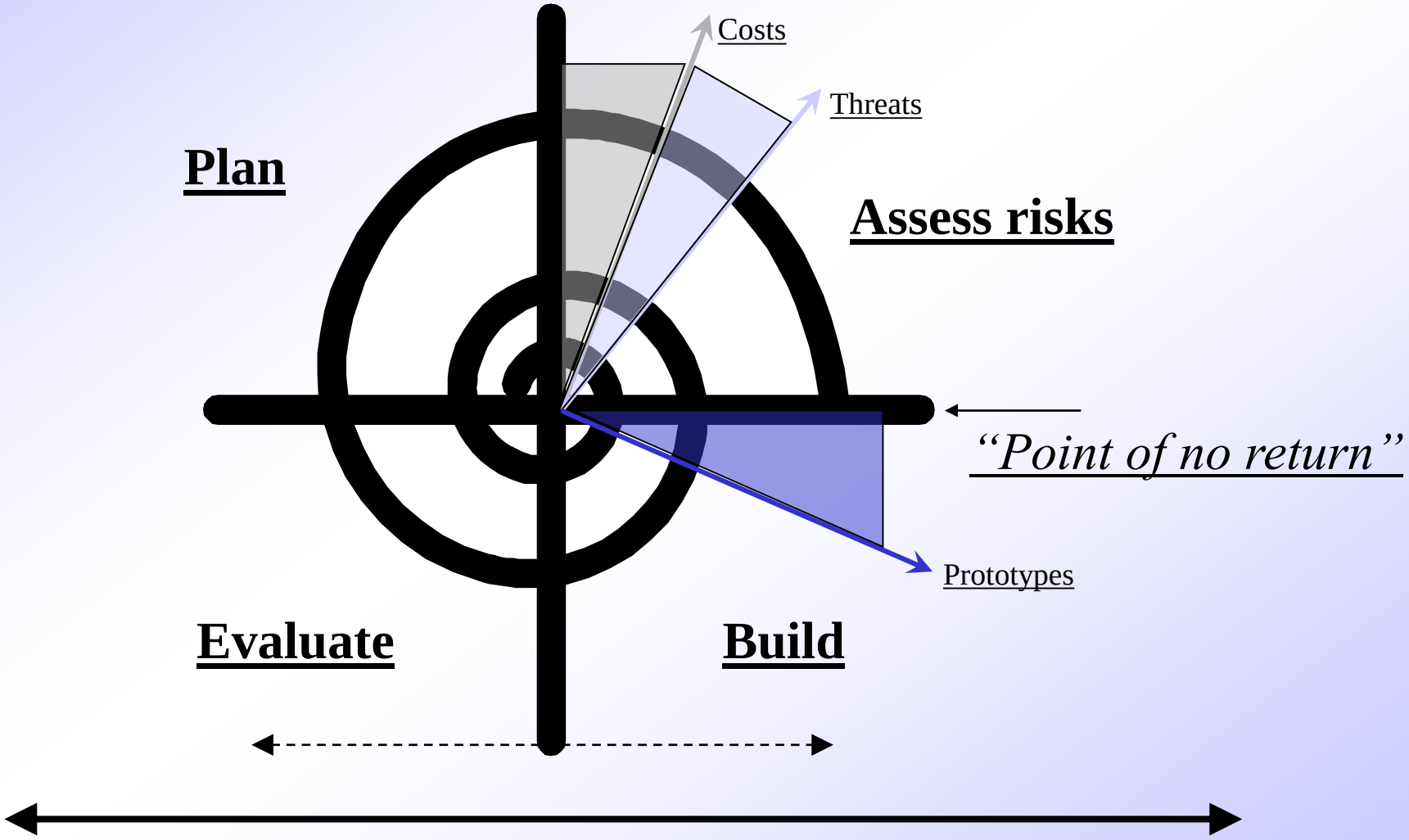
Basic concepts of Spiral Development



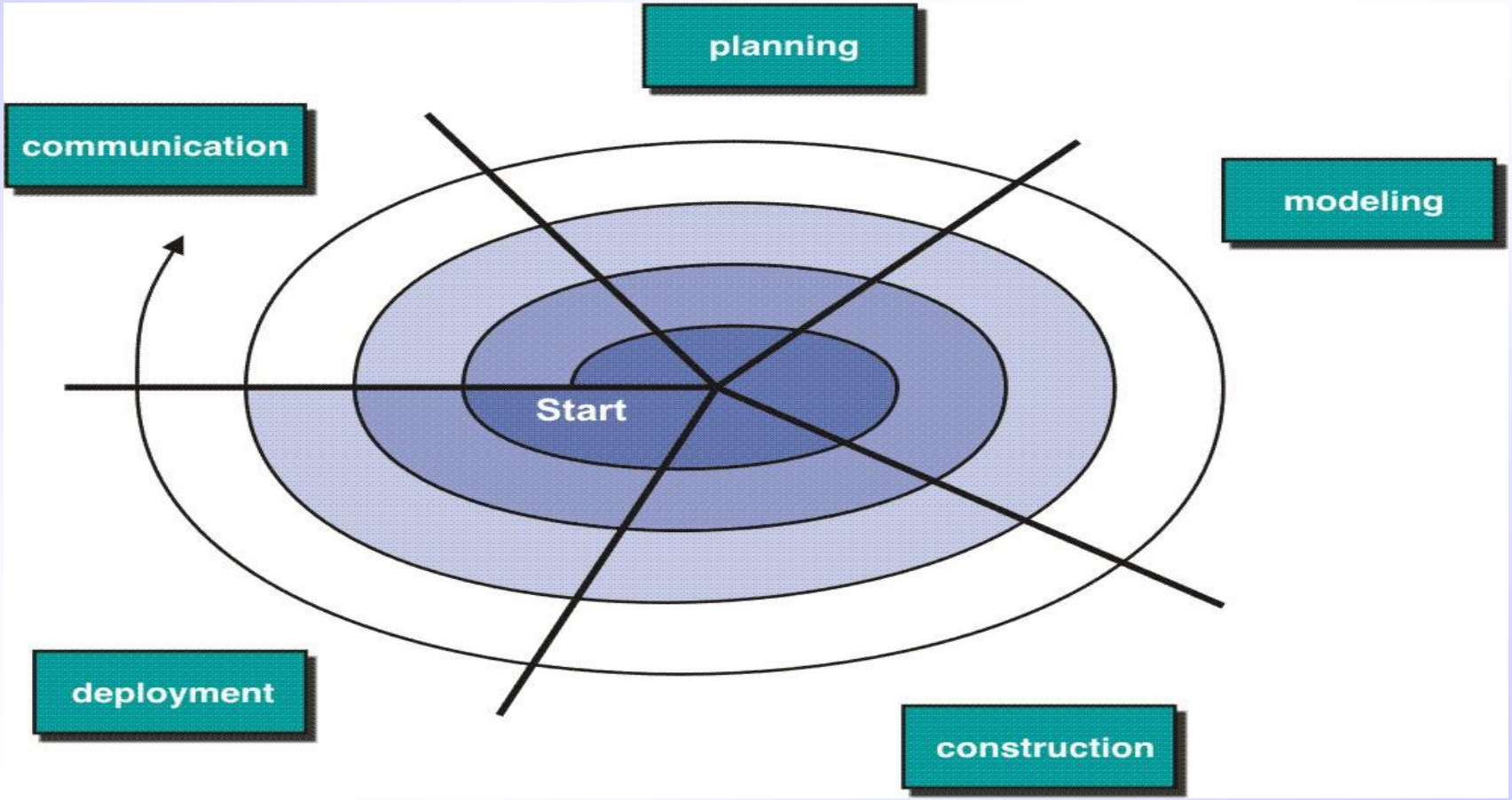
Spiral “areas”

- **Planning**
  - Getting requirements
  - Project planning (based on initial requirements.)
  - Project planning (based on customer evaluation.)
- **Risk analysis**
  - Cost/Benefit and threats/opportunities analysis
  - Based on initial requirements and later on customer feedback
- **Engineering**
  - Preview it
  - Do it
- **Customer evaluation**

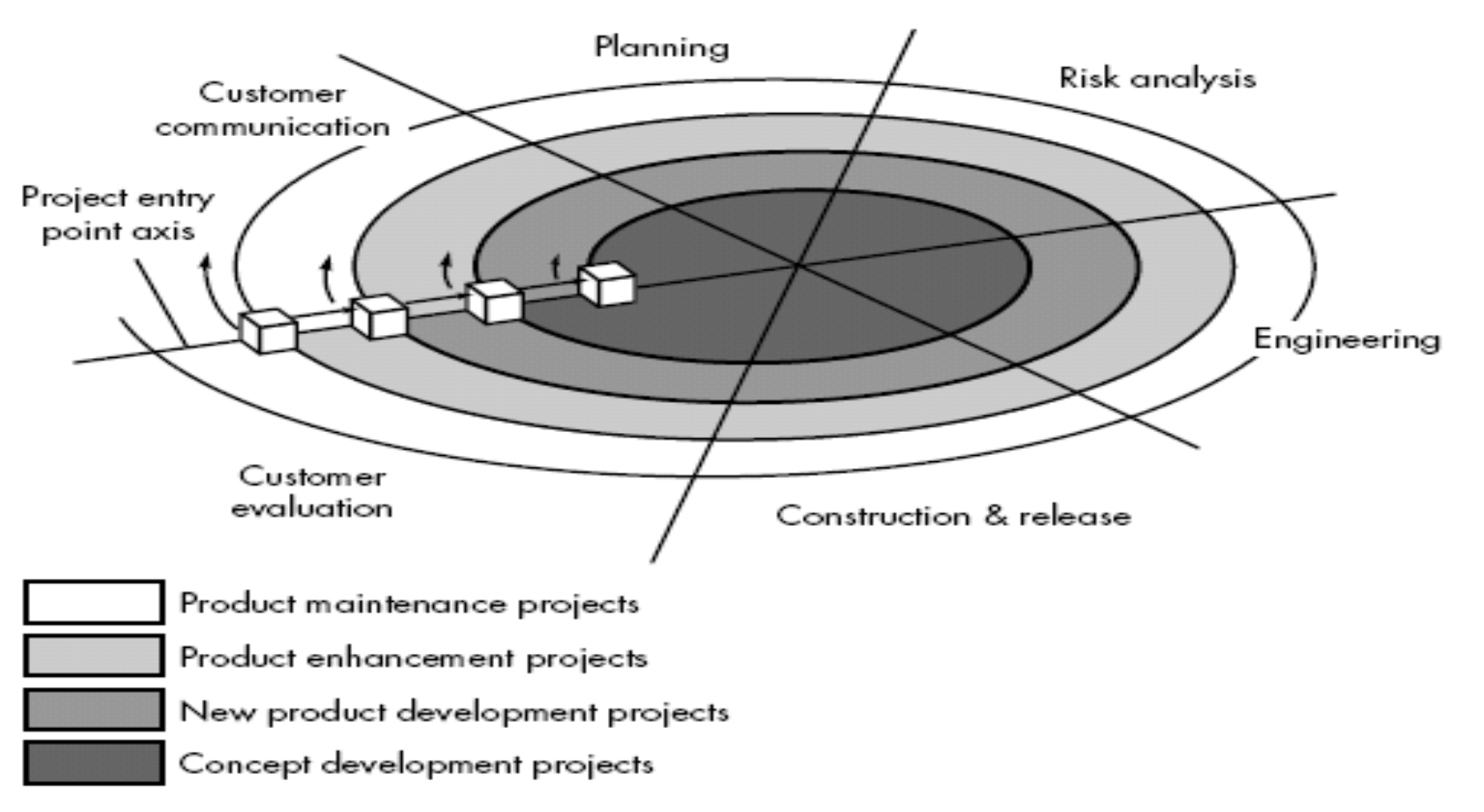
Diagrammatically said…



Evolutionary Model: Spiral Model



Spiral Model



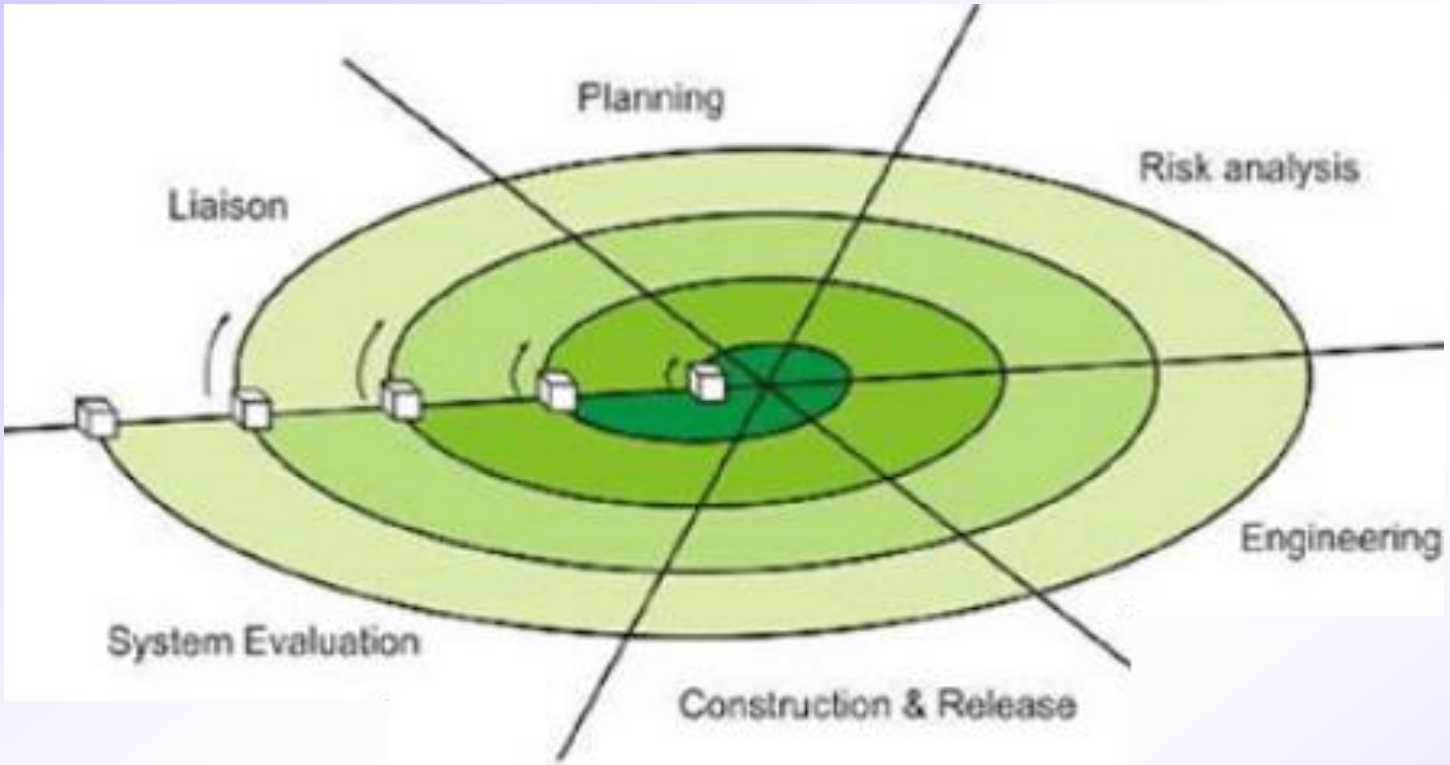
Spiral Model

- The spiral model is **similar to the incremental model**, with more **emphasis** placed on **risk analysis.**
- The spiral model can be **adapted to apply**

  throughout the entire life cycle of an application, from concept development to maintenance.
- Spiral **models uses prototyping** as a **risk reduction mechanism** but, more important, enables the developer to apply the prototyping approach at each stage in the evolution of the product.

Spiral Model

- It maintains the **systematic stepwise approach** suggested by the classic life cycle but also incorporates it into an **iterative framework activity.**



System Maintenance

System Development

System Enhancement

Concept Development

Spiral Model

Spiral Model phases and activities performed during phase:

- **Planning:** It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer
- **Risk Analysis:** Identification of potential risk is done while risk mitigation strategy is planned and finalized
- **Engineering :** It includes testing, coding and deploying software at the customer site
- **Evaluation:** Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun

When to use Spiral model?

- When project is large
- When releases are required to be frequent
- When creation of a prototype is applicable
- When risk and costs evaluation is important
- For medium to high-risk projects
- When requirements are unclear and complex
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

Advantages of Spiral model

- High amount of **risk analysis** hence, avoidance of Risk is enhanced.
- **Good for large** and mission-critical projects.
- Strong approval and documentation control.
- Additional **Functionality can be added** at a later date.
- **Software is produced early** in the software life cycle.

Disadvantages of Spiral model

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

5. Concurrent Development Model

- It represents schematically as series of major technical activities, tasks, along with their associated **states.**
- It is often more **appropriate** for **system engineering projects** where **different engineering teams are involved.**
- An activity- for ex: Modeling- may be in any one of the states for a given time.
- **All activities exist concurrently** but reside in different states.

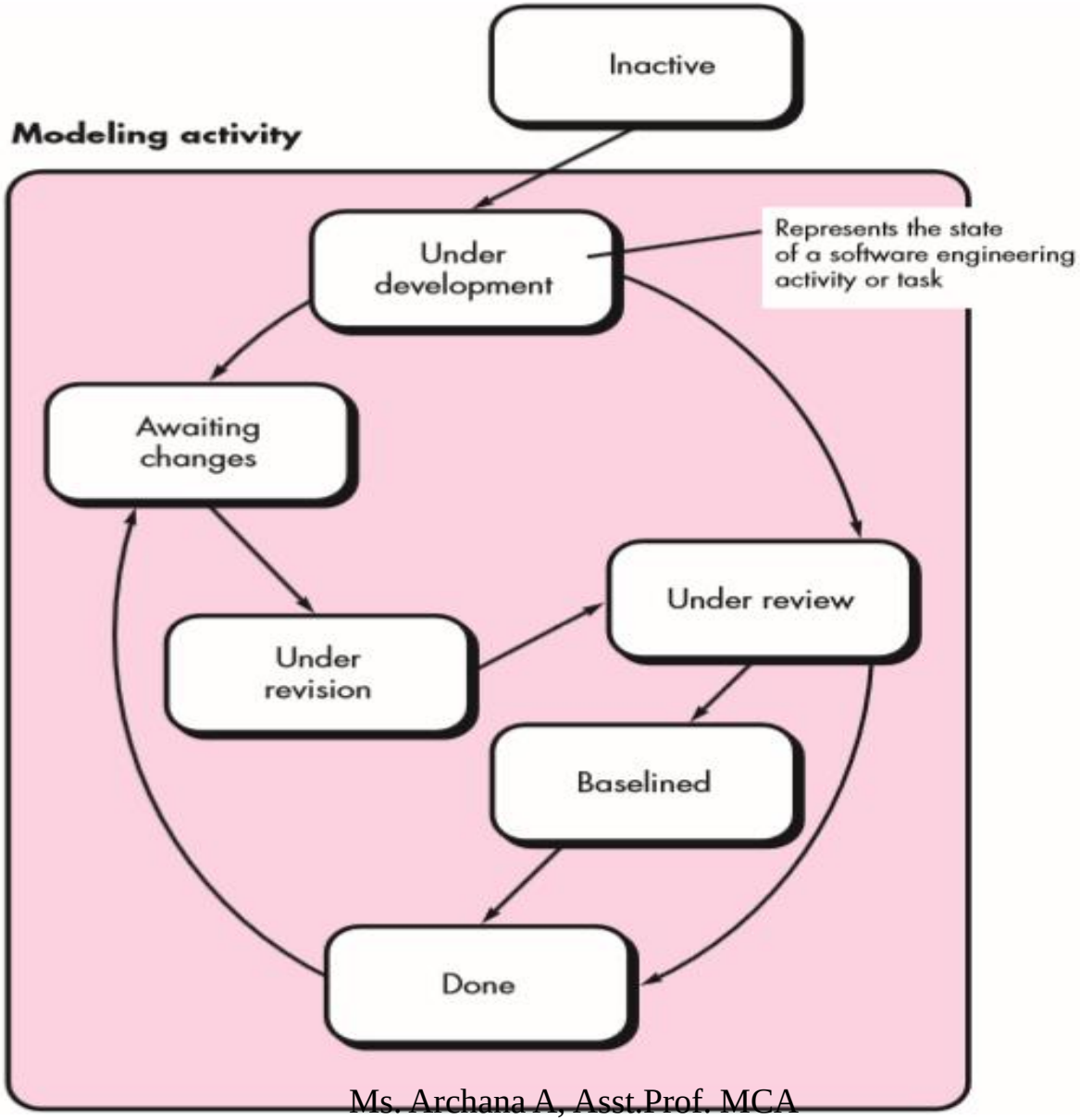
- **Eg.--**The *analysis* activity (existed in the **none** state while initial customer communication was completed) now makes a transition into the **under development** state.

- Then *Analysis* activity moves from the **under development** state into the **awaiting changes** state only if customer indicates changes in requirements.

- A series of event will trigger transition from state to state.

E.g. During initial stage there was **inconsistency in design** which was uncovered. This will triggers the analysis action from the **Done** state into **Awaiting Changes** state.

Concurrent Model



Concurrent Development advantages:

- **Visibility** of current state of project.
- It define **network of activities** .
- Each activities, actions and tasks on the network exists simultaneously with other activities, actions and tasks.
- Events generated at one point in the process network **trigger transitions** among the states.

Specialized Process Models

- These models take on many of the characteristics of one or more of traditional models .

- However, these models **tend to be applied** when a specialized software engineering approach is chosen.

Few **Specialized process models** are:

1. Component Based Development(CBD) model .
2. The Formal methods model.
3. Aspect-Oriented Software Development -AOSD.
4. Unified Process

1. Component Based Development

Characteristics:

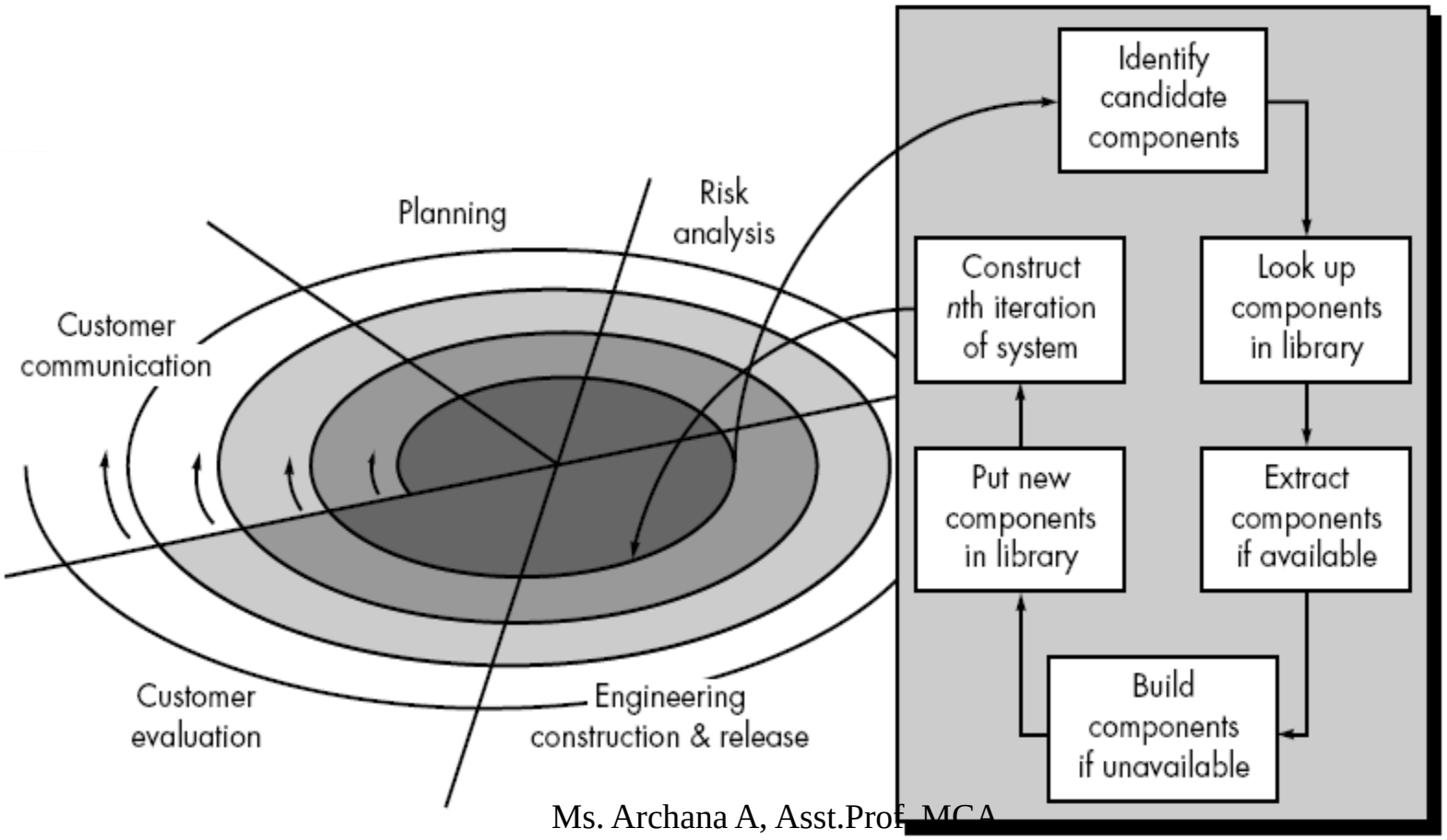
- Component-based development (CBD) model incorporates many of the **characteristics of the spiral model.**
- It is **evolutionary** by nature and **iterative** approach to create software.
- CBD model **creates applications** from **prepackaged software components** (called *classes).*

Component Based Development

Commercial off-the-shelf (COTS)

- 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.

CBD Model


Planning
Risk analysis
Customer communication
Customer evaluation
Engineering construction & release
Identify candidate components
Construct nth iteration of system
Look up components in library
Put new components in library
Extract components if available
Build components if unavailable

CBD model…

- Modeling and construction 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.
- Any new classes built to meet the unique needs of the application. Now process flow return to the spiral activity.

CBD model (cont.)

- 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**.

2. The Formal Methods Model

- The model encompasses **a set of activities that leads to formal mathematical specification** of computer software.
- Formal methods enable us to **specify, develop, and verify** a computer-based system **by applying a rigorous, mathematical notation.**
- Formal methods can be
  - be a foundation for **designing safety critical systems.**
  - be a foundation for **describing complex systems.**
  - Provide support for **program development.**

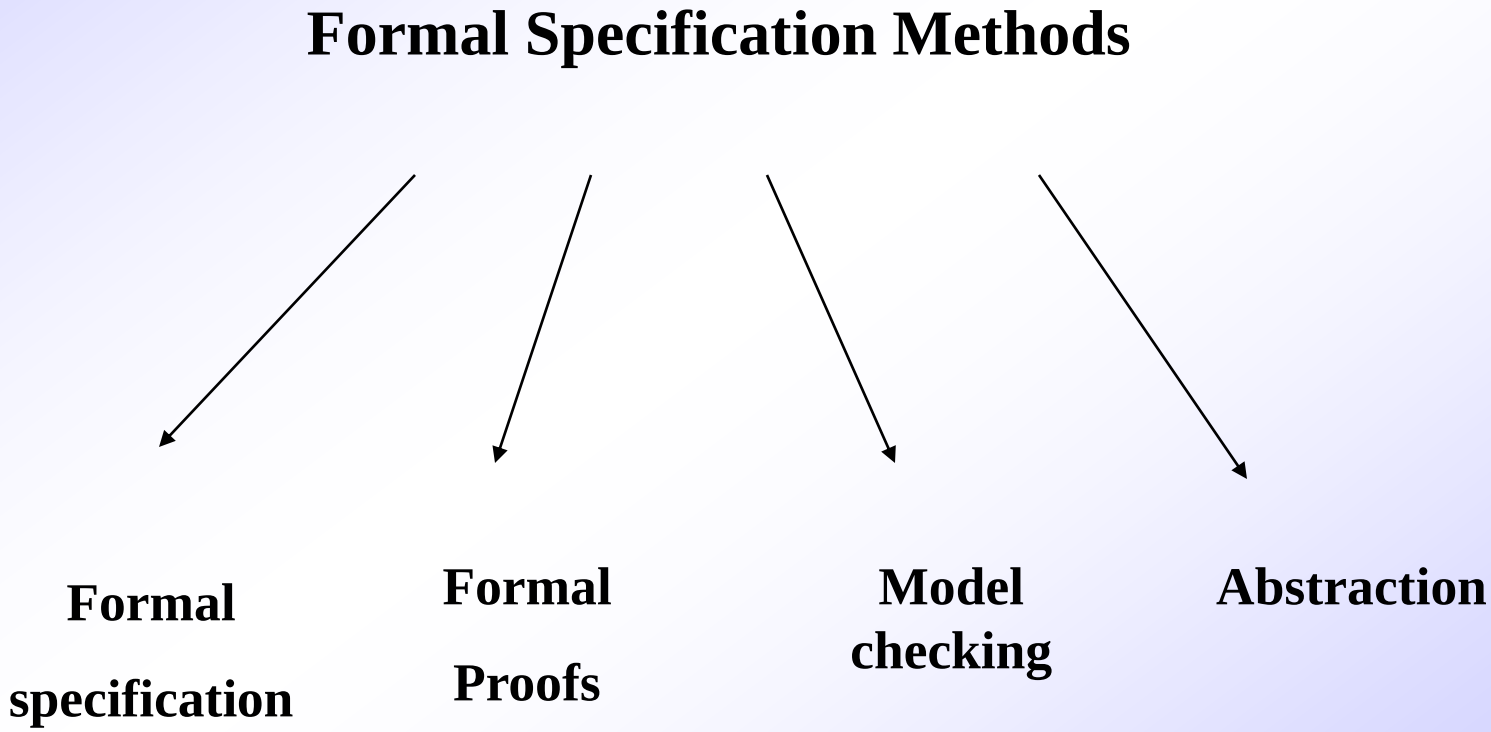
What are Formal Methods?

- **Techniques and tools** based on mathematics and formal logic.

Why Consider Formal Methods?

- The development of a formal specification provides insights understanding of the software requirements and software design
  - **Clarify** customers requirements.
  - **Reveal** and **remove ambiguity, inconsistency and incompleteness.**
  - Facilitate communication of requirement or design.
  - Provides a basis for an **elegant software design**.
  - Traceability
    - System-level requirements should be traceable to subsystems or components

Formal Methods Concepts



Formal Methods model…

- When formal methods are used during development, they provide a **mechanism** for eliminating many of problems that are difficult to overcome using other SE paradigms.
- **Ambiguity, incompleteness & inconsistency** can be **discovered & corrected** more easily – through the **application of mathematical analysis.**
- They serve as a **basis for program verification** & therefore enable to **discover & correct errors** that might otherwise **go undetected**.

The Formal Methods Model…

- Formal methods model offers **defect-free software**.

- Quite **time consuming and expensive**.
- Because few software developers have the necessary background to apply formal methods, **extensive training is required**.

- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

- These approach can be applied while building safety-critical software (aircraft & medical devices)

Cleanroom software development

- A variation on this (Formal methods) approach is Cleanroom software engineering.
- Spend a lot of effort "up-front" to **prevent defects.**
- Formal specification.
- Incremental development.
- **Statistical methods** to ensure reliability.

3. Aspect-oriented software development (AOSD)

- AOSD is a relatively **new software engineering** paradigm that provides a **process and methodological** approach for
- **Defining,**
- **Specifying,**
- **Designing,** and
- **Constructing *aspects.***

AOSD…

- As modern computer-based systems become more complex, **certain concerns** – customer's required properties or **areas of technical interest** – span the entire architecture.
- Some concerns are **high-level properties** of system

  Example: security, fault tolerance.
- Other concerns affect **functions** e.g., application of business rules,
- While others are systemic e.g., task synchronization or memory management.

AOSD…

- **crosscutting concerns:** When concerns cut across multiple system functions, features, and information, they are often referred to as ***crosscutting concerns.***

- ***AOSD*** **defines "aspects" that express customer concerns that cut across multiple system functions, features and information.**

AOSD…

- Aspect oriented process adopt characteristics of both **Evolutionary & concurrent process models.**
- Evolutionary model is appropriate as **aspects are identified & then constructed**.
- The parallel nature of concurrent development is essential because **aspects are engineered independently of localized** software components

4. The Unified Process

Background:

- 1990--Many **object-oriented** analysis and design methods were **proposed.**

- They eventually worked together on a unified method, called the Unified Modeling Language **(UML)**
  - UML is a **robust notation** for the **modeling and development of object-oriented systems**
  - UML became an industry standard in 1997
  - However, UML does not provide the process framework, only the necessary technology for object-oriented development.

The UP-background…

- Booch, Jacobson, and Rumbaugh later developed the **unified process**, which is a **framework** for object-oriented software engineering using UML

  – Draws on the **best features and characteristics** of conventional software process models.

  – Emphasizes the important role of software **architecture.**

  – Consists of a process flow that is **iterative and incremental,** thereby providing an **evolutionary feel.**

The Unified Process

- **Unified Process** — a "**use-case driven, architecture-centric, iterative and incremental**" software process closely aligned with the Unified Modeling Language (UML).

- It implements many of the **best principles of agile software development**.

Rational Unified Process

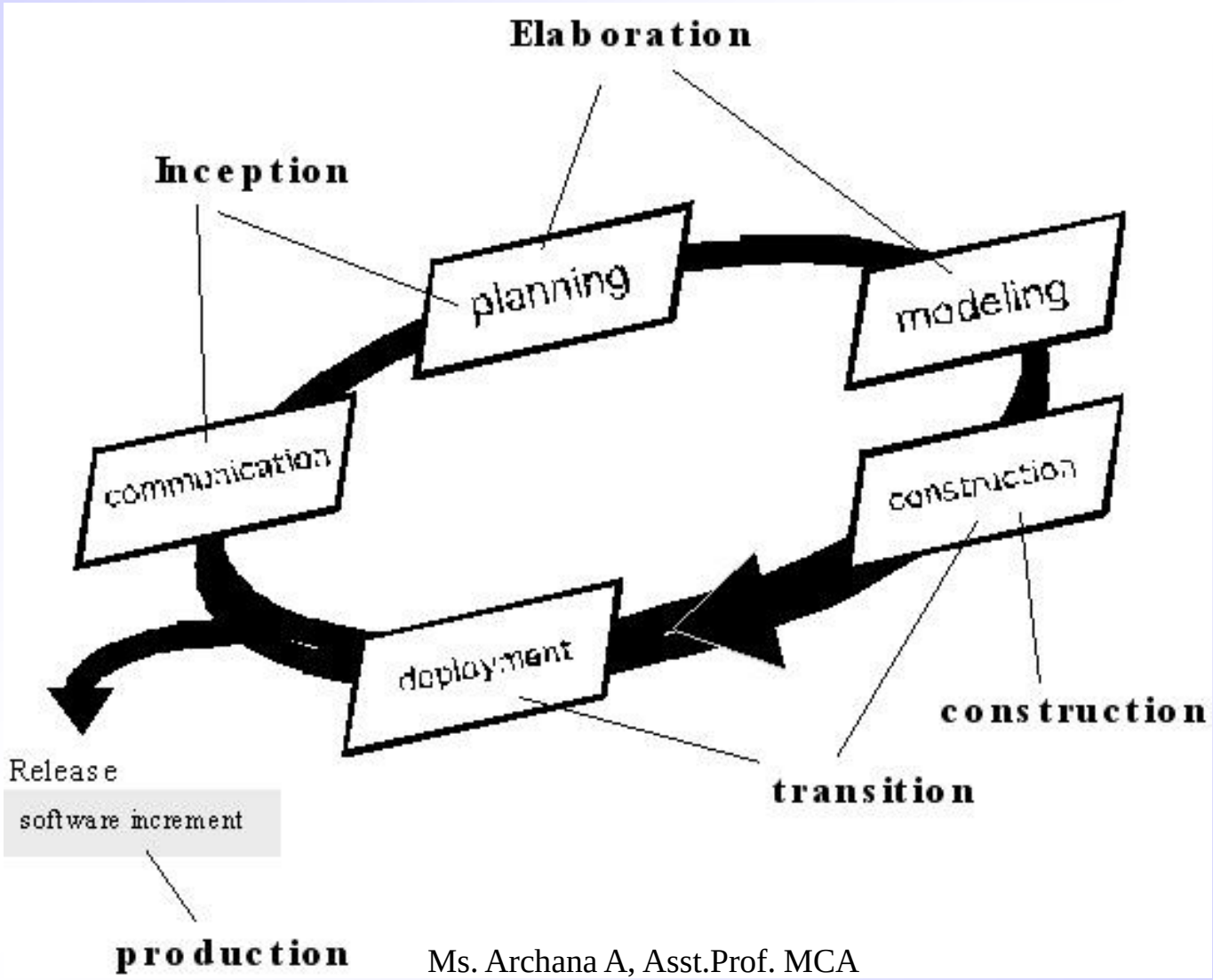
- The **Rational Unified Process®** is a ***Software Engineering Process.*** It provides a **disciplined approach** in assigning tasks and responsibilities within a development organization.

- Its goal is to ensure the **production of high-quality software** that meets the needs of its end-users, within a predictable **schedule and budget.**

Phases of the Unified Process

- The ***Inception*** *phase*
- The ***Elaboration*** *phase*
- The ***Construction*** *phase*
- The ***Transition*** *phase*
- The ***Production*** *phase*

Phases of the Unified Process



Advantages of RUP

- Well-documented and complete methodology.
- Open and Public.
- Training readily available.
- Changing Requirements.
- Reduced Integration Time and Effort.
- Higher Level of Re-use

Disadvantages:

- Process is too complex.
- Does not capture the sociological aspects of software development.

End of unit-1

Process Models

Summary:

1. Waterfall model
2. V-model
3. Incremental Process Models
4. Evolutionary Process Models
5. Concurrent Models
6. Component Based Development(CBD) model
7. The Formal methods Model
8. Aspect-Oriented Software Development(AOSD)
9. Unified Process

Process Flow

1. Linear process flow
2. Iterative process flow
3. Evolutionary process flow
4. Parallel process flow