The background is a solid blue color. On the left side, there is a large white rounded rectangle. To the right of this rectangle, the title 'Introduction to Software Engineering' is written in a bold, black, sans-serif font. Below the title, there is a thick yellow horizontal bar with rounded ends.

Introduction to Software Engineering

What is Software?

- Software is a set of instructions to acquire inputs and to manipulate them to produce the desired output in terms of functions and performance as determined by the user of the software
- Also include a set of documents, such as the software manual , meant for users to understand the software system

Description of the Software

- A software is described by its capabilities. The capabilities relate to the functions it executes, the features it provides and the facilities it offers.

EXAMPLE

Software written for **Sales-order processing** would have different functions to process different types of sales order from different market segments .

- ✓ The features for example , would be to handle multi-currency computing, updating product , sales and Tax status.
- ✓ The facilities could be printing of sales orders, email to customers and reports to the store department to dispatch the goods.

Classes of Software

Software is classified into two classes:

- **Generic Software:**

is designed for broad customer market whose requirements are very common, fairly stable and well understood by the software engineer.

- **Customized Software:**

is developed for a customer where domain , environment and requirements are being unique to that customer and cannot be satisfied by generic products.

What is Good Software?

- Software has number of attributes which decide whether it is a good or bad .
- The definition of a good software changes with the person who evaluates it.
- The software is required by the customer , used by the end users of an organization and developed by software engineer .
- Each one will evaluate the different attributes differently in order to decide whether the software is good.

What are the attributes of good software?

The software should deliver the required functionality and performance to the user and should be **maintainable**, **dependable** and **usable**.

- **Maintainability**
 - Software must evolve to meet changing needs
- **Dependability**
 - Software must be trustworthy
- **Efficiency**
 - Software should not make wasteful use of system resources
- **Usability**
 - Software must be usable by the users for which it was designed

Software - Characteristics

- Software has a dual role. It is a product, but also a vehicle for delivering a product.
- Software is a logical rather than a physical system element.
- Software has characteristics that differ considerably from those of hardware.
 - ✓ Software is developed or engineered, it is not manufactured in the classical sense
 - ✓ Software doesn't "wear out"
 - ✓ Most software is custom-built, rather than being assembled from existing components.

Changing nature of software(Types)

- **System Software-** A collection of programs written to service other programs at system level.
For example, compiler, operating systems.
- **Real-time Software-** Programs that monitor/analyze/control real world events as they occur.
- **Business Software-** Programs that access, analyze and process business information.
- **Engineering and Scientific Software -** Software using “number crunching” algorithms for different science and applications. System simulation, computer-aided design.

Changing nature of software(Types)

- **Embedded Software-:**

Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. It has very limited and esoteric functions and control capability.

- **Artificial Intelligence (AI) Software:**

Programs make use of AI techniques and methods to solve complex problems. Active areas are expert systems, pattern recognition, games

- **Internet Software :**

Programs that support internet accesses and applications. For example, search engine, browser, e-commerce software, authoring tools.

Software Engineering

- “A systematic approach to the analysis, design, implementation and maintenance of software.”
(*The Free On-Line Dictionary of Computing*)
- “The systematic application of tools and techniques in the development of computer-based applications.”
(Sue Conger in *The New Software Engineering*)
- “Software Engineering is about designing and developing high-quality software.”
(Shari Lawrence Pfleeger in *Software Engineering -- The Production of Quality Software*)

What is Software Engineering?

- Engineering: The Application of Science to the Solution of Practical Problems
- Software Engineering: The Application of CS to Building Practical Software Systems
- Programming
 - Individual Writes Complete Program
 - One Person, One Computer
 - Well-Defined Problem
 - Programming-in-the-Small
- Software Engineering
 - Individuals Write Program Components
 - Team Assembles Complete Program
 - Programming-in-the-Large

What is the difference between software engineering and computer science?

Computer Science

Software Engineering

is concerned with

- theory
- fundamentals

- the practicalities of developing
- delivering useful software

Computer science theories are currently insufficient to act as a complete underpinning for software engineering, BUT it is a foundation for practical aspects of software engineering

What is Software Engineering?

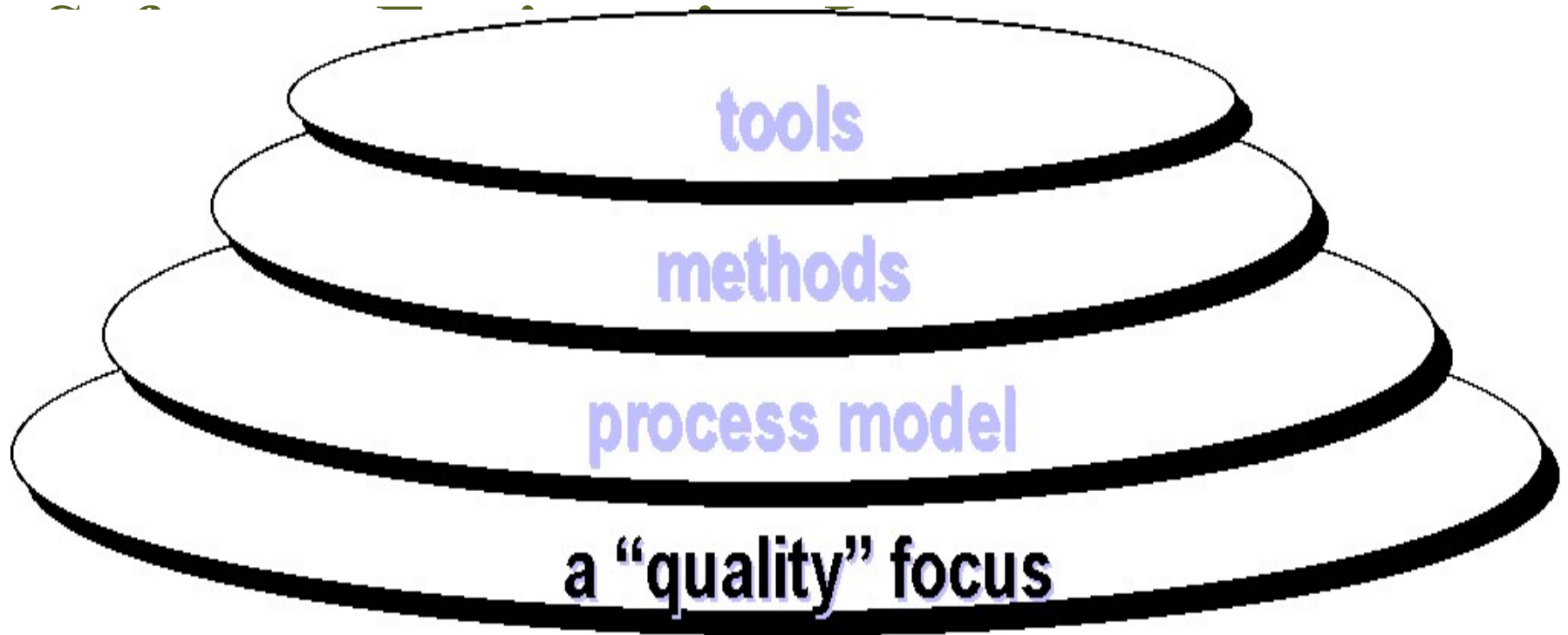
Although hundreds of authors have developed personal definitions of software engineering, a definition proposed by Fritz Bauer[NAU69] provides a basis:

- **“[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.”**

The IEEE [IEE93] has developed a more comprehensive definition when it states:

- **“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 is a layered technology - *Pressman's view*:



What is Software Engineering?

- **Software methods:**
- Software engineering methods provide the technical “how to’s” for building software.
- Methods --> how to encompass a broad array of tasks:
 - requirements analysis, design, coding, testing, and maintenance
- Software engineering methods rely on a set of basic principles.

What is Software Engineering?

- **Software process:**

Software engineering process is the glue that holds:

- technology together
- enables rational and timely development of computer software.

Software engineering process is a framework of a set of key process areas.

It forms a basis for:

- project management, budget and schedule control
- applications of technical methods
- product quality control

What is Software Engineering?

- **Software tools:**

- programs provide automated or semi-automated support for the process and methods.
- programs support engineers to perform their tasks in a systematic and/or automatic manner.

Why Software Engineering?

● Objectives:

- Identify new problems and solutions in software production.
- Study new systematic methods, principles, approaches for system analysis, design, implementation, testing and maintenance.
- Provide new ways to control, manage, and monitor software process.
- Build new software tools and environment to support software engineering.

Why Software Engineering?

Major Goals:

- To increase software **productivity** and **quality**.
- To effectively control software **schedule** and planning.
- To reduce the **cost** of software development.
- To meet the **customers'** needs and requirements.
- To enhance the conduction of software engineering **process**.
- To improve the current **software engineering practice**.
- To support the engineers' activities in a systematic and efficient manner.

A Process Framework

Common process framework

Framework activities

work tasks

work products

milestones & deliverables

QA checkpoints

Umbrella Activities

Process Framework Activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

Umbrella Activities

- Software project tracking & control
- Risk Management
- Formal Technical reviews
- Software configuration management
- Reusability management

- In recent years, there has been a significant emphasis on “process maturity.” The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity.
- To determine an organization’s current state of process maturity, the SEI uses an assessment that results in a five point grading scheme.

- The grading scheme determines compliance with a capability maturity model (CMM) [PAU93] that defines key activities required at different levels of process maturity.
- The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

- **Level 1: Initial.** The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
- **Level 2: Repeatable.** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

- **Level 3: Defined.** The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

- **Level 4: Managed.** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

- **Level 5: Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4

Process maturity level 2

Software configuration management

Software quality assurance

Software subcontract management

Software project tracking and oversight

Software project planning

Requirements management

Process maturity level 3


- Peer reviews
 - Intergroup coordination
- Software product engineering
 - Integrated software management
- Training program
- Organization process definition
 - Organization process focus

Process maturity level 4

- Software quality management
- Quantitative process management

Process maturity level 5

- Process change management
- Technology change management
- Defect prevention



PROCESS MODELS

SDLC Process Model

SDLC – Software Development Life Cycle

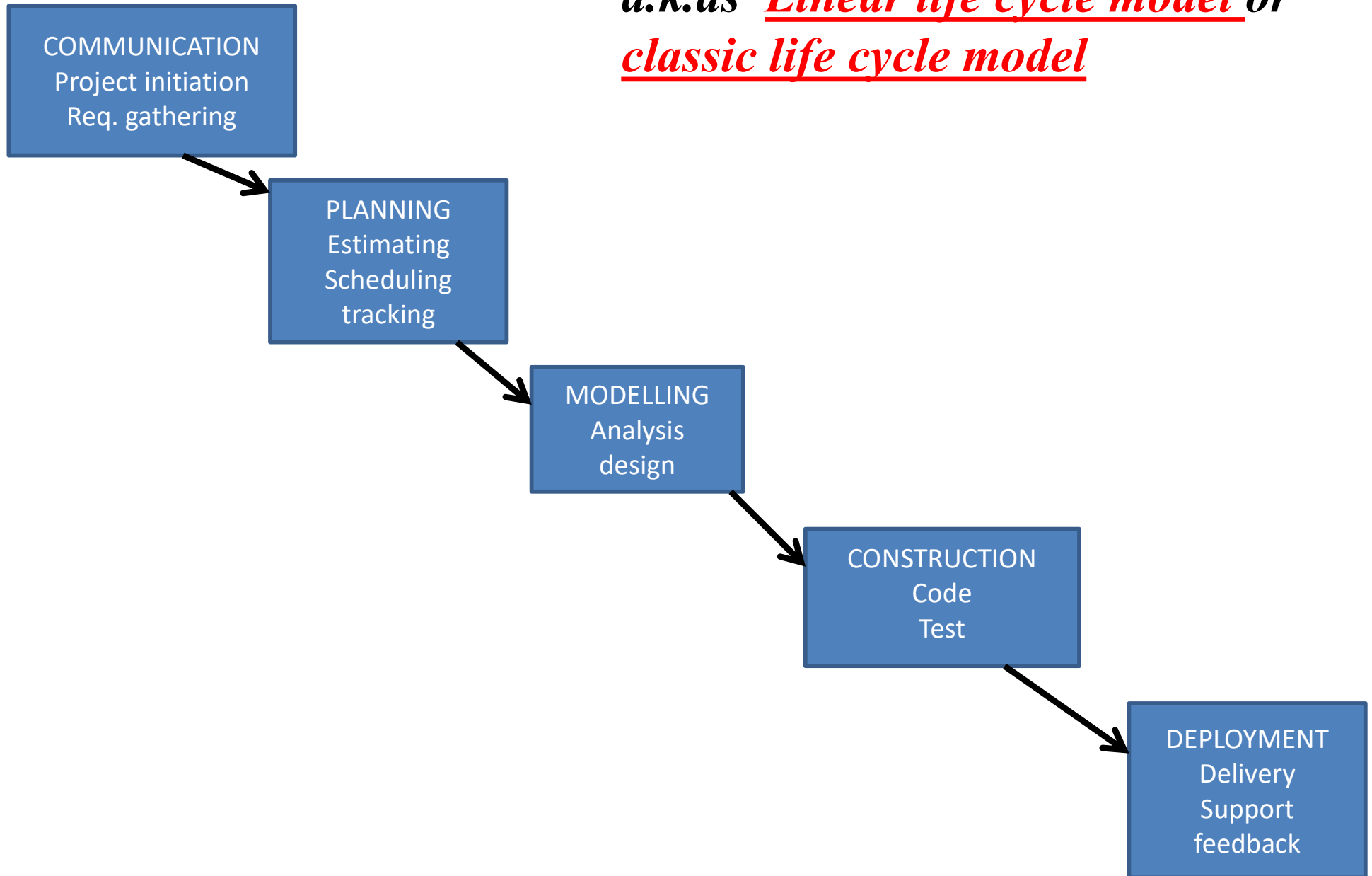
Process model is a framework that describes the activities, actions, tasks, milestones, and work products performed at each stage of a software development project that leads to a high quality software

Life cycle models

- Waterfall model
- Incremental process models
 - Incremental model
 - RAD model
- Evolutionary Process Models
 - Prototyping model
 - Spiral model
- Object oriented process model

WATERFALL MODEL

a.k.as Linear life cycle model or classic life cycle model



WATERFALL MODEL

- Project initiation & requirement gathering
 - What is the Problem to Solve?
 - What Does Customer Need/Want?
 - Interactions Between SE and Customer
 - Identify and Document System Requirements
 - Generate User Manuals and Test Plans
- Planning
 - Prioritize the requirements
 - Plan the process

WATERFALL MODEL

- Analysis and design
 - How is the Problem to be Solved?
 - High-Level Design
 - Determine Components/Modules
 - Transition to Detailed Design
 - Detail Functionality of Components/Modules
- Coding and Testing
 - Writing Code to Meet Component/Module Design Specifications
 - Individual Test Modules in Isolation
 - Integration of Components/Modules into Subsystems
 - Integration of Subsystems into Final Program

WATERFALL MODEL

- Deployment
 - System Delivered to Customer/Market
 - Bug Fixes and Version Releases Over Time

Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Waterfall Drawbacks

- All projects cannot follow linear process
- All requirements must be known upfront
- Few business systems have stable requirements.
- The customer must have patience. A working version of the program will not be available until late in the project time-span
- Leads to 'blocking states'
- Inappropriate to changes

When to use the Waterfall Model

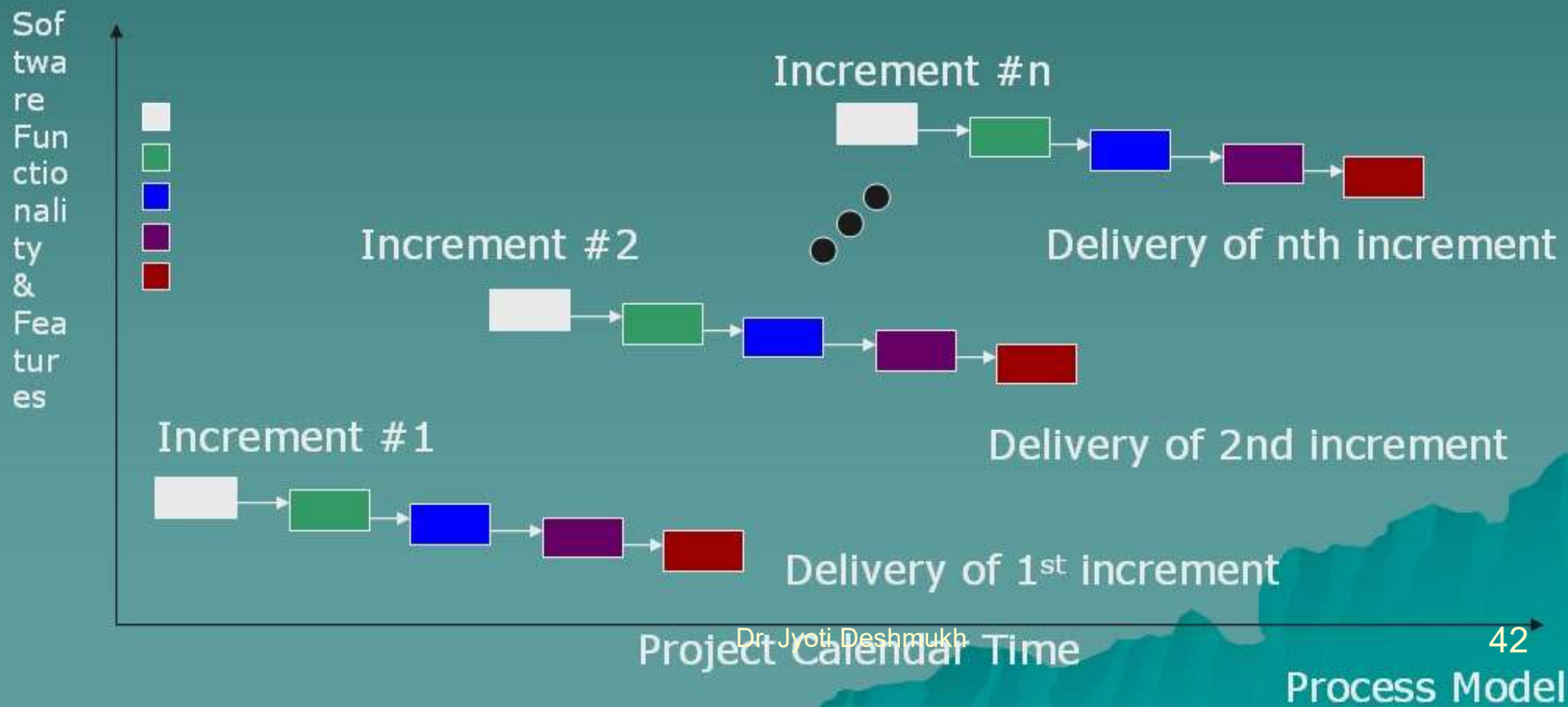
- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Incremental SDLC Model

- Combines the elements of waterfall model in an iterative fashion
- Construct a partial implementation of a total system
- Then slowly add increased functionality
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Process Model

- ◆ Combines the elements of the waterfall model applied in an iterative fashion.



Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

When to use the Incremental Model

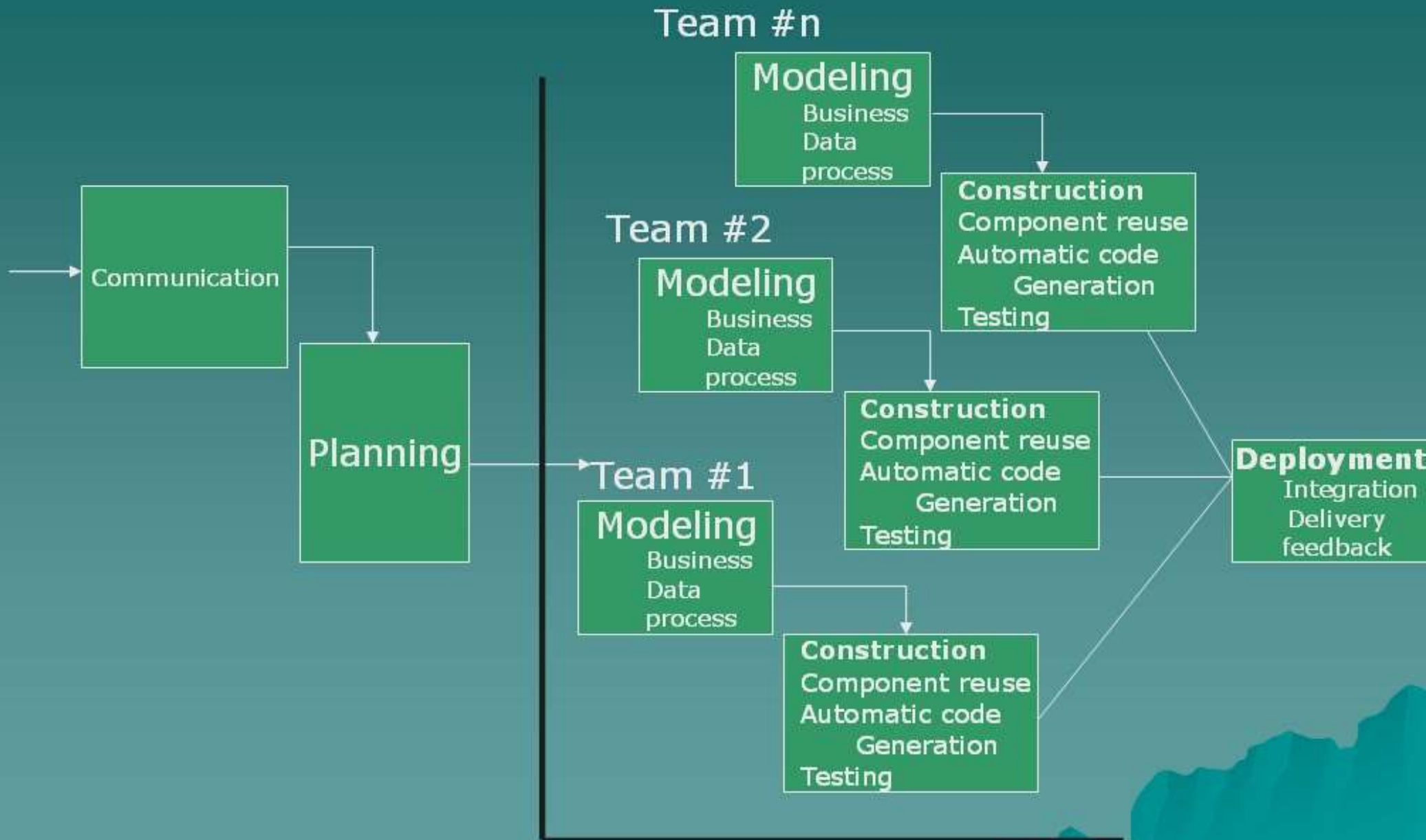
- When staffing is not available by deadline
- Most of the requirements are known up-front but are expected to evolve over time
- When the software can be broken into increments and each increment represent a solution
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology

RAD MODEL

Rapid Application Development Model

- An incremental process model that emphasizes short development cycle
- “High-speed” adaptation of the waterfall model.
- RAD approach also maps into the generic framework activities

RAD Model



Drawbacks of RAD

- For large projects, RAD requires sufficient human resources to create the right number of RAD teams.
- If developers & customers are not committed to rapid-fire activities, RAD projects will fail.
- If the system cannot be properly modularized, building the components will be problematic
- If high-performance is an issue, RAD may not work.
- RAD may be inappropriate when technical risks are high

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

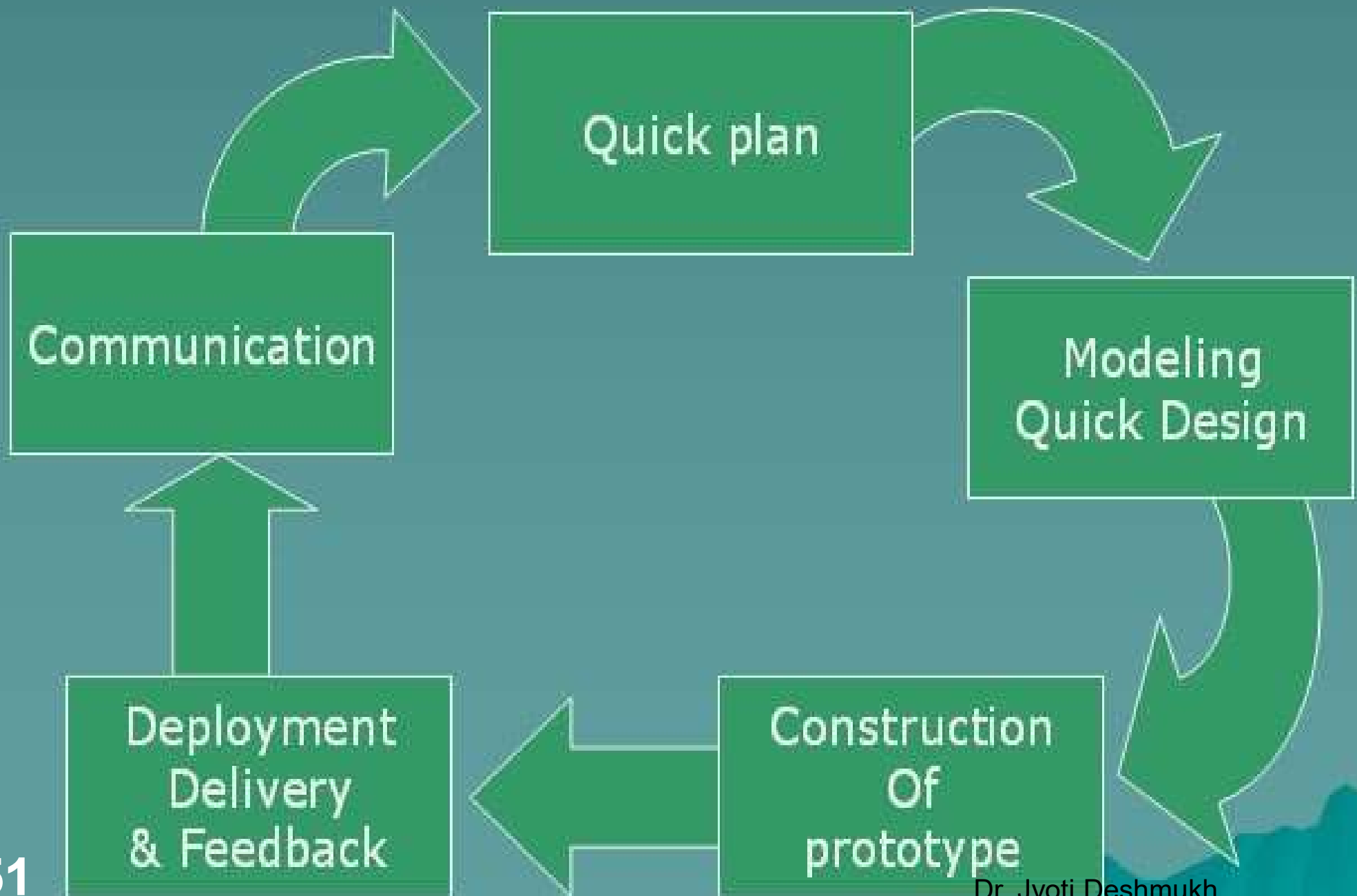
EVOLUTIONARY PROCESS MODEL

- These models produce an increasingly more complete version of the software with each iteration
- When to use
 - Tight market deadlines
 - Well defined system requirements
 - No details about system definition

PROTOTYPING MODEL

- A prototype is a partially developed product that enables customers and developers to examine some aspects of the proposed system and decide if it is suitable or appropriate for the finished product
 - Start with what is known about requirements.
 - Do a quick design.
 - Build the prototype by focusing on what will be seen by the user.
 - Use the prototype to show the user and help refining requirements

PROTOTYPING MODEL



Drawbacks of Prototyping Model

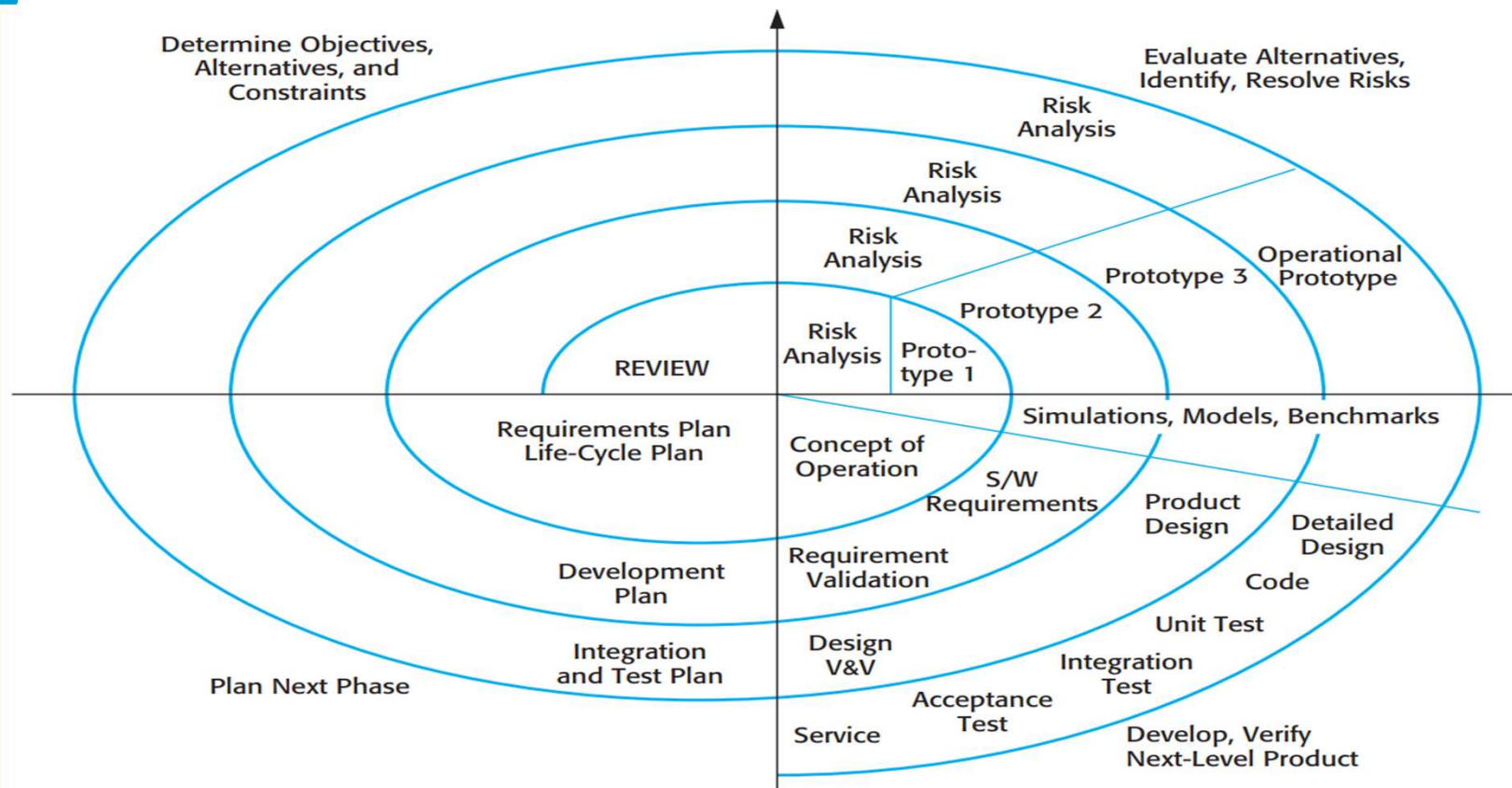
- Customer sees what appears to be a working version of the software and presumes that it is the final thing.
- The developer often makes implementation compromises in order to get a prototype working quickly

Only one advantage is actual software is engineered with an eye toward quality

SPIRAL MODEL

- It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model Process
- Adapted to complete life cycle
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.

SPIRAL MODEL



Spiral Model Strengths

- Focuses attention on reuse options.
- Focuses attention on early error elimination.
- Puts quality objectives up front.
- Integrates development and maintenance.
- Provides a framework for hardware/software Development.

Drawbacks of the Spiral Model

- It may be difficult to convince customers that the evolutionary approach is controllable.
- It demands risk assessment expertise and relies on this expertise for success.
- If a major risk is uncovered and managed, problems will occur.

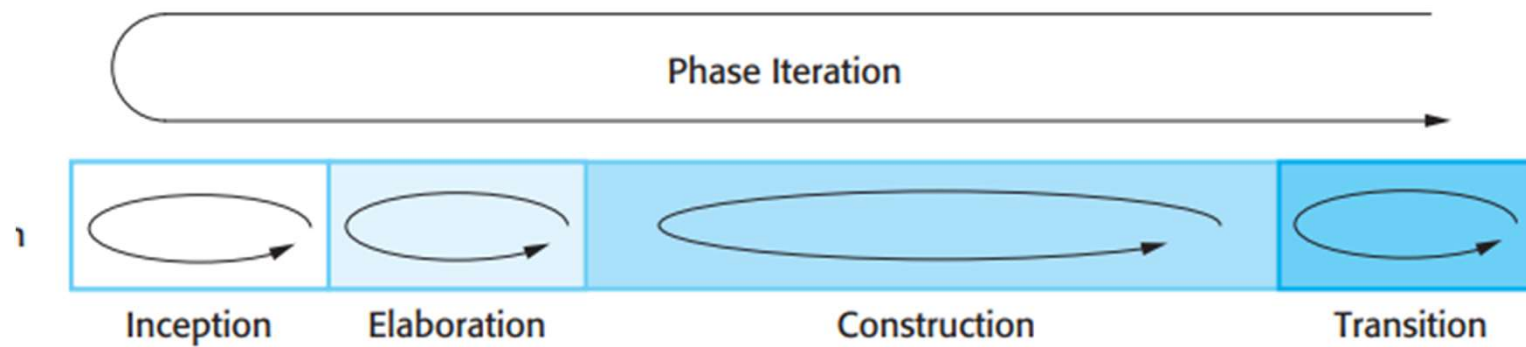
When to use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected

The Rational Unified Process

1. A **dynamic** perspective, which shows the phases of the model over time.
2. A **static** perspective, which shows the process activities that are enacted.
3. A **practice** perspective, which suggests good practices to be used during the process

Phases



1. Inception The goal of the inception phase is to establish a business case for the system.
2. Elaboration The goals of the elaboration phase are to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan, and identify key project risks. On completion of this phase you should have a requirements model for the system

3. Construction The construction phase involves system design, programming, and testing.

4. Transition The final phase of the RUP is concerned with moving the system from the development community to the user community and making it work in a real environment.

Static Workflows in RUP

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models, and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users, and installed in their workplace.
Configuration and change management	This supporting workflow manages changes to the system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

The practice perspective

Six fundamental best practices are recommended:

1. **Develop software iteratively-** Plan increments of the system based on customer priorities and develop the highest-priority system features early in the development process.
2. **Manage requirements-** Explicitly document the customer's requirements and keep track of changes to these requirements. Analyze the impact of changes on the system before accepting them.

3. **Use component-based architectures**
Structure the system architecture into
components

4. **Visually model software-** Use graphical UML models to present static and dynamic views of the software.

5. **Verify software quality-** Ensure that the software meets the organizational quality standards.

6. **Control changes to software-** Manage changes to the software using a change management system and configuration management procedures and tools.

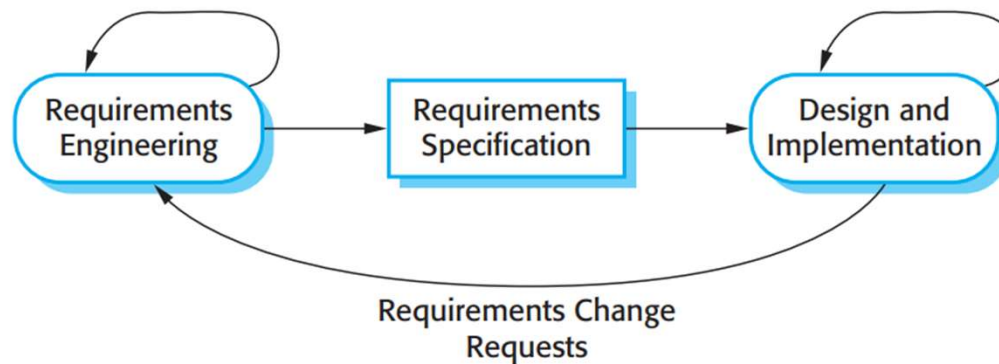
Dr. Jyoti Deshmukh

Agile Software Development

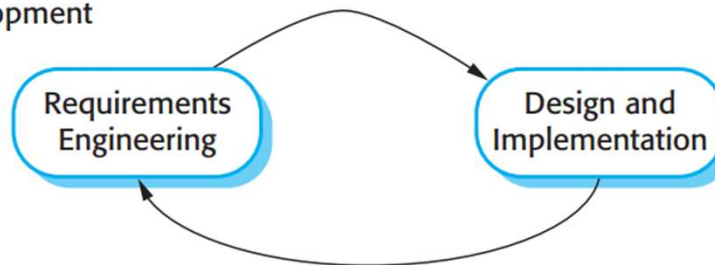
Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Plan Driven and Agile development

Plan-Based Development



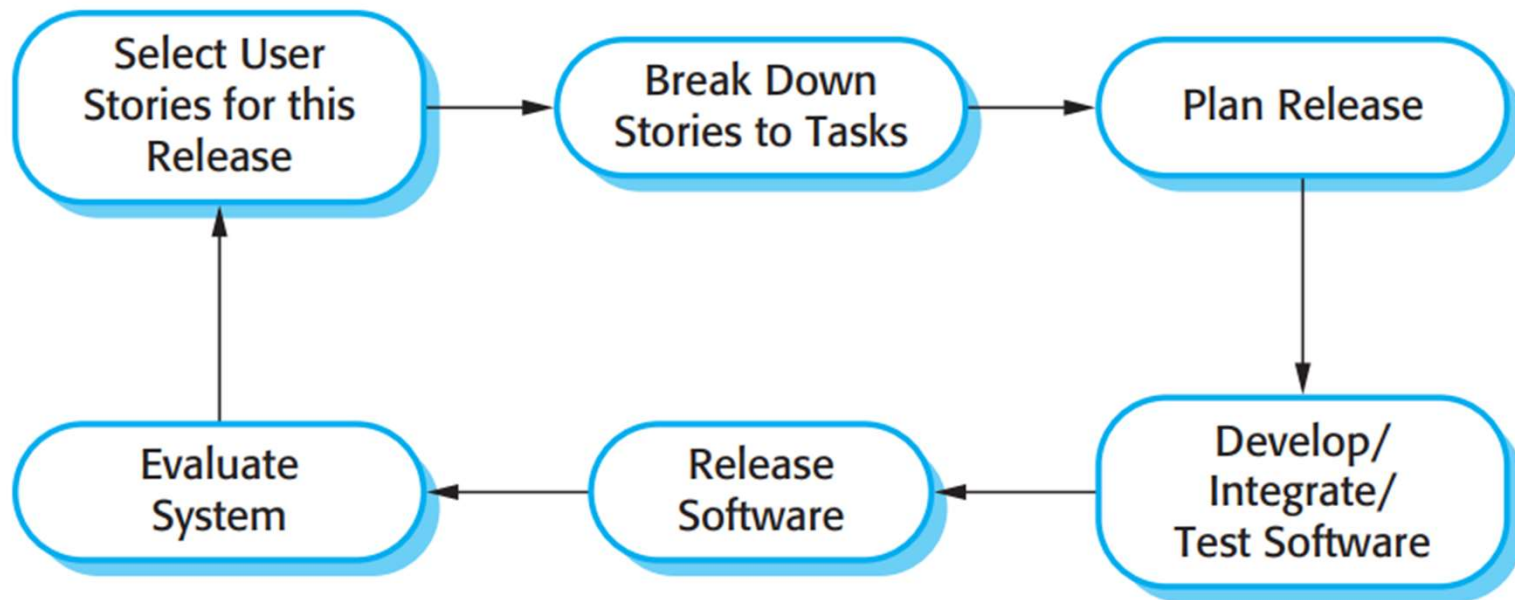
Agile Development



Agile Methods

- extreme programming (Beck, 1999; Beck, 2000)
- Scrum (Cohn, 2009; Schwaber, 2004; Schwaber and Beedle, 2001)
- Crystal (Cockburn, 2001; Cockburn, 2004)
- Adaptive Software Development (Highsmith, 2000),
- DSDM (Stapleton, 1997; Stapleton, 2003), and
- Feature Driven Development (Palmer and Felsing, 2002).
- The success of these methods has led to some integration with more traditional development methods based on system modelling, resulting in the notion of agile modelling (Ambler and Jeffries, 2002) and agile instantiations of the Rational Unified Process (Larman, 2002).

Extreme Programming



Exercises

1. Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car
- A virtual reality system to support software maintenance
- A university accounting system that replaces an existing system
- An interactive travel planning system that helps users plan journeys with the lowest environmental impact

2. Historically, the introduction of technology has caused profound changes in the labor market and, temporarily at least, displaced people from jobs. Discuss whether the introduction of extensive process automation is likely to have the same consequences for software engineers. If you don't think it will, explain why not. If you think that it will reduce job opportunities, is it ethical for the engineers affected to passively or actively resist the introduction of this technology?

3. Explain why incremental development is the most effective approach for developing business software systems. Why is this model less appropriate for real-time systems engineering?

4. Propose a specific software project that would be amenable to the incremental model. Present a scenario for applying the model to the software.

5. As you move outward along the process flow path of the spiral model, what can you say about the software that is being developed or maintained?

6. Provide five examples of software development projects that would be amenable to prototyping. Name two or three applications that would be more difficult to prototype