JGI JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Master of Computer Applications

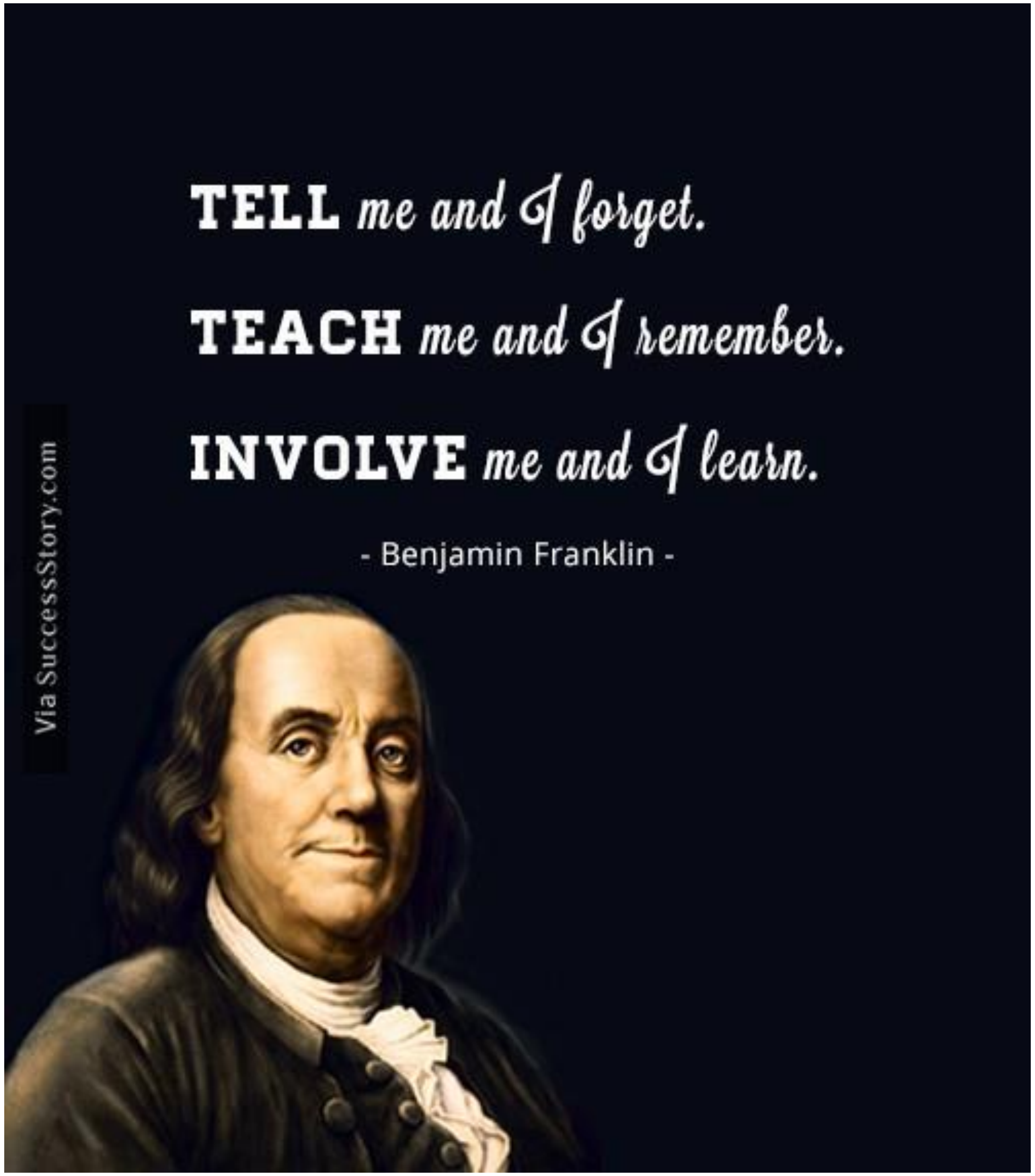
Minor Specialization – Advanced Software Engineering

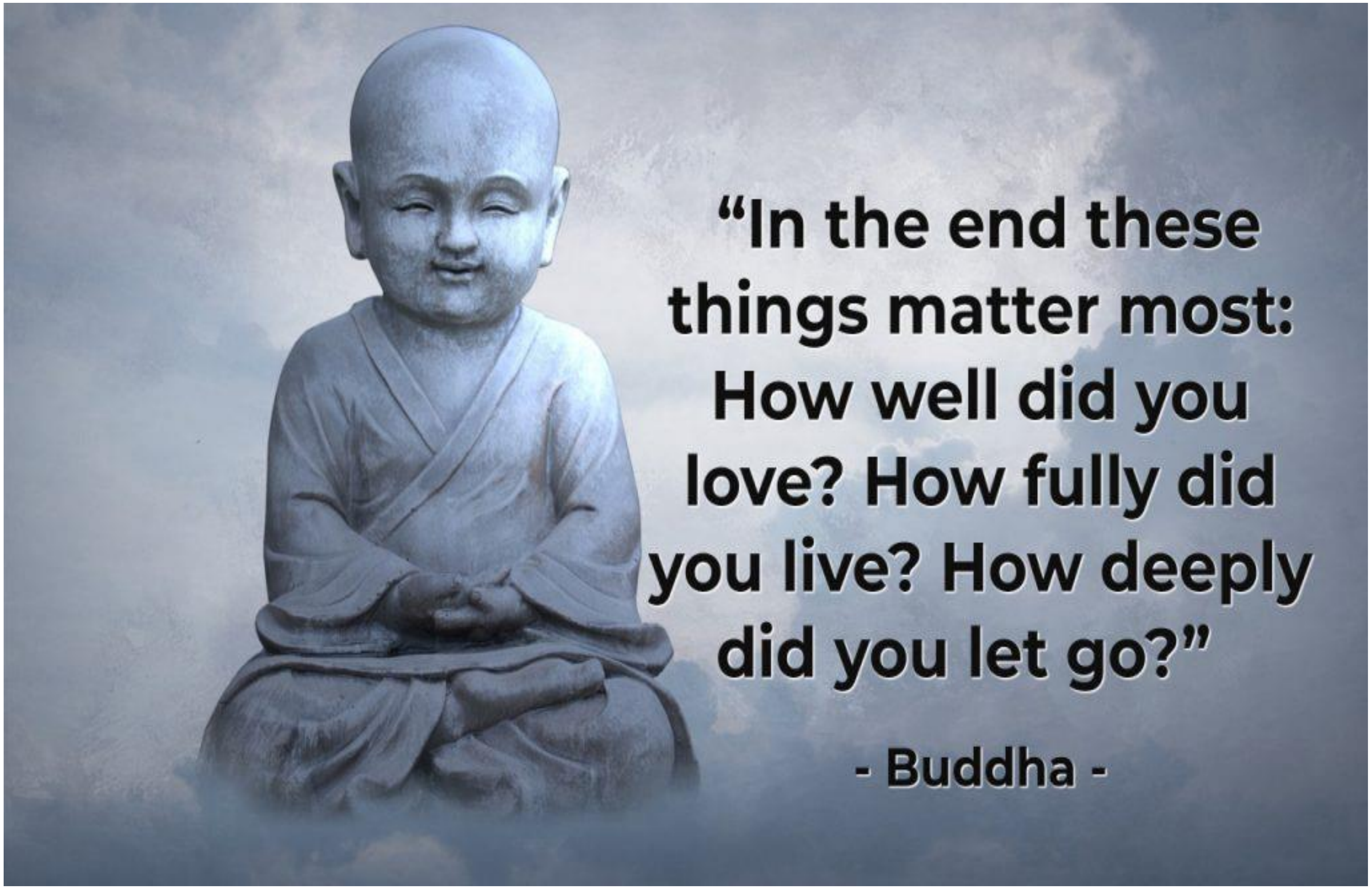
Subject code – 23MCAM11

Assistant Professor – Prof. Rahul Pawar

Module I

Introduction


TELL me and I forget.
TEACH me and I remember.
INVOLVE me and I learn.
- Benjamin Franklin -
Via SuccessStory.com



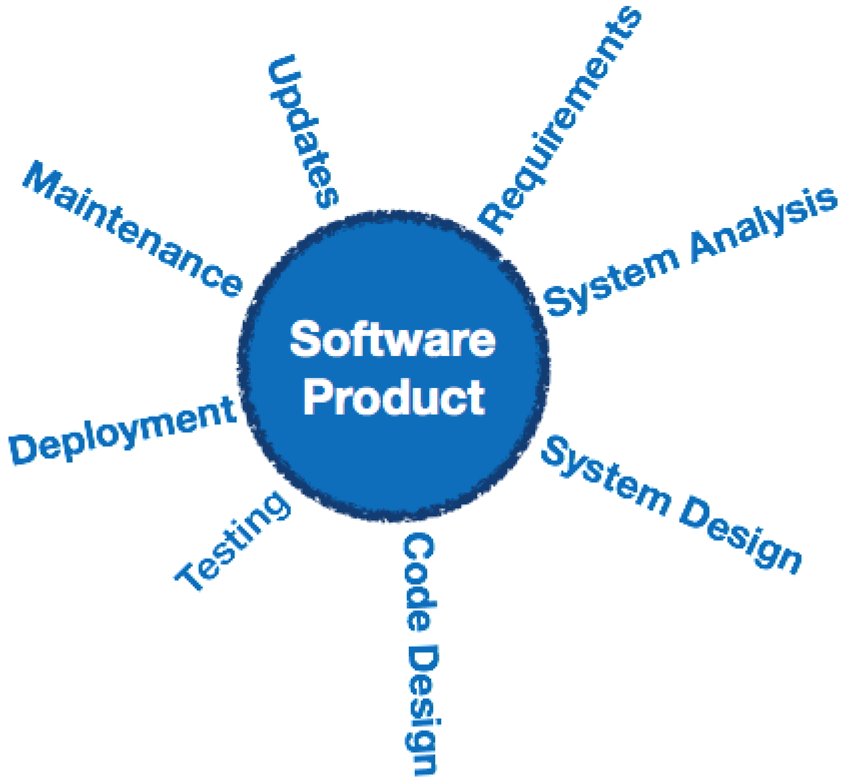
"In the end these things matter most: How well did you love? How fully did you live? How deeply did you let go?"

- Buddha -

Software Engineering

- **Software** is more than just a program code.
- A program is an executable code, which serves some computational purpose.
- Software is considered to be collection of executable programming code, associated libraries and documentations.
- Software, when made for a specific requirement is called **software product.**
- **Engineering** on the other hand, is all about **developing products, using well-defined, scientific principles and methods.**

Unit : 1


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

- **Software engineering** is an engineering branch associated with development of software product using well-defined **scientific principles, methods and procedures**.
- The outcome of software engineering is an efficient and **reliable software product.**
- **Understand the problem** before you build a solution
- **Design** is a **pivotal software Engineering Activity**
- Both **Quality and maintainability** are a result of good design.

Definitions

IEEE defines software engineering as:

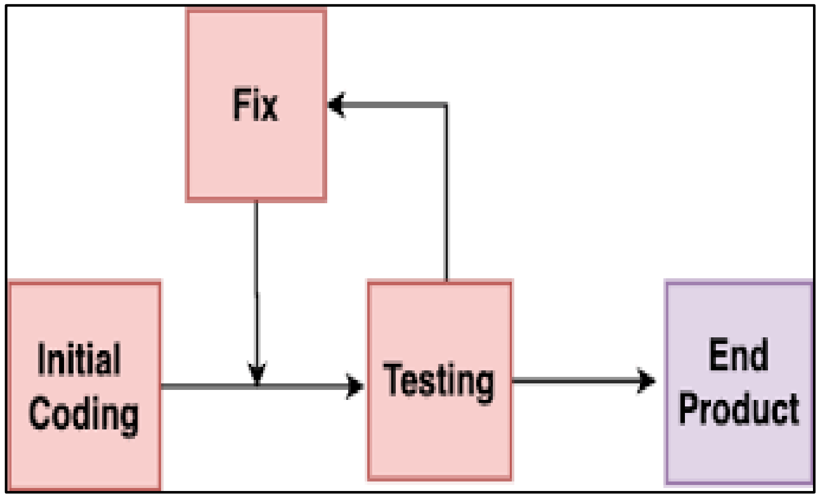
- (1) The application of a **systematic, disciplined, quantifiable approach** to the development, operation and maintenance of software; that is, the application of engineering to software.

- (2) The study of approaches as in the above statement.

Another Definition:

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

Exploratory Style



- A ‘dirty’ program is quickly developed
- The bugs are fixed as and when they are encountered/noticed by the novice programmer or developer
- Similar to how a junior student develops programs…

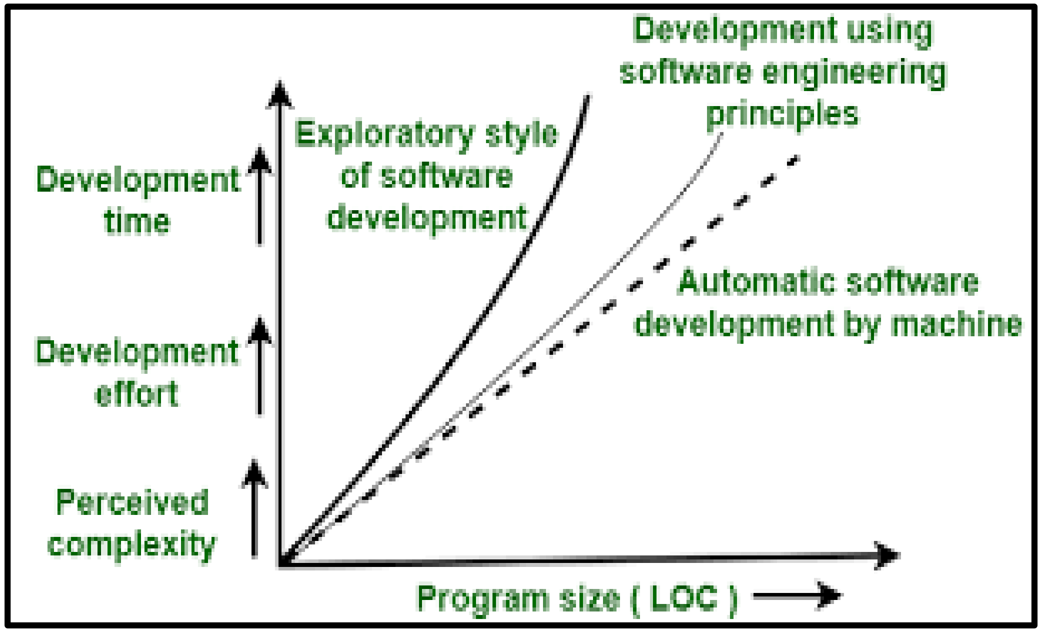


Fig. Exploratory Style of Software Development

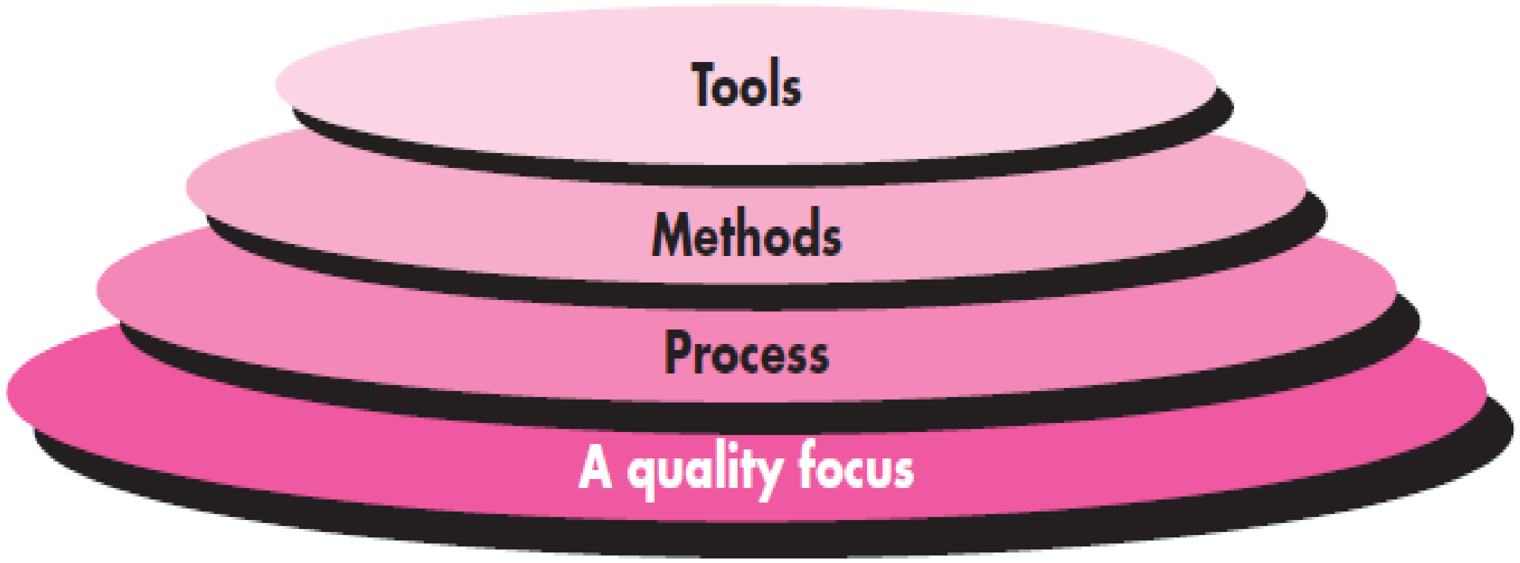
Knowledge Check

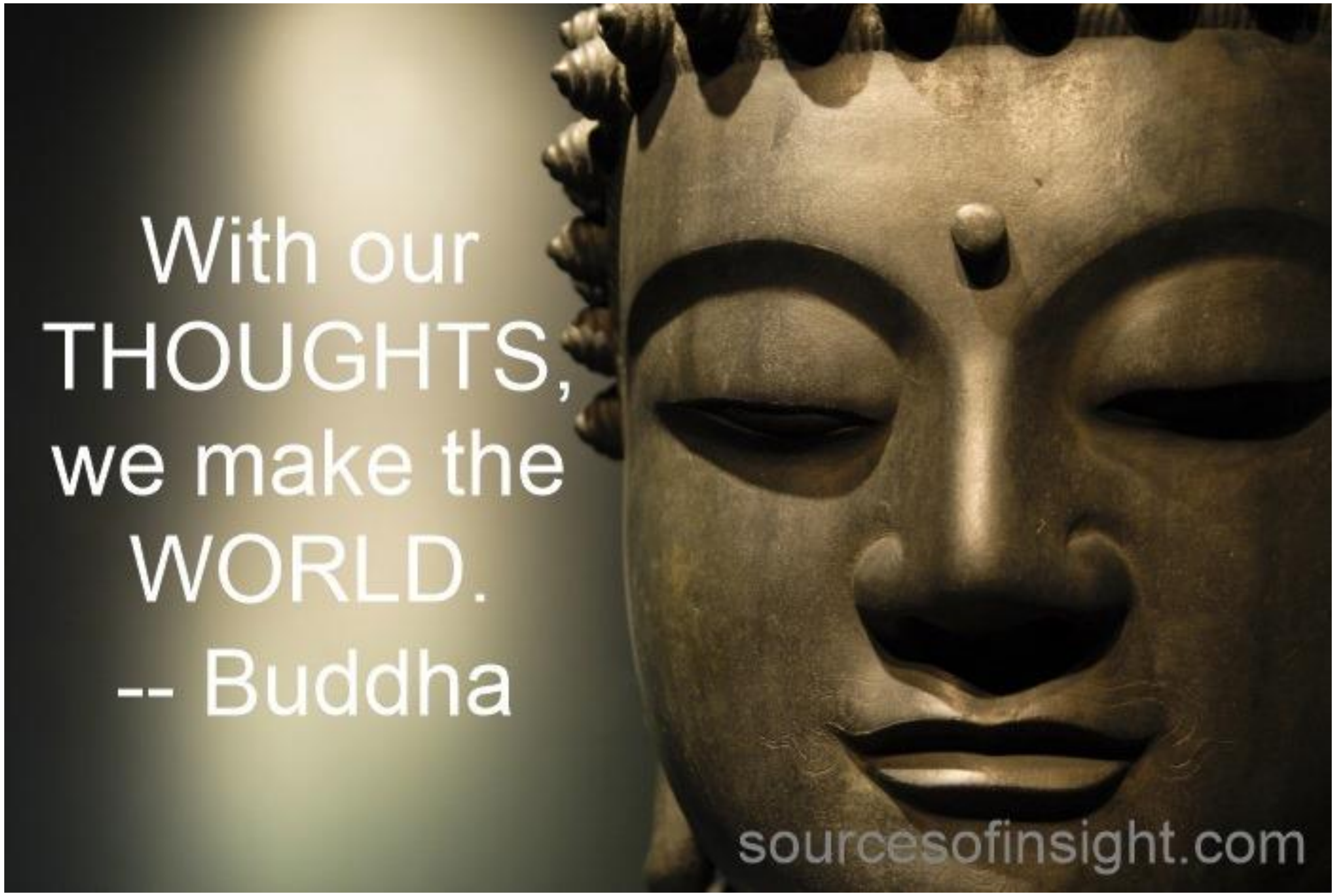
1. Stages in Exploratory Style of software development

a) Initial Coding, Testing, Fix, Final Product b) Testing, Initial Code, Fix, Final Product

c) Fix, Initial Coding, Testing, Final Product d) Intial Coding, Testing, Fix, Final Product

Software Engineering Layers




With our THOUGHTS, we make the WORLD.
-- Buddha
sourcesofinsight.com

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 to achieve a broad objective** (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An **action** (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A **task** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

- In the context of software engineering, a process is not a rigid prescription for how to build computer software.
- Rather, it is an adaptable approach that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks.
- 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, regardless of their size or complexity.
- In addition, the process framework encompasses a set of **umbrella activities** that are applicable across the entire software process.
- A **generic process framework** for software engineering encompasses **five activities**:

- **Communication:** Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders) The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **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 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.

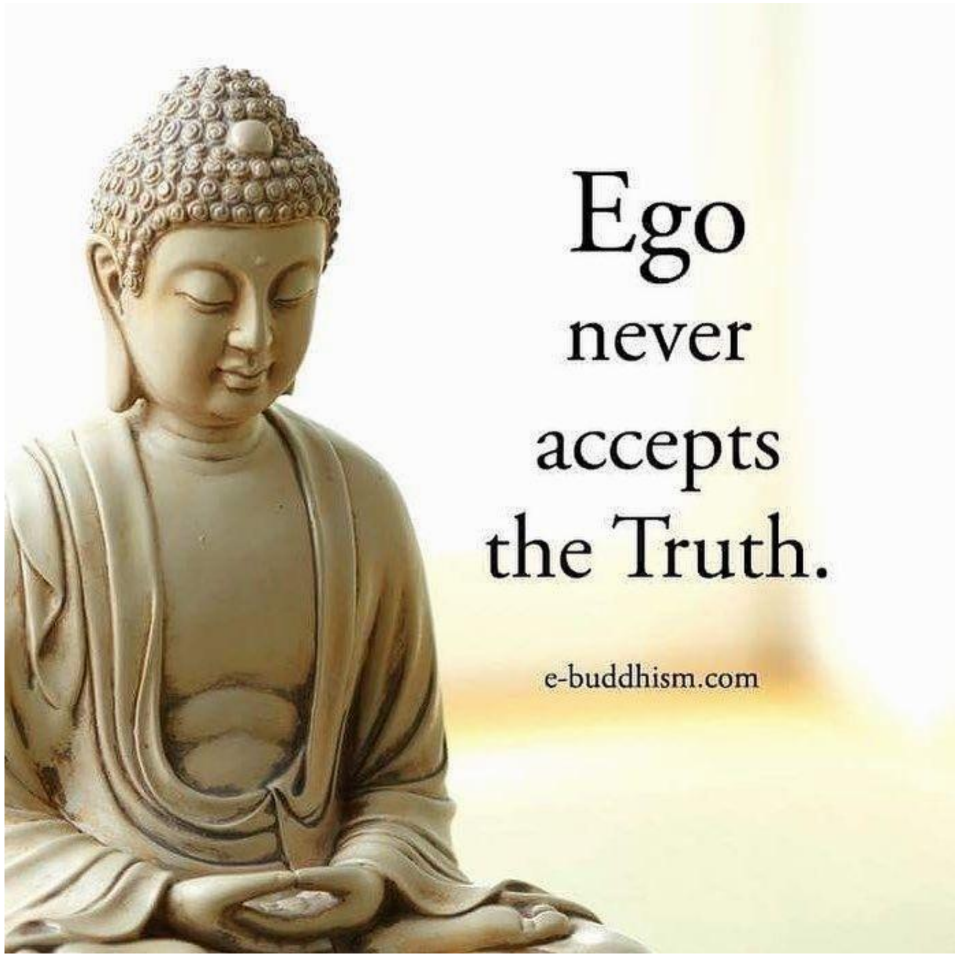
- **Modeling:** Whether you're a landscaper, a bridge builder, an aeronautical engineer, a carpenter, or an architect, you work with models every day.
- You create a **"sketch"** of the thing so that you'll understand the big picture—what it will look like architecturally, how the constituent parts fit together, and many other characteristics.
- If required, you refine the sketch into greater and greater detail in an effort to better understand the problem and how you're going to solve it.
- A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.

- **Construction:** This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
- **Deployment:** The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small, simple programs, the creation of large Web applications, and for the engineering of large, complex computer-based systems.

- Software engineering process framework activities are complemented by a number of ***umbrella activities*.**
- In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:
- **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.
- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**—defines and conducts the activities required to ensure software quality.

- **Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.


Ego
never
accepts
the Truth.
e-buddhism.com

Generic process model

- A generic process framework for software engineering defines five framework activities—communication, planning, modeling, construction, and deployment.

  1. Communication:
  The software development starts with the communication between customer and developer.

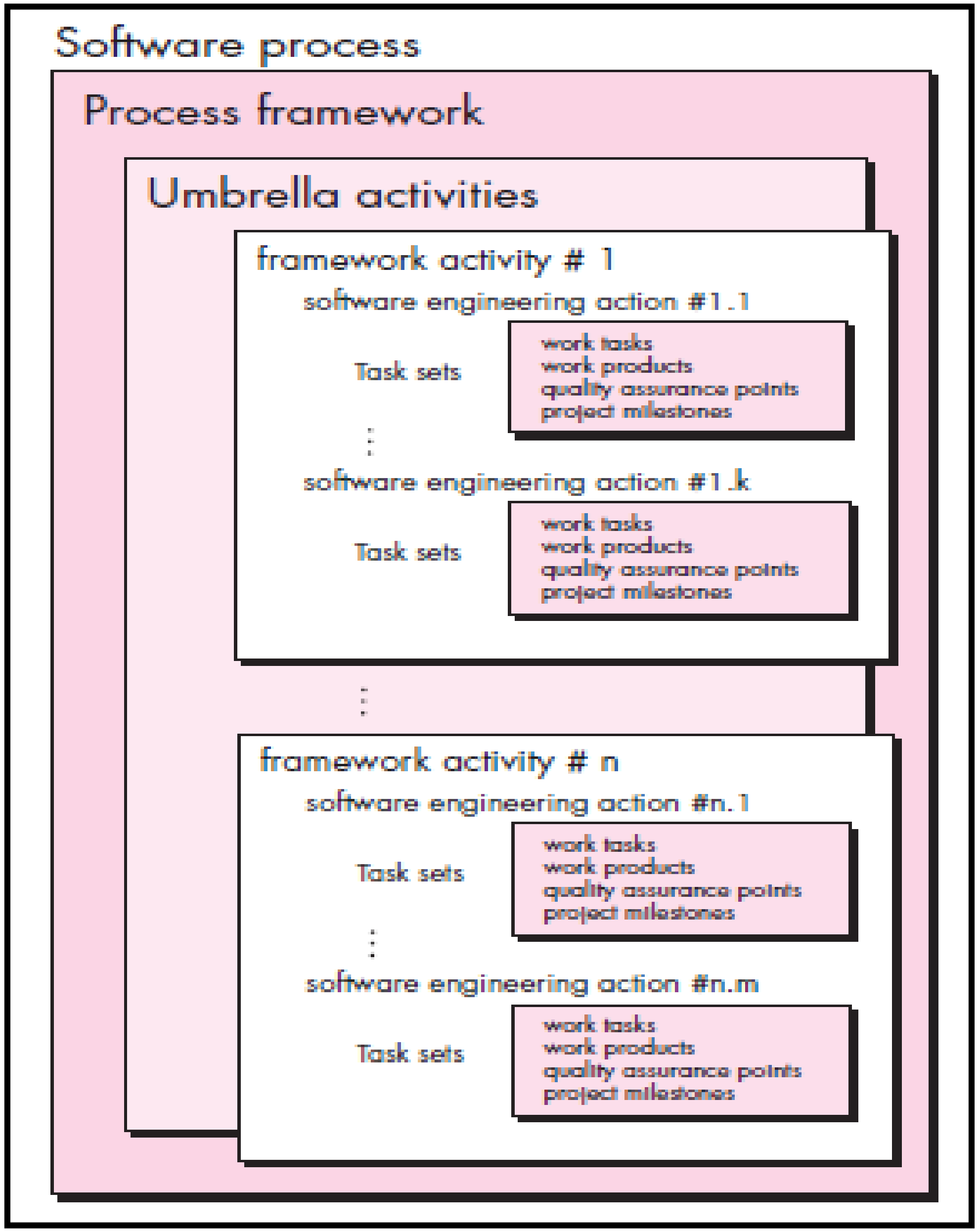
  2. Planning:
  It consists of complete estimation, scheduling for project development and tracking.

  3. Modeling: Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

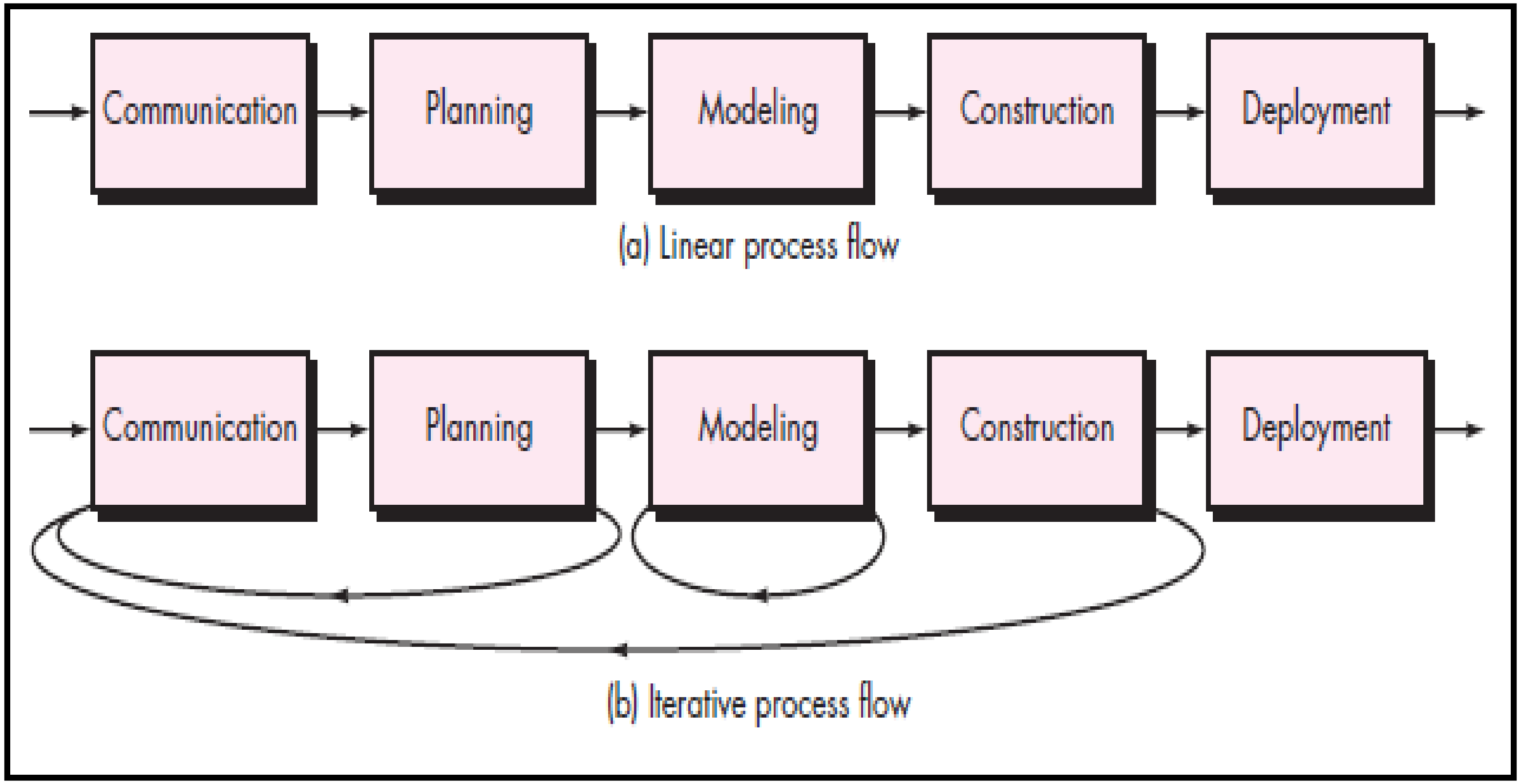
4. **Construction:** Construction consists of code generation and the testing part.
   - Coding part implements the design details using an appropriate programming language.
   - Testing is to check whether the flow of coding is correct or not.
   - Testing also check that the program provides desired output.
5. **Deployment:** Deployment step consists of delivering the product to the customer and take feedback from them.
   - If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

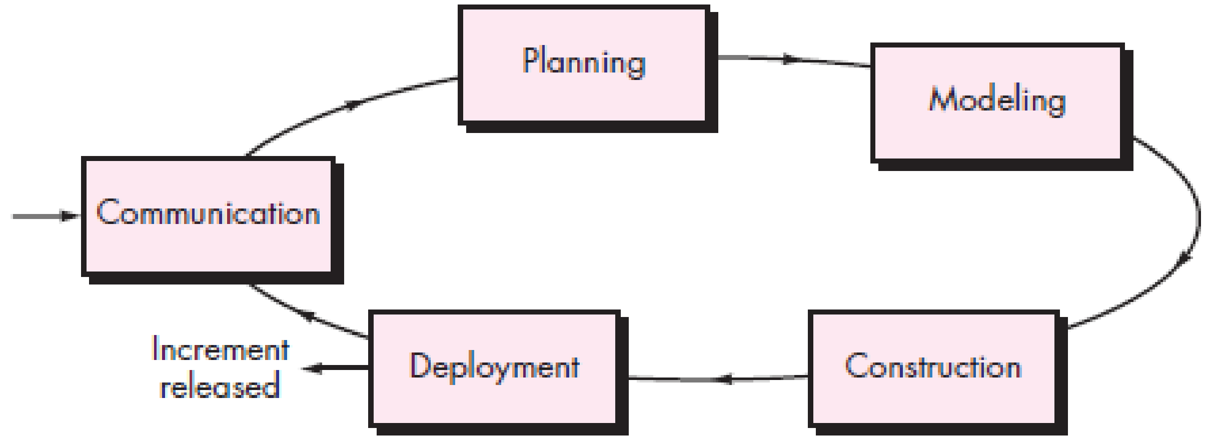
A Software Process Framework

SCHOOL OF COMPUTER SCIENCE AND IT

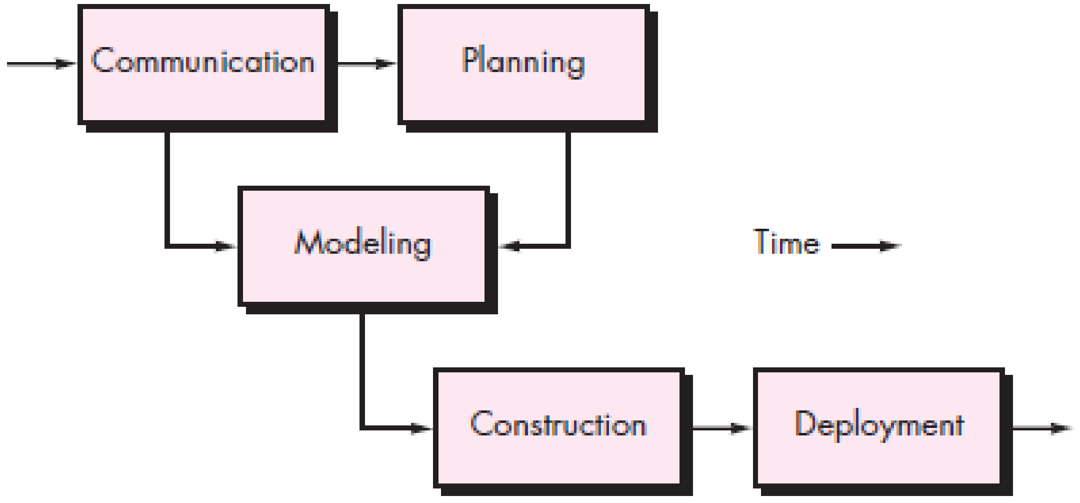


Process Flow types





(c) Evolutionary process flow



(d) Parallel process flow

Knowledge Check

1. Steps in Software Process Framework

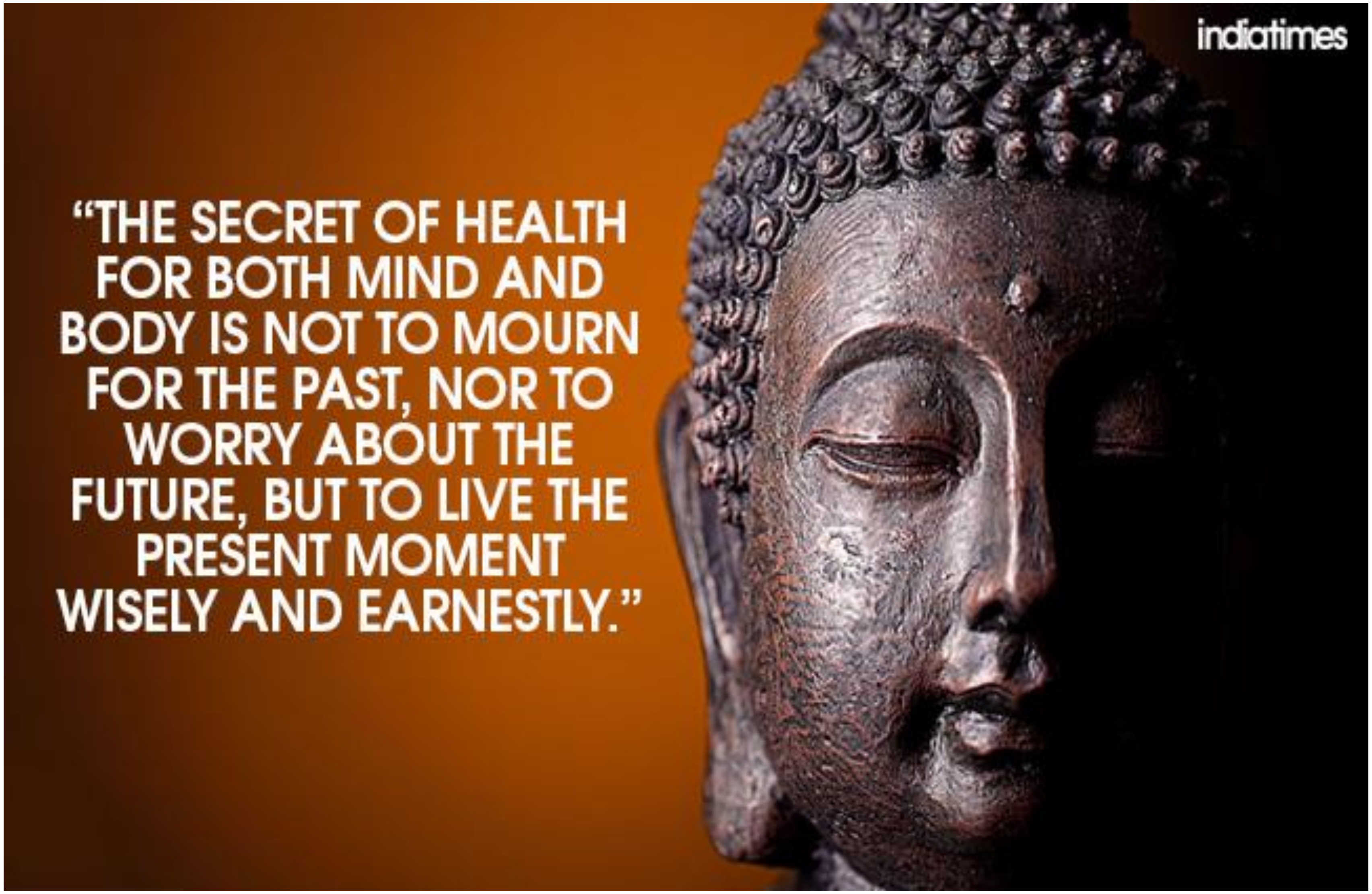
a) process framework, framework activities, umbrella activities, S E actions

b) process framework, umbrella activities, framework activities, S E actions

c) process framework, framework activities, umbrella activities, S E actions

d) umbrella activities, process framework, framework activities, S E actions

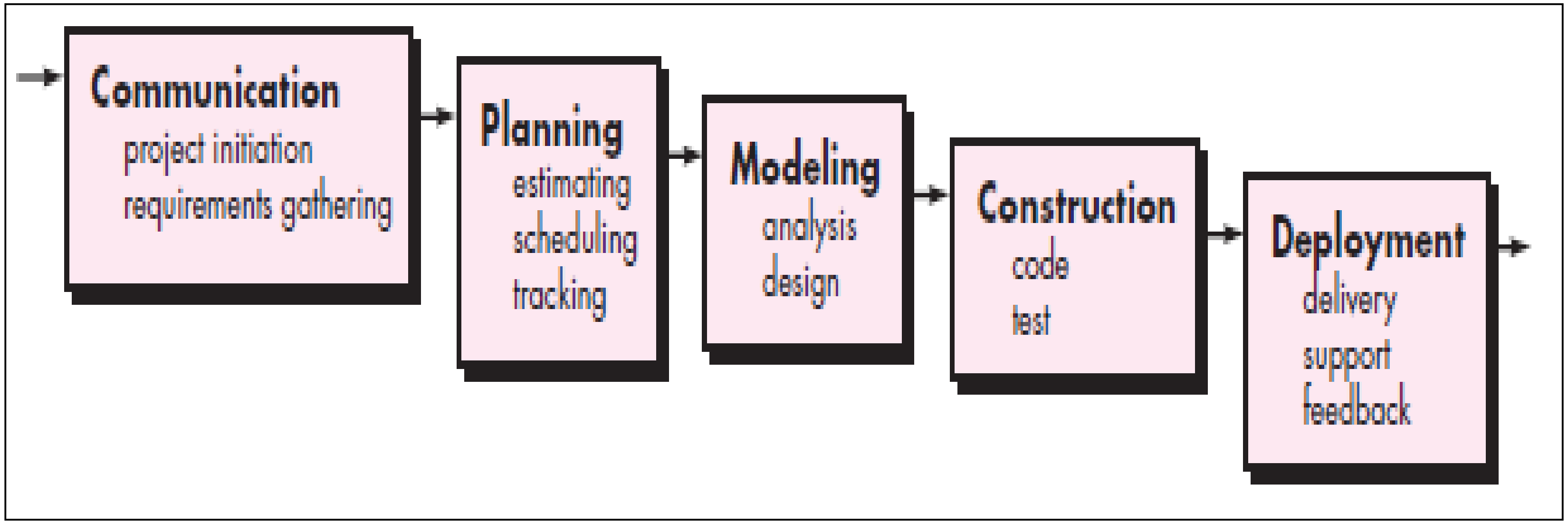
2. Steps in Generic Process Model


indiatimes
"THE SECRET OF HEALTH FOR BOTH MIND AND BODY IS NOT TO MOURN FOR THE PAST, NOR TO WORRY ABOUT THE FUTURE, BUT TO LIVE THE PRESENT MOMENT WISELY AND EARNESTLY."

Prescriptive Process Models

- Prescriptive process models were originally proposed to bring order to the **chaos** of software development.
- "Prescriptive" because they prescribe a set of process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project.

The Waterfall Model



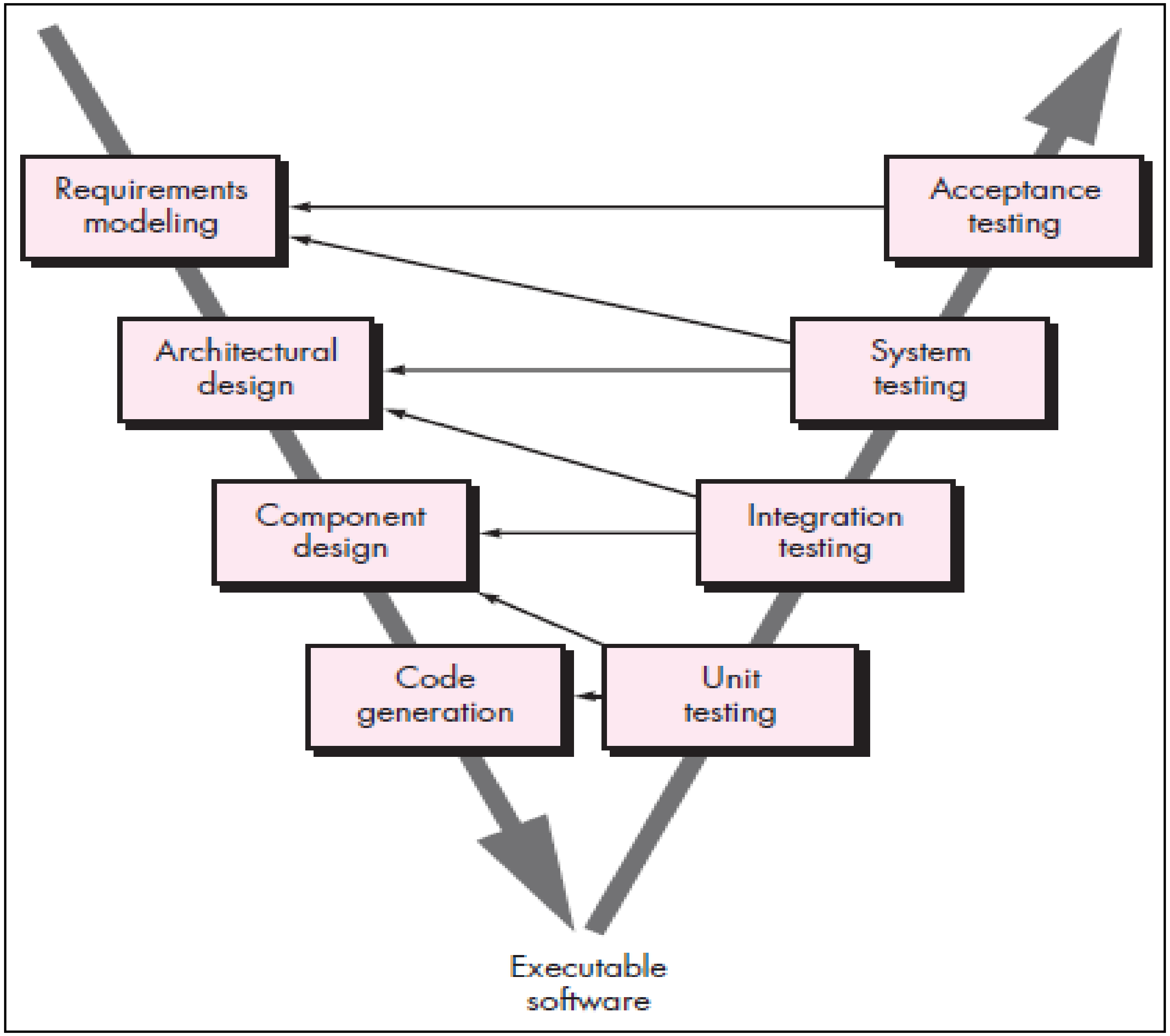
Although the original waterfall model proposed by Winston Royce [Roy70] made provision for "feedback loops," the vast majority of organizations that apply this process model treat it as if it were strictly linear.

SCHOOL OF COMPUTER SCIENCE AND IT

Problems when the Waterfall Model is applied

- Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
- It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
- The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

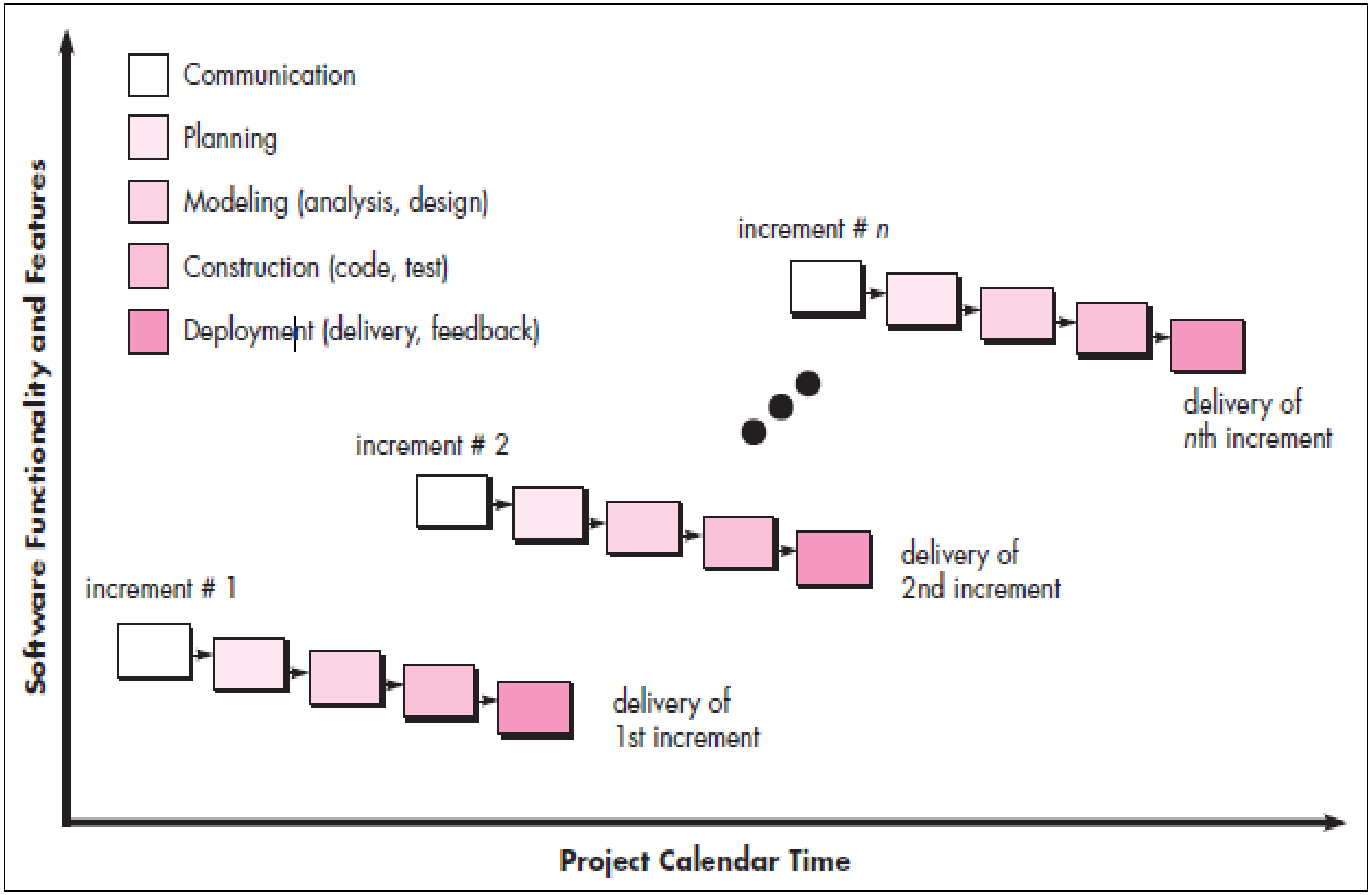
Variation of Waterfall Model
V- Model



- The V-model [Buc99] depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- 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 as the team moved down the left side.

Incremental Process Models

SCHOOL OF COMPUTER SCIENCE AND IT

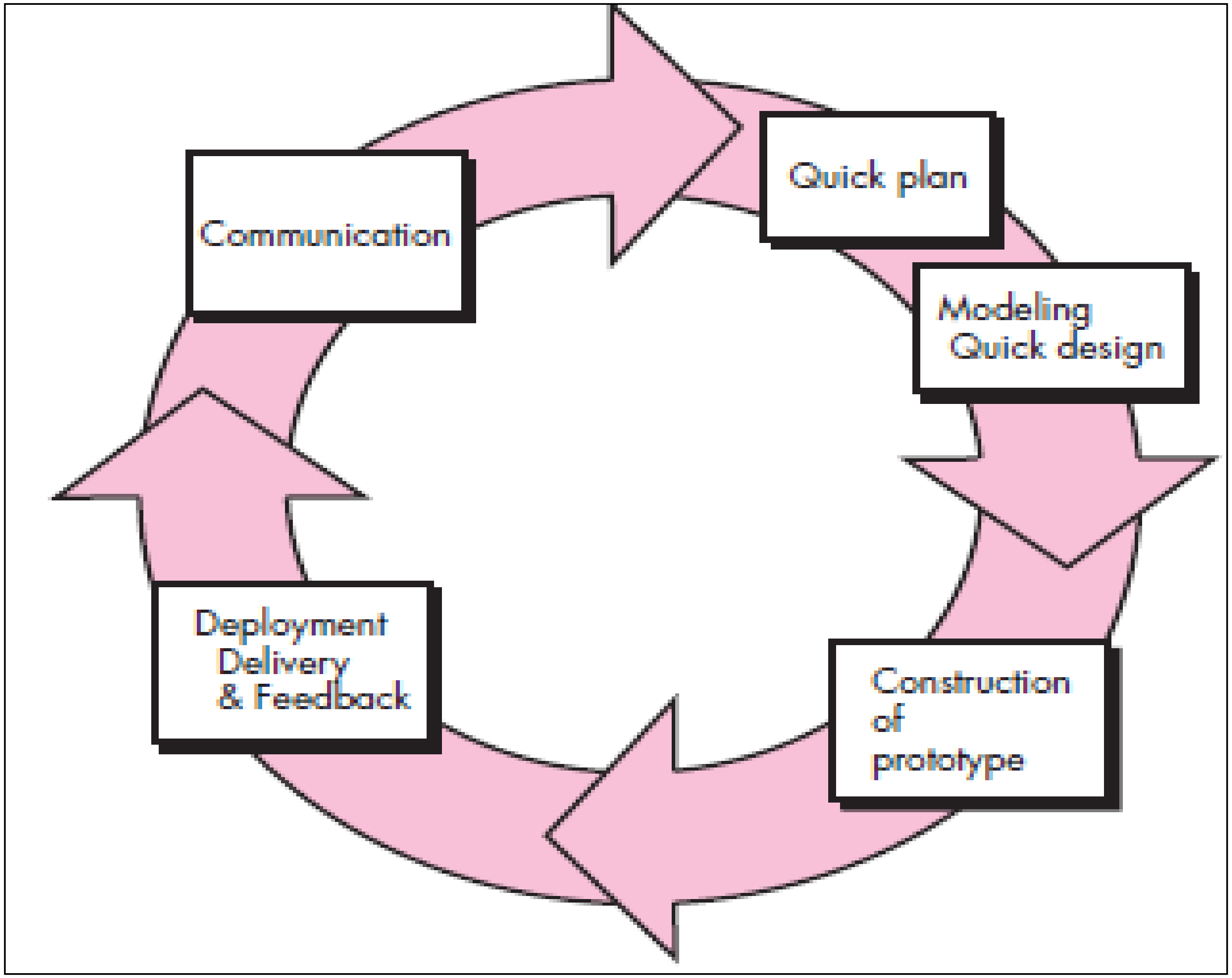

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

- The *incremental* model combines elements of linear and parallel process flows.
- the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software in a manner that is similar to the increments produced by an evolutionary process flow.
- When an incremental model is used, the first increment is often a *core product.* That is, basic requirements are addressed but many supplementary features remain undelivered.

Evolutionary Process Models

- Software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic.
- Tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure; a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.
- Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

The Prototyping Paradigm



- Prototyping Paradigm:
  - begins with communication.
  - meet with other stakeholders to define the overall objectives for the software
  - prototyping iteration is planned quickly, and modeling (quick design) occurs

A quick design focuses on a representation of those aspects of the software that will be visible to end users

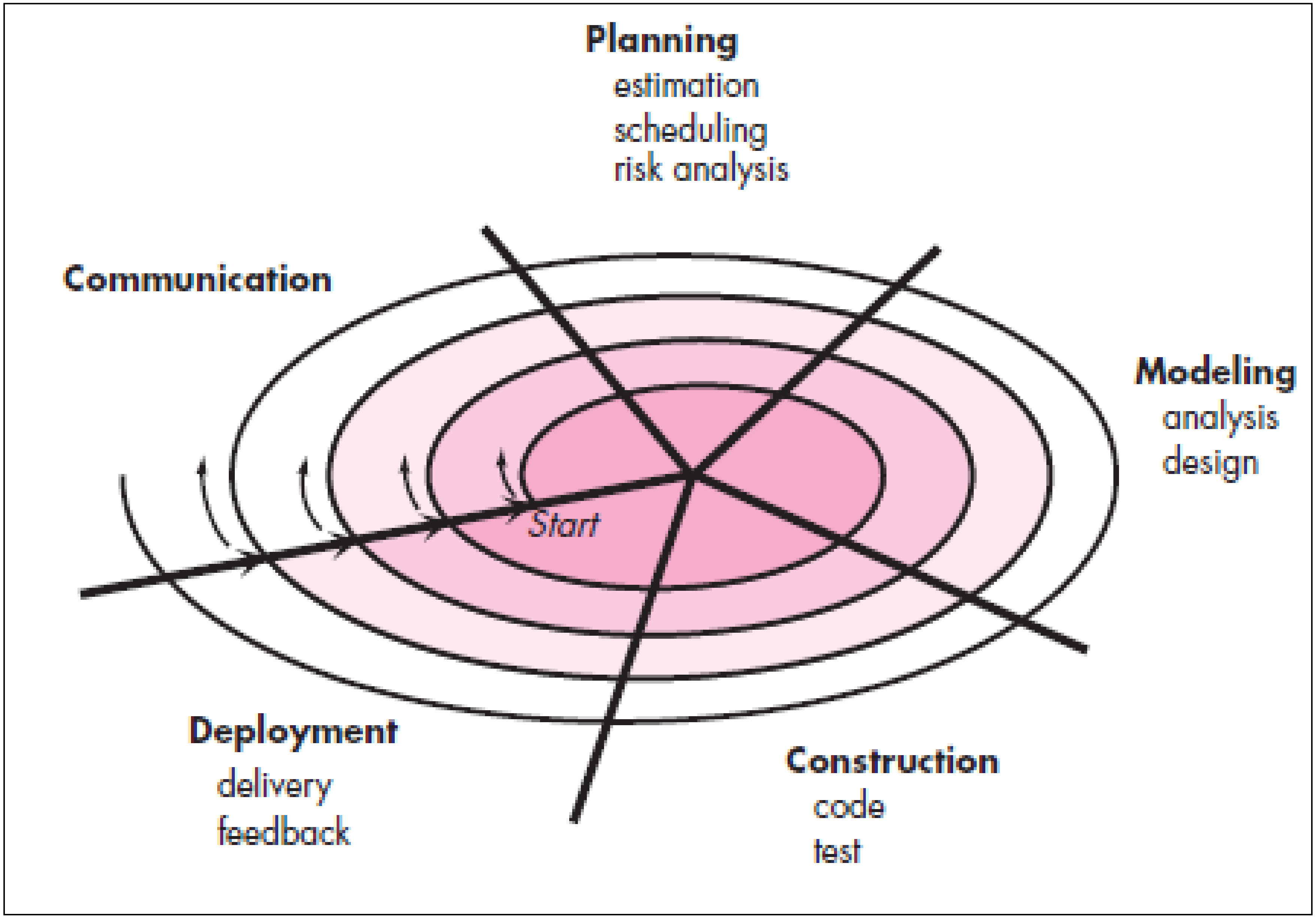
Ex: human interface layout or output display formats

The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements

Prototyping can be problematic for the following reasons:

- Confusion on main product with prototype
- An inappropriate operating system or programming language may be used
- An inefficient algorithm may be implemented simply to demonstrate capability.

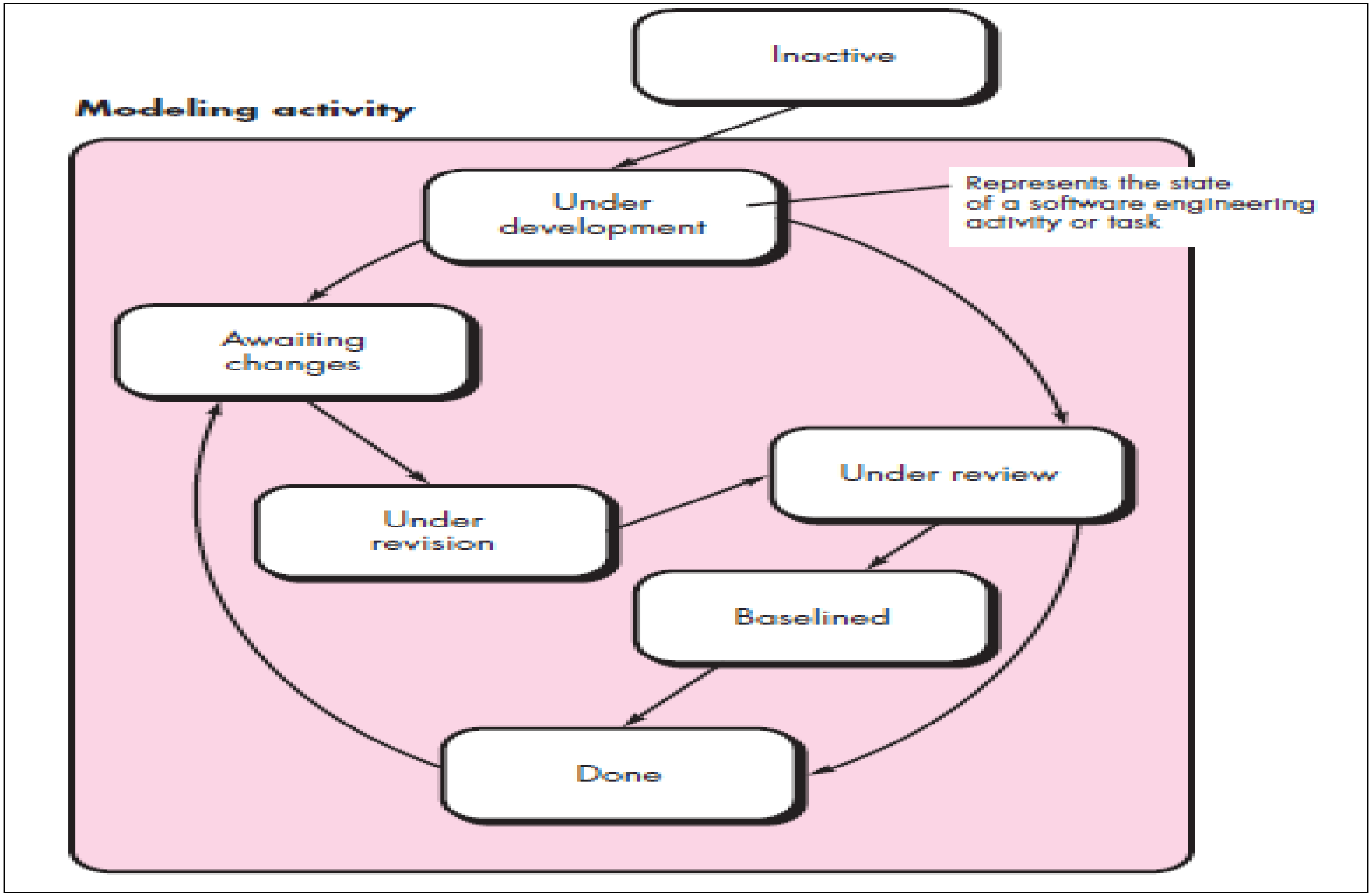
A Typical Spiral Model



- **The Spiral Model.** Originally proposed by Barry Boehm [Boe88], the *spiral model* is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.
- A spiral model is divided into a set of framework activities defined by the software engineering team.

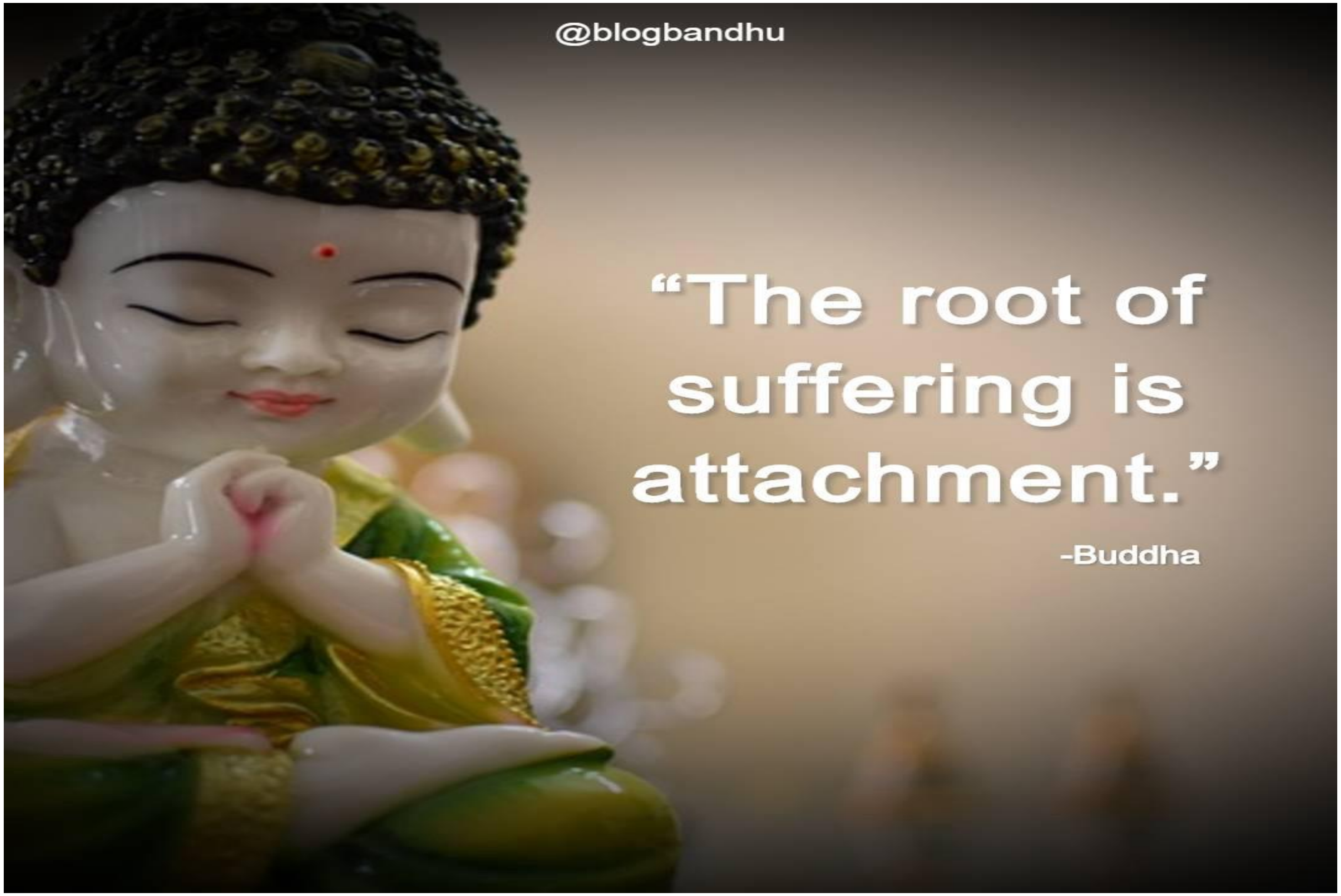
- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.

One element of the concurrent process model



- The *concurrent development model*, sometimes called *concurrent engineering*, allows a software team to represent iterative and concurrent elements of any of the process models.
- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.

Unit : 1



SPECIALIZED PROCESS MODELS

- Specialized models tend to be applied when a narrowly defined software engineering approach is chosen.
  - COMPONENT-BASED DEVELOPMENT
  - THE FORMAL METHODS MODEL
  - ASPECT-ORIENTED SOFTWARE DEVELOPMENT

COMPONENT-BASED DEVELOPMENT

- Commercial off-the-shelf (COST) software components, developed by vendors who offer them as products, can be used when software is to be built.
- It provides targeted functionality with well-defined interfaces that enable the component to be integrated into the software.
- Incorporates many of the characteristics of the spiral model
- It is evolutionary in nature, demanding an iterative approach to the creation of software.

- It incorporates the following steps:
  - Available component-based products are researched and evaluated for the application domain
  - Component integration issues are considered.
  - A software architecture is designed to accommodate the components.
  - Components are integrated into the architecture.
  - Comprehensive testing is conducted to ensure proper functionality. This model leads to software reuse.

THE FORMAL METHODS MODEL

- The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.
- It enable a software engineer to specify, develop, and verify a computer-based system by applying rigorous, mathematical notation.

Advantages:

- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through the application of mathematical analysis.
- During design, they serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might otherwise go undetected.

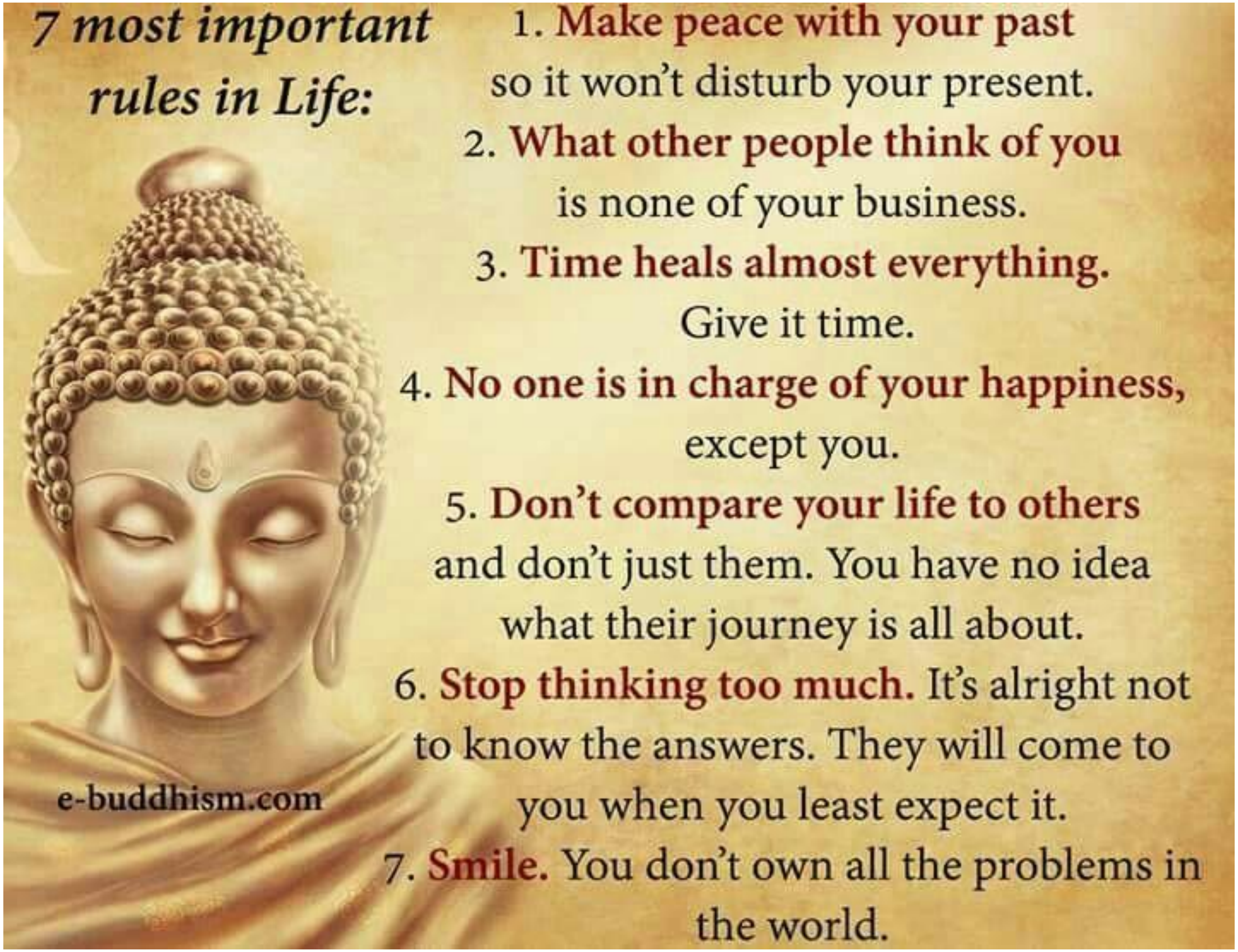
Disadvantages:

- The development of formal models is currently quite time-consuming and expensive.
- extensive training is required to apply formal methods as few software developers have the necessary background to work with this method.
- It is difficult to use the model as a communication mechanism for technically unsophisticated customers.

ASPECT-ORIENTED SOFTWARE DEVELOPMENT

- Complex software are being implemented as set of localized features, functions and information content referred as Components.
- But it becomes crosscutting concerns when it flows across multiple systems.
- Aspectual requirements define these crosscutting concerns that have impact across the software architecture.
- Aspect-oriented software development (AOSD), often referred to as aspect-oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
- The purpose of Aspect-Oriented Software Development is to provide systematic means to modularize crosscutting concerns(Cross-cutting concerns are parts of a program that rely on or must affect many other parts of the system).

- AOSD aims to address crosscutting concerns by providing means for systematic identification, separation, representation and composition.
- Crosscutting concerns are encapsulated in separate modules, known as aspects, so that localization can be promoted. This results in better support for modularization hence reducing development, maintenance and evolution costs.



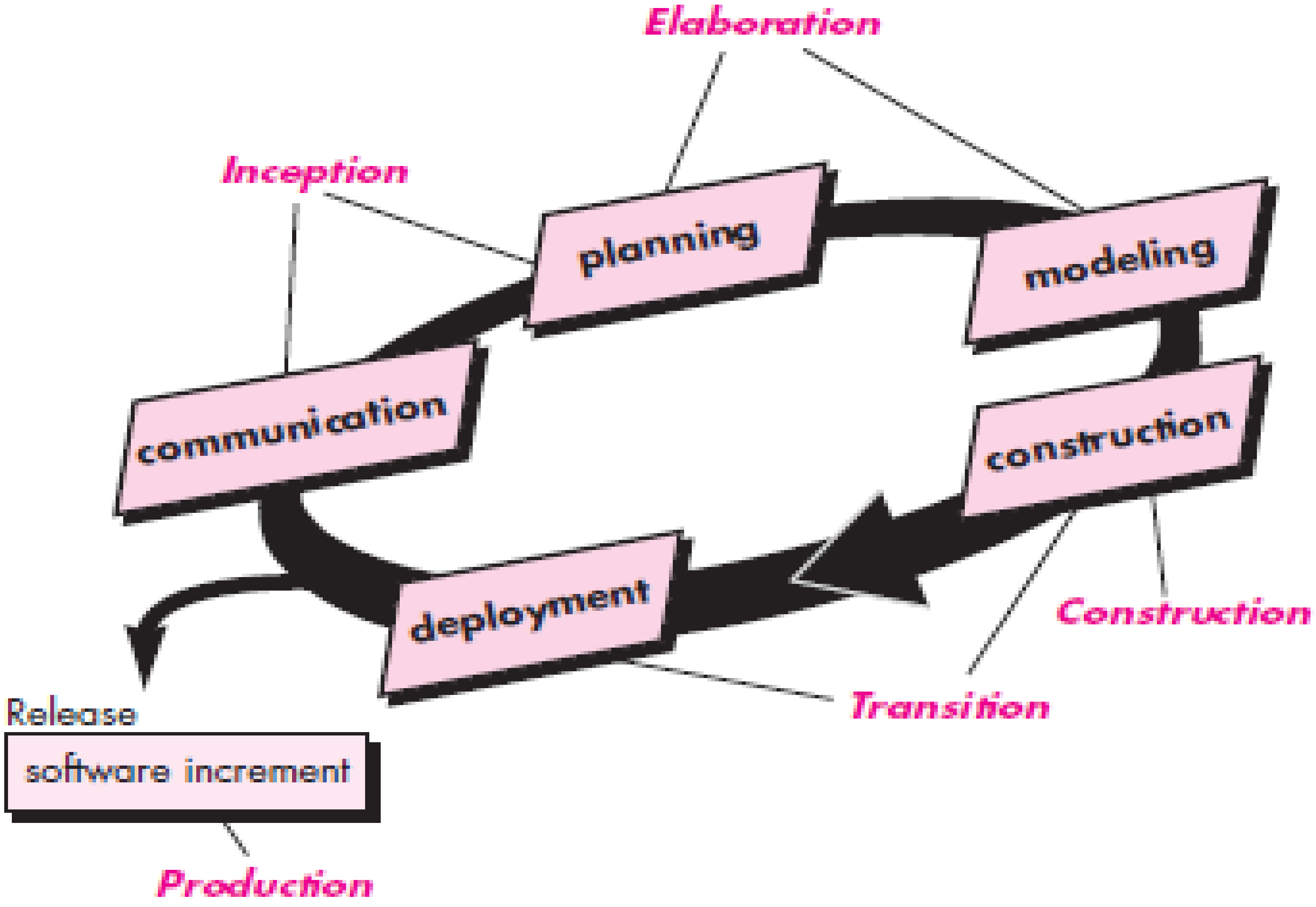
7 most important rules in Life:

1. **Make peace with your past** so it won't disturb your present.
2. **What other people think of you** is none of your business.
3. **Time heals almost everything.** Give it time.
4. **No one is in charge of your happiness,** except you.
5. **Don't compare your life to others** and don't just them. You have no idea what their journey is all about.
6. **Stop thinking too much.** It's alright not to know the answers. They will come to you when you least expect it.
7. **Smile.** You don't own all the problems in the world.

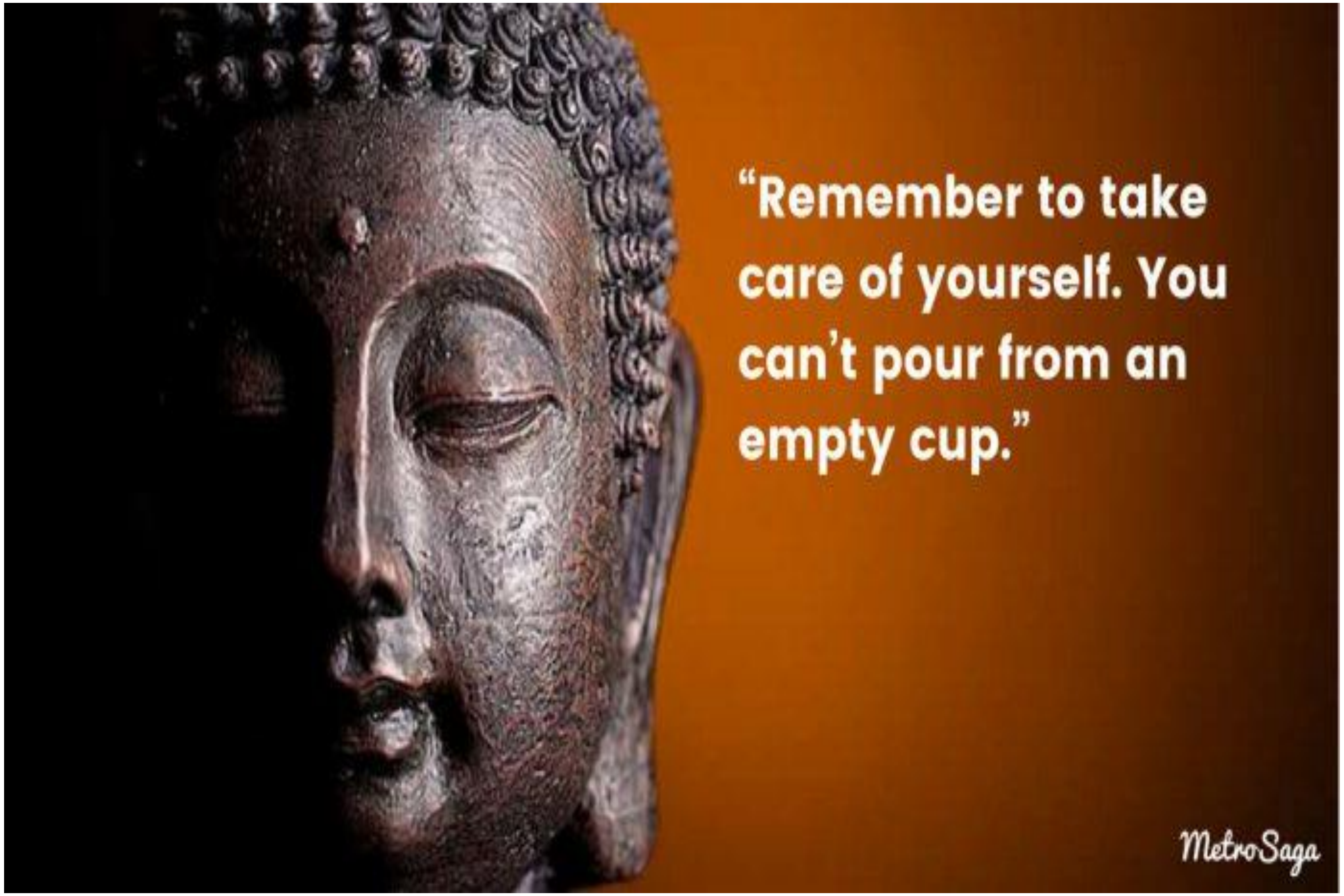
e-buddhism.com

THE UNIFIED PROCESS

- Jacabson, Rumbaugh and Booch developed the Unified Process, a framework for object-oriented software engineering using UML.
- Comprises best features and characteristics of conventional software process models.
- Recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system.
- Phases of Unified Process:
  - Inception-Involves Customer communication and planning activities.
  - Elaboration- Encompasses the customer communication and Modeling activities of the generic process model. Architectural representation using Use case model, Analysis model, Design and Implementation and Deployment model.


Elaboration
Inception
planning
modeling
communication
construction
deployment
Construction
Transition
Release
software increment
Production

- Construction = Develops or acquires the software components that will make each use case operational for end users.
- Transition = Software is given to end user for beta testing and user feedback reports both defects and necessary changes.
- Production = Coincides with the deployment activity of process. The on going use of software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.


"Remember to take care of yourself. You can't pour from an empty cup."
MetroSaga

AGILE DEVELOPMENT

- What is Agility?
  - Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
  - Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.
  - Everyone is agile. An agile team is a nimble (quick) team able to appropriately respond to changes.
  - Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product.

- The most popular Agile methods include: Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995).

These are now collectively referred to as Agile Methodologies.

Agile Process

- Any agile software process is characterized in a manner that addresses a number of key assumptions about the majority of software projects:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.

2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.

3. Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

Contd..

How do we create a process that can manage unpredictability?

Agile process, therefore, must be adaptable

Agile software process must adapt incrementally

Agile team requires customer feedback

Agility Principles: (12)

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Contd..

4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
7. Working software is the primary measure of progress.
8. Agile process, promotes sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

Contd..

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self– organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

SCHOOL OF COMPUTER SCIENCE AND IT

Human Factors – Agile Process

"Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams."

Competence: Encompasses innate talent, specific software-related skills, and overall knowledge of the process that the team has chosen to apply.

Common focus: Members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised.

Collaboration: Team members must collaborate—with one another and all other stakeholders

Decision-making ability: Good software team must be allowed the freedom to control its own destiny. given autonomy—decision-making authority for both technical and project issues

Contd..

- **Fuzzy problem-solving ability:** :Lessons learned from any problem-solving activity may be of benefit to the team later in the project.
- **Mutual trust and respect:** Trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts"
- **Self-organization:**
  - (1) the agile team organizes itself for the work to be done
  - (2) the team organizes the process to best accommodate its local environment
  - (3) the team organizes the work schedule to best achieve delivery of the software increment

Nothing motivates a team as much as accepting the responsibility for fulfilling commitments that it made itself."

Extreme Programming

- The Extreme Programming is commonly used agile process model.
- It uses the concept of object-oriented programming.
- A developer focuses on the framework activities like planning, design, coding and testing. XP has a set of rules and practices.

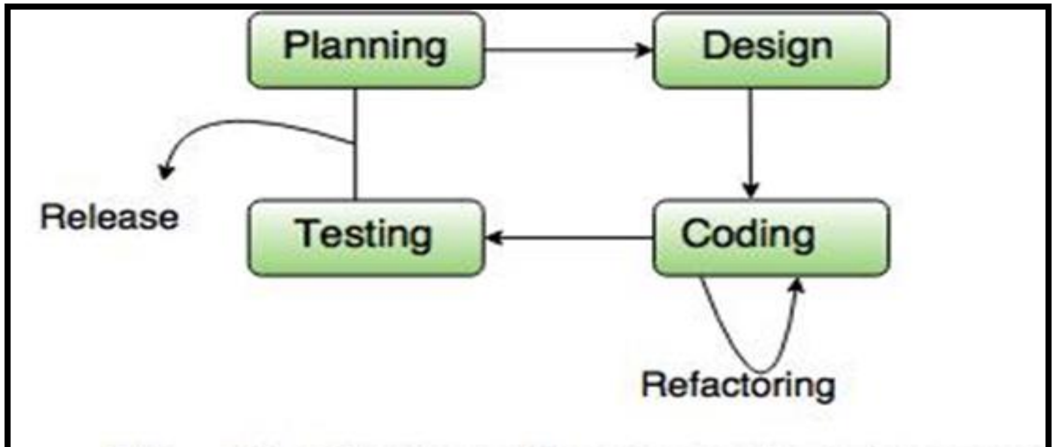


Fig. - The Extreme Programming Process

XP Values

- Beck defines a set of five values that establish a foundation for all work performed as part of XP—communication, simplicity, feedback, courage, and respect.
- **Following are the values for extreme programming:**
  1. Communication: Building software development process needs communication between the developer and the customer.

  Communication is important for requirement gathering and discussing the concept.

  2. Simplicity - The simple design is easy to implement in code.
  3. Feedback - guides the development process in the right direction.
  4. Courage - In every development process there will always be a pressure situation. The courage or the discipline to deal with it surely makes the task easy.
  5. Respect - Agile process should inculcate the habit to respect all team members, other stake holders and customer.

The XP Process

The XP process comprises 4 framework activities:

Planning : Planning starts with the requirements gathering which enables XP team to understand the rules for the software.

- The customer and developer work together for the final requirements.

Design: The XP design follows the 'keep it simple' principle.

- A simple design always prefers the more difficult representation.

Coding : The coding is started after the initial design work is over.

- After the initial design work is done, the team creates a set of unit tests which can test each situation that should be a part of the release.
- The developer is focused on what must be implemented to pass the test.
- Two people are assigned to create the code. It is an important concept in coding activity.

Testing:

- Validation testing of the system occurs on a daily basis. It gives the XP team a regular indication of the progress.
- 'XP acceptance tests' are known as the customer test.

Other Agile Process models

Scrum

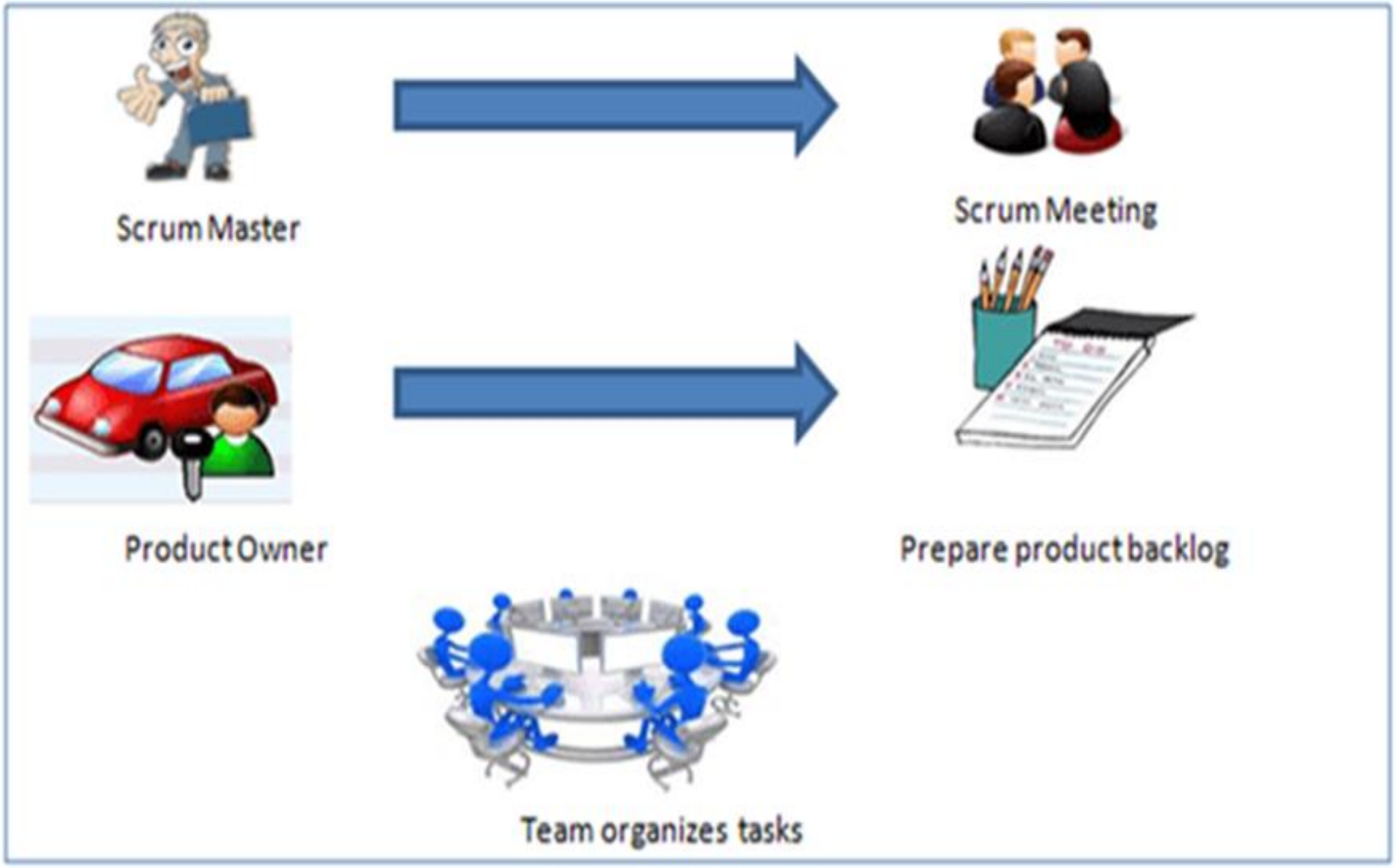
Scrum principles are consistent with the agile platform that are used to guide development activities within a process.

It includes the framework activities like requirement analysis, design, evolution and delivery.

Work tasks occur within a process pattern in each framework activity called as **'sprint'.**

Scrum highlights the use of a set of software process pattern that are effective for the projects with tight timelines, changing requirements and business criticality.

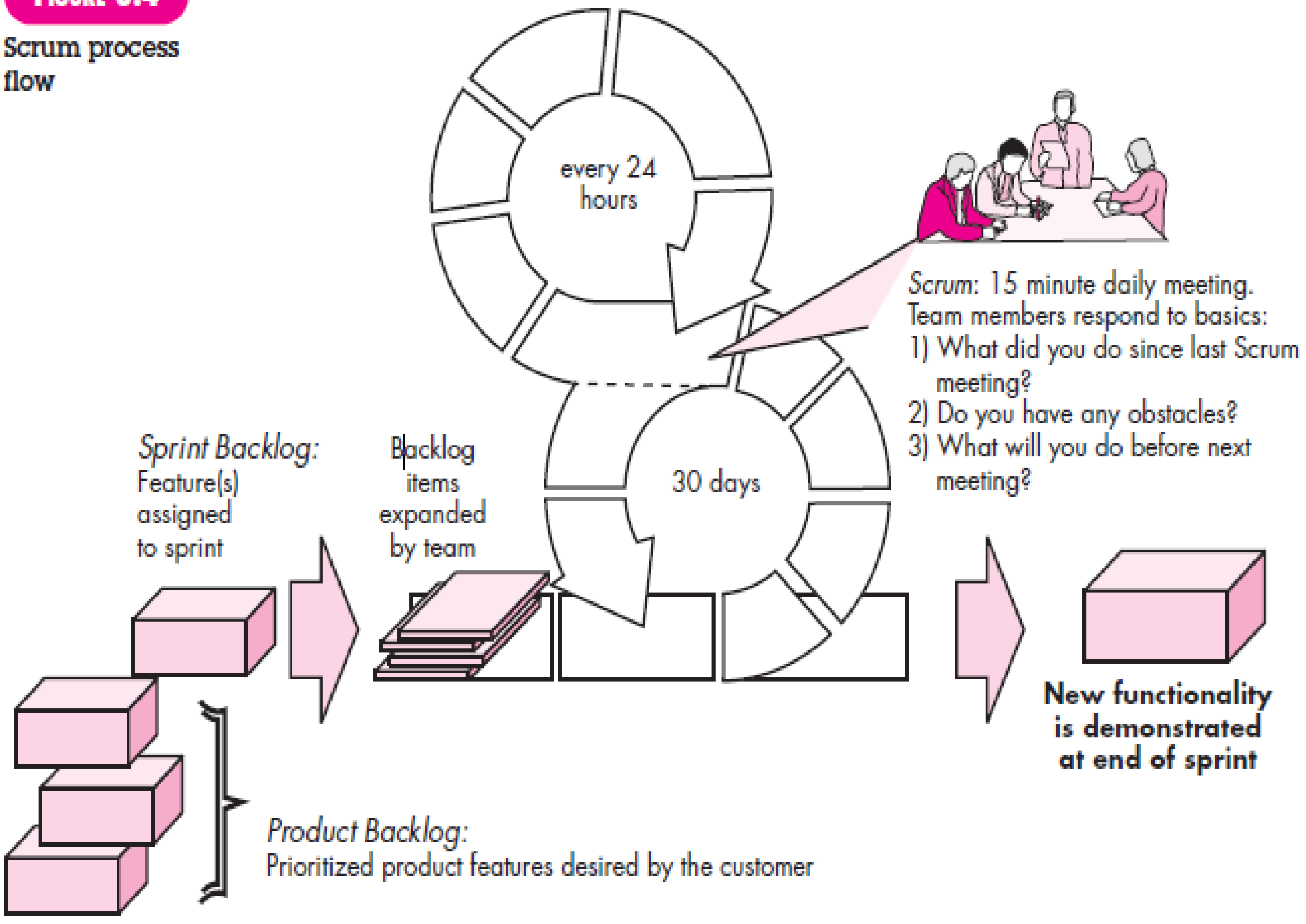
It consists of three roles, and their responsibilities are explained as follows:



- Scrum Master
  - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- Product owner
  - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- Scrum Team
  - Team manages its own work and organizes the work to complete the sprint or cycle.

FIGURE 3.4

Scrum process flow



every 24 hours
30 days
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Each process patterns defines a set of development actions which are as follows:

- **Backlog**
  - A prioritized list of project requirements or features that provide business value for the customer.
  - Items can be added to the backlog at any time.
  - The product manager accesses the backlog and updates priorities, as required.
- **Sprints**
  - It consists of work units that are required to achieve a requirement defined in the backlog.
  - Changes are not introduced during the sprints. It allows team members to work in short-term but in the stable environment.

- **Scrum meeting**
  - The short meetings are held daily by the scrum team.
  - The key questions are asked and answered by all team members.
  - Daily meetings guide to 'knowledge socialization' and that encourages a self-organizing team structure.
- **Demos**
  - Deliver the software increment to the customer. Using which the customer evaluates and demonstrates the functionalities.

DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM)

- It is an agile software development
- "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment"
- Pareto principle —80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application
- DSDM is an iterative software process in which each iteration follows the 80 percent
- That is, only enough work is required for each increment to facilitate movement to the next increment.

DSDM (contd…)

DSDM life cycle that defines 3 different iterative cycles

- **Feasibility study**
  - establishes the basic business requirements and constraints associated with the application to be built
  - then assesses whether the application is a viable candidate for the DSDM process
- **Business study**

  - establishes the functional and information requirements will allow the application to provide business value

  - defines the basic application architecture and identifies the maintainability requirements for the application.

DSDM (contd…)

- **Functional model iteration**
  - —produces a set of incremental prototypes that demonstrate functionality for the customer associated with the application to be built
  - – is to gather additional requirements by eliciting feedback from users

- **Design and build iteration**

  - revisits prototypes to ensure that provide operational business value for end users.

  - **Functional model** iteration and **Design and build iteration occur concurrently**

DSDM (contd…)

- **Implementation**—places the latest software increment into the operational environment.

  It should be noted that

  (1) the increment may not be 100 percent complete

  (2) changes may be requested as the increment is put into place

  DSDM development work continues by returning to the functional model iteration activity

Crystal Family of Agile methods

SCHOOL OF COMPUTER SCIENCE AND IT

Cockburn and Jim Highsmith created the Crystal family of agile methods in order to achieve a software development approach that puts a premium on "maneuverability" (skillfully & carefully)

To achieve maneuverability, Cockburn and Highsmith have defined a set of methodologies

- roles, process patterns, work products, and practice that are unique to each.

Crystal family is actually a set of example agile processes that have been proven effective for different types of projects.

The intent is to allow agile teams to select the member of the crystal family that is most appropriate for their project and environment.

Feature Driven Development (FDD)

practical process model for object-oriented software engineering

FDD adopts a philosophy that

(1) emphasizes collaboration among people on an FDD team

(2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and

(3) communication of technical detail using verbal, graphical, and text-based means

In FDD, a **feature** "is a client-valued function that can be implemented in two weeks or less"

Feature Driven Development (FDD)

The emphasis on the definition of features provides the following benefits:

•Features are small blocks of deliverable functionality,

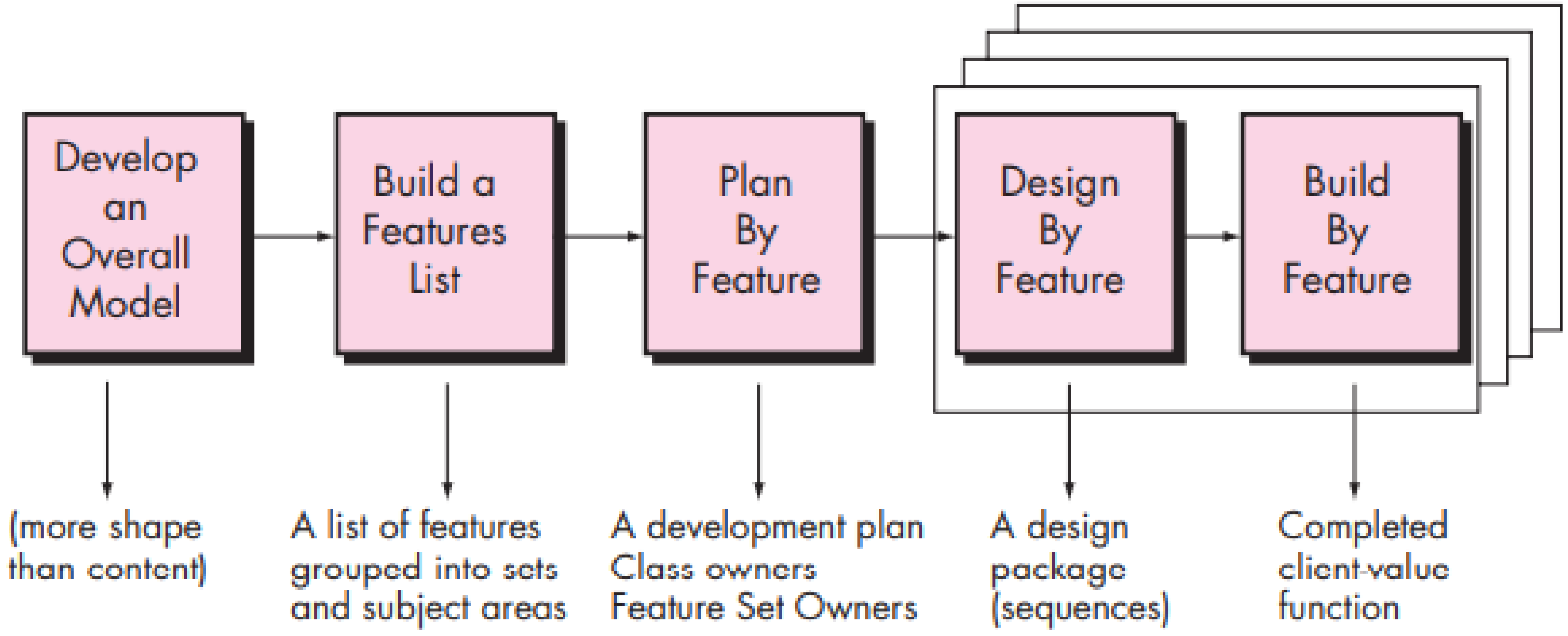
- users can describe them more easily;
- understand how they relate to one another more readily; and
- better review them for ambiguity, error, or omissions.

• Features can be organized into a hierarchical business-related grouping.

• Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.

Feature Driven Development (FDD)

- Because features are small, their design and code representations are easier to inspect effectively.
- Project planning, scheduling, and tracking are driven by the feature hierarchy


Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
A list of features grouped into sets and subject areas
A development plan Class owners Feature Set Owners
A design package (sequences)
Completed client-value function

Lean Software Development

- **Lean principles:**

  - eliminate waste, build quality, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole

- **LSD process:**

(1) **adding no extraneous features** or functions

(2) **assessing the cost and schedule impact** of **any newly requested requirement**

(3) **removing any surplus or extra process steps**

(4) **establishing mechanisms to improve the way** team members find information

(5) **ensuring the testing** finds as **many errors** as possible

(6) **reducing the time required** to request and **get a decision that affects the software or the process** that is applied to create it

(7) **streamlining the manner** in which **information is transmitted to all stakeholders**

SOFTWARE ENGINEERING KNOWLEDGE

- Many software practitioners think of software engineering knowledge almost exclusively as knowledge of specific technologies: Java, Perl, html, C, Linux, Windows NT, and so on. Knowledge of specific technology details is necessary to perform computer programming.
- If someone assigns you to write a program in C, you have to know something about C to get your program to work.

CORE PRINCIPLES

- Software engineering is guided by a collection of core principles that help in the application of a meaningful software process and the execution of effective software engineering methods.
- General principles that span software engineering process and practice:
  - provide value to end users,
  - keep it simple,
  - maintain the vision (of the product and the project),
  - recognize that others consume (and must understand) what you produce,
  - be open to the future,
  - plan ahead for Reuse

Principle 1. ***Be agile***

Principle 2. ***Focus on quality at every step***

Principle 3. ***Be ready to adapt***

Principle 4. ***Build an effective team***

Principle 5. ***Establish mechanisms for communication and coordination***

Principle 6. ***Manage change***

Principle 7. ***Assess risk***

Principle 8. ***Create work products that provide value for others***

JGi JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Principles That Guide Practice

- Software engineering practice has a single overriding goal—to deliver on-time, high quality, operational software that contains functions and features that meet the needs of all stakeholders. To achieve this goal, you should adopt a set of core principles that guide your technical work.
- **Principle 1. Divide and conquer**
- **Principle 2. Understand the use of abstraction**
- **Principle 3. Strive for consistency-**a familiar context make software easier to use.
- **Principle 4. Focus on the transfer of information**
- **Principle 5. Build software that exhibits effective modularity**-Modularity
- **Principle 6. Look for patterns.**

- **Principle 7. When possible, represent the problem and its solution from a number of different perspectives.**
- **Principle 8. Remember that someone will maintain the software.**

SCHOOL OF COMPUTER SCIENCE AND IT

PRINCIPLES THAT GUIDE EACH FRAMEWORK ACTIVITY

Communication Principles

- **Principle 1.** ***Listen.***
- **Principle 2.** ***Prepare before you communicate***

  -Spend the time to understand the problem before you meet with others.
- **Principle 3.** ***Someone should facilitate the activity*****.** have a leader (a facilitator) to keep the conversation moving in a productive direction, (2) to mediate any conflict that does occur, and (3) to ensure than other principles are followed.
- **Principle 4.** ***Face-to-face communication is best.***
- **Principle 5.** ***Take notes and document decisions***
- **Principle 6.** ***Strive for collaboration*****.**

- **Principle 7. *Stay focused; modularize your discussion*.**
- **Principle 8. *If something is unclear, draw a picture***
- **Principle 9. *(a) Once you agree to something, move on. (b) If you can't agree to something, move on. (c) If a feature or function is unclear and cannot be clarified at the moment, move on.***
- **Principle 10. *Negotiation is not a contest or a game. It works best when both parties win.***

Planning Principles

- The **planning activity** - a set of management and technical practices that enable the software team to define a road map.
- **Principle 1. Understand the scope of the project-** provides the software team with a destination.
- **Principle 2. Involve stakeholders in the planning *activity*-** Stakeholders define priorities and establish project constraints.
- **Principle 3. Recognize that planning is iterative*.***
- **Principle 4. Estimate based on what you know-** intent is to provide an indication of effort, cost, and task duration
- **Principle 5. Consider risk as you define the plan*.***
- **Principle 6. Be realistic-** Even the best software engineers make mistakes.

- **Principle 7. Adjust granularity as you define the plan**
  - Granularity refers to the level of detail that is introduced as a project plan is developed.
  - A "high-granularity" plan provides significant work task detail that is planned over relatively short time increments (so that tracking and control occur frequently).
  - A "low-granularity" plan provides broader work tasks that are planned over longer time periods.

- **Principle 8. Define how you intend to ensure quality-** technical reviews, pair programming should be explicitly defined within the plan if required.

- **Principle 9. Describe how you intend to accommodate change**-can the customer request a change at any time? If a change is requested, is the team obliged to implement it immediately?

- **Principle 10. Track the plan frequently and make adjustments as required.**
  - Track progress on a daily basis, looking for problem areas and situations in which scheduled work does not conform to actual work conducted.

Modeling Principles

- Design models provide a concrete specification for the construction of the software.
- **Principle 1.** ***The primary goal of the software team is to build software, not create models.***
    - Models that slow the process down or provide little new insight should be avoided.
- **Principle 2.** ***Travel light—don't create more models than you need.***
- **Principle 3.** ***Strive to produce the simplest model that will describe the problem or the software.***
- **Principle 4.** ***Build models in a way that makes them open to change.***
- **Principle 5.** ***Be able to state an explicit purpose for each model that is created.***

- **Principle 6.** ***Adapt the models you develop to the system at hand.***
- **Principle 7.** ***Try to build useful models, but forget about building perfect models.***
- **Principle 8.** ***Don't become strict about the syntax of the model. If it communicates content successfully, representation is secondary.***
- **Principle 9.** ***If your instincts tell you a model isn't right even though it seems okay on paper, you probably have reason to be concerned.***
  - If you are an experienced software engineer, trust your instincts.
- **Principle 10.** ***Get feedback as soon as you can***

Requirement modeling principles

- **Principle 1.** ***The information domain of a problem must be represented and understood.***
- **Principle 2.** ***The functions that the software performs must be defined.***
- **Principle 3.** ***The behavior of the software (as a consequence of external events) must be represented.***
- **Principle 4.** ***The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.***
- **Principle 5.** ***The analysis task should move from essential information toward implementation detail.***

Design Modeling Principles

- **Principle 1.** ***Design should be traceable to the requirements model.***
- **Principle 2.** ***Always consider the architecture of the system to be built.***
- **Principle 3.** ***Design of data is as important as design of processing functions.***
- **Principle 4.** ***Interfaces (both internal and external) must be designed with care.***
- **Principle 5.** ***UI design should be tuned to the needs of the end user.***
- **Principle 6.** ***Component-level design should be functionally independent*.**

- **Principle 7.** ***Components should be loosely coupled to one another and to the external environment.***
- **Principle 8.** ***Design representations (models) should be easily understandable.***
- **Principle 9.** ***The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.***

Construction Principles

- The construction activity encompasses a set of coding and testing tasks that lead to operational software that is ready for delivery to the customer or end user.
- The initial focus of testing is at the component level, often called unit testing. Other levels of testing include
  - (1) integration testing (conducted as the system is constructed),
  - (2) validation testing that assesses whether requirements have been met for the complete system (or software increment), and
  - (3) acceptance testing that is conducted by the customer in an effort to exercise all required features and functions.

Coding Principles

- The principles that guide the coding task are closely aligned with programming style, programming languages, and programming methods.
- **Preparation principles:** Before you write one line of code, be sure you
  - Understand the problem you're trying to solve.
  - Understand basic design principles and concepts.
  - Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.
  - Select a programming environment that provides tools that will make your work easier.
  - Create a set of unit tests that will be applied once the component you code is completed.

- **Programming principles:** ***As you begin writing code, be sure you***
  - Constrain your algorithms by following structured programming practice.
  - Consider the use of pair programming.
  - Select data structures that will meet the needs of the design.
  - Understand the software architecture and create interfaces that are consistent with it.
  - Keep conditional logic as simple as possible.
  - Create nested loops in a way that makes them easily testable.
  - Select meaningful variable names and follow other local coding standards.
  - Write code that is self-documenting.
  - Create a visual layout (e.g., indentation and blank lines) that aids understanding.

- **Validation Principles**: ***After you've completed your first coding pass, be sure you***
  - Conduct a code walkthrough when appropriate.
  - Perform unit tests and correct errors you've uncovered.
  - Refactor the code.

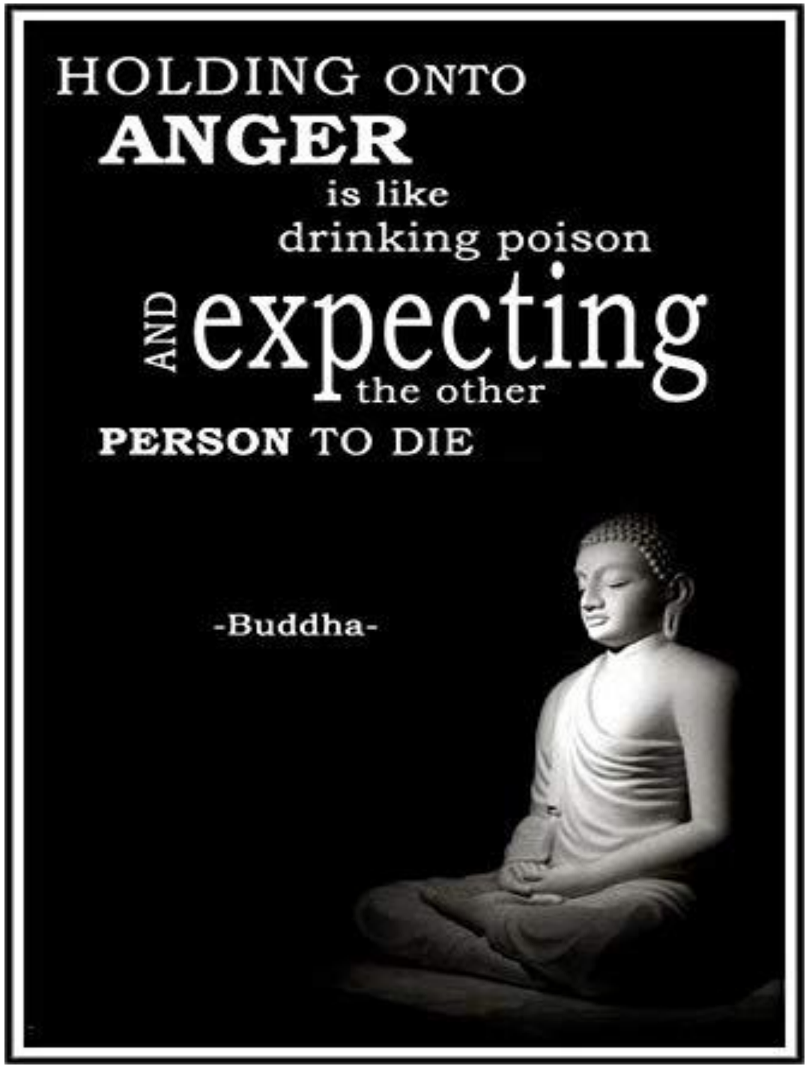
Testing Principles

- Testing is a process of executing a program with the intent of finding an error.
- good test case is one that has a high probability of finding an as-yet undiscovered error.
- A successful test is one that uncovers an as-yet-undiscovered error.

- **Principle 1. *All tests should be traceable to customer requirements***
- **Principle 2. *Tests should be planned long before testing begins***
- **Principle 3. *The Pareto principle applies to software testing.***
  - In this context the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components.
- **Principle 4. *Testing should begin "in the small" and progress toward testing "in the large."***
- **Principle 5. *Exhaustive testing is not possible.***

Deployment Principles

- The deployment activity encompasses three actions: delivery, support, and feedback.
- **Principle 1.** ***Customer expectations for the software must be managed***
- **Principle 2.** ***A complete delivery package should be assembled and tested.***
- **Principle 3.** ***A support management must be established before the software is delivered.***
- **Principle 4.** ***Appropriate instructional materials must be provided to end users.***
- **Principle 5.** ***Buggy software should be fixed first, delivered later***


HOLDING ONTO
ANGER
is like
drinking poison
AND expecting
the other
PERSON TO DIE
-Buddha-

THANK YOU