SOFTWARE ENGINEERING AND PROJECT MANAGEMENT (CSE 227)

FACILITATOR:

DR.S.SENTHILKUMAR

Department of Computer Science and Engineering

School of Engineering,

PRESIDENCY UNIVERSITY

TEXT BOOK AND REFERENCE BOOKS

- **REFERENCE MATERIALS:**
- **Text book(s):**
- Roger S. Pressman, “Software Engineering – A Practitioner’s Approach”, VII Edition, McGraw-Hill, 2017.
- Bob Hughes, Mike Cotterell, Rajib Mall, “Software Project Management”, VI Edition, McGraw-Hill, 2018.
- **Reference book(s):**
- Ian Sommerville, “Software Engineering”, IX Edition, Pearson Education Asia, 2011.
- Rajib Mall, “Fundamentals of Software Engineering”, VI Edition, PHI learning private limited, 2014.

Assessment Schedule

Sl. no	Assessment type	Contents	Course outcome Number	Duration In Hours	Marks	weightage	Venue, DATE &TIME
1	Quiz I [Informal]	Modules 1	CO1	1	10	5%	Announce Later
2	Quiz II [Informal]	Modules 2	CO2	1	10	5%	
3	Test-1 [Formal]	Modules 1 & 2	CO1 CO2	1	40	20%	
4	Assignment	Module 3	CO3	1	10	5%	
5	Test-2 [Formal]	Modules 3 & 4	CO3 CO4	1	40	20%	
6	Case Study	Module 4	CO4	1	10	5%	
7	End Term Exam [Formal]	Modules 1 – 4	CO1-CO4	3	80	40%	

CHAPTER 1 - CONTENTS

1. What is Software?
2. Nature of Software or Software characteristics
3. Software Application Domains
4. Legacy Software
5. Characteristics of WebApps
6. What is Software Engineering?
7. Layered Technology
8. Software Process Framework
9. Umbrella activities in the Software Process
10. Essence of Software Practice
11. Software Myths

1. What is 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.

Software plays a dual role of being a "product" **and also a** "vehicle for delivering a product".

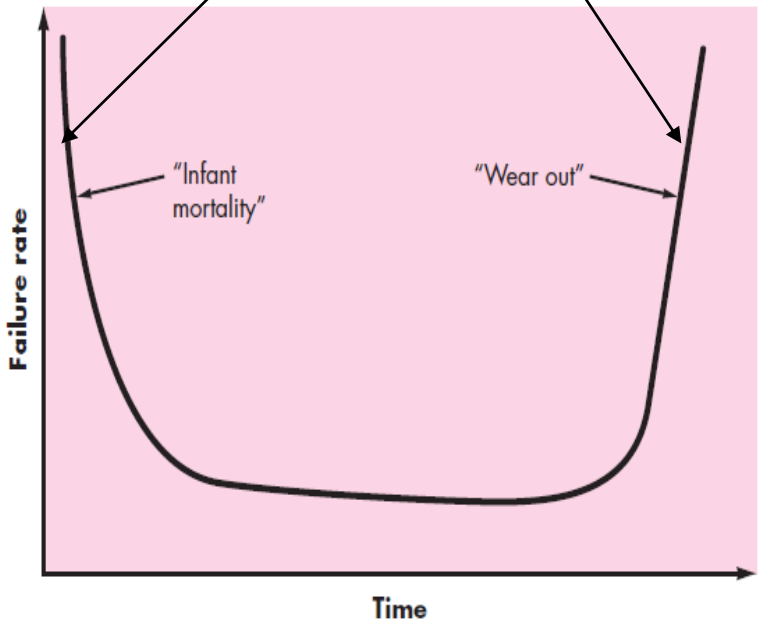
2. Nature of Software or Characteristics

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

HARDWARE VS. SOFTWARE FAILURES

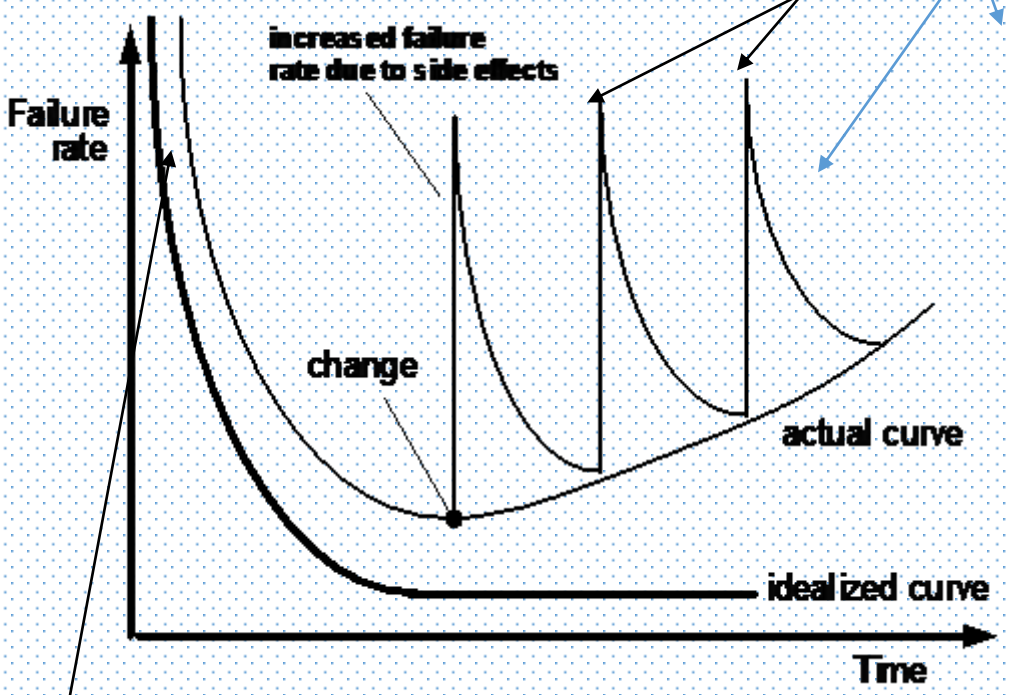
Hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes

Design or manufacturing defects



Failure Curve for Hardware

Changes in the software



Uncovering defects

Failure Curve for Software

3. Software Application Domains

- System Software : Require heavy interaction with computer hardware
  Ex: Operating Systems like Windows, Ubuntu, Android, iOS
- Application Software : Stand alone programs that solve business need
  Ex: Microsoft Office applications such as Word, Excel, PowerPoint
- Engineering/Scientific software : Required for high performance scientific computing
  Ex: Computer Aided Design (CAD) software, MATLAB, Julia
- Embedded Software : Software that controls machines or devices
  Ex: ATMs, Braking system in car, Key pad control in Microwave Oven

Software Application Domains

- Product-line Software : Satisfies particular market segment
  Ex: Walmart POS (Point of Sale) System, Metro POS System, Word processing apps

- Web Applications : Client-Server type of applications
  Ex: Gmail, Google Docs, Facebook

- AI Software : Expert Systems, Robotics, Pattern Recognition
  Ex: Apple's Siri (virtual assistant), IBM Watson, Google Maps

Software - New Categories

- Open world computing - pervasive, distributed computing
- Ubiquitous computing - wireless networks
- Netsourcing - 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

4. Legacy Software

Legacy software are developed decades ago and have been continually modified to meet changes in business requirements and computing platforms.

Why must it change?

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

5.1 Characteristics of WebApps - I

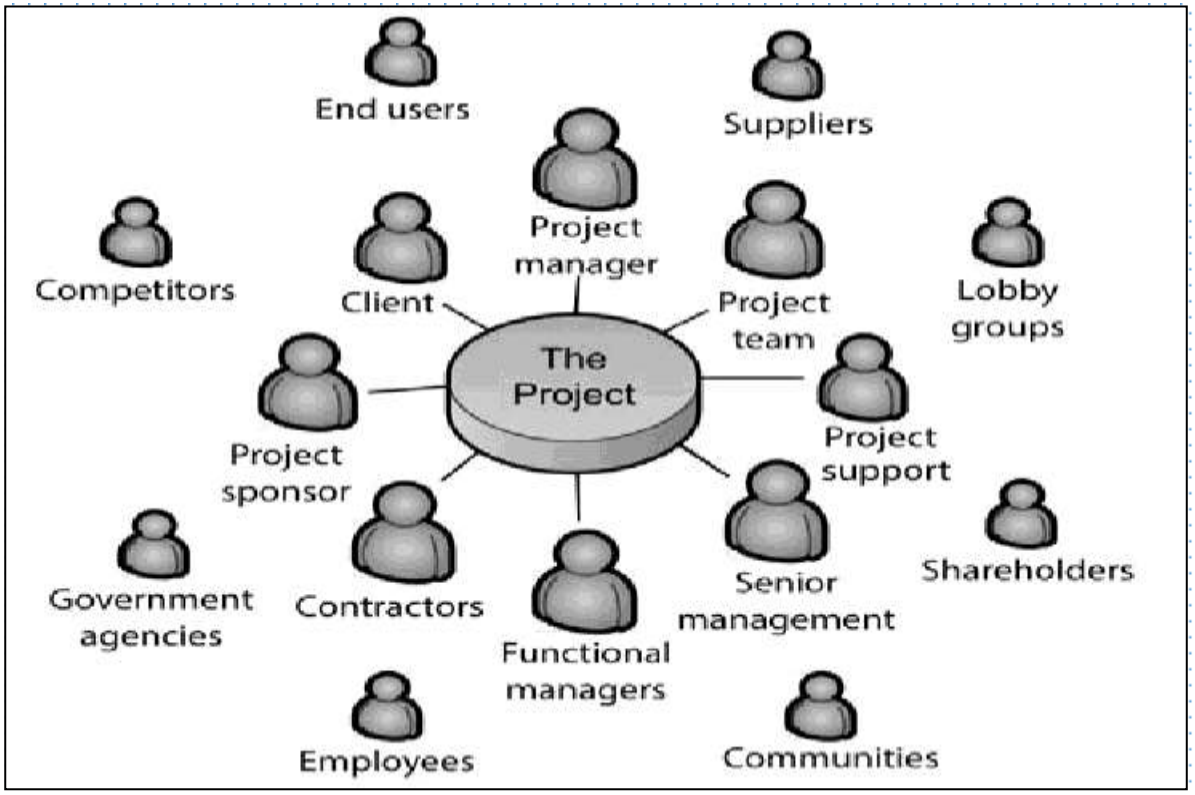
- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a “24/7/365” basis.

5.2 Characteristics of WebApps - II

- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application.
- **Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel.

6. Software Engineering

The word **Engineer** means "**To Build**"



Software Engineering

- 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
- Seminal definition by Fritz Bauer
  - *[Software engineering is] the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines.*

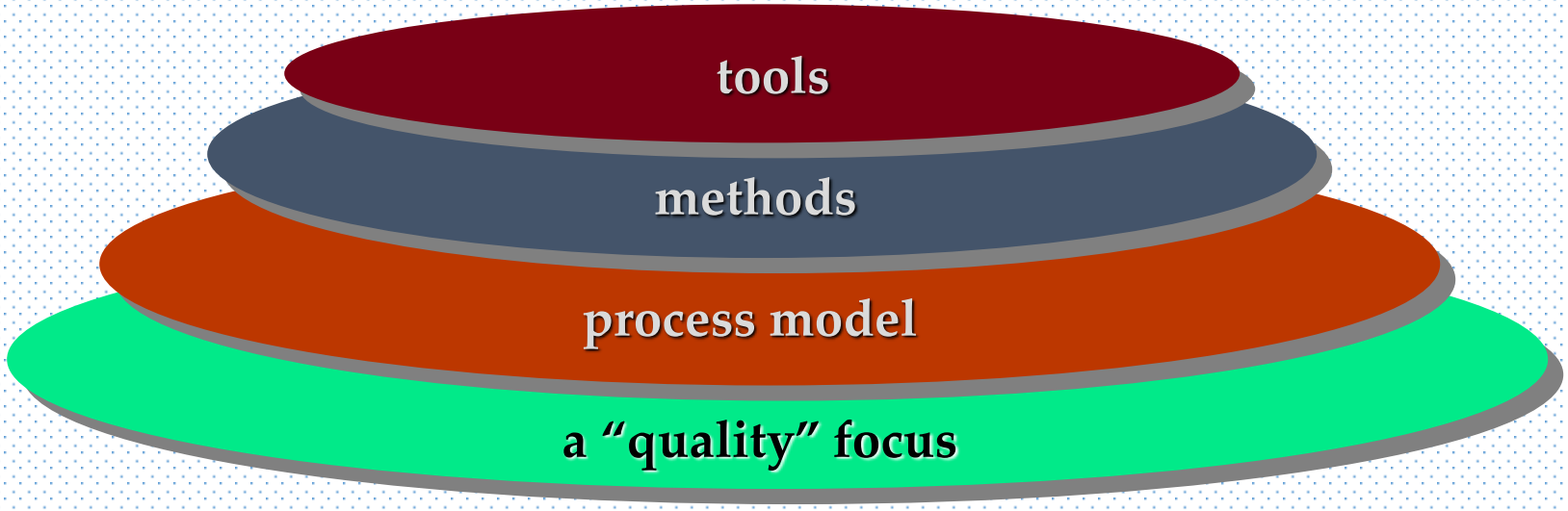
Software Engineering

- 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.*

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.
- 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.

7. A Layered Technology



Software Engineering Layers

Life-Cycle Activity	Models	Methods & Tools	Standards
Software Management	•Life-Cycle Process Model •Work Breakdown Structure •Constructive Cost Model (COCOMO) •Project Plan •Configuration Management (CM) Plan •Risk Management Plan	•Effort, Schedule and Cost Estimation •Risk Analysis •Data Collection •Project Tracking •CM Management •Iterative/Incremental Development •Agile Development	•[IEEE 828] •[IEEE 1058] •[IEEE 1540] •[IEEE 12207]
Software Requirements	•Functional Model •User Class Model •Data Flow Diagram •Object Model •Formal Model •User Stories	•Requirements Elicitation •Prototyping •Structural Analysis •Data-Oriented Analysis •Object-Oriented Analysis •Object Modeling Language (OML) •Formal Methods •Requirements Specification •Requirements Inspection	•[IEEE 830] •[IEEE 1012] •[IEEE 12207]
Software Design	•Architectural Model •Structure Diagram •Object Diagram •Class Specification •Data Model	•Structured Design •Object-Oriented Design •OML •Modular Design •Integrated Development Environment (IDE) •Database Management System (DBMS) •Design Review •Refinement	•[IEEE 1012] •[IEEE 1016] •[IEEE 12207] •[IEEE 42010]
Software Construction	•Detail Design Document •Pseudocode •Flow Chart •Program Code •Unit Test Plan •Integration Test Plan	•Detailed Design •Functional Programming •Object-Oriented Programming •IDE •DBMS •Black Box/White Box Testing •Basic Path Testing •Unit Testing •Code Review •Proof of Correctness •Software Reuse •Integration •Integration Testing	•[IEEE 1008] •[IEEE 1012] •[IEEE 1016] •[IEEE 12207]

8. A Process Framework

Process framework

Framework activities

- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella Activities

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

9. Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Why Software Projects Fail?

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication: clients, developers, & users
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

10. The Essence of Software Practice

- Polya suggests:
  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).**

10.1 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?

10.2 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?

10.3 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?

10.4 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?

Hooker's General Principles

- Principle refers to "an important underlying law or assumption required in a system of thought"
- 1: The Reason It All Exists
- 2: KISS (Keep It Simple, Stupid!)
- 3: Maintain the Vision
- 4: What You Produce, Others Will Consume
- 5: Be Open to the Future
- 6: Plan Ahead for Reuse
- 7: Think!

11. Software Myths

Myths refer to “erroneous beliefs about software and the process that is used to build it”

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

2. ***If we get behind schedule, we can add more programmers and catch up (sometimes called the “Mongolian horde” concept).***

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

Software Myths

CUSTOMER MYTHS

- *Software requirements continually change, but change can be easily accommodated because software is flexible.*

PRACTITIONER'S MYTHS

1. *Once we write the program and get it to work, our job is done.*
2. *The only deliverable work product for a successful project is the working program.*
3. *Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.*

CHAPTER 2
Process Models

Department of Computer Science and Engineering

School of Engineering, Presidency University

CHAPTER 2 - CONTENTS

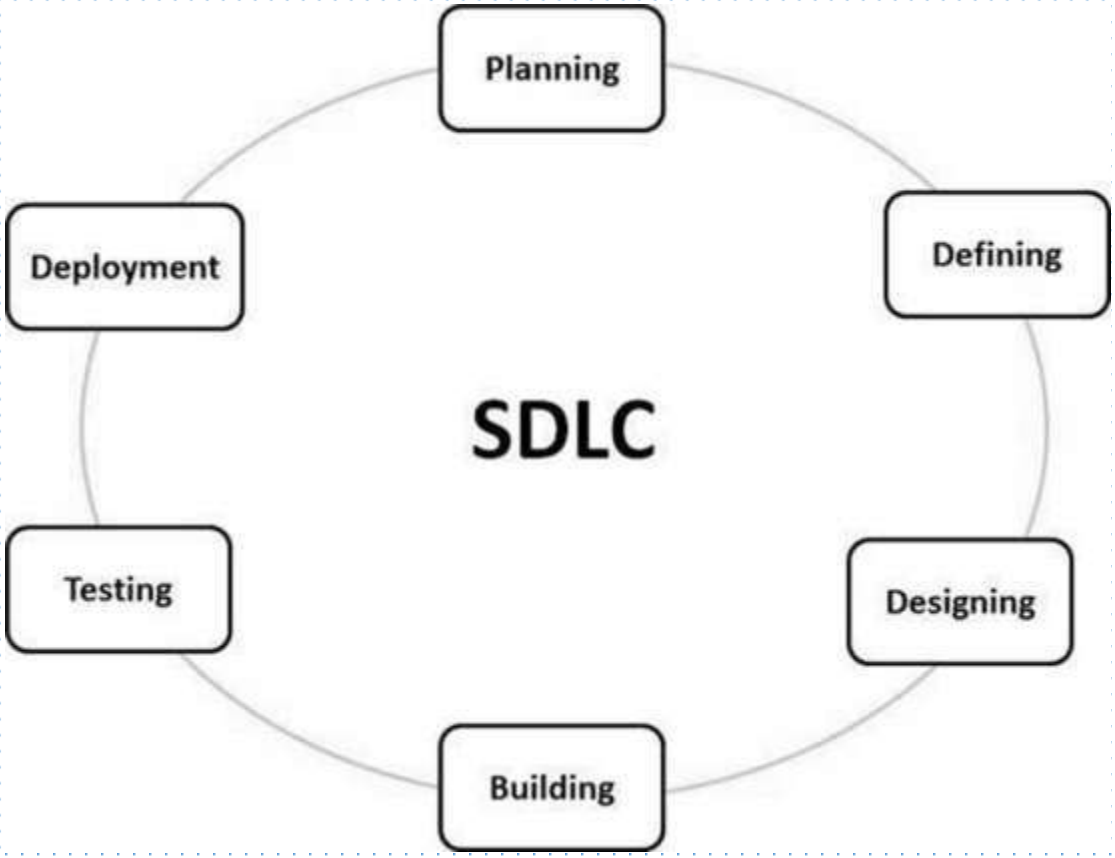
I. What is Software Process

- Software Process (SP) is a **framework** for the activities, actions, and tasks that are required to build high-quality software.
- SP defines the approach that is taken as software is **engineered**.
- Is not equal to software engineering, which also encompasses **technologies** that populate the process - technical methods and automated tools.

II. Software Development Life Cycle(SDLC)

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software's.
- The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- It is also called as Software Development Process.

Software Development Life Cycle(SDLC)



LIFE CYCLE STAGES–

- Stage 1: Planning and Requirement Analysis
- Stage 2: Defining Requirements
- Stage 3: Designing the Product Architecture
- Stage 4: Building or Developing the Product
- Stage 5: Testing the Product
- Stage 6: Deployment in the Market and Maintenance

SDLC models followed in the industry

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

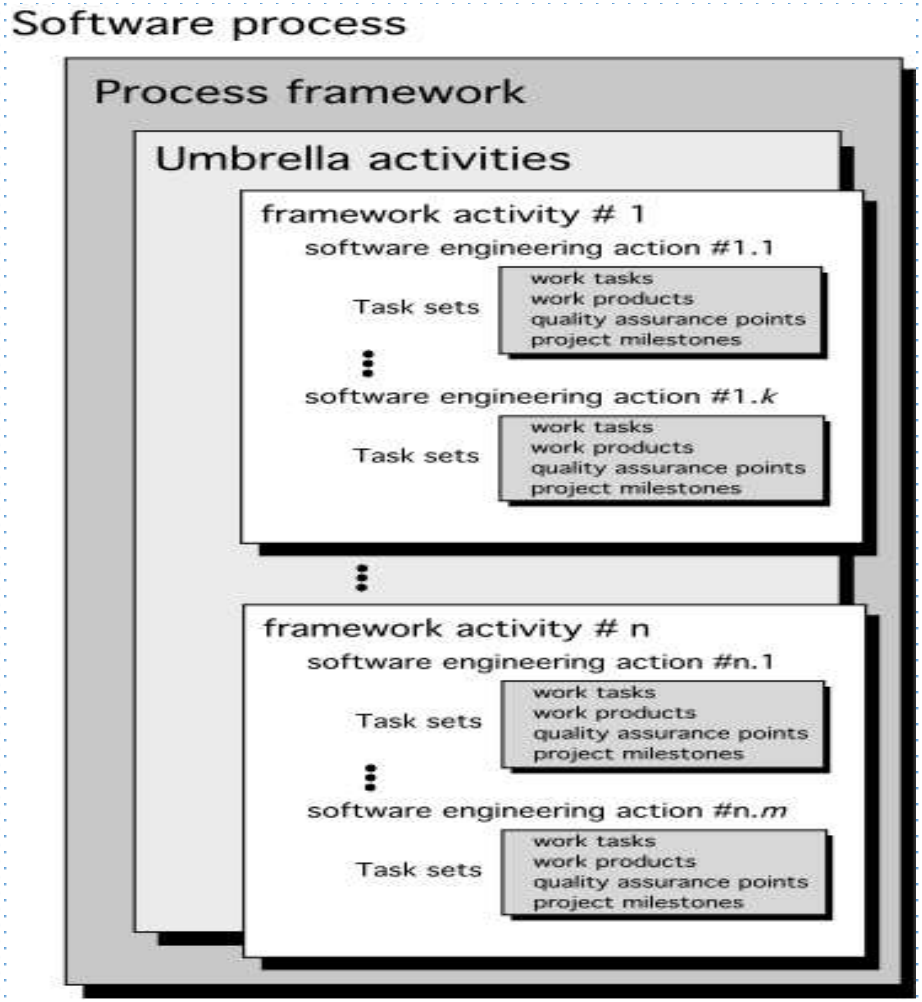
III. What is Software Process Model

- **What**: Go through a series of predictable steps - a **road map** that helps you create a timely, high-quality results.
- **Who**: Software engineers and their managers, clients also. People adapt the process to their needs and follow it.
- **Why**: Provides stability, control, and organization to an activity that can if left uncontrolled, become quite chaotic. However, modern software engineering approaches must be agile and **demand ONLY** those activities, controls and work products that are appropriate.

What is Software Process Model

- **What Work products**: Programs, documents, and data
- **What are the steps:** The process you adopt depends on the software that you are building. One process might be good for aircraft avionic system, while an entirely different process would be used for website creation.
- **How to ensure right:** A number of software process assessment mechanisms that enable us to determine the maturity of the software process. However, the quality, timeliness and long-term viability of the software are the best indicators of the efficacy of the process you use.

IV. Generic Process Model



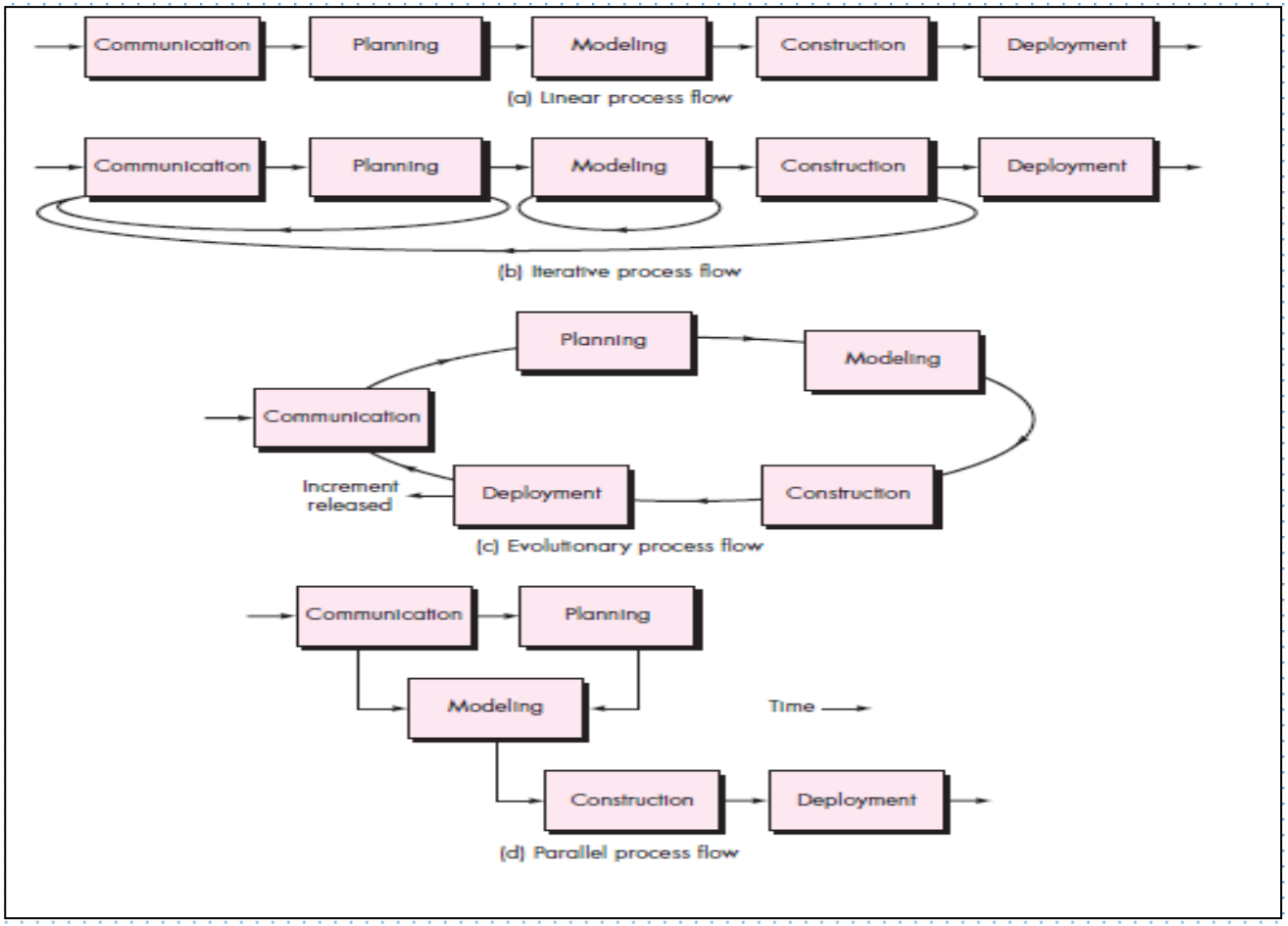
Generic Process Model

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

- In addition, a set of umbrella activities - project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

- Next question is: how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time? See the **process flow** for answer.

Process Flow in a Generic Process Model



Process Flow in a Generic Process Model

- ***Linear process flow*** executes each of the five activities in sequence.
- An ***iterative process flow*** repeats one or more of the activities before proceeding to the next.
- An ***evolutionary process flow*** executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A ***parallel process flow*** executes one or more activities in parallel with other activities (Ex: modeling for one aspect of the software in parallel with construction of another aspect of the software).

V. Identifying a Task Set

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?

- 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

Identifying a Task Set – Example 1

For a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone call with the stakeholder. Therefore, the only necessary action is phone conversation, the task set for communication action includes:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

Identifying a Task Set – Example 2

The task sets for requirements elicitation action for a **simple** project may include:

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.

Identifying a Task Set – Example 3

The task sets for requirements elicitation action for a BIG project may include:

1. Make a list of stakeholders for the project.
2. Interview each stakeholders separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder 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.

VI. Prescriptive Models

Prescriptive models are used as **guidelines** to organize and structure how software developmental activities should be performed, and in what order.

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

6.1. Waterfall Model

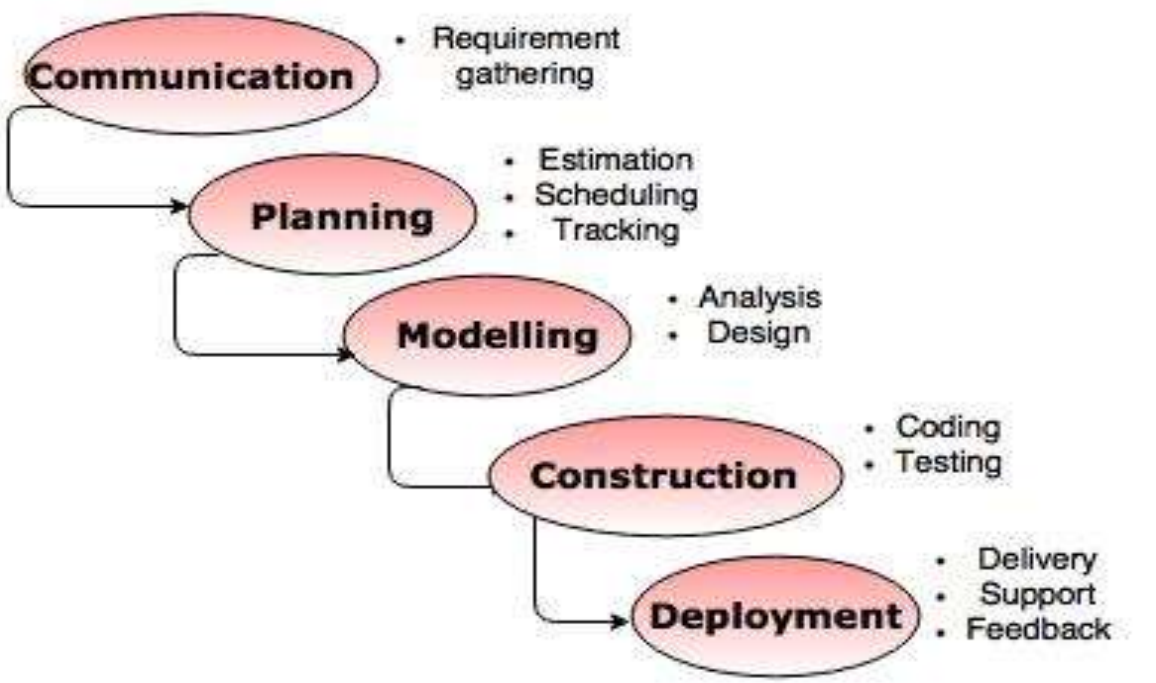
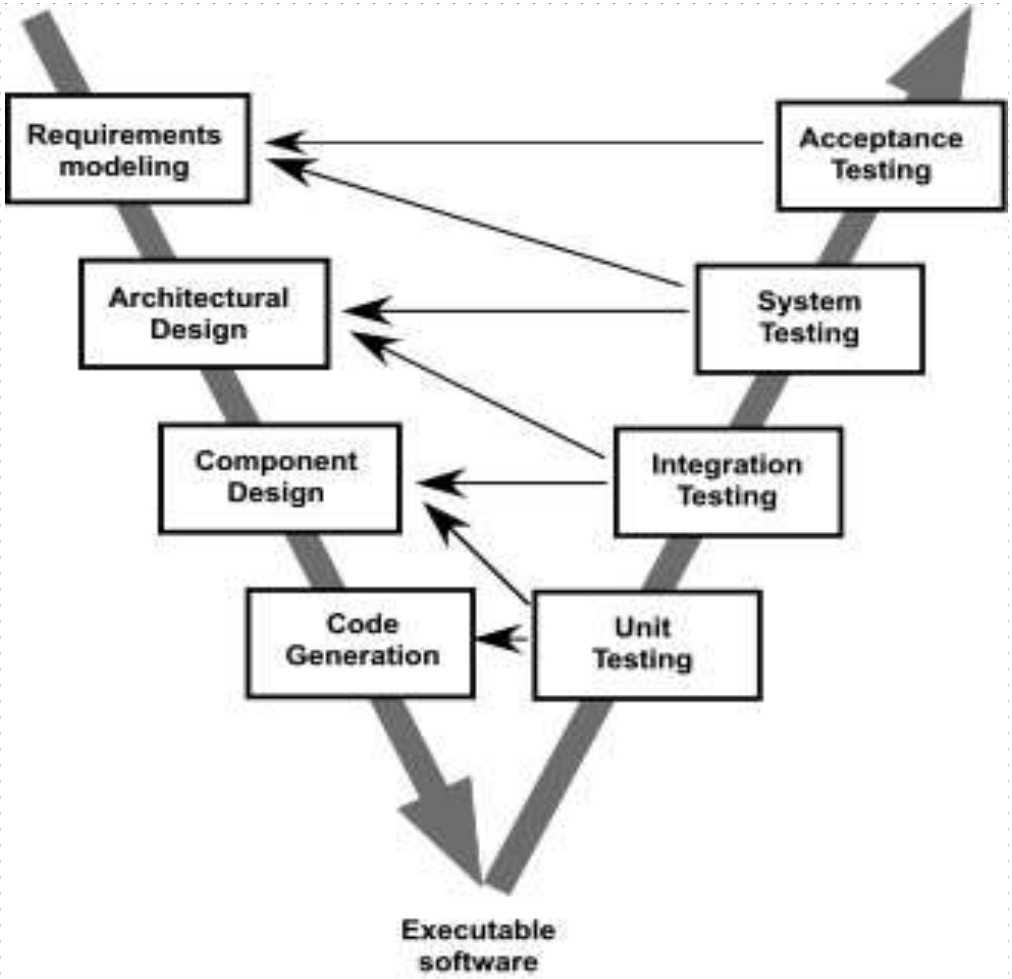


Fig. - The Waterfall model

6.1. Waterfall Model

- The waterfall model is also called as **'Linear sequential model'.**
- In this model, *each phase is fully completed* before the beginning of the next phase.
- This model is used for small projects.
- Assumes that requirements are defined and reasonably stable.
- Testing part starts only after the development is complete.

6.2. The V-Model

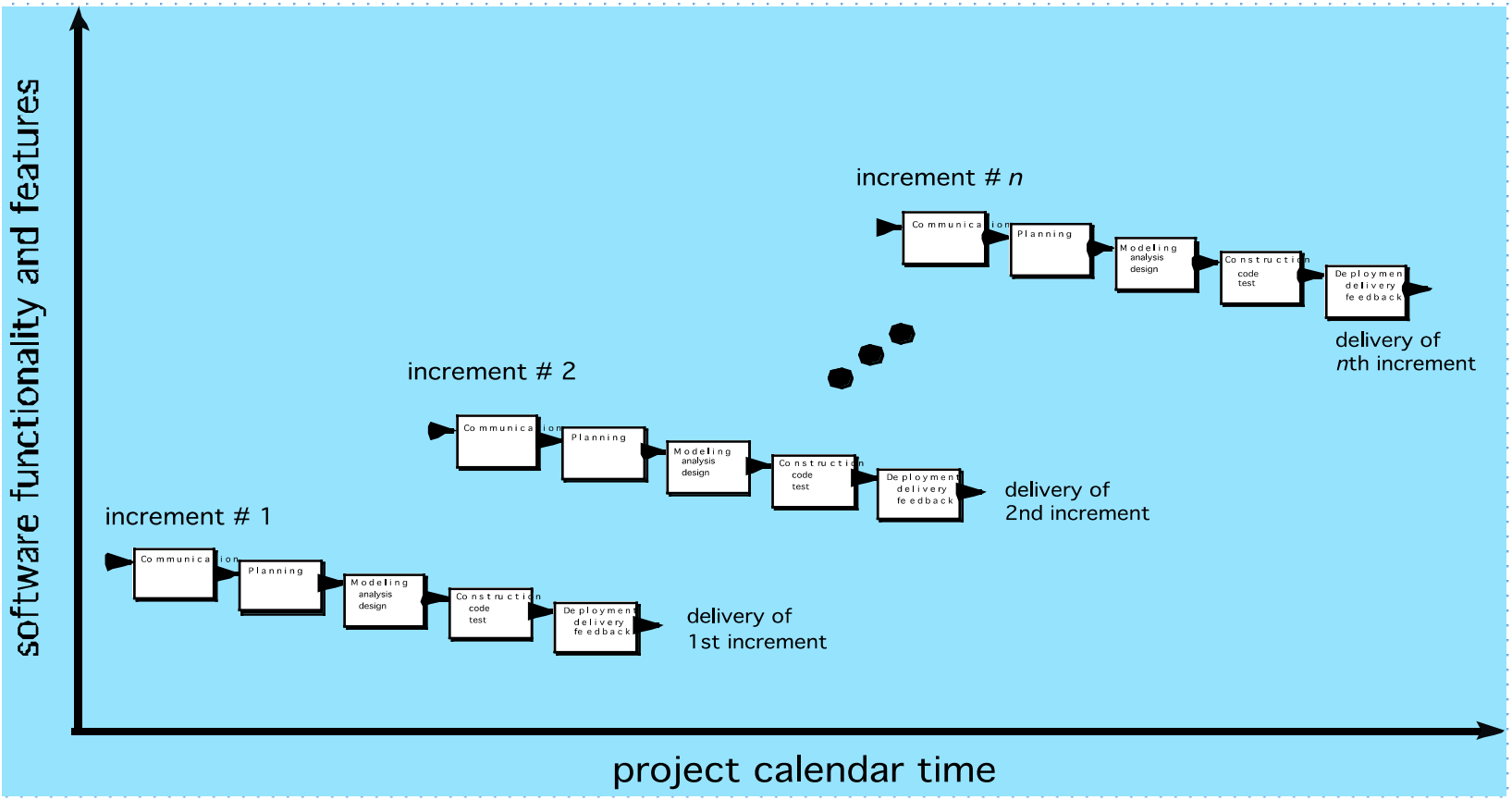


6.2. The V-Model

- A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

- Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

6.3. The Incremental Model



6.3. The Incremental Model

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.

6.4. Evolutionary Models

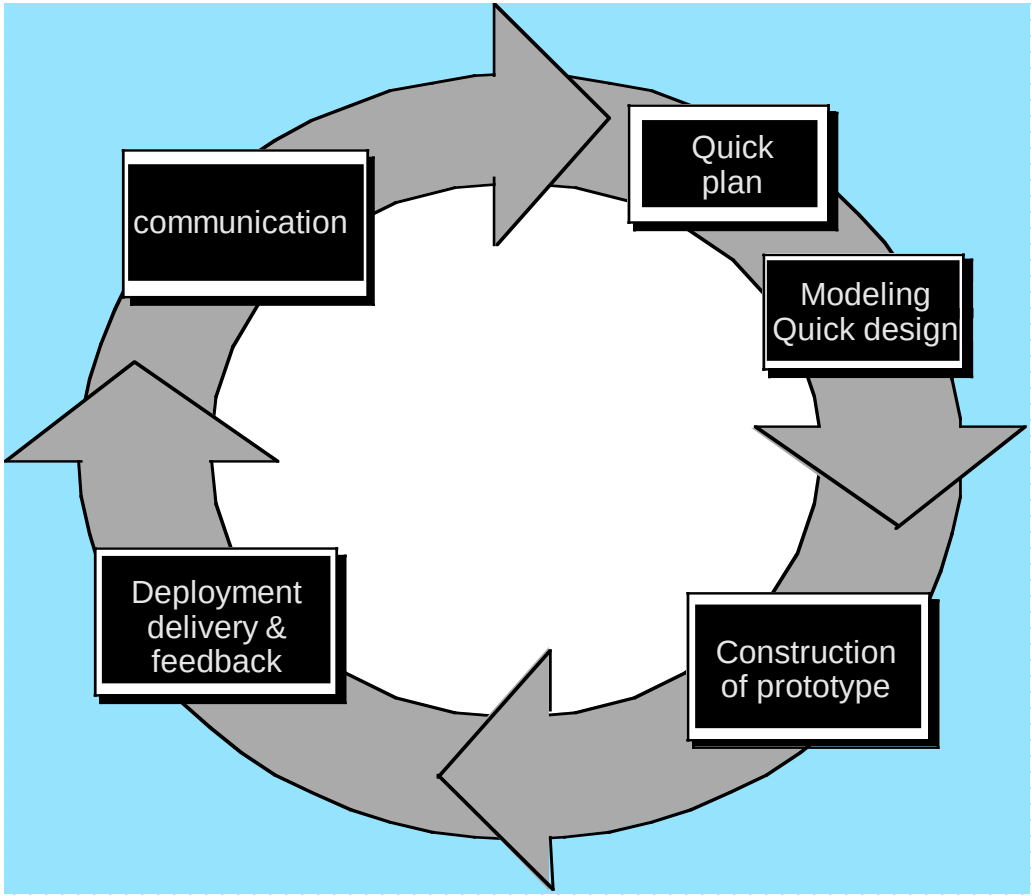
- Software system evolves over time as requirements often change as development proceeds.
- Usually a set of core product or system requirements is well understood, but the details and extension have yet to be defined.
- You need a process model that has been explicitly designed to accommodate a product that evolved over time.
- It is iterative that enables you to develop increasingly more complete version of the software.
- Types: Prototyping and Spiral models.

6.4a. Evolutionary Models - Prototyping

- **When to use**: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. Or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.

- **Advantages**: Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately. However, engineers may make compromises in order to get a prototype working quickly. The less-than-ideal choice may be adopted forever after you get used to it.

6.4a. Evolutionary Models - Prototyping

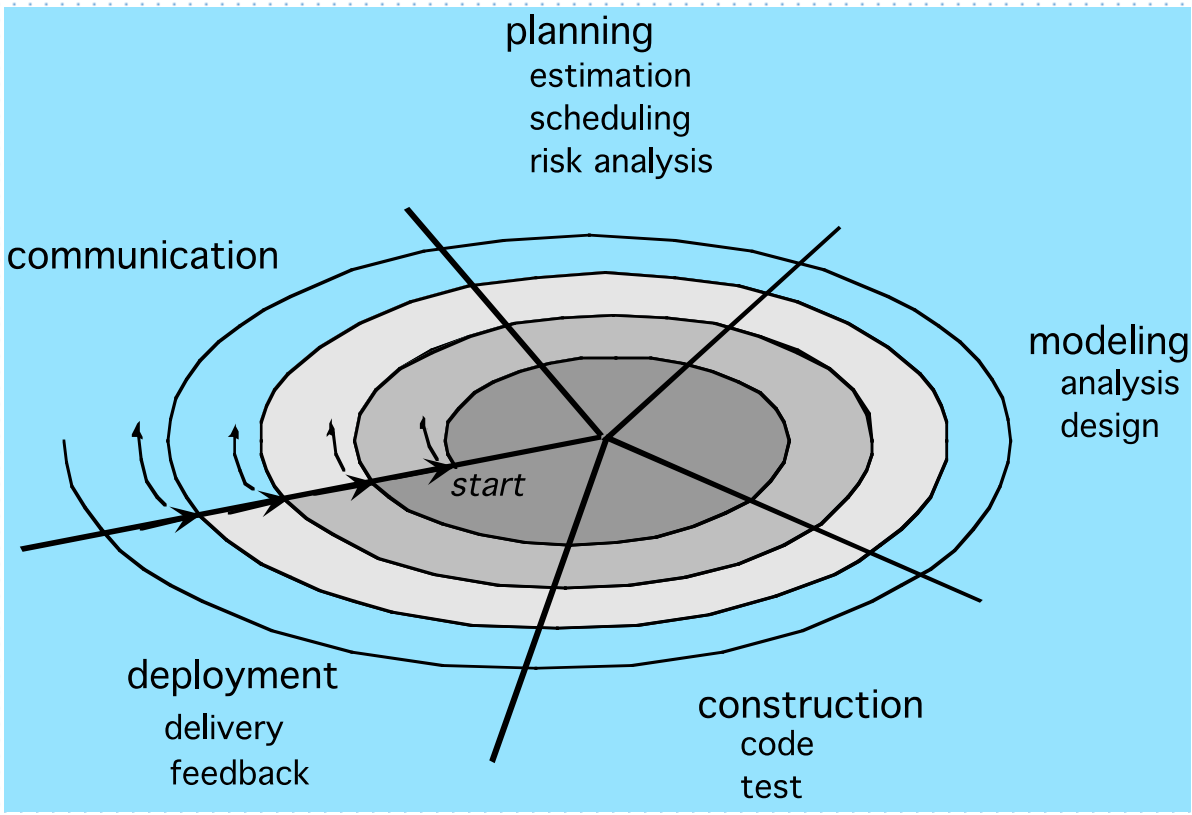


6.4b. Evolutionary Models - Spiral

- It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model and is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.

- Two main distinguishing features: one is cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

6.4b. Evolutionary Models - Spiral



Three Concerns of Evolutionary Models

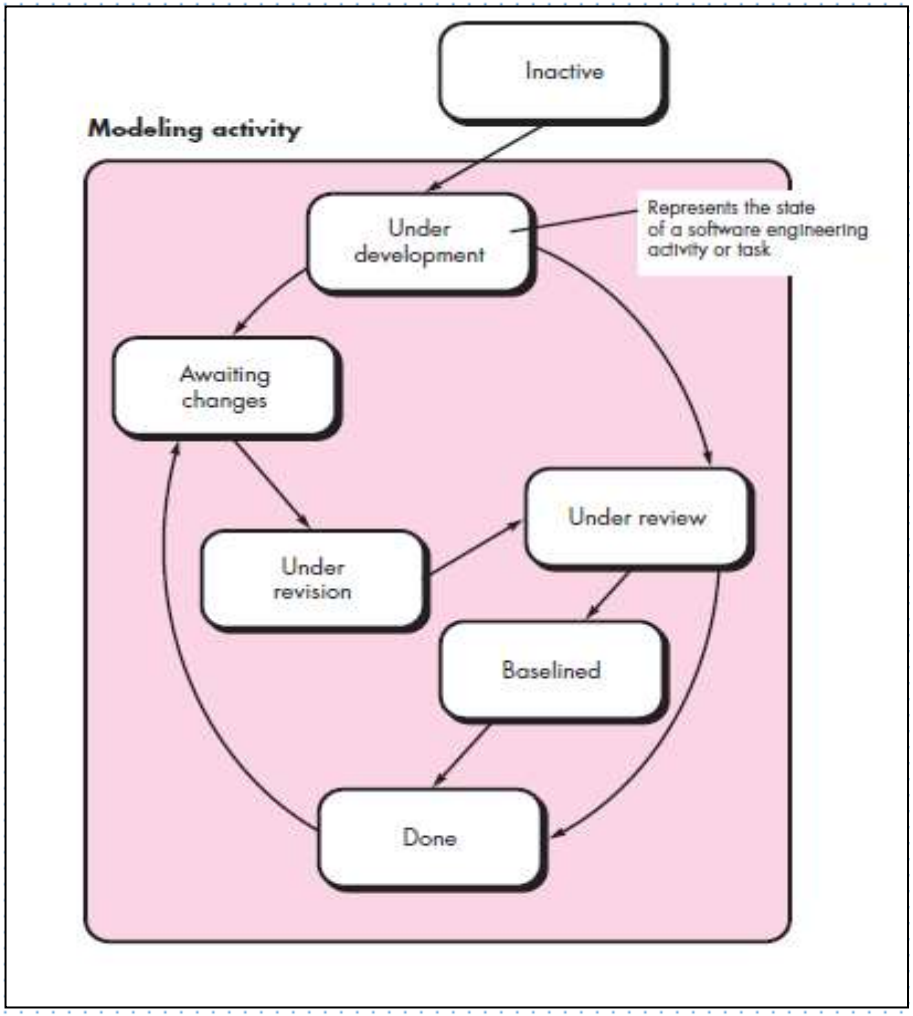
- First concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product.
- Second, it does not establish the maximum speed of the evolution. If the evolution occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then productivity could be affected.
- Third, software processes should be focused on flexibility and extensibility rather than on high quality. We should prioritize the speed of the development over zero defects. Extending the development in order to reach high quality could result in a late delivery of the product when the opportunity niche has disappeared.

6.5. Concurrent Model

- Allow a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.

- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather than confining software engineering activities, actions and tasks to a sequence of events, it defines a process network. Each activity, action or task on the network exists simultaneously with other activities, actions or tasks. Events generated at one point trigger transitions among the states.

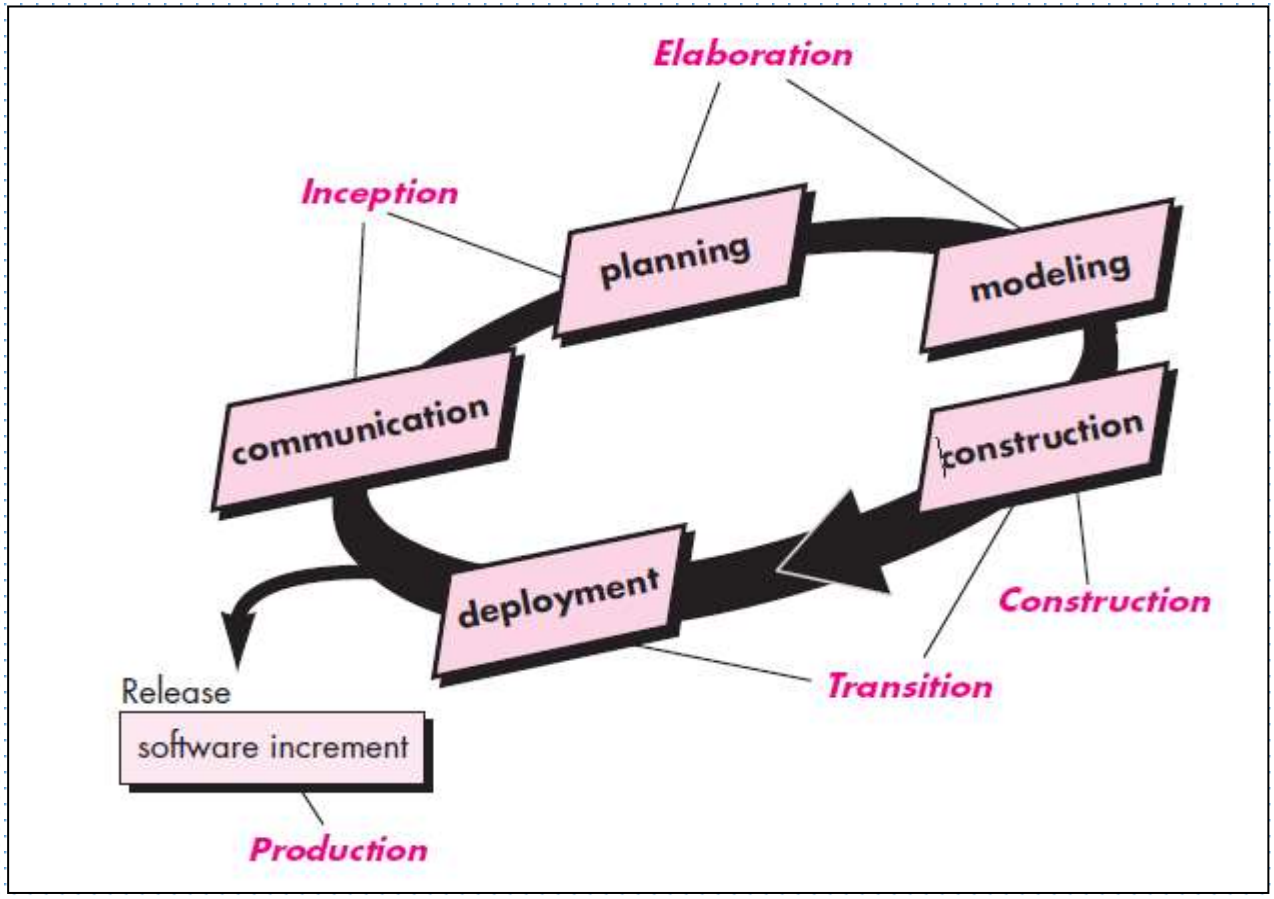
6.5. Concurrent Model



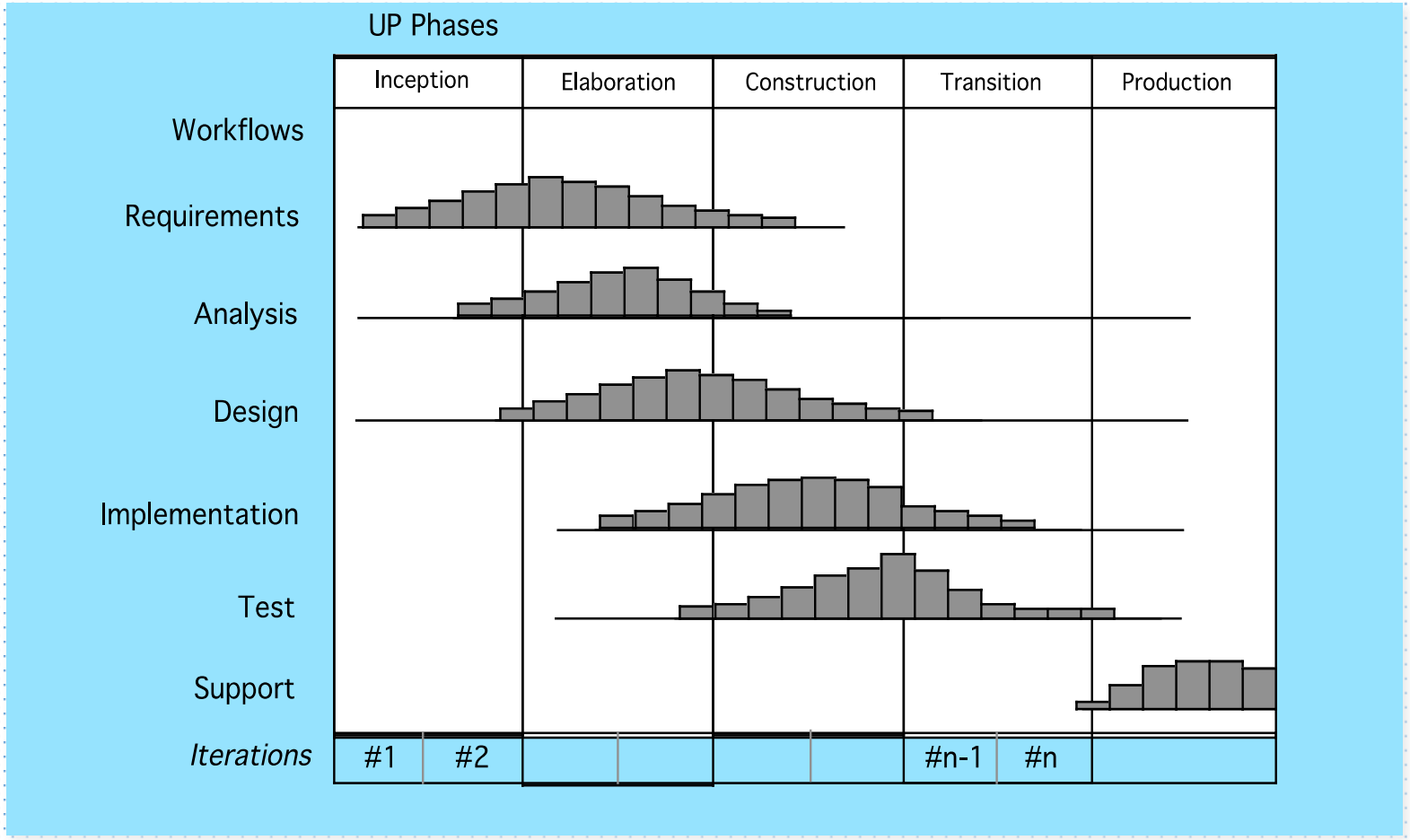
Other Process Models…

- **Component based development** - the process to apply when reuse is a development objective (like spiral model)
- **Formal methods** - emphasizes the mathematical specification of requirements (easy to discover and eliminate ambiguity, incompleteness and inconsistency)
- **Aspect Oriented software development (AOSD)** - provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- **Unified Process** - a "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML) to model and develop object-oriented system iteratively and incrementally.

7. The Unified Process (UP)



UP Phases


UP Phases
Inception
Elaboration
Construction
Transition
Production
Workflows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

UP Work Products

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
  phases and iterations.
Business model,
  if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
  including non-functional
Analysis model
Software architecture
  Description.
Executable architectural
  prototype.
Preliminary design model
Revised risk list
Project plan including
  iteration plan
  adapted workflows
  milestones
  technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
  increment
Test plan and procedure
Test cases
Support documentation
  user manuals
  installation manuals
  description of current
    increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

8. Personal Software Process (PSP)

The **Personal Software Process** (PSP) is a structured **software development process** that is intended (planned) to help **software** engineers better understand and improve their performance by tracking their predicted and actual **development** of code.

PSP helps software engineers to:

- Improve their estimating and planning skills.
- Make commitments they can keep.
- Manage the quality of their projects.
- Reduce the number of defects in their work.
- Develop metrics for all project related activities, i.e., code, test etc.

9. Team Software Process (TSP)

- Is Intended to **improve the levels of quality** and **productivity** of a team's software development project
- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.

Criteria for selecting the right process model

Factors	Waterfall	V-Shaped	Evolutionary Prototyping	Spiral	Iterative and Incremental	Agile
Unclear User Requirement	Poor	Poor	Good	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Excellent	Good	Poor
Reliable system	Good	Good	Poor	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Good	Poor	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Poor	Excellent	Excellent
Visibility of Stakeholders	Good	Good	Excellent	Excellent	Good	Excellent
Skills limitation	Good	Good	Poor	Poor	Good	Poor
Documentation	Excellent	Excellent	Good	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Poor	Excellent	Poor

CASE STUDIES…

Case 1: Company X has purchased a CRM (Customer Relationship Management) solution to handle its FMCG (Fast Moving Consumer Goods) operations. The CRM solution is deployed within Company X, but needs to be customized according to its needs. The customized requirements are defined thoroughly by the company management. What process model is right for this requirement?

Best fit model – Waterfall model.

Rationale: Since the **requirements** for the **customization** of the **CRM** solutions is **well defined**, the **complexity** and **risk** of project execution is **low**, hence waterfall model is a good fit solution.

CASE STUDIES…

Case 2: A banking customer needs a website to be developed. The basic set of features are well defined, however, the advanced features are thought to be evolving in nature. The basic feature set includes development of retail portal for customer login, checking of account status and account balance. Advanced features include: account transfer, change of customer pin etc. What process model is right for this requirement?

Best fit model: Incremental model

Rationale: All the basic features of the website can be developed and deployed as Iteration 1, so that customers can start using the website. Subsequent evolving requirements can be handled in other iteration phases, as and when the features are freezed.

CASE STUDIES…

Case 3: A Company Y which is a manufacturer of medical equipments' wants to model a three-dimensional visualization of anatomical part, which improves the quality of preoperative planning and assists in the selection of optimal surgical approach for prosthetic implants. What process model is right for this requirement?

Best fit model: Evolutionary Model - Prototyping

Rationale: A **3D prototype** of **any anatomical part** can be developed based on the requirements and examined. Once the 3D prototype is approved, the system can be developed to incorporate other anatomical parts.

(Chapter 3)
Agile Software Development

Department of Computer Science and Engineering

School of Engineering, Presidency University

Chapter 3 - Contents

1. **Manifesto for Agile Software development**
2. **What is "Agility"?**
3. **Agility and the Cost of Change**
4. **Agile Process and Principles**
5. **Agile Models**
    1. ***Extreme Programming (XP)***
    2. ***Adaptive Software Development (ASD)***
    3. ***Scrum***
    4. ***Feature Driven Development***
6. **Agile Modelling**
7. **Criteria for selecting the right Process Model**

1. Manifesto for Agile S/W Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** **over processes and tools**
- ***Working software*** **over comprehensive documentation**
- ***Customer collaboration*** **over contract negotiation**
- ***Responding to change*** **over following a plan**

That is, while there is value in the items on the right, we value the items on the left more."

- Kent Beck et al.

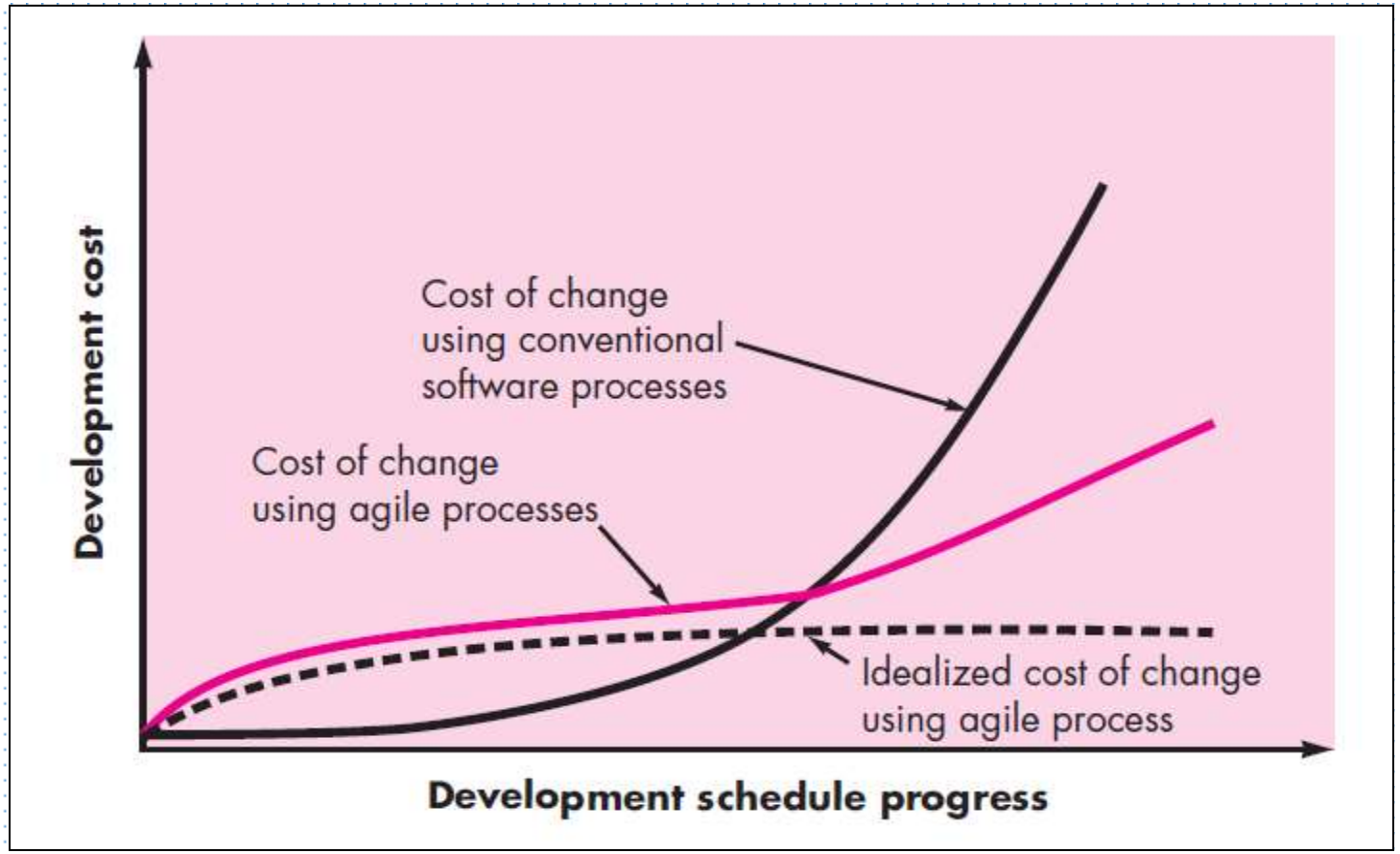
2. What is “Agility”

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software

3. Agility and the Cost of Change



4. An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agile Principles - I

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.
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.

Agile Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
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.

Human Factors

- *The process molds to the needs of the people and team*, not the other way around
- Key traits must exist among the people on an agile team and the team itself:
  - Competence.
  - Common focus.
  - Collaboration.
  - Decision-making ability.
  - Fuzzy problem-solving ability.
  - Mutual trust and respect.
  - Self-organization.

5. AGILE MODELS

5.1 Extreme Programming

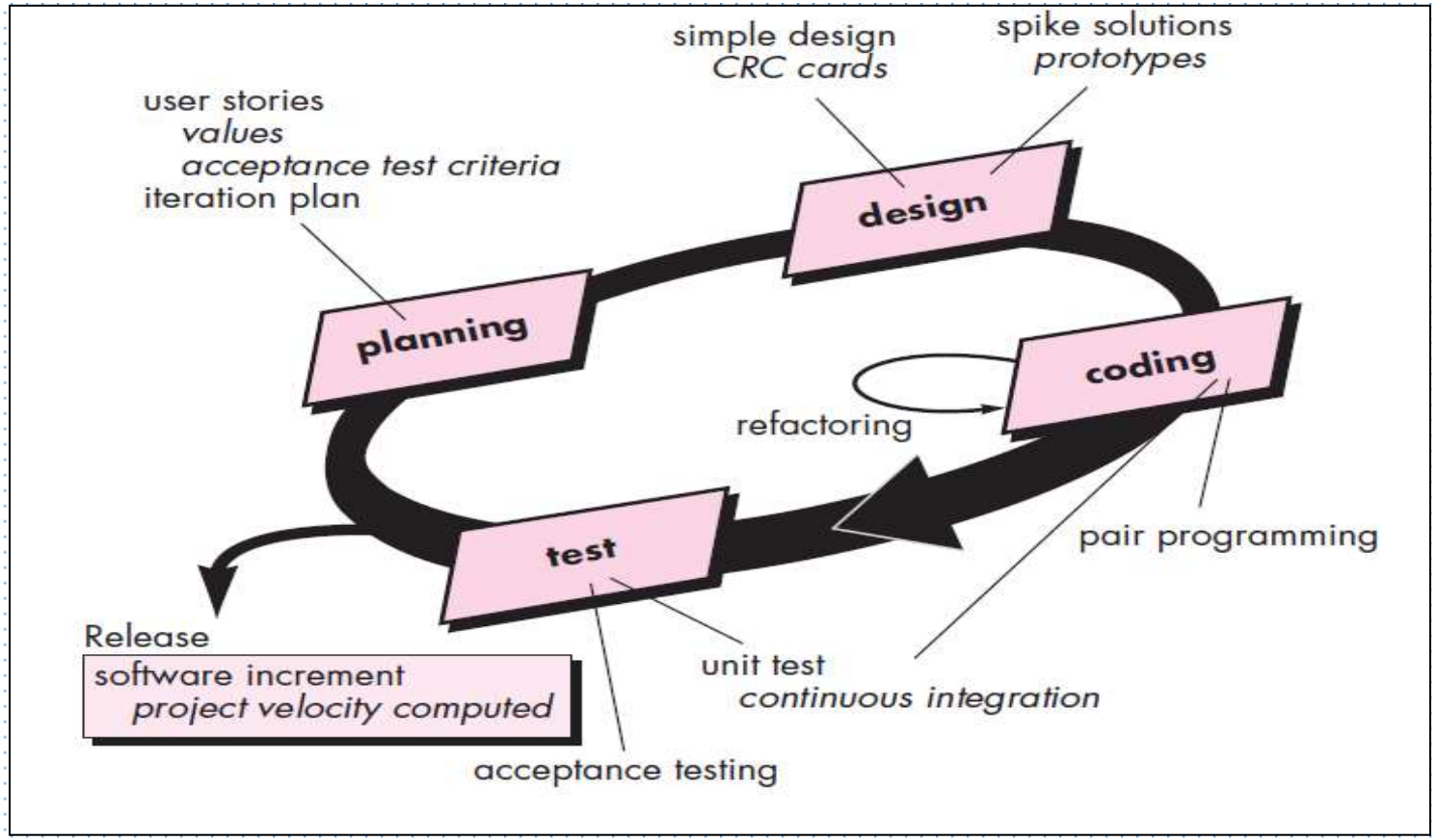
- The most widely used agile process, originally proposed by Kent Beck

- **XP Planning**
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to form a deliverable increment
  - A commitment is made on delivery date
  - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

Extreme Programming

- **XP Design**
  - Follows the KISS principle
  - Encourage the use of CRC cards
  - For difficult design problems, suggests the creation of "spike solutions" - a design prototype
  - Encourages "refactoring" - an iterative refinement of the internal program design
- **XP Coding**
  - Recommends the construction of a unit test for a user story *before* coding commences
  - Encourages "pair programming"
- **XP Testing**
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

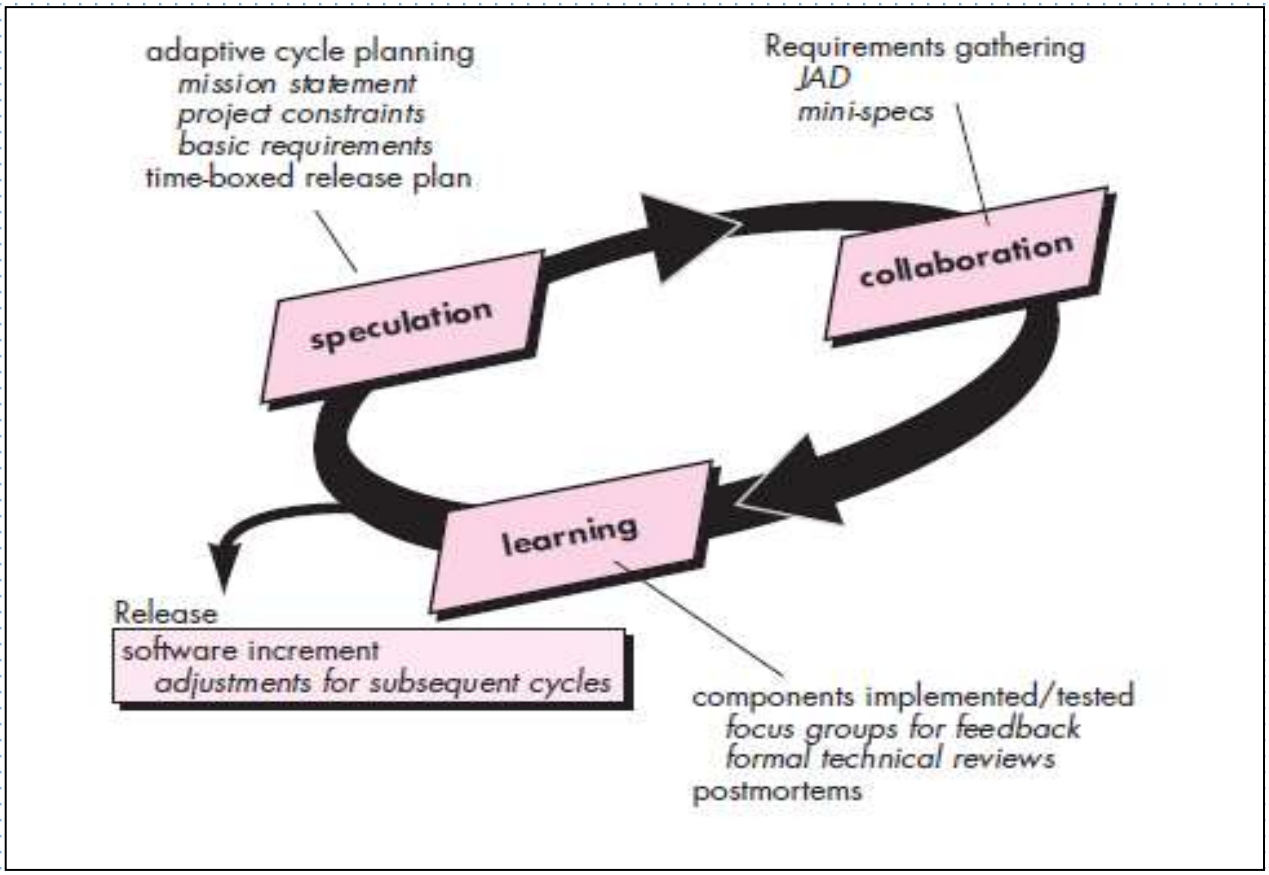
Extreme Programming Process



5.2. Adaptive Software Development (ASD)

- Originally proposed by Jim Highsmith for building complex software and systems.
- Distinguishing features of ASD:
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing"
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes "learning" throughout the process

Adaptive Software Development (ASD)



5.3. SCRUM

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Items can be added to backlogs anytime, introducing changes to the system.
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated

SCRUM

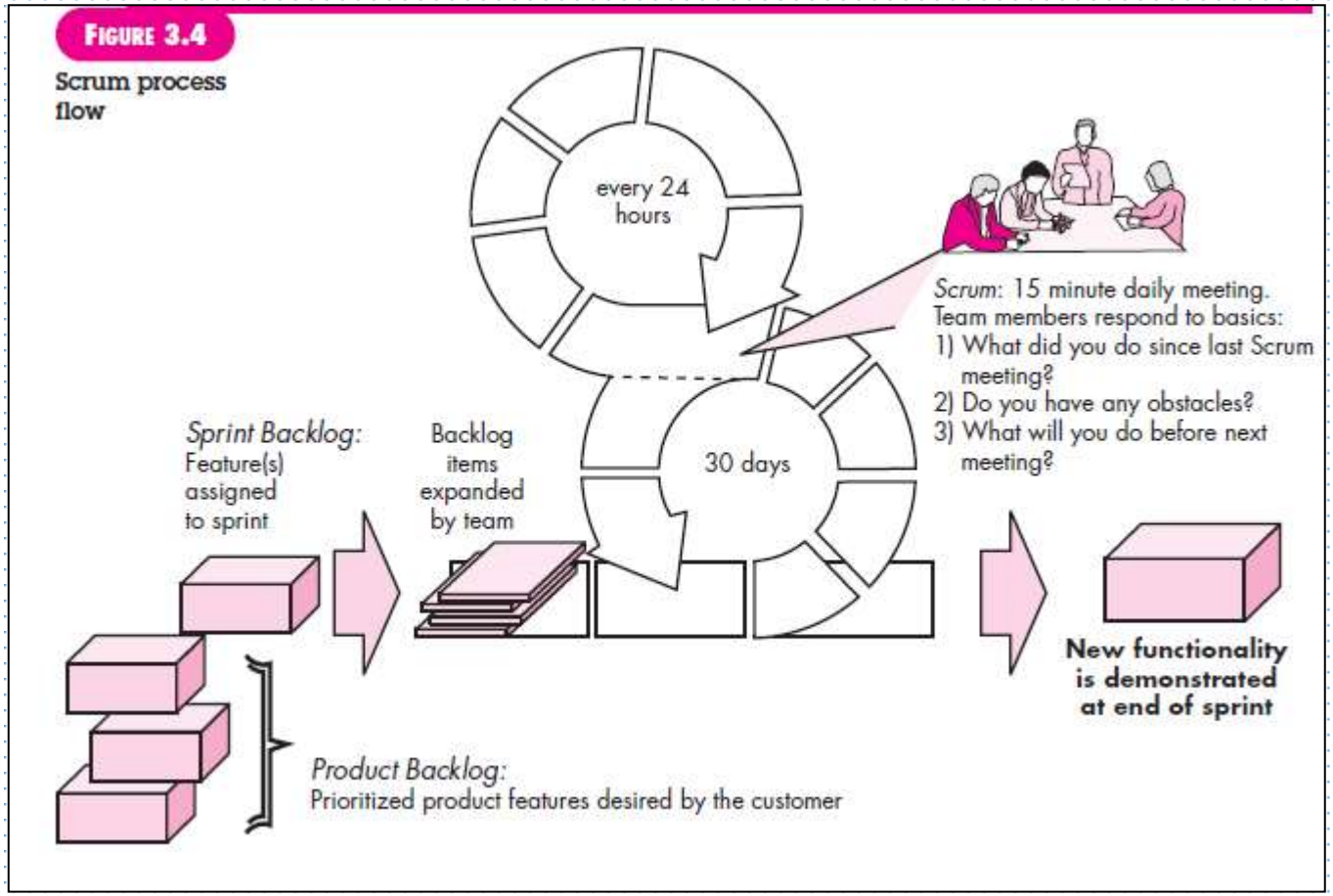


FIGURE 3.4
Scrum process flow

5.4. Feature Driven Development

- Originally proposed by Peter Coad et al
- FDD - distinguishing features
  - Emphasis is on defining "features"
    - a *feature* "is a client-valued function that can be implemented in two weeks or less."
    - Feature is a small block of deliverable functionality
  - Uses a feature template
    - <action> the <result> <by | for | of | to> a(n) <object>

5.4. Feature Driven Development

- Uses a feature template
  - <action> the <result> <by | for | of | to> a(n) <object>

  Ex 1: Add the Product to shopping cart

  Ex 2: Display the technical specifications of the product

  Ex 3: Store the shipping information for the customer

- A features list is created and "plan by feature" is conducted

- Design and construction merge in FDD

Feature Driven Development

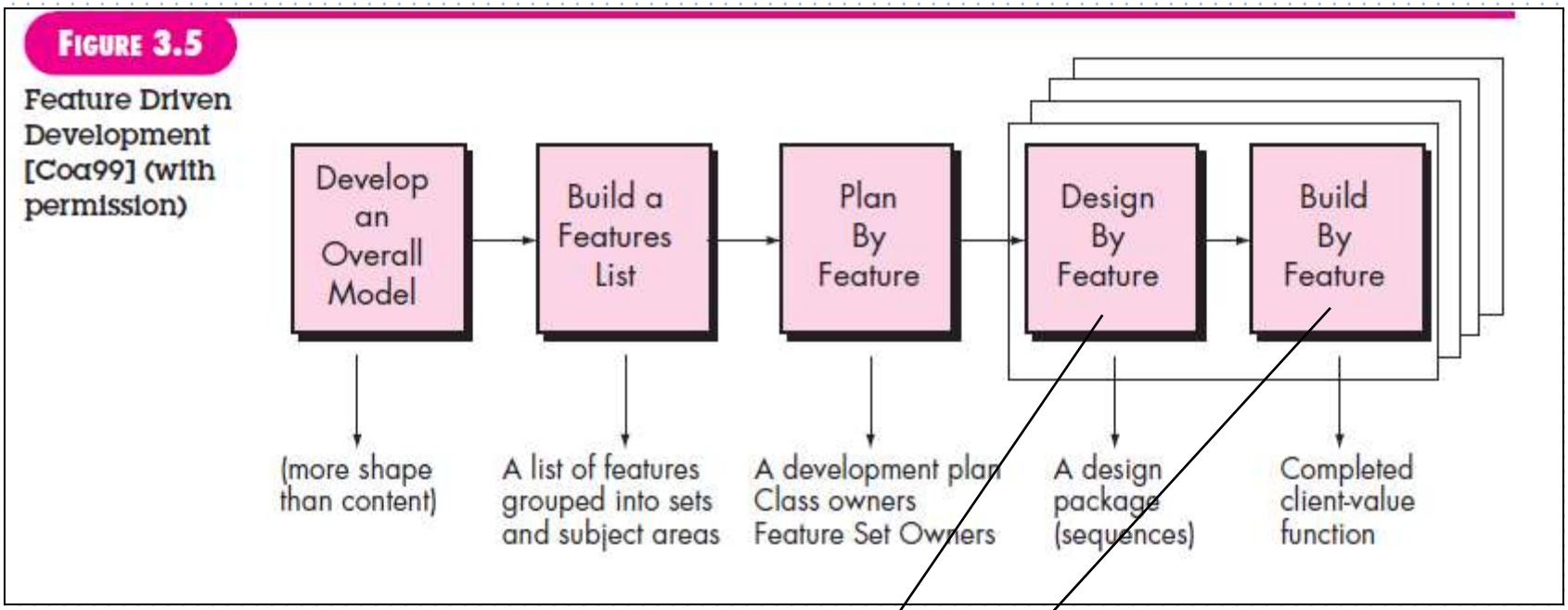


FIGURE 3.5
Feature Driven Development [Coa99] (with permission)

Design walkthrough, Design, Design Inspection, Code
Code inspection

6. Agile Modelling

- Originally proposed by Scott Ambler
- Is a practice-based methodology for effective modeling and documentation of software-based systems
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light
  - Content is more important than representation
  - Know the models and the tools you use to create them
  - Adapt locally

7. Criteria for selecting the right Process Model

Factors	Waterfall	V-Shaped	Evolutionary Prototyping	Spiral	Iterative and Incremental	Agile
Unclear User Requirement	Poor	Poor	Good	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Excellent	Good	Poor
Reliable system	Good	Good	Poor	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Good	Poor	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Poor	Excellent	Excellent
Visibility of Stakeholders	Good	Good	Excellent	Excellent	Good	Excellent
Skills limitation	Good	Good	Poor	Poor	Good	Poor
Documentation	Excellent	Excellent	Good	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Poor	Excellent	Poor