Introduction to Software Engineering

CO1: Student will be able to **understand** and **use** basic knowledge in software engineering.

CO2: Student will be able to **identify** requirements, **analyze** and **prepare** models.

Contents

- Nature of Software, Software Engineering, Software Process,
- Generic Process Model,
- Prescriptive Process Models: The Waterfall Model, V-model, Incremental Process Models,
- Evolutionary Process Models, Concurrent Models, Capability Maturity Model (CMM)
- Agile process, Agility Principles,
- Extreme Programming (XP),
- Scrum, Kanban model
- **Self-learning Topics: Personal and Team Process Models**

What is Software?

The product that software professionals build and then support over the long term.

Software encompasses:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately store and manipulate information and

(3) documentation that describes the operation and use of the programs.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs **more to maintain** than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

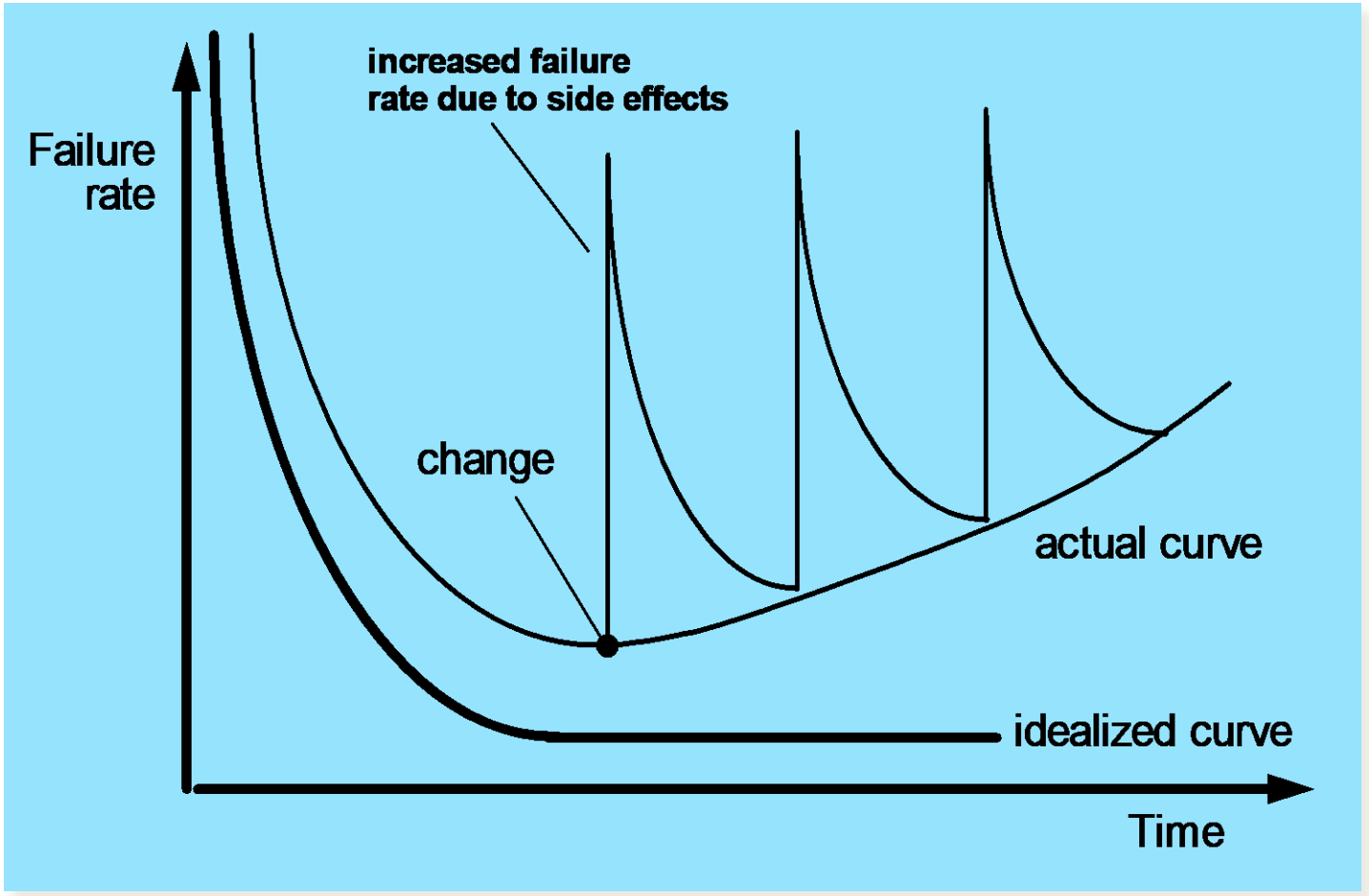
Features of Software?

- Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

Wear vs. Deterioration



Software Applications

1. System software: such as compilers, editors, file management utilities

2. Application software: stand-alone programs for specific needs.

3. Engineering/scientific software: Characterized by "number crunching"algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc

4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)

5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)

6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Software—New Categories

- Open world computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to **communicate across vast network**.
- Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- 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

Software Engineering Definition

The seminal definition:

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

The IEEE definition:

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

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

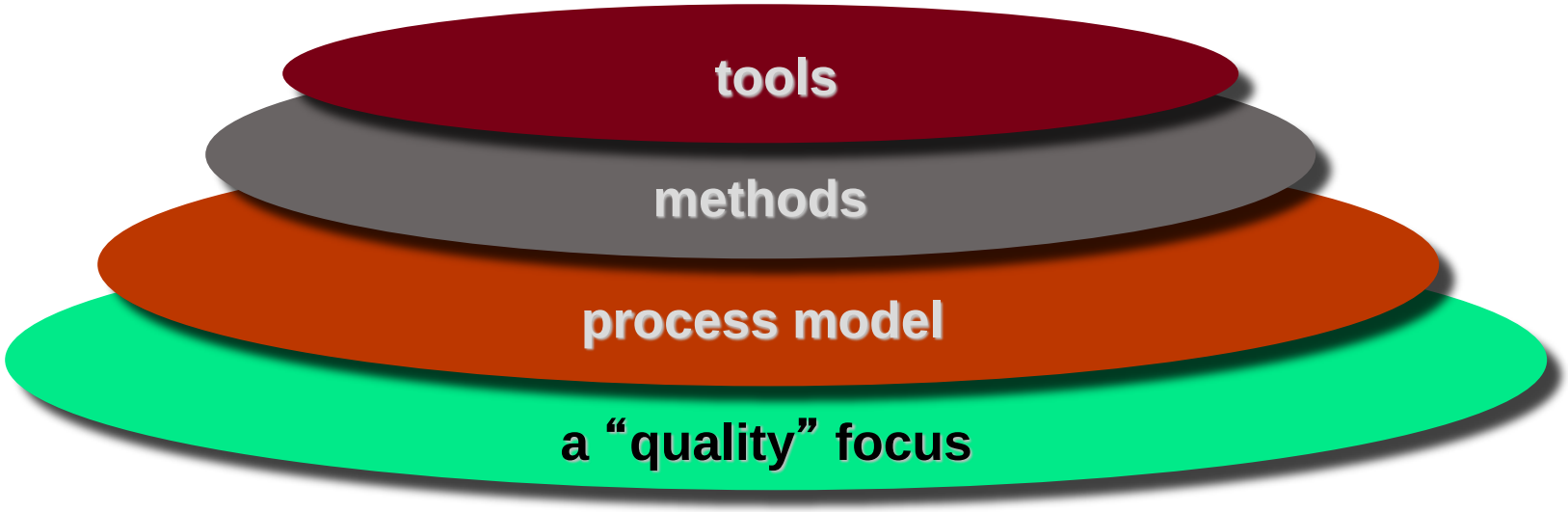
FAQ about Software Engineering

Question	Answer
What is software?	Computer programs, data structures and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Software Engineering *A Layered Technology*



- Any engineering approach must rest on organizational commitment to **quality** which fosters a continuous process improvement culture.
- **Process** layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- **Method** provides technical how-to’s for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.
- **Tools** provide automated or semi-automated support for the process and methods.

Software Process

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is **not a rigid prescription** for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the **appropriate** **set of work actions** and tasks.

- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

Five Activities of a Generic Process framework

1. Communication: communicate with customer to understand objectives and gather requirements
2. Planning: creates a "map" defines the work by describing the tasks, risks and resources, work products and work schedule.
3. Modeling: Create a "sketch", what it looks like architecturally, how the constituent parts fit together and other characteristics.
4. Construction: code generation and the testing.
5. Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.

- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Umbrella Activities

Complement the five process framework activities and help team manage and control progress, quality, change, and risk.

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- Software configuration management: manage the effects of change throughout the software process.
- Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

Adapting a Process Model

The process should be **agile and adaptable** to problems. Process adopted for one project might be significantly different than a process adopted from another project. (to the problem, the project, the team, organizational culture). Among the differences are:

- the overall flow of activities, actions, and tasks and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity
- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the manner in which project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described
- the degree to which the customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

Prescriptive and Agile Process Models

- The **prescriptive process** models stress detailed definition, identification, and application of process activities and tasks. Intent is to improve system quality, make projects more manageable, make delivery dates and costs more predictable, and guide teams of software engineers as they perform the work required to build a system.

- **Agile process models** emphasize project “agility” and follow a set of principles that lead to a more informal approach to software process. It emphasizes maneuverability and adaptability. It is particularly useful when Web applications are engineered.

The Essence of Practice

- How does the practice of software engineering fit in the process activities mentioned above?
- Namely, communication, planning, modeling, construction and deployment.
- George Polya outlines the essence of problem solving, 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).

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?

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 sub-problems be defined?* If so, are solutions readily apparent for the sub-problems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

Carry Out the Plan

- *Does the solutions 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?

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?

How It all Starts

- *SafeHome:*
  - Every software project is precipitated by some business need—
    - the need to correct a defect in an existing application;
    - the need to the need to adapt a ‘legacy system’ to a changing business environment;
    - the need to extend the functions and features of an existing application, or
    - the need to create a new product, service, or system.

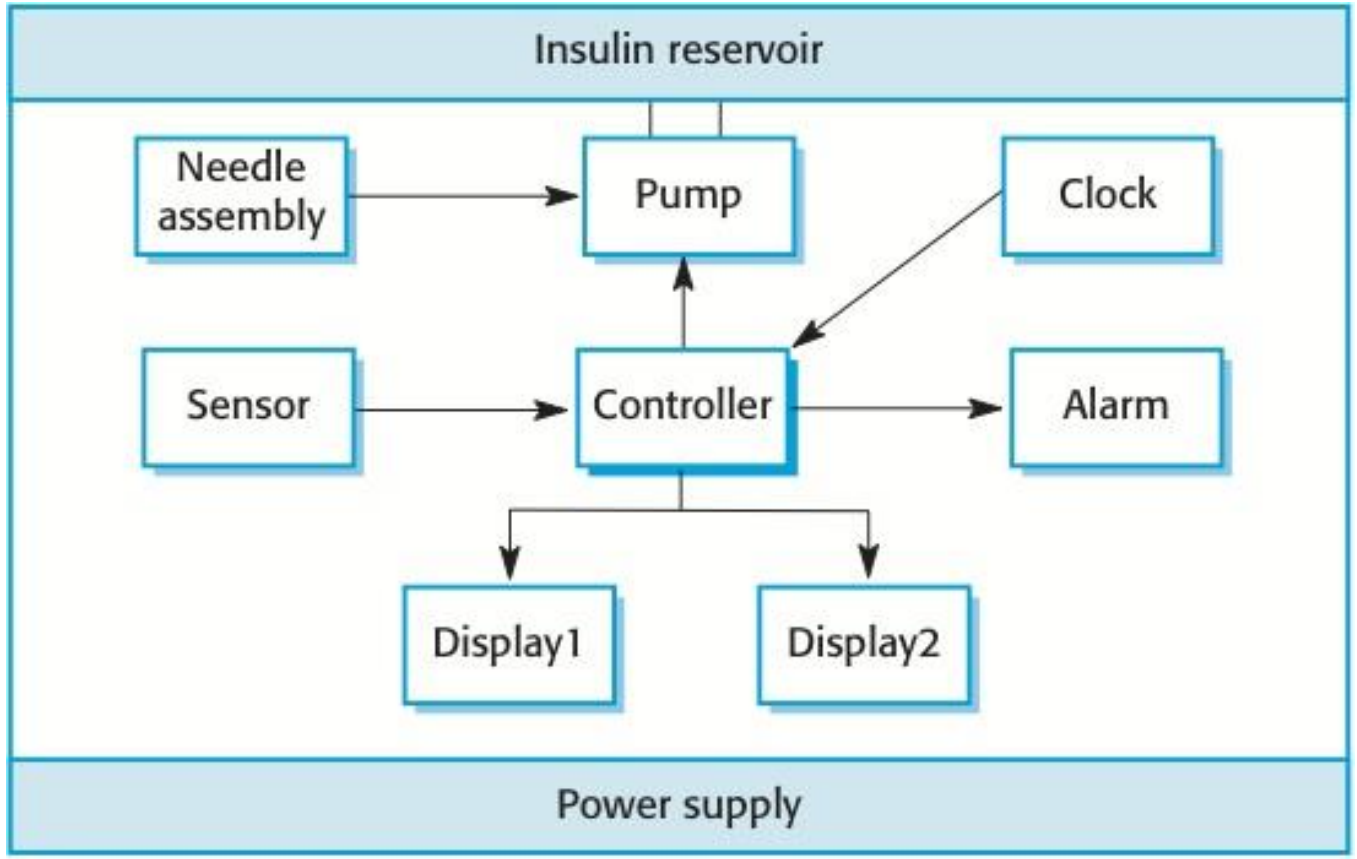
Case study

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.

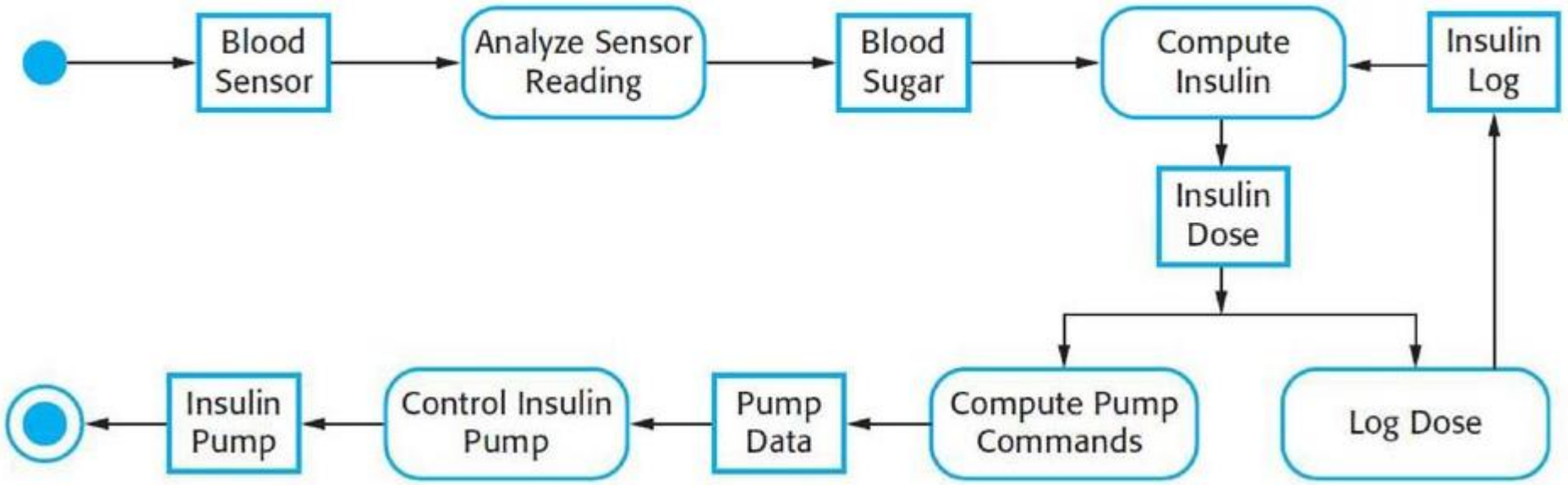
Insulin pump control system

1. Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
2. Calculation based on the rate of change of blood sugar levels.
3. Sends signals to a micro-pump to deliver the correct dose of insulin.
4. Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin pump hardware architecture



Activity model of the insulin pump



Essential high-level requirements

1. The system shall be available to deliver insulin when required.
2. The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
3. The system must therefore be designed and implemented to ensure that the system always meets these requirements.

Process Models

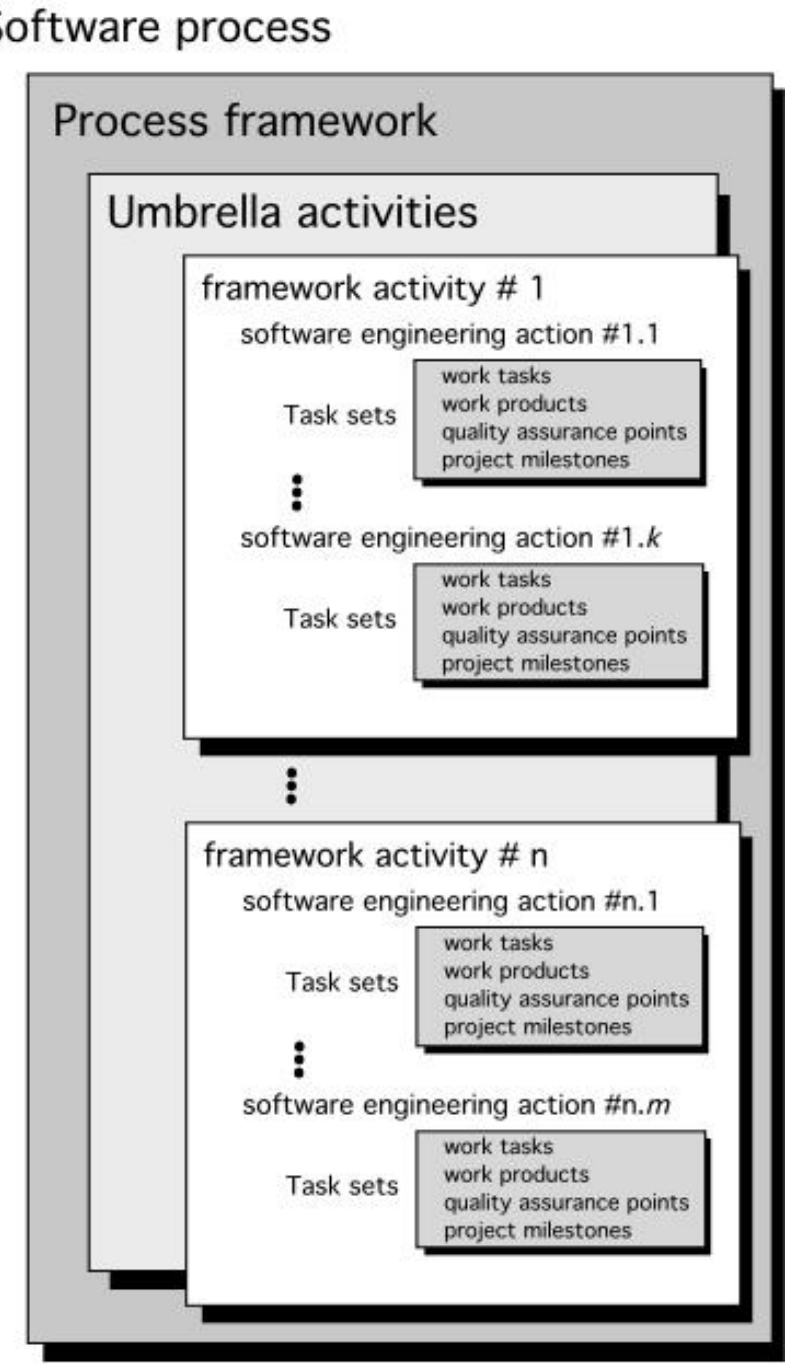
What / who / why is Process Models?

- **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 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.

Definition of Software Process

- 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.

A Generic Process Model



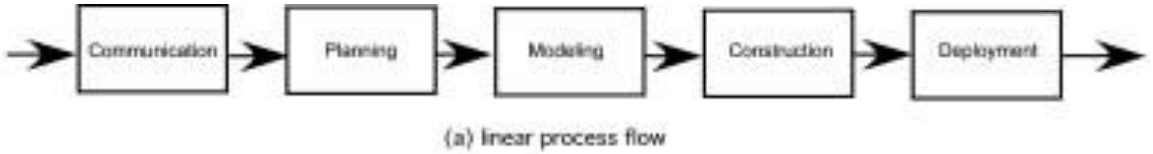
A Generic Process Model

- As we discussed before, a generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.

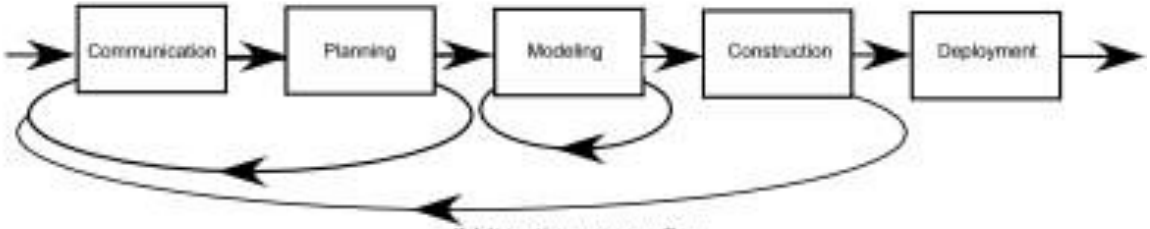
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

- 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.

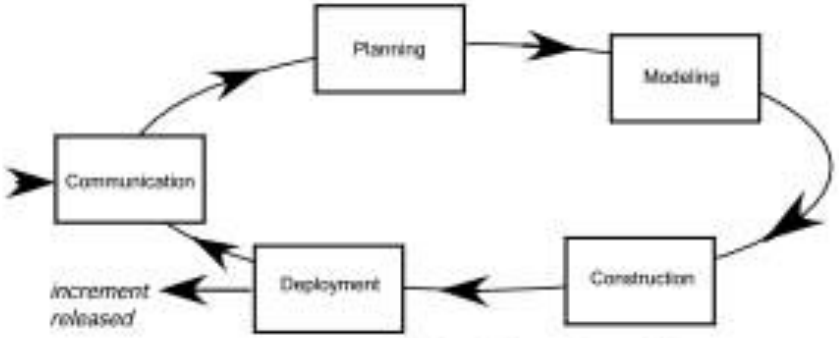
Process Flow



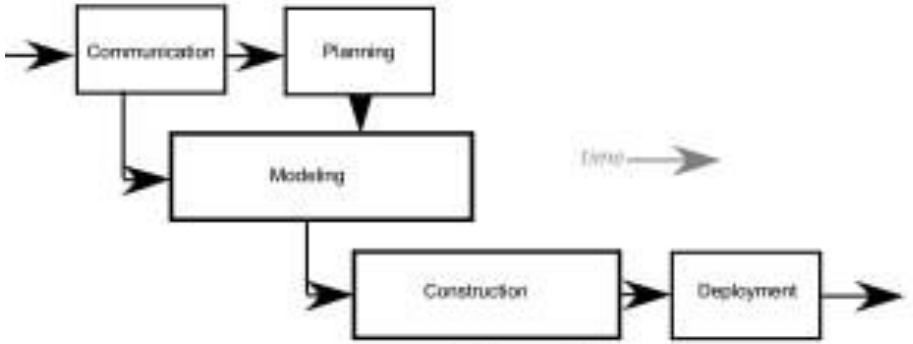
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



(d) parallel process flow

Process Flow

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

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

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

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.

Example of a Task Set for Elicitation

- The task sets for Requirements gathering 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.

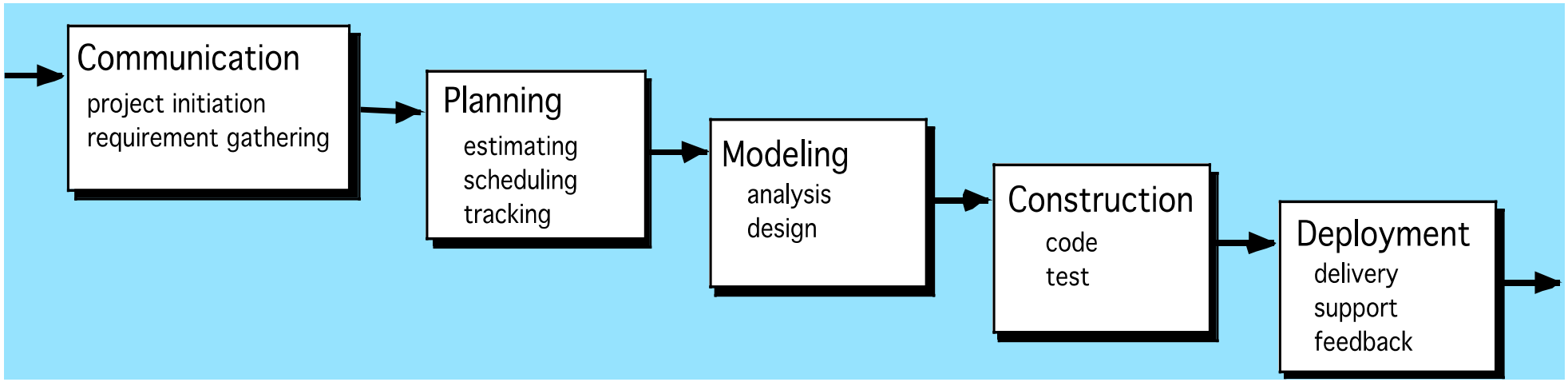
Example of a Task Set for Elicitation

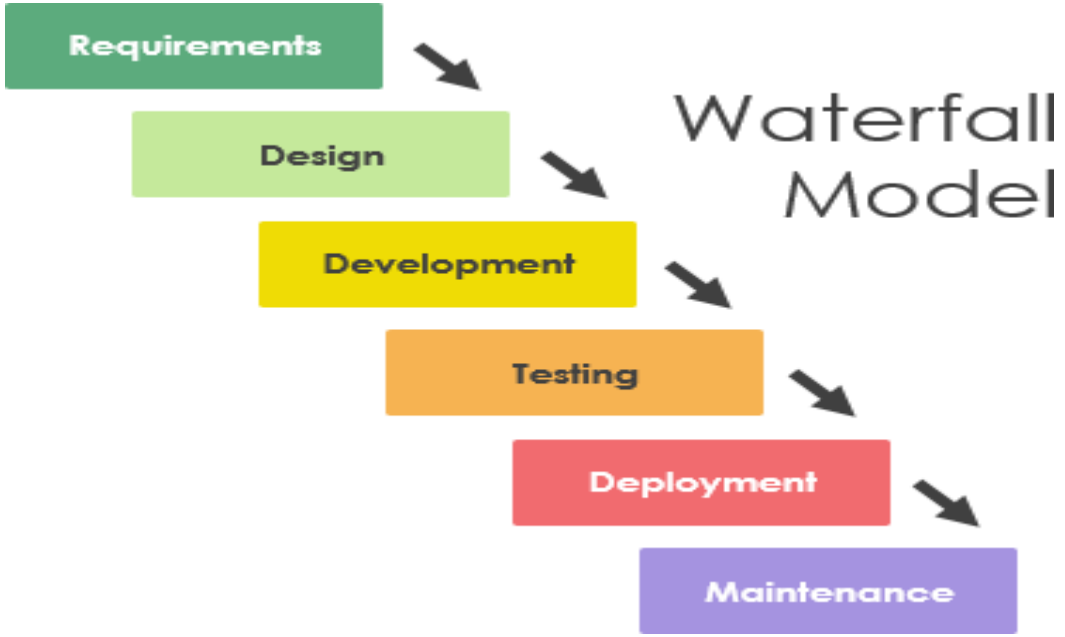
- The task sets for Requirements gathering action for a **big** project may include:
  1. Make a list of stakeholders for the project.
  2. Interview each stakeholders separately to determine overall wants and needs.
  3. Build a preliminary list of functions and features based on stakeholder input.
  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.

- Both do the same work with different depth and formality. Choose the task sets that achieve the goal and still maintain quality and agility.

Prescriptive Models

The Waterfall Model





Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors,

Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

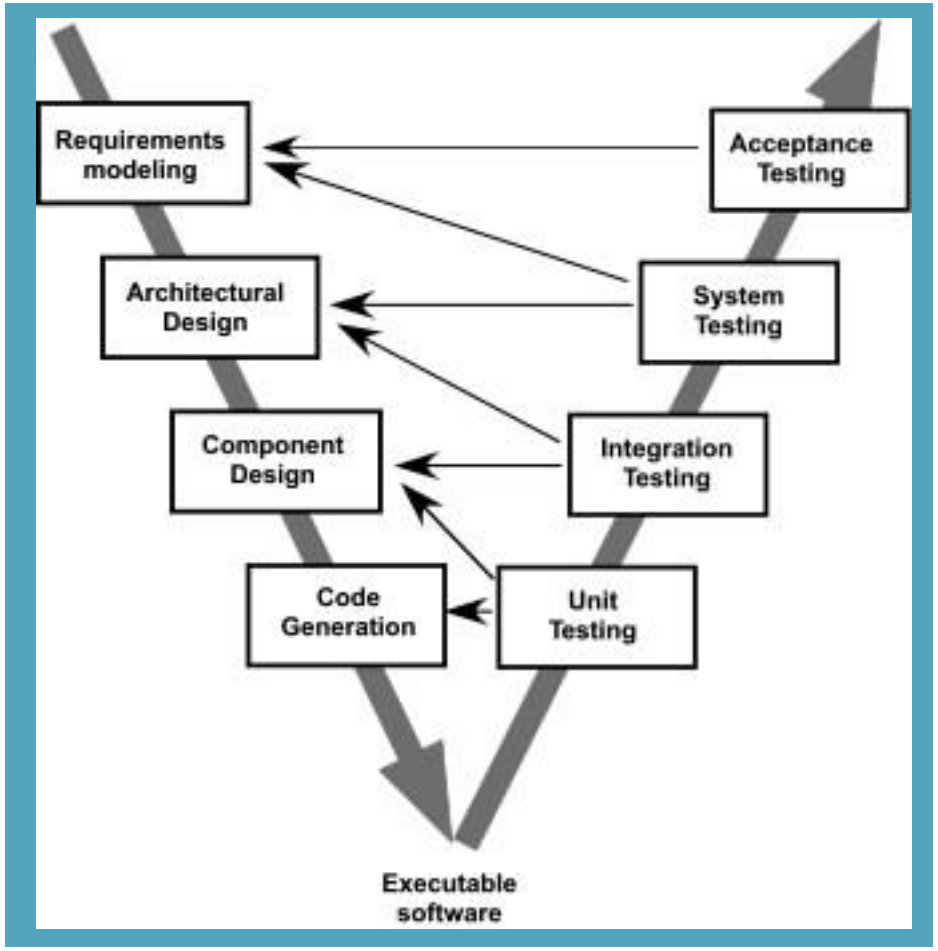
Waterfall Model - Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

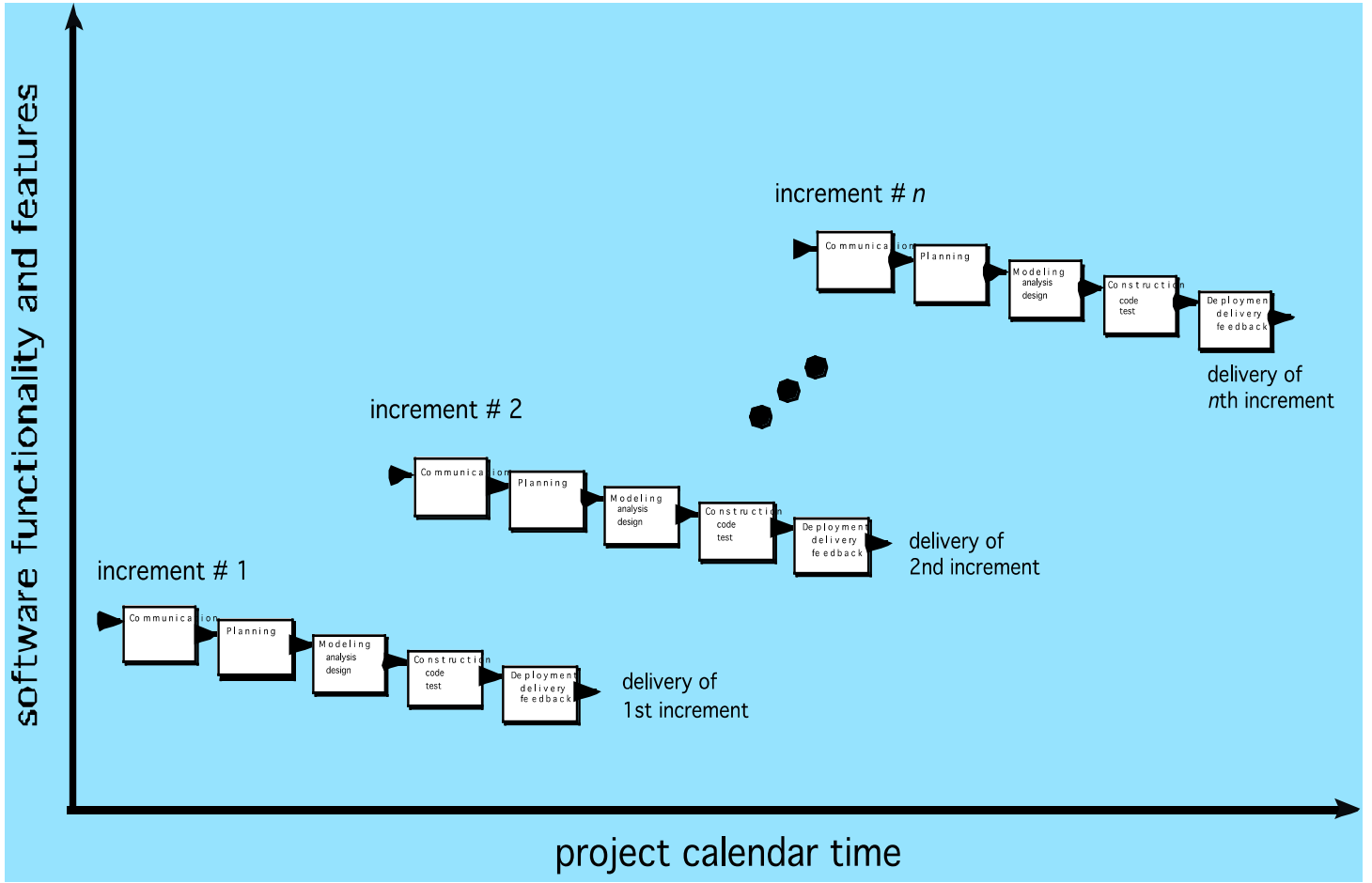
The V-Model

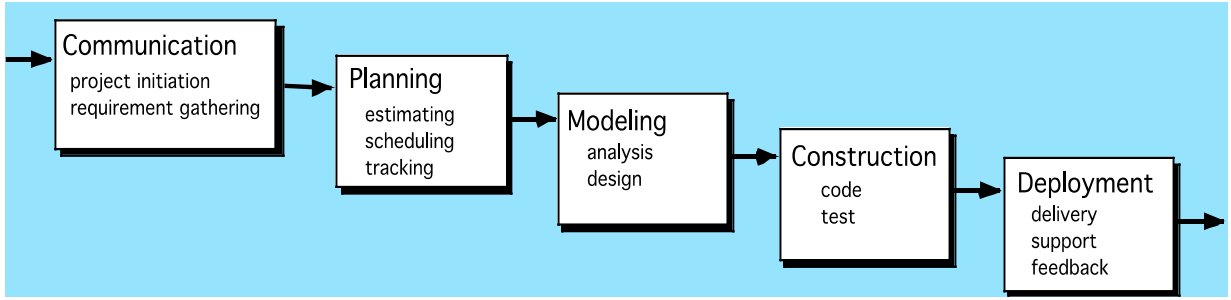


A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

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.

The Incremental Model





The Incremental Model

- Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle.
- In this model, each module goes through the requirements, design, implementation and testing phases.
- Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantages & Disadvantages of Incremental Model

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed

Evolutionary Models

Evolutionary Models

- Software system evolves over time as requirements often change as development proceeds. Thus, a straight line to a complete end product is not possible. However, a limited version must be delivered to meet competitive pressure.
- 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.
- **Following are the evolutionary process models.**

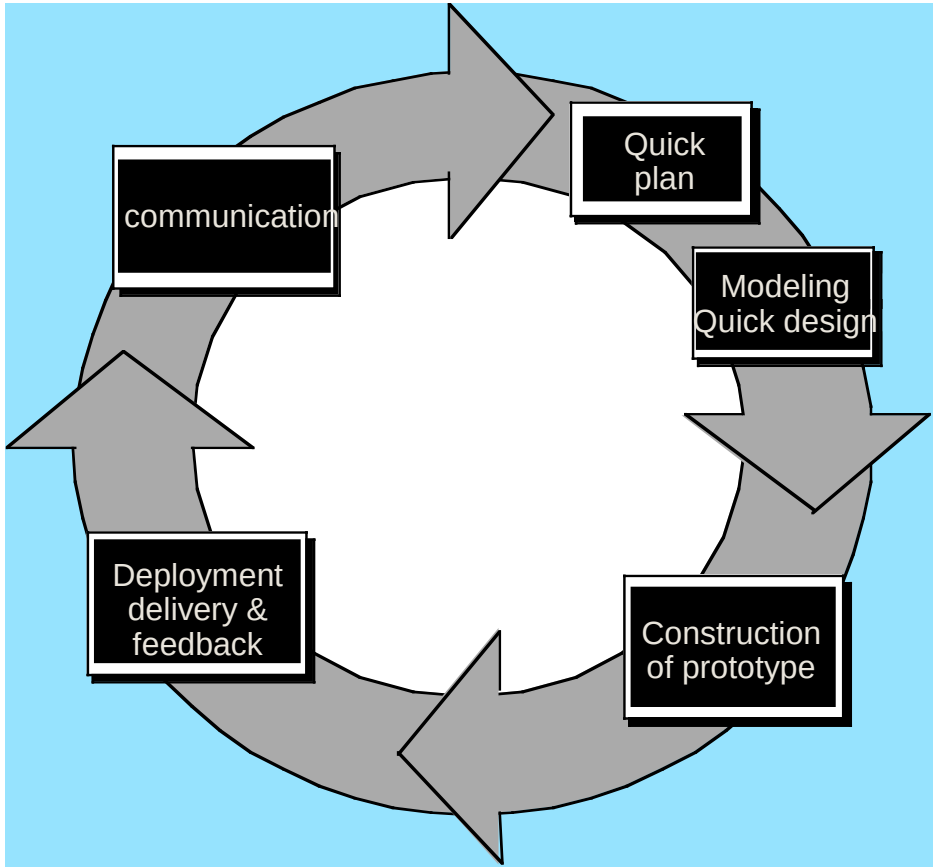
1. The prototyping model
2. The spiral model
3. Concurrent development model

The prototyping model

- 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.
- What step: Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory. A quick plan for prototyping and modeling (quick design) occur. Quick design focuses on a representation of those aspects the software that will be visible to end users. (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder’ s comments will be used to refine requirements.
- 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.

The prototyping model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.

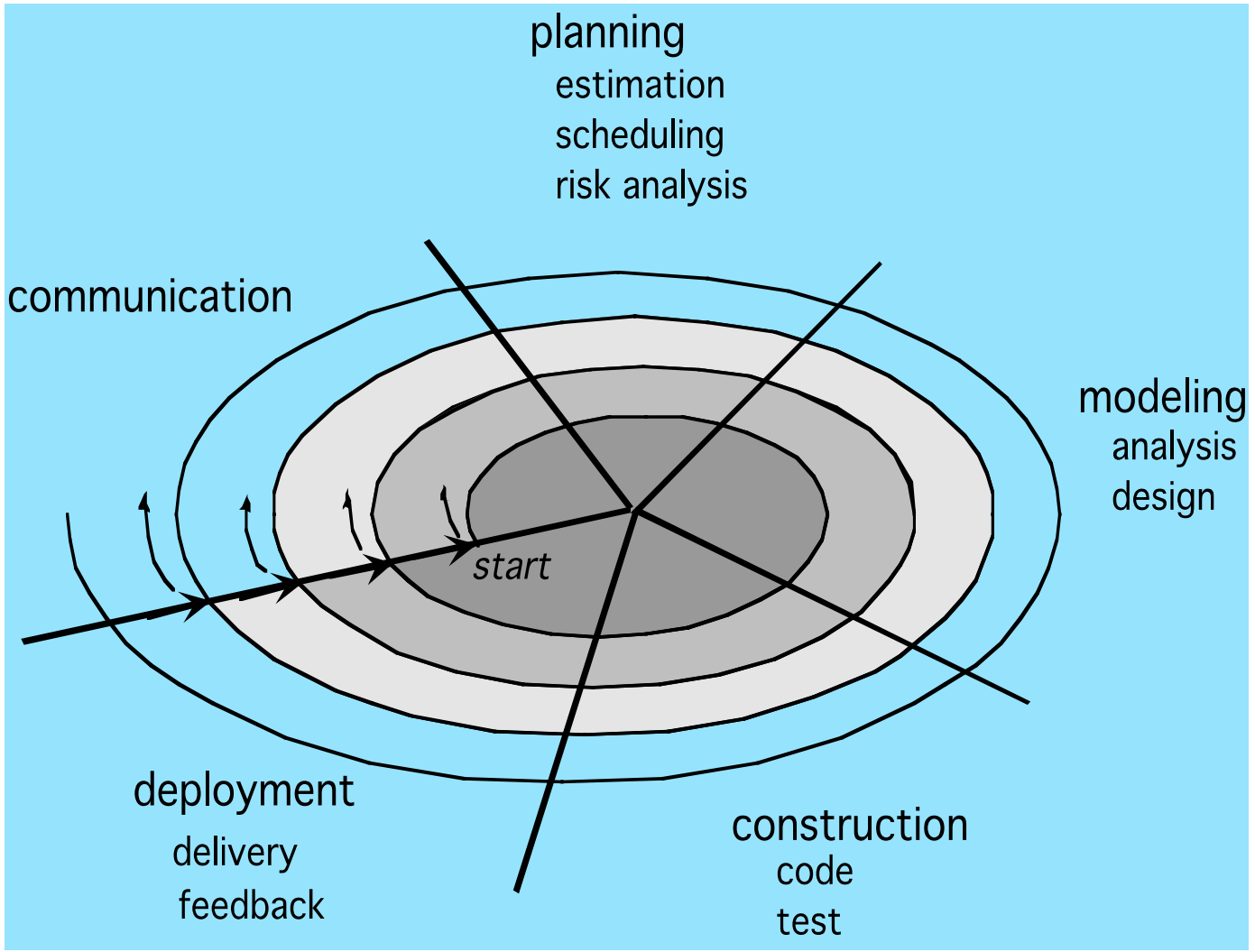


Advantages & Disadvantages of Prototyping Model

- **Advantages of Prototyping Model**
- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

- **Disadvantages of Prototyping Model**
- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.

The Spiral model



The Spiral model

- 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.
- A series of evolutionary releases are delivered. During the early iterations, the release might be a model or prototype. During later iterations, increasingly more complete version of the engineered system are produced.
- The first circuit in the clockwise direction might result in the 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 results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.
- **Good to develop large-scale system as software evolves as the process progresses and risk should be understood and properly reacted to. Prototyping is used to reduce risk.**
- **However, it may be difficult to convince customers that it is controllable as it demands considerable risk assessment expertise**.

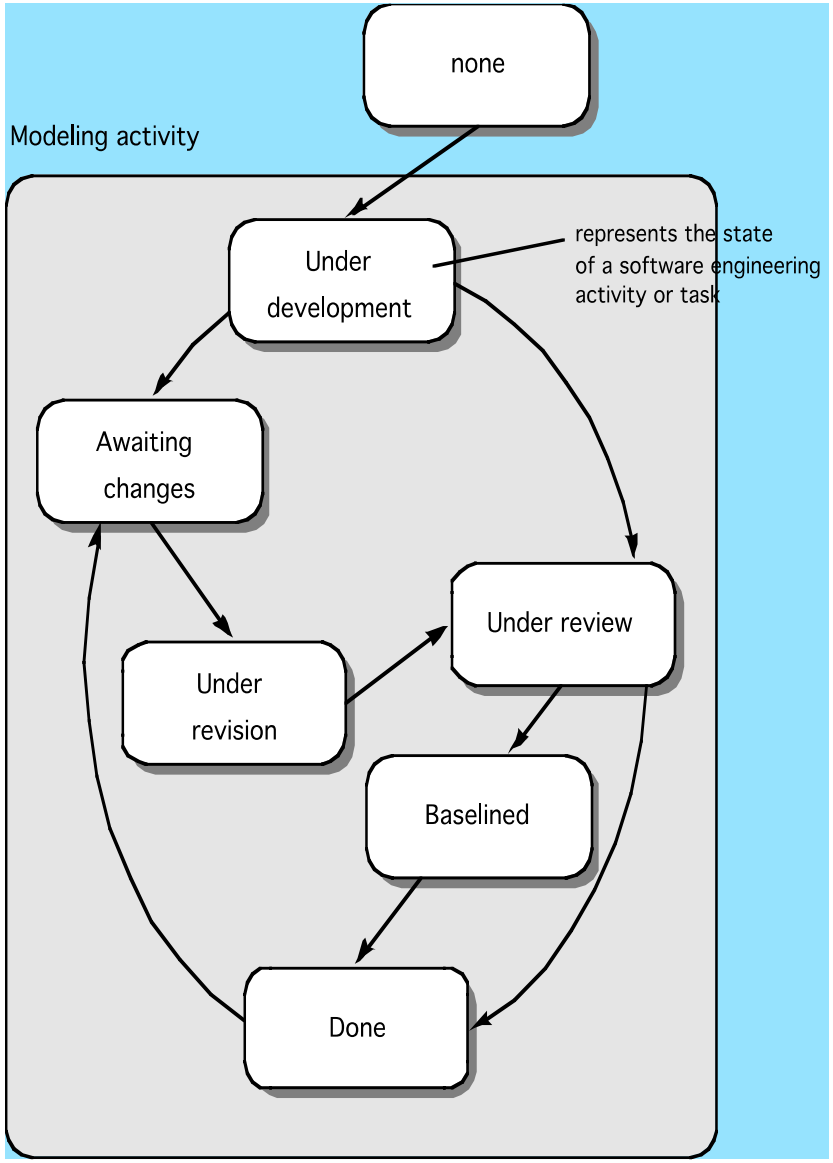
Three Concerns on Evolutionary Processes

- **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.

Concurrent development model



Concurrent development model

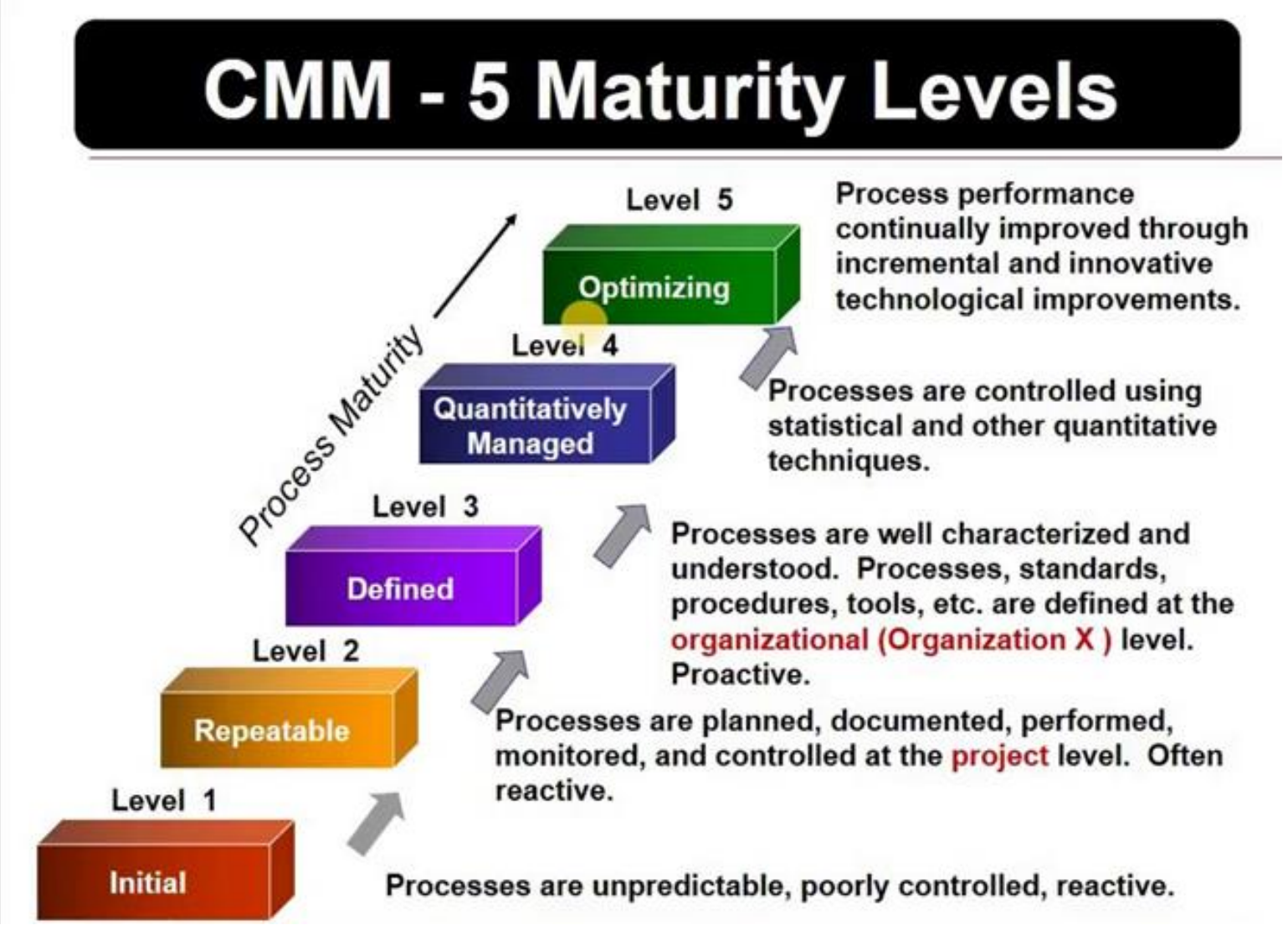
- Allow a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.
- The Figure shows modeling may be in any one of the states at any given time. For example, communication activity has completed its first iteration and in the awaiting changes state. The modeling activity was in inactive state, now makes a transition into the under development state. If customer indicates changes in requirements, the modeling activity moves from the under development state into the awaiting changes state.
- 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.

Capability Maturity Model

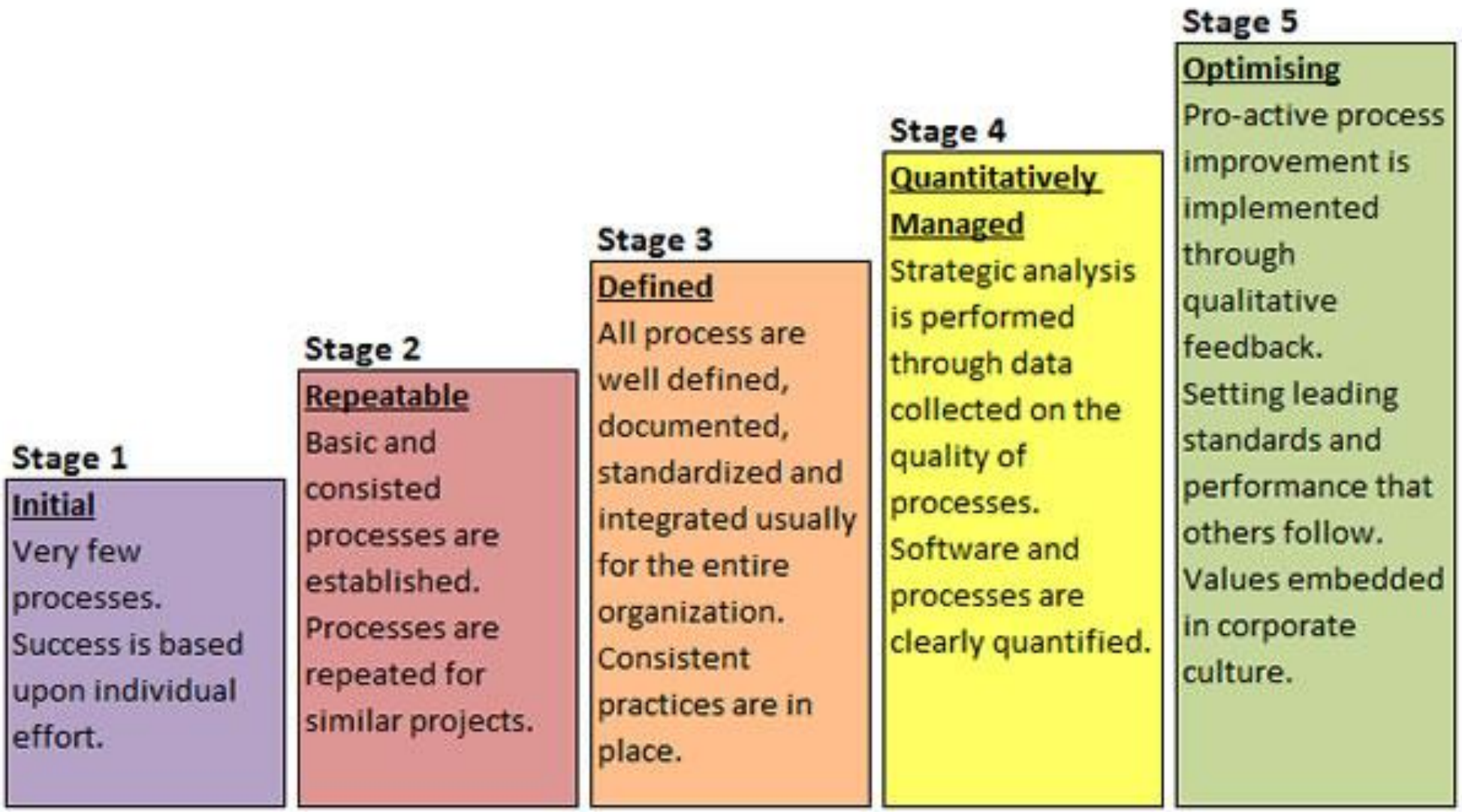
What is Capability Maturity Model?

- The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

Levels of CMM



Capability Maturity Model Maturity Levels



CMM Levels

- **Level One :Initial** - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.

- **Level Two: Repeatable** - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.

CMM Levels

- **Level Three: Defined** - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing, testing and maintaining the application.

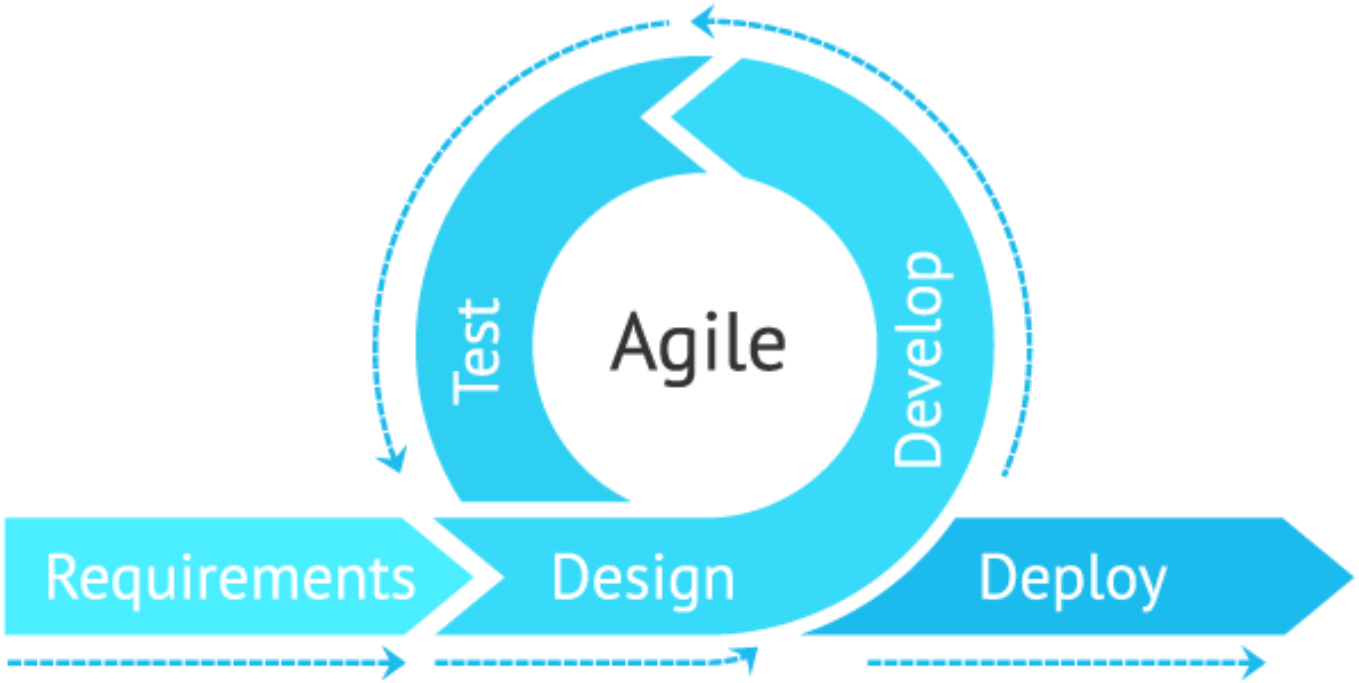
- **Level Four: Managed** - Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.

CMM Levels

- **Level Five: Optimizing** - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

WHAT IS AGILE?

- AGILE methodology is a practice that promotes **continuous iteration** of development and testing throughout the software development lifecycle of the project. In the Agile model, both development and testing activities are concurrent, unlike the Waterfall model.



What is Agile Software Development?

- The **Agile software development** methodology is one of the simplest and effective processes to turn a vision for a business need into software solutions. Agile is a term used to describe software development approaches that employ continual planning, learning, improvement, team collaboration, evolutionary development, and early delivery. It encourages flexible responses to change.
- **The agile software development emphasizes on four core values.**

1. Individual and team interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Agility Principles:

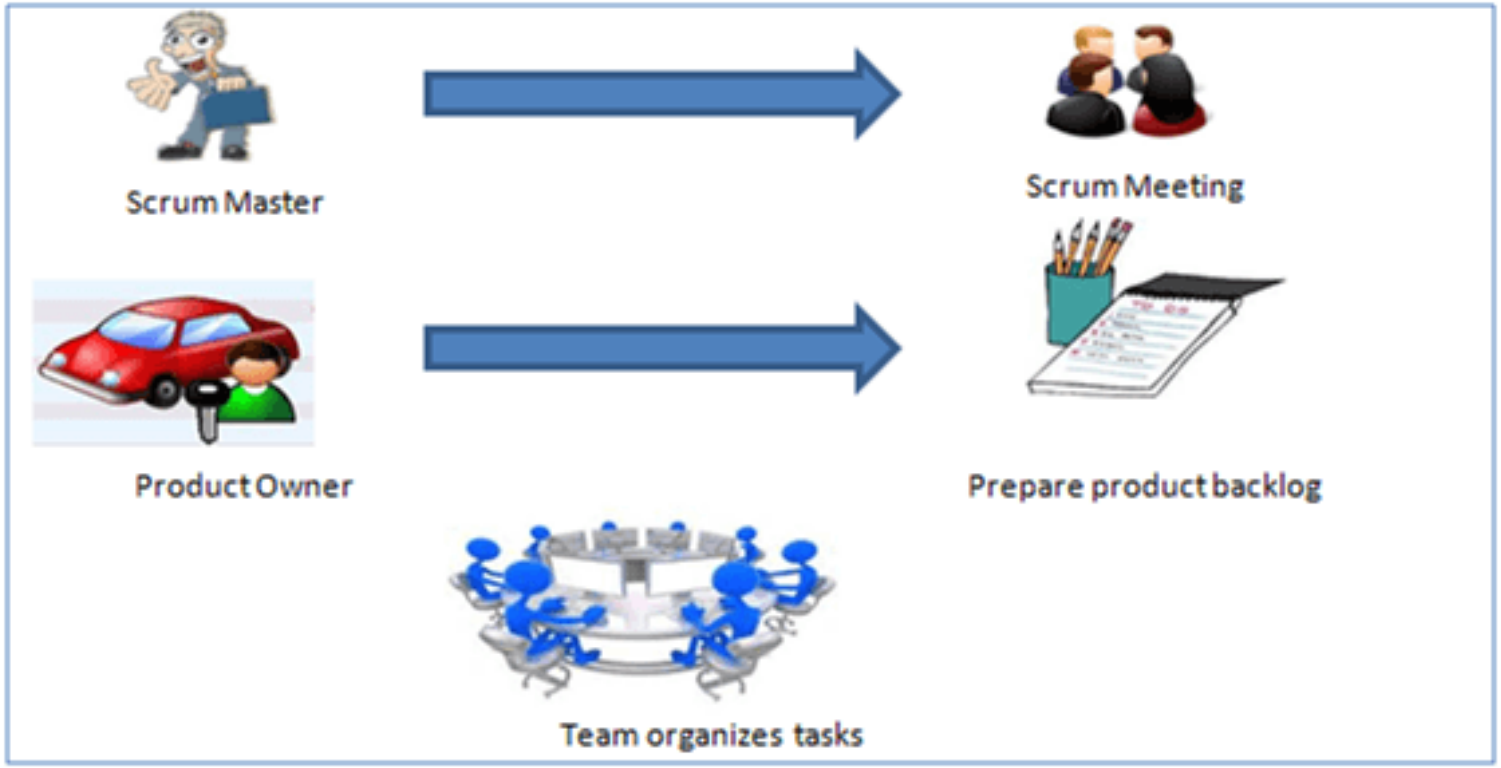
1. Our highest priority is to satisfy the client through early and continuous delivery of valuable computer software.
2. Welcome dynamical necessities, even late in development. Agile processes harness modification for the customer's competitive advantage.
3. Deliver operating computer software often, from a pair of weeks to a couple of months, with a preference to the shorter timescale.
4. Business individuals and developers should work along daily throughout the project.
5. The build comes around actuated people. offer them the setting and support they have, and trust them to urge the task done.
6. the foremost economical and effective methodology of conveyancing info to and among a development team is face-to-face speech.

Agility Principles:

7. working computer software is the primary live of progress.
8. Agile processes promote property development. The sponsors, developers, and users got to be able to maintain a relentless pace indefinitely.
9. Continuous attention to technical excellence and smart style enhances nimbleness.
10. Simplicity—the art of maximizing the number of work not done—is essential.
11. the most effective architectures, necessities, and styles emerge from self–organizing groups.
12. At regular intervals, the team reflects on a way to become simpler, then tunes and adjusts its behavior consequently.

SCRUM

- SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment. Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). It consists of three roles, and their responsibilities are explained as follows:

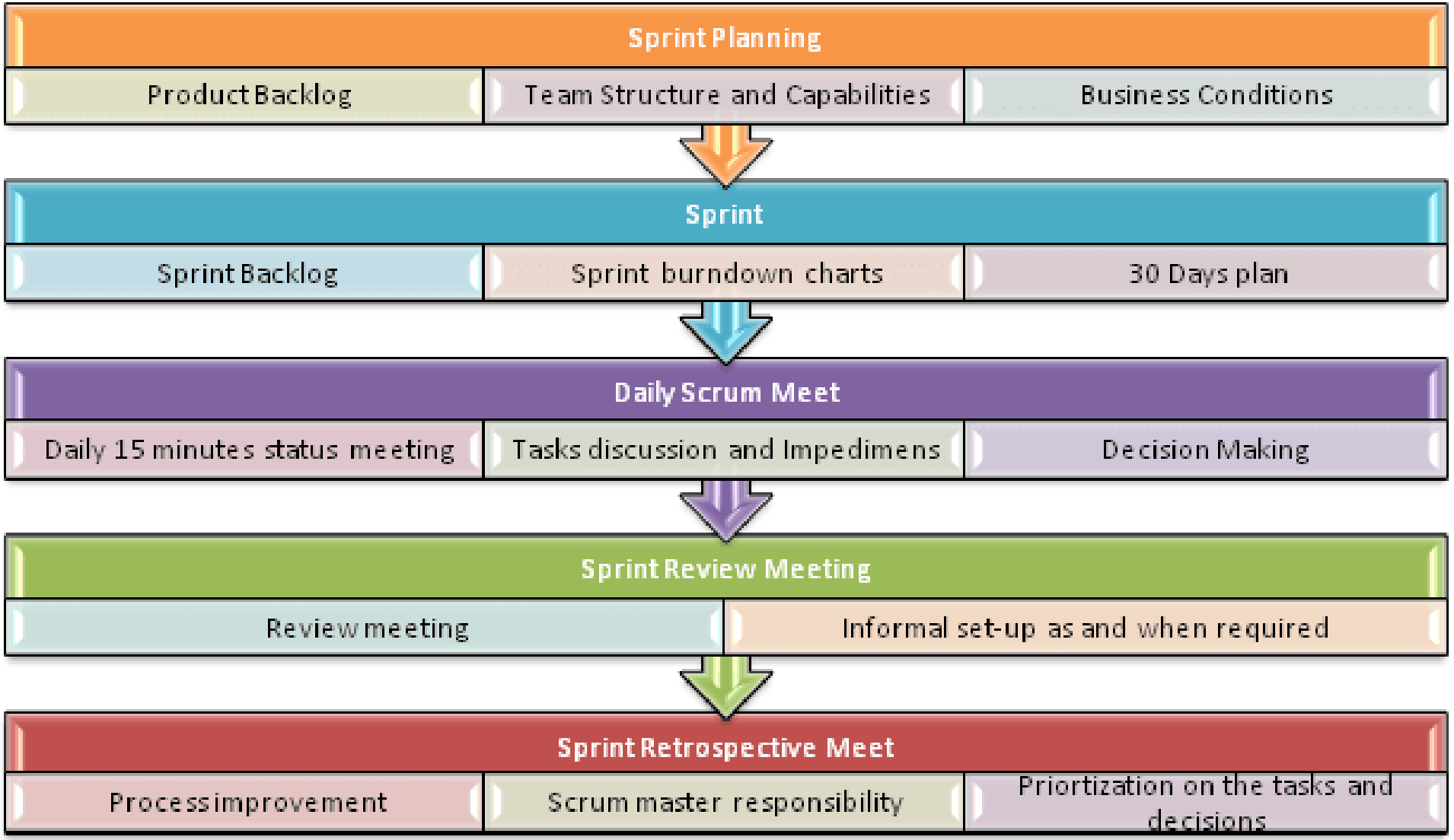


SCRUM

- 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

- **Product Backlog**
  - This is a repository where requirements are tracked with details on the no of requirements(user stories) to be completed for each release. It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team. Team can also request for a new requirement addition or modification or deletion

Scrum Practices



Process flow of Scrum Methodologies:

Process flow of scrum testing is as follows:

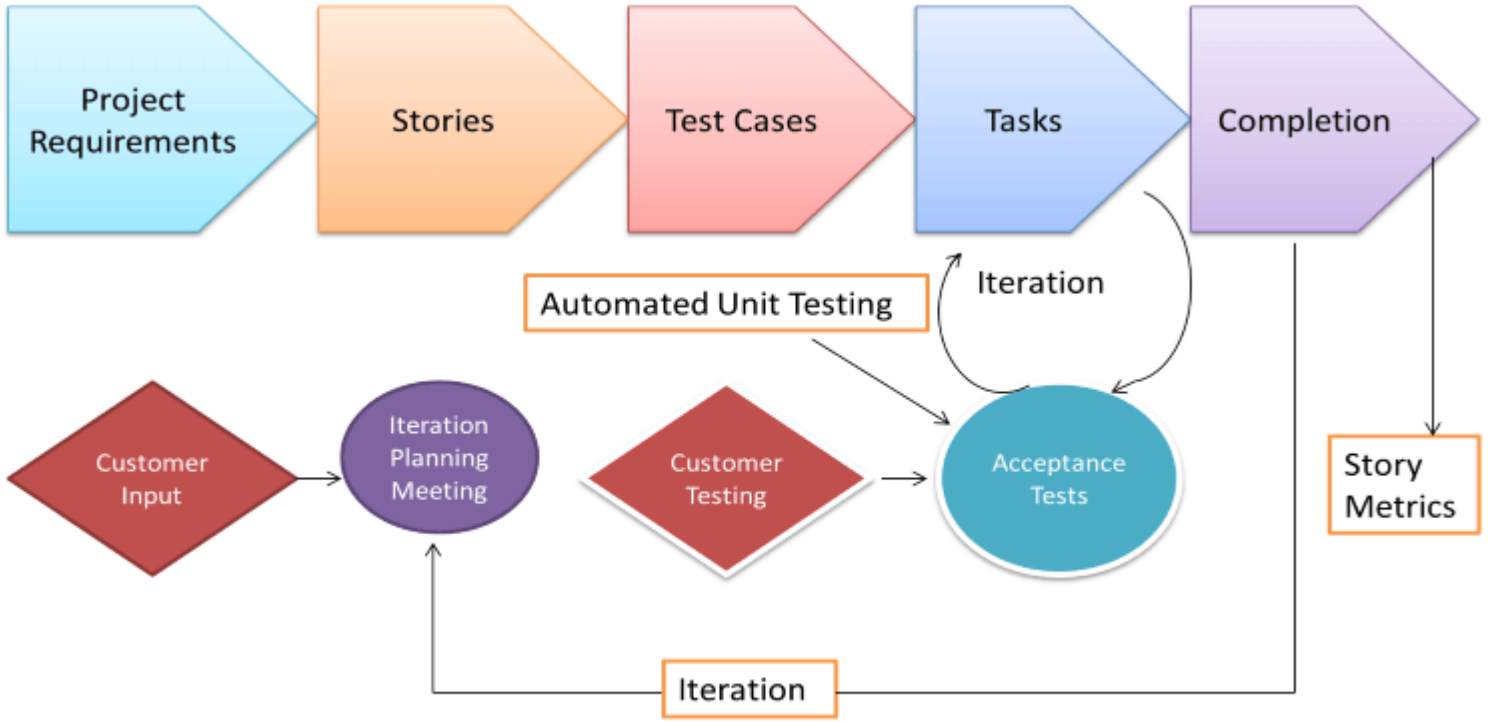
- Each iteration of a scrum is known as Sprint
- Product backlog is a list where all details are entered to get the end-product
- During each Sprint, top user stories of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

eXtreme Programming (XP)

- Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.

- It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented. The XP develops software keeping customer in the target.

- Business requirements are gathered in terms of stories. All those stories are stored in a place called the parking lot.
- In this type of methodology, releases are based on the shorter cycles called Iterations with span of 14 days time period. Each iteration includes phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.



Extreme Programming (XP)

Phases of eXtreme programming:

- There are 6 phases available in Agile XP method

1. ***Planning***

- Identification of stakeholders and sponsors
- Infrastructure Requirements
- Security related information and gathering
- Service Level Agreements and its conditions

2. Analysis

- Capturing of Stories in Parking lot
- Prioritize stories in Parking lot
- Scrubbing of stories for estimation
- Define Iteration SPAN(Time)
- Resource planning for both Development and QA teams

Phases of eXtreme programming:

3. Design

- Break down of tasks
- Test Scenario preparation for each task
- Regression Automation Framework

4. Execution

- Coding
- Unit Testing
- Execution of Manual test scenarios
- Defect Report generation
- Conversion of Manual to Automation regression test cases
- Mid Iteration review
- End of Iteration review

Phases of eXtreme programming:

5. Wrapping

- Small Releases
- Regression Testing
- Demos and reviews
- Develop new stories based on the need
- Process Improvements based on end of iteration review comments

6. Closure

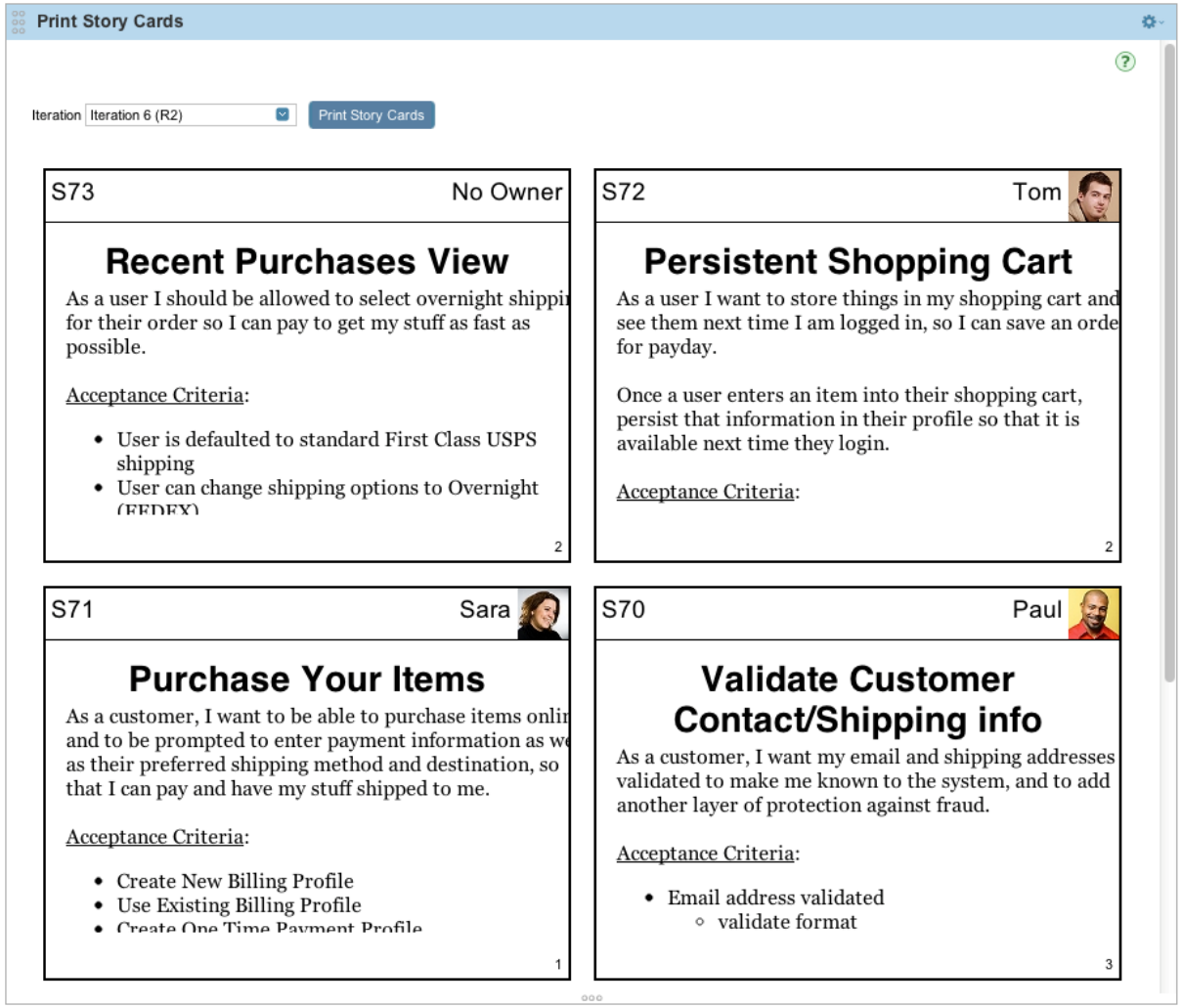
- Pilot Launch
- Training
- Production Launch
- SLA Guarantee assurance
- Review SOA strategy
- Production Support

eXtreme programming

There are two storyboards available to track the work on a daily basis, and those are listed below for reference.

- Story Cardboard
  - This is a traditional way of collecting all the stories in a board in the form of stick notes to track daily XP activities. As this manual activity involves more effort and time, it is better to switch to an online form.

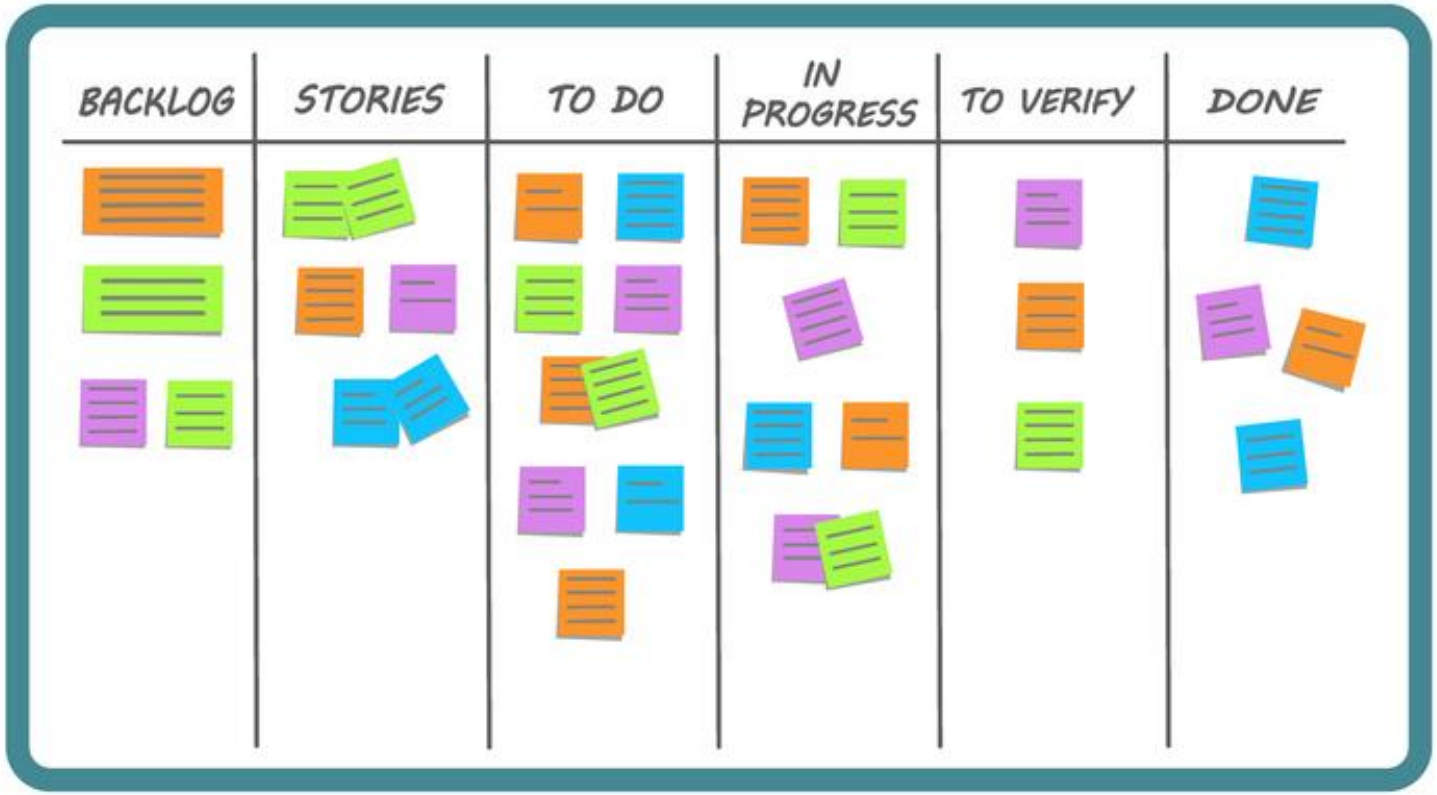
- Online Storyboard
  Online tool Storyboard can be used to store the stories. **Several teams can use it** for different purposes.



What is Kanban?

- Agile Kanban is Agile Software Development with Kanban approach. In Agile Kanban, the Kanban board is used to visualize the workflow.
- The Kanban board is normally put up on a wall in the project room.
- The status and progress of the story development tasks is tracked visually on the Kanban board with flowing Kanban cards.

Kanban Board



Kanban Board

Kanban board is used to depict the flow of tasks across the value stream. The Kanban board –

- Provides easy access to everyone involved in the project.
- Facilitates communication as and when necessary.
- Progress of the tasks are visually displayed.
- Bottlenecks are visible as soon as they occur.

Advantages of Kanban board

- The major advantages of using a Kanban board are –
- **Empowerment of Team** – This means –
  - Team is allowed to take decisions as and when required.
  - Team collaboratively resolves the bottlenecks.
  - Team has access to the relevant information.
  - Team continually communicates with customer.
- **Continuous Delivery** – This means –
  - Focus on work completion.
  - Limited requirements at any point of time.
  - Focus on delivering value to the customer.
  - Emphasis on whole project.

- The tasks and stories are represented by Kanban cards.
- The current status of each task is known by displaying the cards in separate columns on the board.
- The columns are labeled as **To Do, Doing**, and **Done**.
- Each task moves from **To Do** to **Doing** and then to **Done**.

Software Development Kanban

Backlog	Analysis		Design		Development		QA		To Deploy
6	3		5		4		3		6
Feature U	Doing	Done	Doing	Done	Doing	Done	Doing	Done	Feature A
Feature V	Feature S	Feature R	Feature O	Feature M	Feature K	Feature I	Feature G	Feature F	Feature B
Feature W	Feature T		Feature P	Feature N	Feature L	Feature J	Feature H		Feature C
Feature X			Feature Q						Feature D
									Feature E

Summary

- Nature of Software, Software Engineering, Software Process,
- Generic Process Model,
- Prescriptive Process Models: The Waterfall Model, V-model, Incremental Process Models,
- Evolutionary Process Models: Prototyping Model, Spiral Model, Concurrent Models,
- Capability Maturity Model (CMM)
- Agile process, Agility Principles,
- Extreme Programming (XP),
- Scrum, Kanban model