Software Engineering(SE)

CSC 601

Subject Incharge

Varsha Nagpurkar

Assistant Professor

Room No. 407

email: varshanagpurkar@sfit.ac.in

Module- I

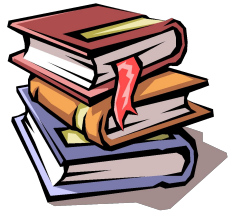
Module No.	Unit No.	Topics
1.0		**Introduction To Software Engineering and Process Models**
	1.1	Nature of Software, Software Engineering, Software Process, Capability Maturity Model (CMM)
	1.2	Generic Process Model, **Prescriptive Process Models:** The Waterfall Model, V-model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Agile process, Agility Principles, Extreme Programming (XP), Scrum, Kanban model

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) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

What is software?

Computer programs and **associated documentation**



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

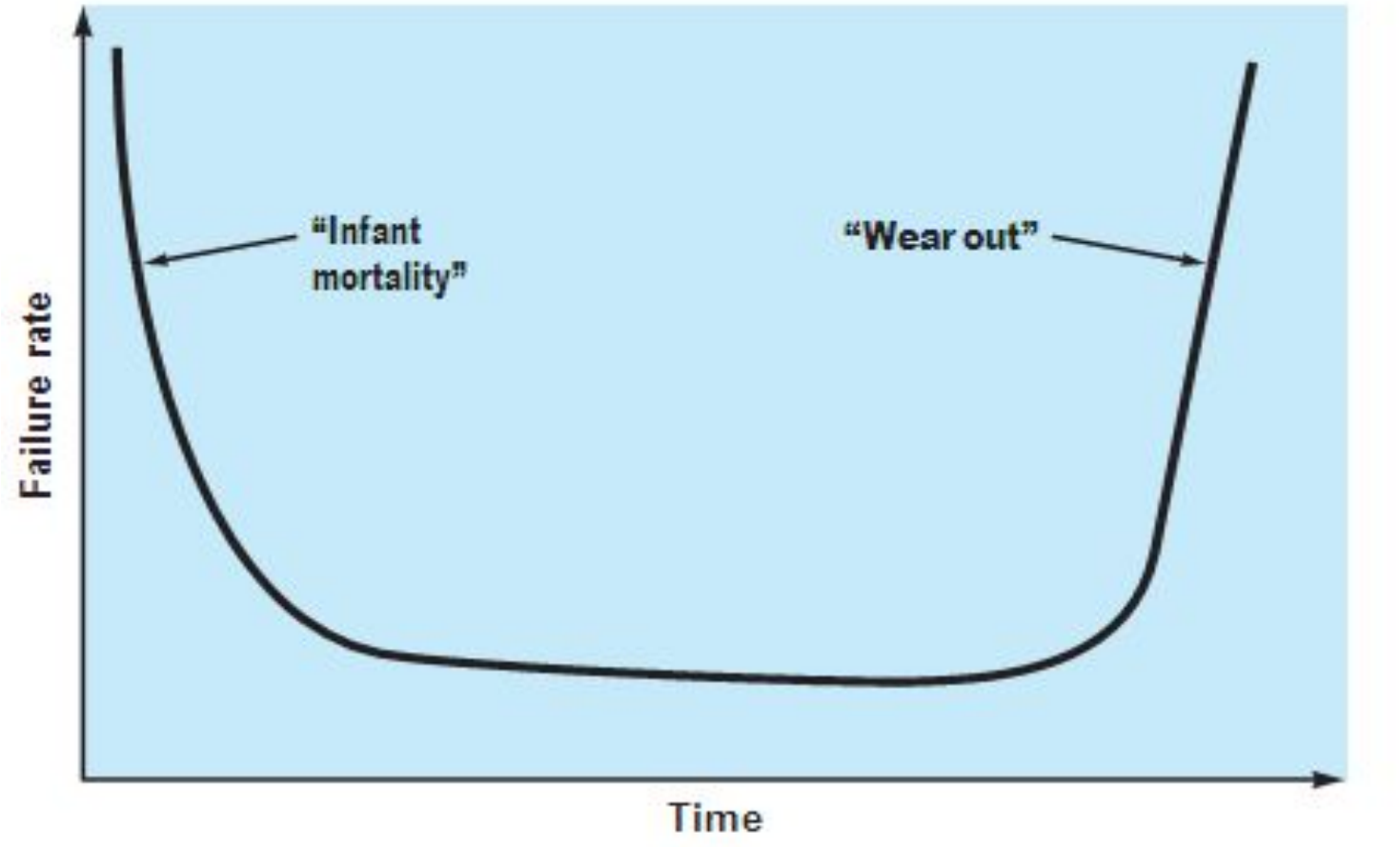
Software products may be

- **Generic - developed to be sold to a range of different customers**
- **Bespoke (custom) - developed for a single customer according to their specification**

Software Characteristic

- Software is a logical rather than a physical system element.
- Software has one fundamental characteristic that makes it considerably different from hardware: *Software doesn't "wear out.*-exhaust"

Failure curve for Hardware



Failure curve for Hardware

- The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects);
- defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time.

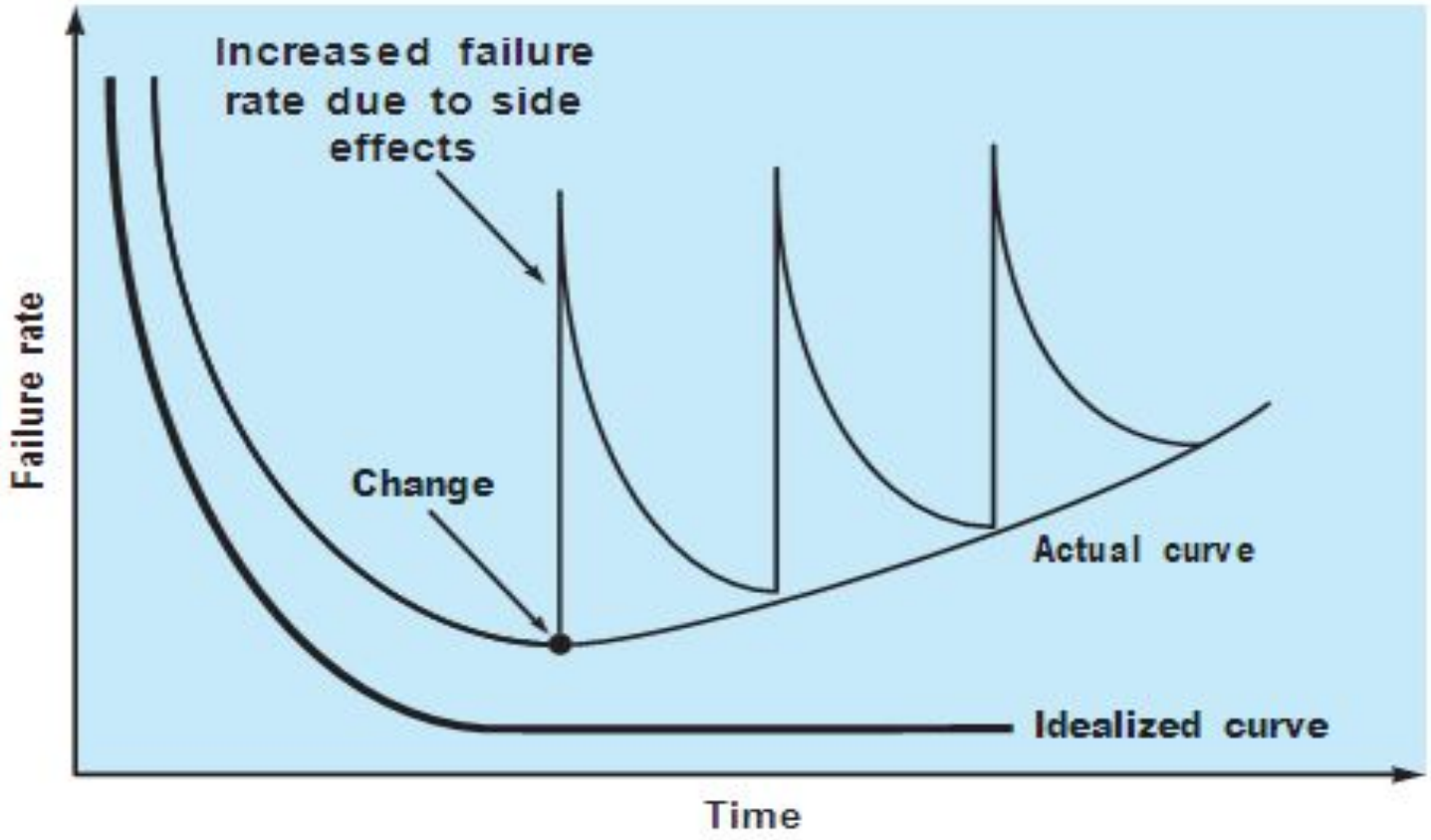
Failure curve for Hardware

- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to *wear out.*

Failure curve for Software

- Software is not susceptible to the environmental maladies that cause hardware to wear out.
- In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in Figure
- Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. However, the implication is clear—software doesn't wear out. But it does *deteriorate!*

Failure curve for Software



Failure curve for Software

- During its life, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the figure .
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again.
- Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

Software Application Domains

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-line software
- Web/Mobile applications
- Artificial intelligence software

Software Application Domains

- **System software—**
- A collection of programs written to service other programs.
- It is characterized by its heavy interaction with computer hardware.
- Some system software examples (e.g., compilers, editors, and file management utilities)operating systems,drivers, networking software

Software Application Domains

- **Application software—**
- It consists of stand-alone programs that solve a specific business need.
- Examples of application software
  - Hospital management system
  - Bank management system
  - A payroll system

Software Application Domains

- **Engineering/scientific software—**
- **In this type application areas are:**
  - Astronomy-Theory of space
  - Volcanology-Science about volcano
  - Molecular biology
  - Computer-aided design(e.g.AutoCAD software)

Software Application Domains

- **Embedded software—**
- **It** resides within a product or system and is used to implement and control features and functions for the end user and for the system itself.
- Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven)

Software Application Domains

- **Product-line software—**designed to provide a specific capability for use by many different customers.
- The personal computer software comes in this category

Examples of product-line software

- Inventory control(A software developed for some supermarket keeping track of inventory database and related manipulations.
- Word processing
- Spreadsheets
- Multimedia(Audio,Video)
- Entertainment(e.g.animation and other movies)

Software Application Domains

- **Web/Mobile applications—**
- This network-centric software category spans a wide range of applications and includes both browser-based apps and software that resides on mobile devices.
- Website related applications with databases,hyperlinks and graphics capabilities
- E-commerce web applications

Software Application Domains

- **Artificial intelligence software—**makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straight- forward analysis.
- Applications within this area include robotics, expert systems, pattern recognition (image and voice), artiftcial neural networks, theorem proving, and game playing.

What is software engineering?

Software engineering is an engineering discipline which is concerned with all aspects of software production

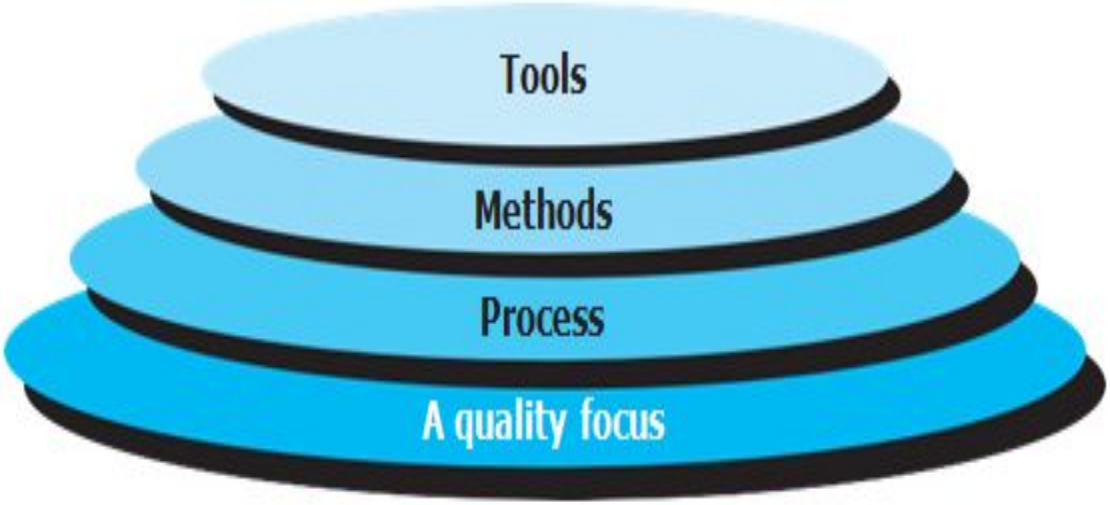
Software engineers should

- adopt a systematic and organised approach to their work
- use appropriate tools and techniques depending on
  - **the problem to be solved,**
  - **the development constraints and**
  - **the resources available**

Software Engineering

- Software Engineering : (1) The application of a systematic, disciplined, quantiftable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- Software engineering is a layered technology. Referring to Figure, any en- gineering approach (including software engineering) must rest on an organiza-tional commitment to quality.

Software Engineering Layers



Quality Focus

- Quality is nothing but ‘degree of goodness’.
- Good quality software has following characteristics:
  - Correctness is degree to which software performs its required function
  - Maintainability is an ease with which software is maintained
  - Integrity is a security provided so that unauthorized user can not access data or information.e.g.password authentication
  - Usability is the efforts required to use or operate the software.

Software Engineering Layers-Process Layer

- The foundation for software engineering is the *process* layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software
- A *process* is a collection of activities, actions, and tasks that are performed when some work product is to be created.

Software Engineering Layers-Process Layer

- Software process defines following 'what' activities,actions and tasks for software development:
  - What activities are to be carried out?
  - What actions will be taken?
  - What tasks are to be carried out in a given action?

*

Software Engineering Layers-Process Layer

- An *activity* refers to achieve a broad objec-tive (e.g., communication with stakeholders)
- An *action* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural model).
- A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

Software Engineering Layers-Methods

- Software engineering *methods* provide the technical way to implement the software.
- Methods consist of collection of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

Software Engineering Layers-Tools

- Software engineering *tools* provide automated or semi-automated support for the process and the methods.
- For example
  - Microsoft front page or Microsoft Publisher can be used as web designing tool.
  - Rational Rose can be used as object oriented analysis and design tool.

The Process Framework-A Generic Process Model

- A generic process framework for software engineering encompasses five activities:
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment

A software process framework

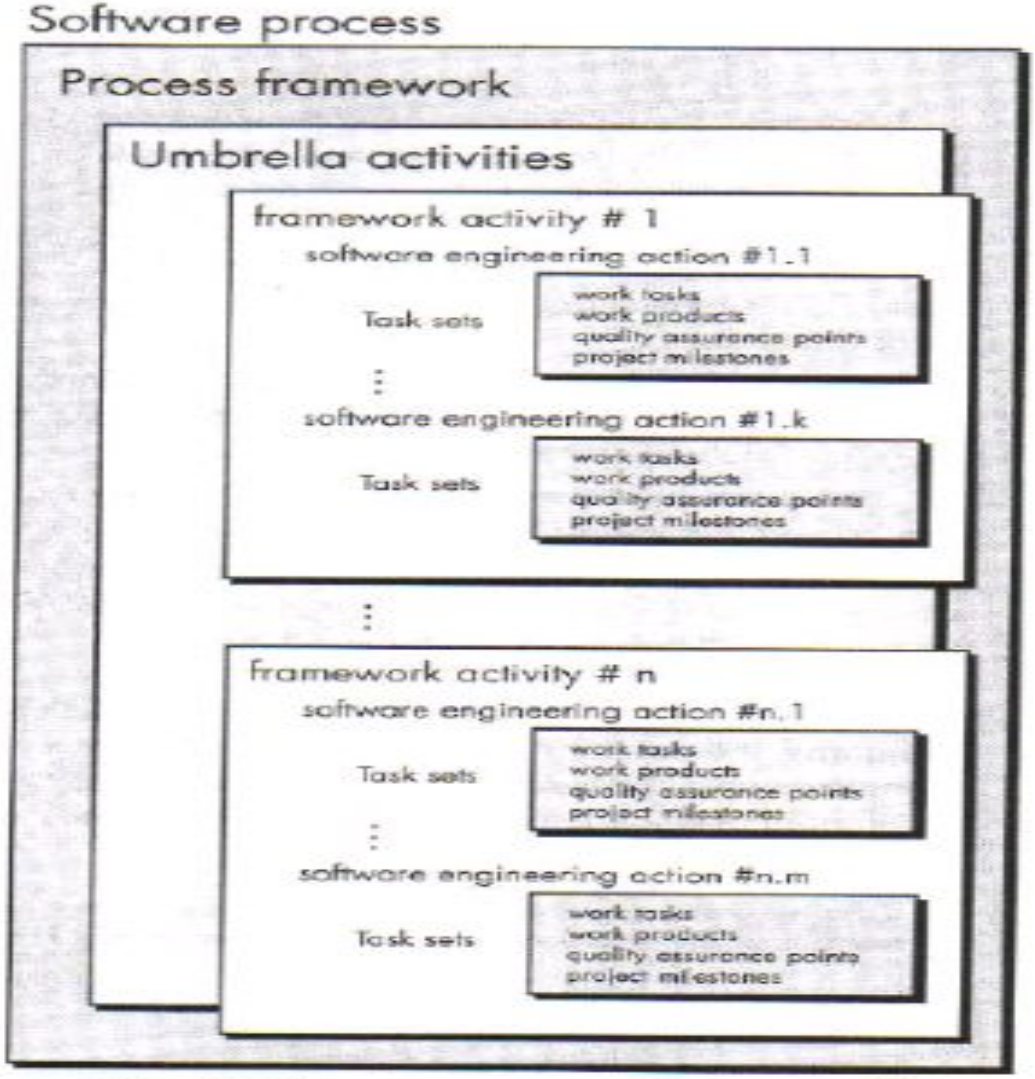


Fig. Software process framework

A software process framework

- Each framework activity is populated by a set of software engineering actions-a collection of related tasks that produces a major software engineering work product(e.g.,design is a software engineering action)
- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

Umbrella Activities

- Software engineering process framework activities are complemented by a num- ber of *umbrella activities.*
- *U*mbrella activities are applied throughout a software project and help a software team manage and control progress, qual-ity, change, and risk. Typical umbrella activities include:

*

Umbrella Activities

- **Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**—defines and conducts the activities required to ensure software quality.
- **Technical reviews**—assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Umbrella Activities

- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs
- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (includ-ing software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—includes the activities re-quired to create work products such as models, documents, logs, forms, and lists.

Prescriptive Process Models

- All software organizations should encompass the following set of framework activities in their process model
  - Communication
  - Planning
  - Modelling
  - Construction
  - Deployment

Prescriptive Process Models

- The name ‘prescriptive’ is given since the model prescribes set of activities,actions,tasks,quality assurance and change control mechanism for every project
- It also prescribes a workflow
- Workflow is the flow of process elements and the manner in which they are interrelated
- Each prescriptive model put different emphasis to generic framework activities and give different workflow.

Prescriptive Process Models

- The waterfall model
- Incremental process models
- Evolutionary process models
- The specialized process models

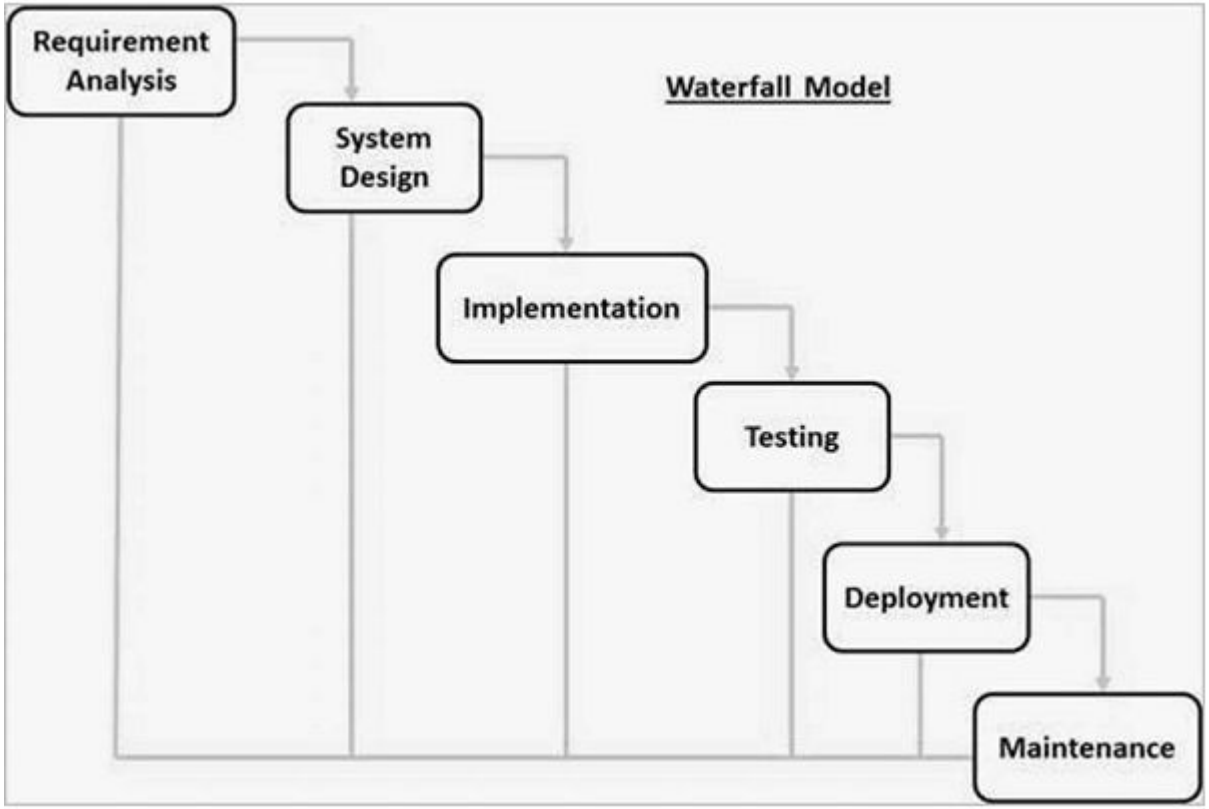
Prescriptive Process Models

- The waterfall model
- Incremental process models
  - The Incremental Model
  - The RAD Model
- Evolutionary process models
  - Prototyping
  - The Spiral Model
  - The Concurrent Development Model
- The specialized process models
  - Component-Based Development
  - The Formal Methods Model
  - Aspect-Oriented Software Development

The *waterfall model*

- The *waterfall model,* sometimes called the *Classic Life Cycle or Linear Sequential Model*, suggests a system-atic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, con- struction, and deployment,

The *waterfall model*



The *waterfall model*

- Communication:-
  - It starts with communication between customer and developer.
  - According to this model customer must state all requirements at the beginning of project.
- Planning:-
  - It includes complete estimation(e.g cost estimation of project) and scheduling(complete timeline chart for project development) and tracking.

The *waterfall model*

- Modelling:-
  - It includes detail requirement analysis and project design(algorithm,flowchart etc).
  - Flowchart shows complete pictorial flow of program whereas algorithm is step-by-step solution of problem.
- Construction:-
  - It includes coding and testing phases
  - Coding-Design details are implemented using appropriate programming language
  - Testing-It is carried out to check whether flow of coding is correct,to check out the errors of program

The *waterfall model*

- Deployment-
  - It includes software delivery,support and feedback from customer

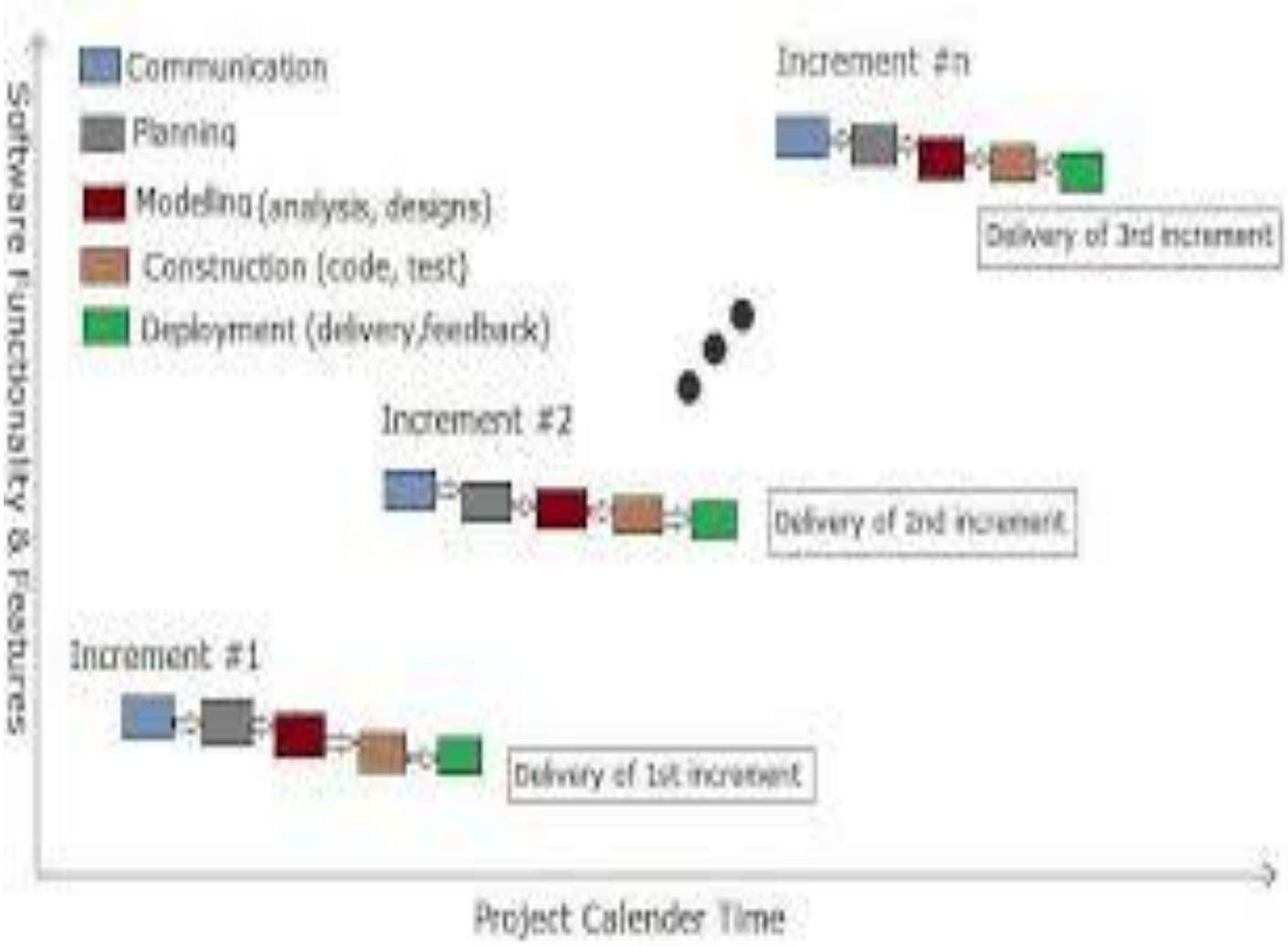
The incremental Model

- It combines the elements of the waterfall model applied in an interactive fashion
- Each linear sequence produces deliverable "increments" of the software
- For example,word processing software developed using incremental paradigm might deliver basic file management,editing and document production functions in the first increment;more sophisticated editing,and document production capabilities in the second increment;spelling and grammer checking in the third increment;and advanced page layout capability in the fourth increment

The incremental Model

- In this model,the first increment is generally a core product
- That is,basic requirements are addressed,but many supplementary features(some known,others unknown)remain undelivered.
- The core product is used by the customer(or undergoes detailed evaluation)
- As a result of use and/or evaluation,a plan is developed for the next increment
- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality
- This process is repeated following the delivery of each increment,until the complete product is produced

The incremental Model



RAD Model

- Rapid Application Developed(RAD) is an incremental software process model that emphasizes a short development cycle.
- The RAD model is a high-speed adaptation of the waterfall model,in which rapid development is achieved by using a component based construction approach
- The RAD process enables a development team to create a "fully functional system" within a very short time period(e.g.,60 to 90 days)
- Like other process models,the RAD approach maps into the generic framework activities

RAD Model

- If a business application can be modularized in a way that enables each major function to be completed in less than three months,it is a candidate for RAD
- Each major function can be addressed by a separate RAD team and then integrated to form a whole

In RAD model,each team carries out the following steps

- Communication- It works to understand the business problem and the information characteristics that the software must accommodate
- Planning-It is essential because multiple software teams work in parallel on different system functions
- Modeling-It consists of 3 major phases-business modeling,data modeling and process modeling

The RAD Model

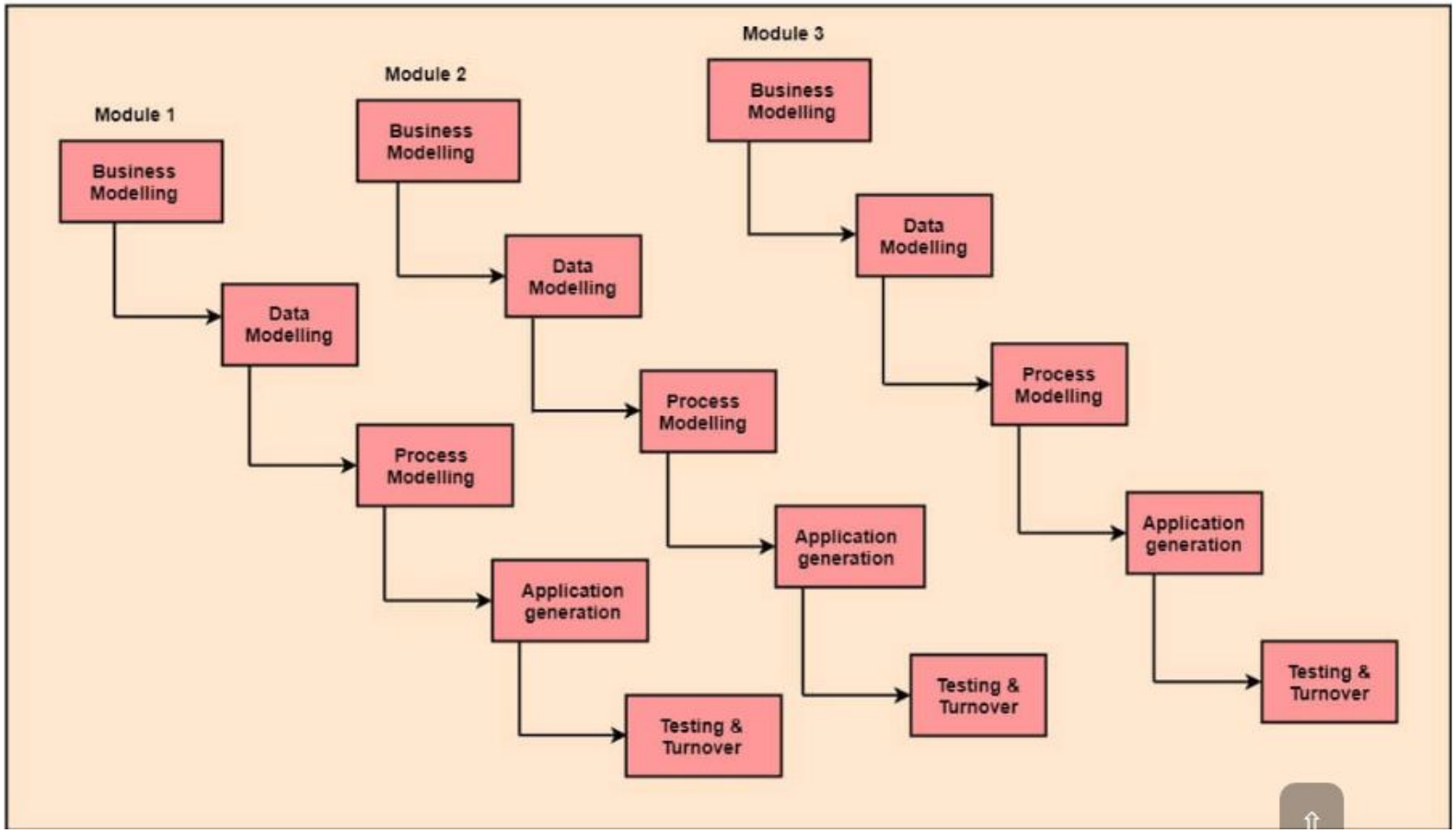
- Modeling
- Modeling establishes design representations that serve as the basis for RAD's construction activity

  - Business Modeling-It includes information flow among different functions in the project e.g.,what information will be produced by each function,which functions handles that information etc.
  - Data Modeling-It includes different data objects used in software and reltionship among different objects
  - Process Modeling-During process modeling,process descriptions are created e.g add,modify,delete etc

RAD Model

- Construction- It emphasizes the use of preexisting software components and the application of automatic code generation
- Deployment- It establishes a basis for subsequent iterations,if required

The RAD Model



Merits/Advantages

- Using the RAD approach,a product can be developed within very short period of time
- It supports increased reusability of the components
- It results in minimal code writing as it supports automatic code generation
- It encourages the customer feedback
- In this approach,quick initial reviews are possible
- Module integration is done from the beginning thus resolving number of integration issues

Drawbacks

- For large,but scalable projects,RAD requires sufficient human resources to create the right number of RAD teams
- If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated time frame,RAD projects will fail
- If a system cannot be properly modularized,building the components necessary for RAD will be problematic
- RAD may not be appropriate when technical risks are high(e.g.,when a new application makes a heavy use of new technology)

Evolutionary process models

- Prototyping
- The Spiral Model
- The Concurrent Development Model

The Prototyping Paradigm

- It assists the software engineer and the customer to better understand what is to be built when requirements are fuzzy-not clear
- The prototyping paradigm begins with communication
- The software engineer and customer meet and define the overall objectives for the software,identify whatever requirements are known
- A prototyping iteration is planned quickly and modeling(in the form of a "quick design")occurs
- A quick design focuses on a representation of those aspects of the software that will be visible to the customer/end-user(e.g human interface layout or output display formats)
- The quick design leads to the construction of a prototype

The Prototyping Paradigm

- The prototype is deployed and then evaluated by the customer/user
- Feedback is used to refine requirements for the software
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer,while at the same time enabling the developer to better understand what needs to be done
- Ideally,the prototype serves as a mechnism for identifying software requirements

The Prototyping Paradigm

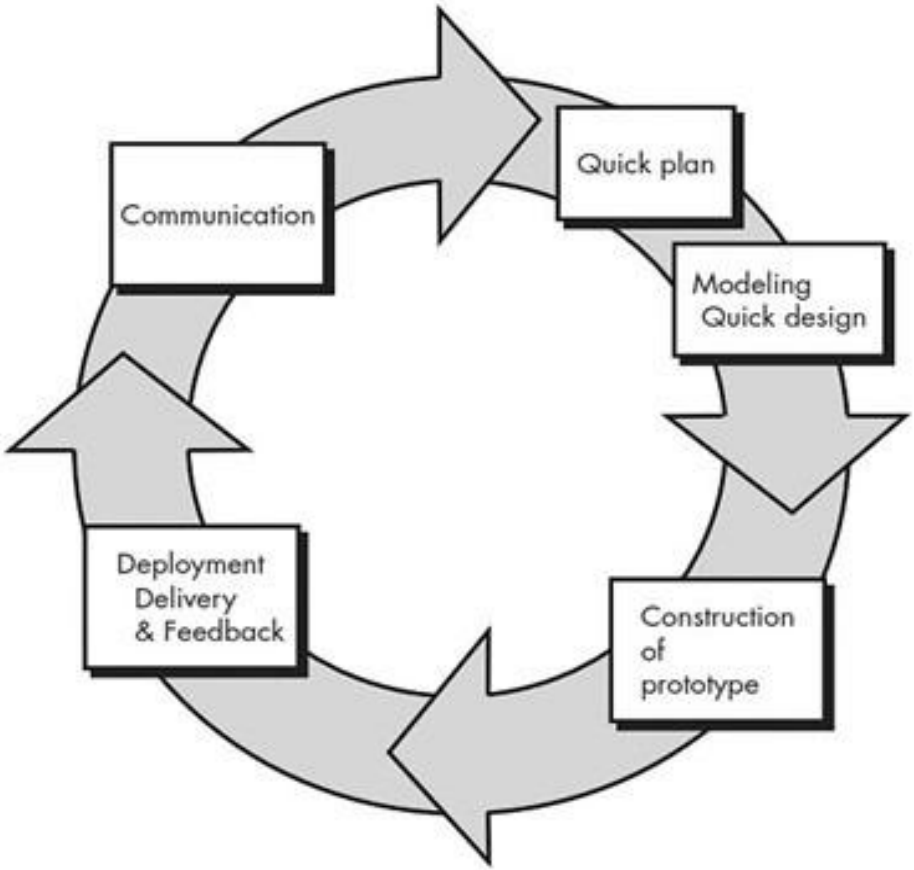
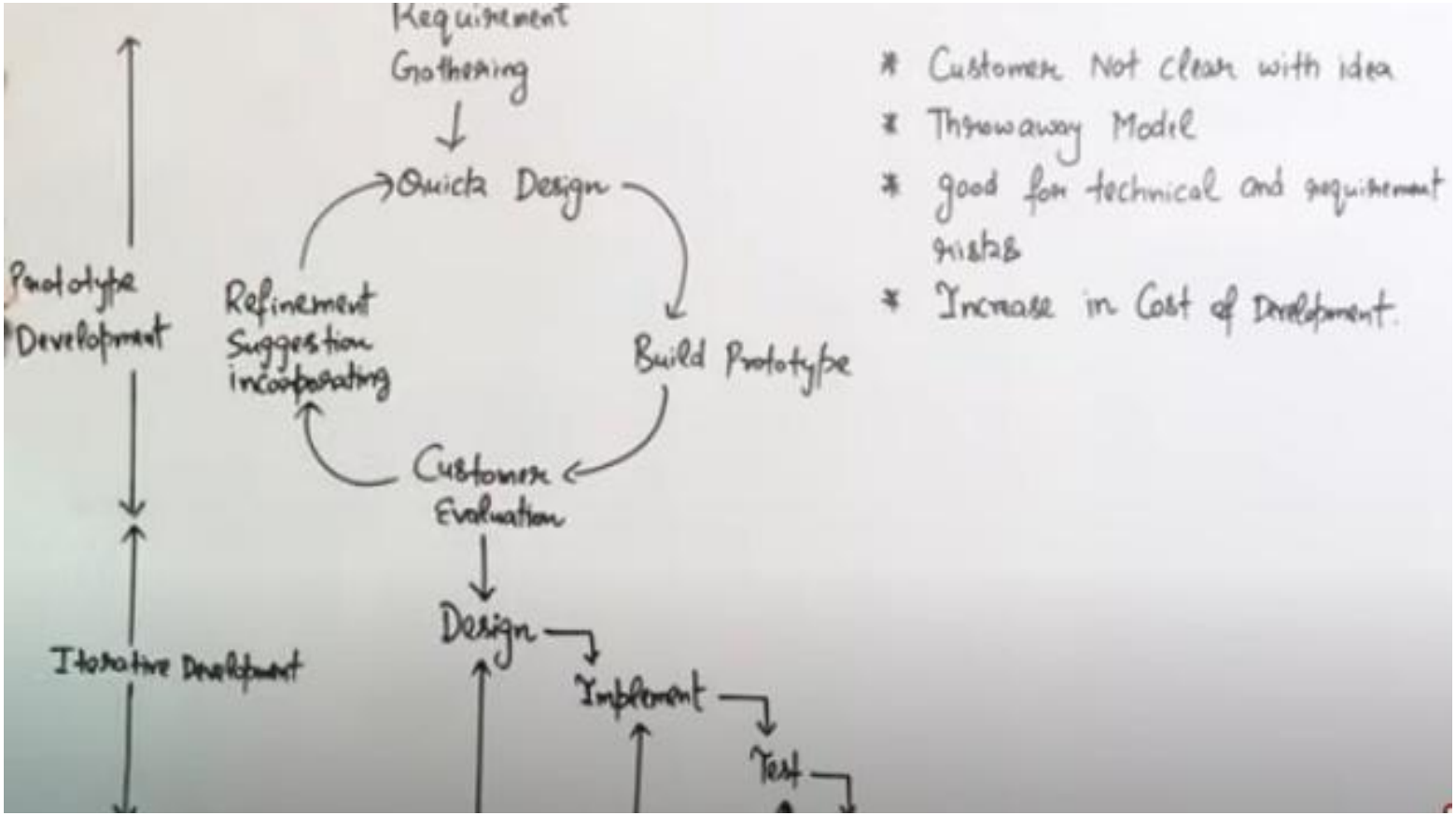


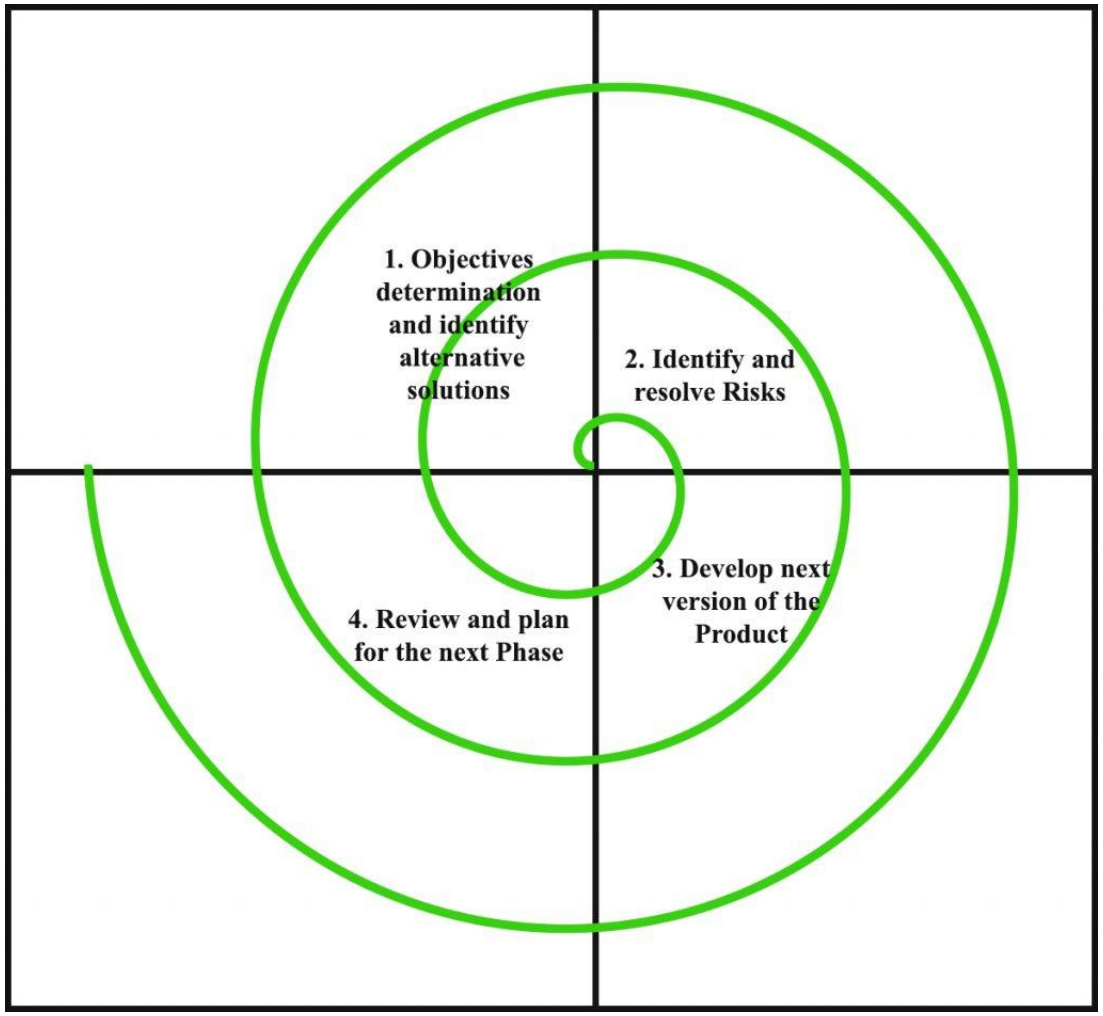
Figure: Prototype Model

The Prototyping Paradigm



*

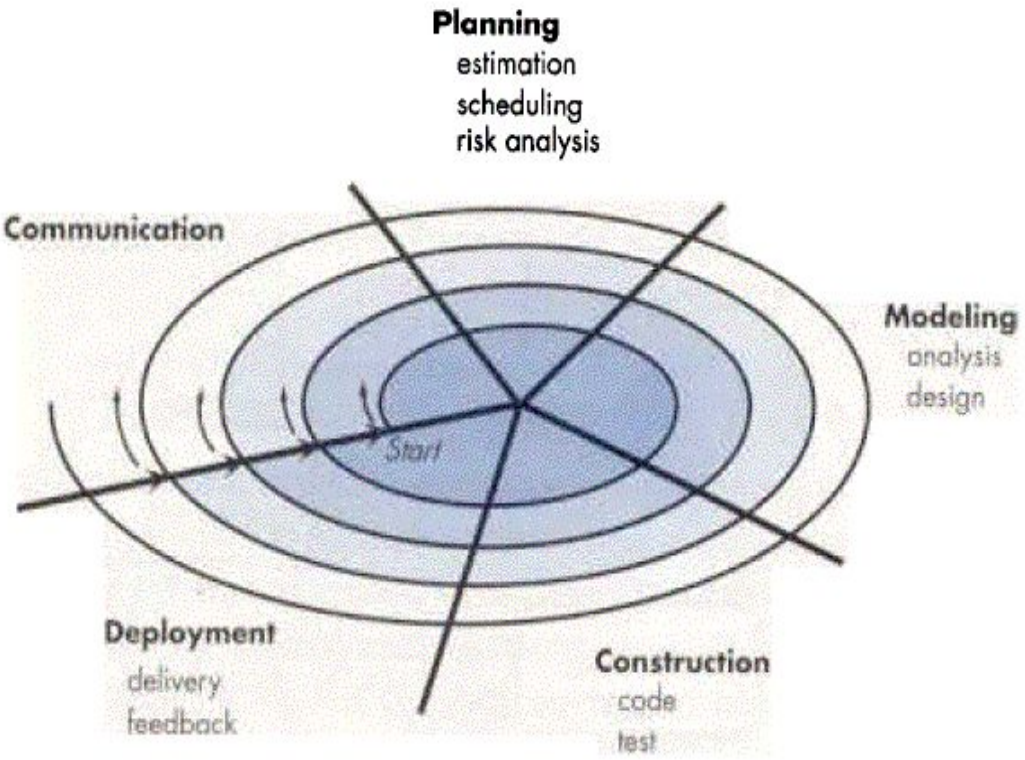
Spiral Model



Spiral Model

FIGURE 3.5

A typical spiral model



*

Spiral Model

- The spiral model,originally proposed by Boehm,is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model
- It provides the potential for rapid development of increasingly more complete versions of the software
- The software is developed as series of evolutionary releases
- During the initial releases,it may be just paperwork or prototype.But during later releases the version goes towards more completed stage

Spiral Model

- A spiral model is divided into a set of the framework activities defined by the software engineering team
- Each of the framework activities represent one segment of the spiral path as shown in fig.
- As this evolutionary process begins,the software team performs activities that are implied by a circuit around the spiral in a clockwise direction,beginning at the center

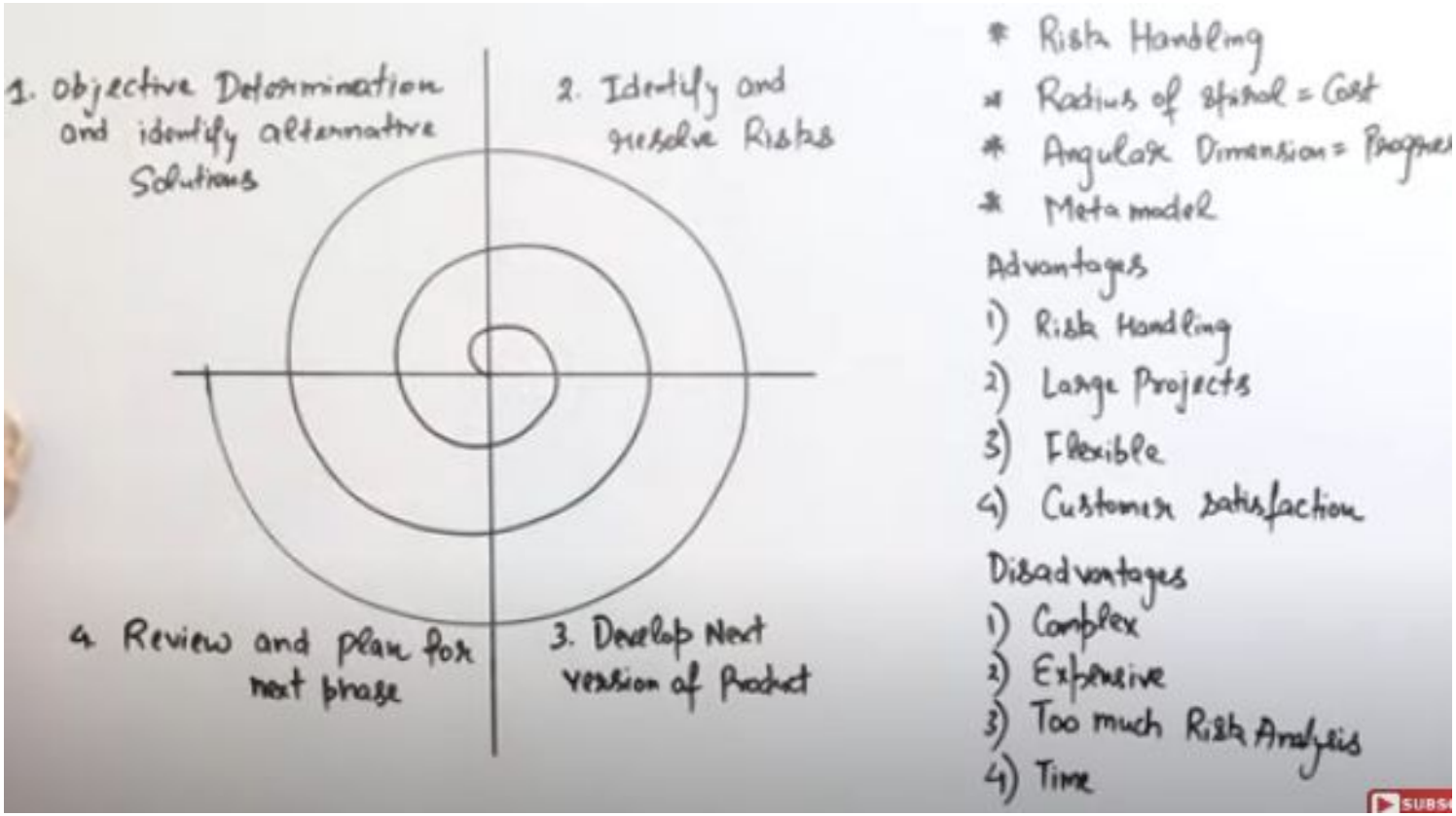
Spiral Model

- The first circuit around the spiral might result in the development of a product specification;subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software

Spiral Model

- The spiral model is a realistic approach to the development of large-scale systems and software
- It uses prototyping as a risk reduction mechanism
- It maintains the systematic stepwise approach suggeted by the classis life cycle but incorporates it into an iterative framework that more realistically reflects the real world

Spiral Model



*

Difference between Waterfall Model and Spiral Model

- Waterfall model works in sequential method.
- In waterfall model errors or risks are identified and rectified after the completion of stages.
- Waterfall model is applicable for small project.
- In waterfall model requirements and early stage planning is necessary.
- Flexibility to change in waterfall model is Difficult.

- While spiral model works in evolutionary method.
- In spiral model errors or risks are identified and rectified earlier.
- While Spiral model is used for large project.
- While in spiral model requirements and early stage planning is necessary if required.
- Flexibility to change in spiral model is not Difficult.

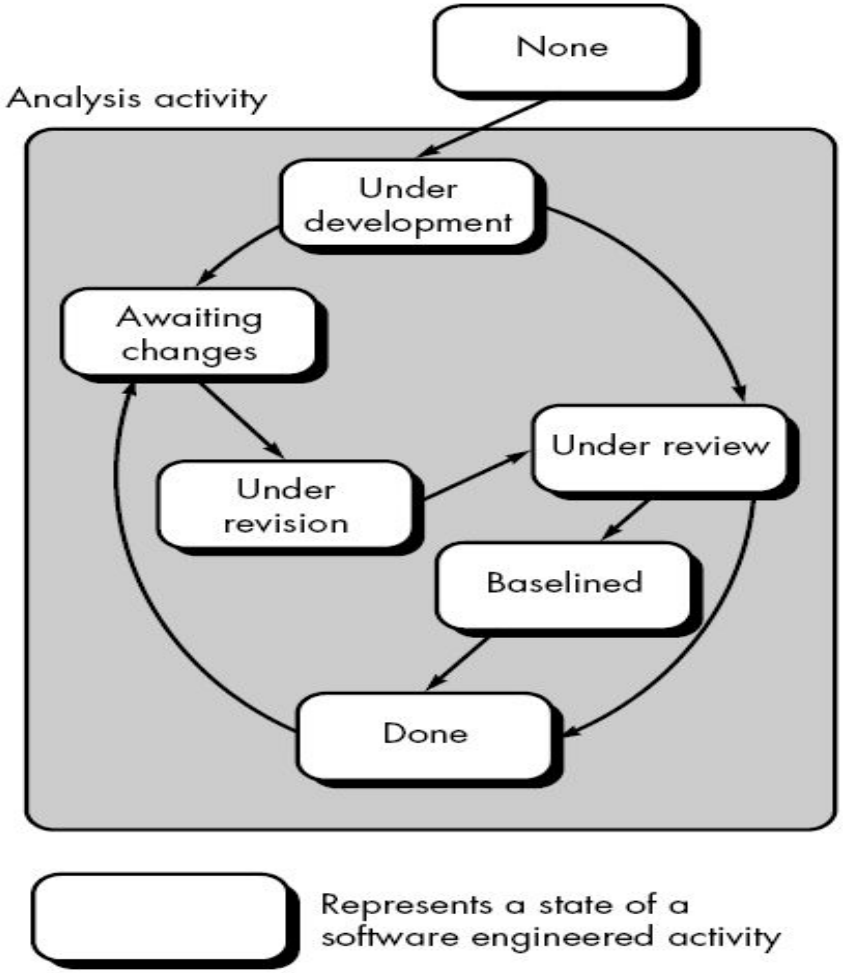
Difference between Waterfall Model and Spiral Model

- There is high amount risk in waterfall model.
- Waterfall model is comparatively inexpensive.
- There is low amount risk in spiral model.
- While cost of spiral model is very expensive.

The Concurrent Development Model

- The concurrent development model,sometimes called concurrent engineering,can be represented as a series of framework activities,software engineering actions and tasks,and their associated states
- This model is more appopriate for system engineering projects where different engineering teams are involved

Fig shows one element of the concurrent process model i.e. modeling activity



The Concurrent Development Model

- The modeling activity is in one of the states and other activities like communication or construction can also be represented in an analogous manner
- Let the communication activity has completed its first iteration and available in awaiting changes state
- The modeling activity has completed its initial communication and ready to move to under development state from none state

The Concurrent Development Model

- During these transitions,if customer indicates some modification in the requirement,then the modelling activity transits to awaiting changes state from under development state
- It defines network of activity instead of considering activities,actions and tasks as sequence of events
- This model is applicable to all types of software development and provides an accurate picture of the current state of a project

The Concurrent Development Model

- A baseline is a fixed point of reference that is used for comparison purposes. In business, the success of a project or product is often measured against a baseline number for costs, sales, or any number of other variables. A project may exceed a baseline number or fail to meet it.

Concurrent Development Model: Contd…

Merits:

- **Applicable to all types of S/W development & provides an accurate picture of the current state of the project.**

Demerits:

- **Problem to Project planning. How many No of iterations are to be planned? Uncertainty…**
- **Process may fall in chaos if the evolutions occurs too fast without a period of relaxation. On the other hand if the speed is too slow productivity could be affected.**
- **S/W processes are focussed on flexibility & extendability, rather than on high quality.**

Specialized Process Models:

Specialized process models use many of the characteristics of one or more of the conventional models presented so far, however they tend to be applied when a narrowly defined software engineering approach is chosen. They include,

- Components based development
- The Formal Methods Model
- Aspect oriented software development

Components Based Development :

In this approach, Commercial Off-The-Shelf (COTS) S/W components, developed by vendors who offer them as products are used in the development of software.

These components provide targeted functionality with well-defined interfaces that enable the components to be integrated into the software

The component-based development model incorporated many of the characteristics of the spiral model.

Components Based Development :

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

Components Based Development :

- **Merits:**
  - Leads to software reuse, which provides number of benefits
    - 70% reduction in development cycle time
    - 84 % reduction in project cost
    - Productivity index goes up to 26.2 (Norm : 16.9)
- **Demerits:**
  - Component Library must be robust.
  - Performance may degrade

The Formal Methods Model:

- The formal methods model encompasses a set of activities that lead to formal mathematical specification of computer software.

- Formal methods enable a software engineer to specify,develop,and verify a computer based system by applying a rigorous,mathematical notation

- The main focus is on defect prevention rather than defect removal

- Example, Clean Room S/W Engineering (CRSE)

The Formal Methods Model:

- **Merits:**
  - Removes many of the problems that are difficult to remove using other S/W Engg. Paradigms.
  - Ambiguity, Incompleteness & Inconsistency can be discovered & corrected easily by using formal methods of mathematical analysis.
- **Demerits:**
  - Development is time consuming & expensive
  - Extensive training is required
  - Difficult to use with technically unsophisticated customers

Aspect Oriented Software Development (AOSD):

- A set of localized features, functions & information contents are used while building complex software.
- These localized s/w characteristics are modeled as components (e.g. OO classes) & then constructed within the context of a system architecture.
- Certain "concerns" (Customer required properties or areas of technical interest) span the entire architecture i.e. Cross cutting concerns like system security, fault tolerance etc.

Merits:

- It is similar to component based development for aspects

Demerits:

- Component Library must be robust.
- Performance may degrade

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

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

An Agile Process

- These models satisfy customer through early and continuous delivery
- It can accommodate changing requirements
- In this development process customer,business people and developer must work together daily
- The messages are conveyed orally i.e.face-to-face.Conversation is essential
- Technical excellence and good design is achieved

Principles of Agility

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome/Accommodate changing requirements, even late in development.
- Deliver working software frequently,in shorter time span.
- The customer,Business people and developers must work together daily throughout the project.

Principles of Agility

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Principles of Agility

- Continuous attention to technical excellence and good design enhances agility.
- There must be simplicity in development
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Models

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process
- Kanban

*

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of a set of stories(also called user stories)that describe required features and functionality for software to be bulit
  - Each story is written by the customer and is placed on an index card
  - The customer assigns a value(i.e. a priority) to the story based on the overall business value of the feature or function
  - Members of the XP team then assess each story and assign a cost-measured in development weeks-to it
  - If the story will require more than 3 development weeks,the customer is asked to split the story into smaller stories ,and assignment of value and cost occurs again
  - Stories are grouped to for a deliverable increment
  - A commitmentt is made on delivery date
  - After the first increment project velocity is used to help define subsequent delivery dates for other increments
  - Project velocity is the number of customer stories implemented during the first release

Extreme Programming (XP)

- XP Design
  - Follows the KIS(Keep it simple) principle
  - A simple design is always preferred over a more complex represention
  - Design provides implementation guidance for a story as it is written
  - Encourage the use of CRC(class-responsibility collaborator) cards
  - CRC cards is an effective mechanism for thinking about the software in an object-oriented context
  - CRC cards identify and organize the object-oriented classes that are relevant to the current software increment
  - For difficult design problems, suggests the creation of spike solutions — a design prototype-the immediate creation of an operational prototype of that portion of the design
  - Encourages refactoring — an iterative refinement of the internal program design
  - Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure

Extreme Programming (XP)

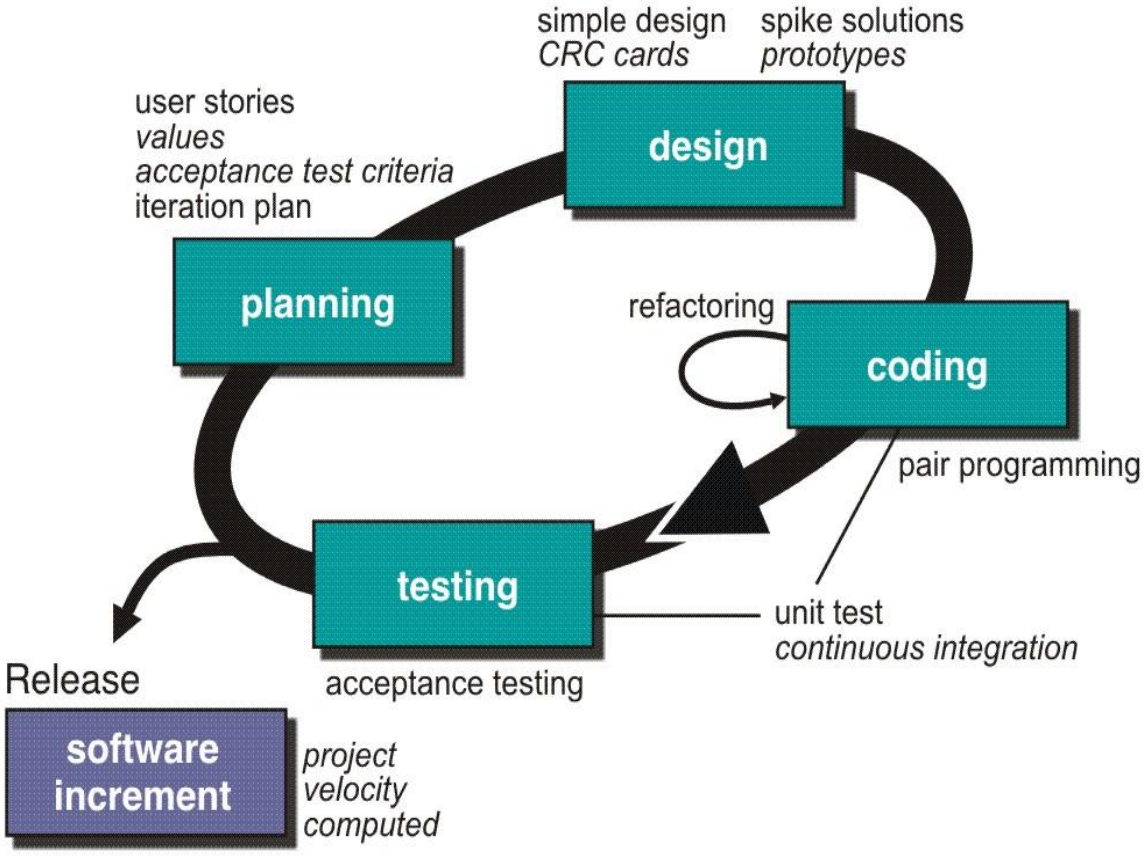
- XP Coding
  - –Recommends the construction of a unit test for a store *before* coding commences
  - –Once the unit test has been created,the developer is better able to focus on what must be implemented to pass the unit test
  - –Once the unit test has been created,the developer is better able to focus on what must be implemented to pass the unit test
  - –Encourages pair programming-It recommends that two people work together at one computer workstation to create code for a story
  - –For example,one person might think about the coding details of a particular portion of the design while the other ensures that coding standards are being followed

Extreme Programming (XP)

- XP Testing
  - All unit tests are executed daily
  - Acceptance tests, are called customer tests,defined by the customer and focus on overall system features and functionality that are visible and reviewable by the customer

*

Extreme Programming (XP)



Scrum

- Scrum is an agile process model developed by Jeff Sutherland and his team in the early 1990s
- Scrum principles
  - Small working teams are organized to "maximize communication,minimize overhead,and maximize sharing of tacit,informal knowledge."
  - The process must be adaptable to both technical and business changes "to ensure the best possible product is produced."
  - The process yields frequent software increments "that can be inspected,adjusted,tested,documented,and built on."

Scrum

- Development work and the people who perform it are partitioned "into clean,low coupling partitions, or packets."
- Constant testing and documentation is performed as the product is built
- It provides the "ability to declare a product 'done' whenever required

Scrum

- Scrum process pattern development activities
  - Backlog-a prioritized list of project requirements or features that provide business value for the customer.Items can be added to the backlog at any time(this is how changes are introduced.The product manager assesses the backlog and updates priorities as required
  - Sprints-consist of work units that are required to achieve a requirement defined in the backlog that must fit into a predefined time-box(typically 30 days).During the sprint,the backlog items that the sprint work units address are frozen(i.e.,changes are not introduced during the sprint).

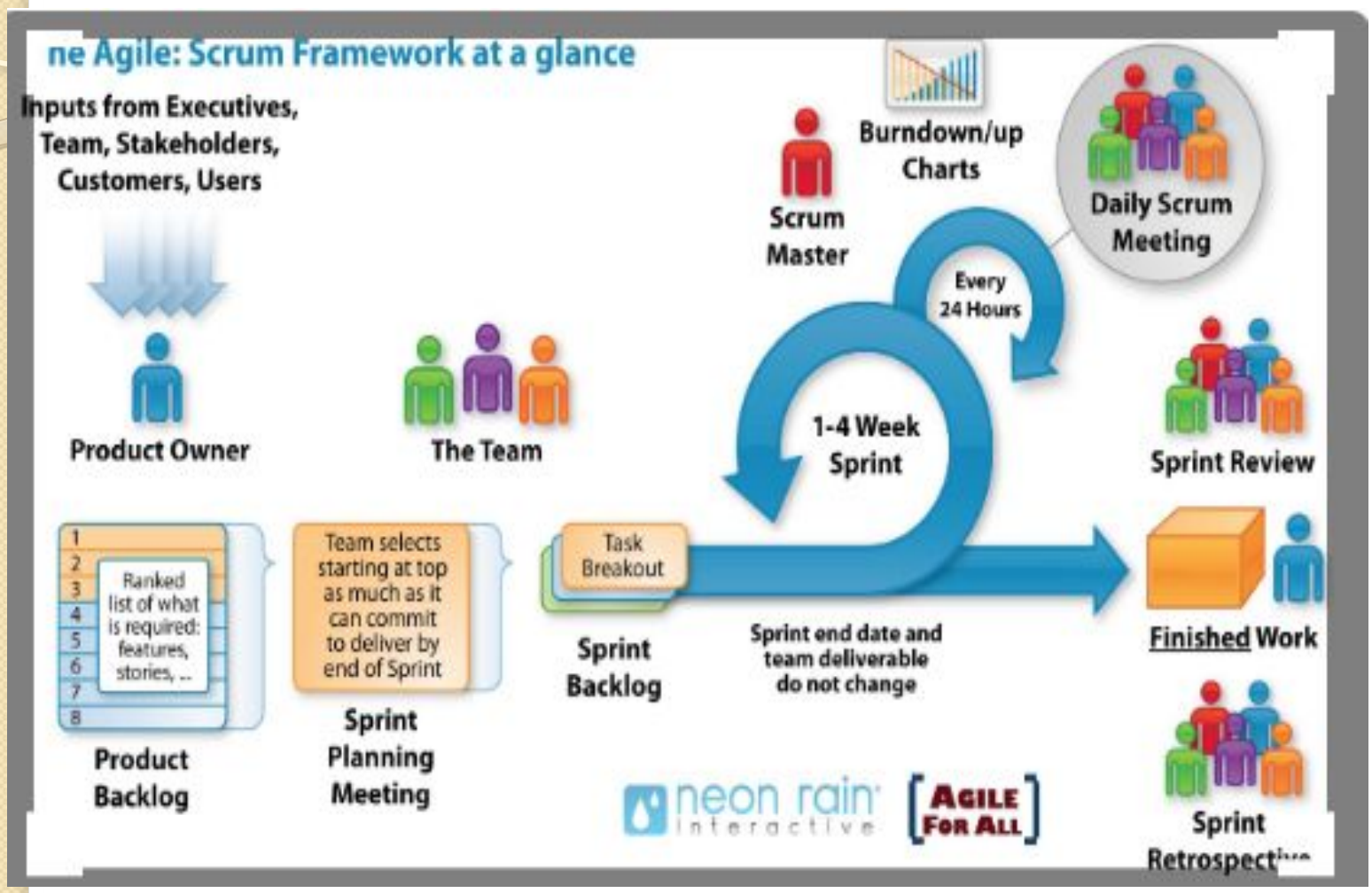
Scrum

- Scrum meetings-are short(typically15 minutes)meetings held daily by the scrum team
- 3 key questions are asked and answered by all team members
  - What did you do since the last team meeting?
  - What obstacles are you encountering?
  - What do you plan to accomplish by the next team meeting?
- A team leader,called a "Scrum master",leads the meeting and assesses the responses from each person

Scrum

- Demos-deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer

Scrum


ne Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
Ranked list of what is required: features, stories, ..
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospect
neon rain interactive
AGILE FOR ALL

SCRUM PROCESS

24 h Scrum

15-minute Daily Scrum Meeting
Team members describe:
- What I've done since the last Scrum meeting
- What I plan to do before the next
- Issues I have that I need help to resolve

Sprint Backlog
Features assigned to sprint

Backlog
Items expanded by team

New functionality

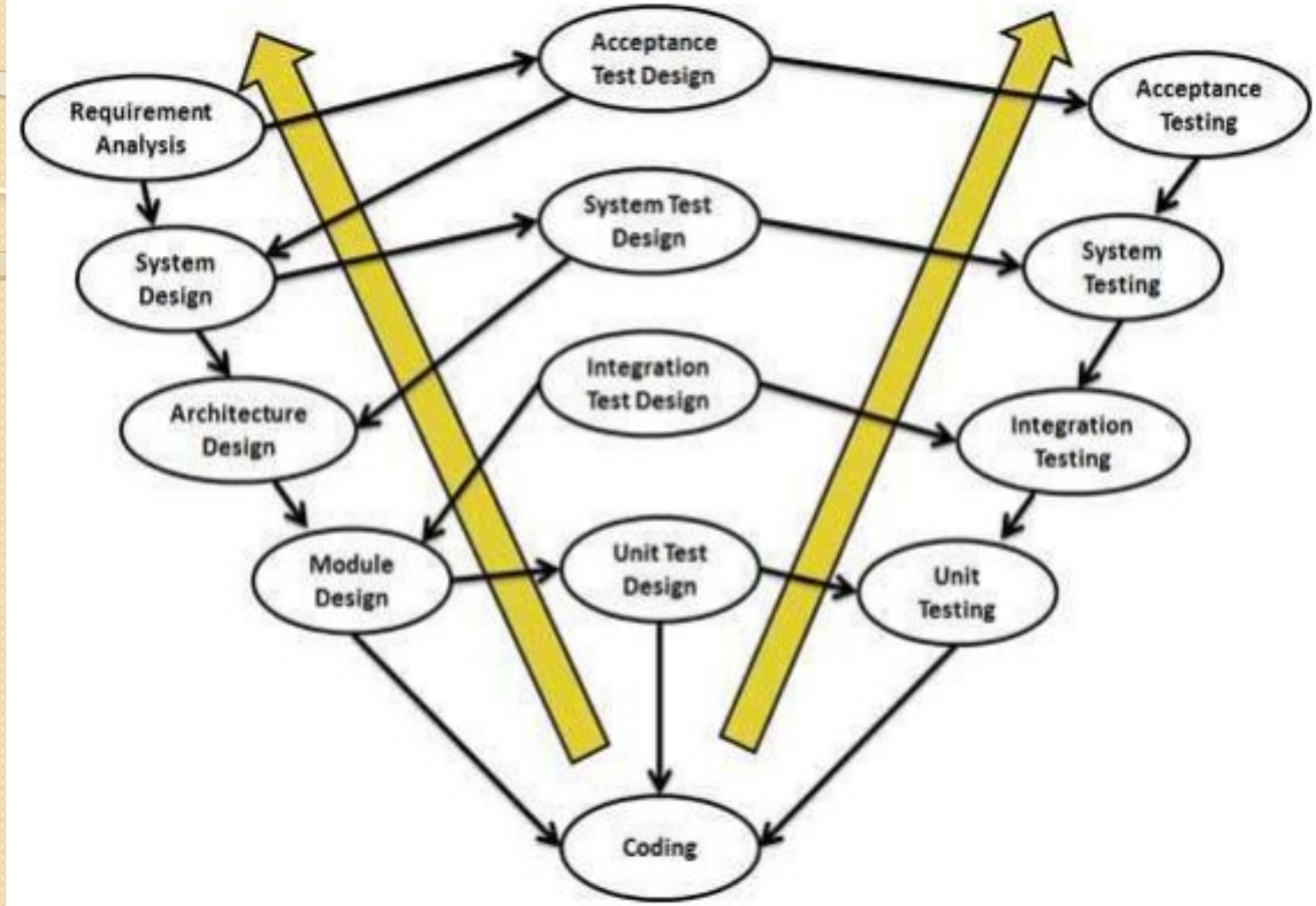
ProductBacklog
Prioritized product features desired by client

The V - model

The V - model is SDLC model where execution of processes happens in a sequential manner in V shape. It is also known as **Verification** and **Validation** model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.

This is a highly disciplined model and next phase starts only after completion of the previous phase.



- Verification phases on one side of the .V. and Validation phases on the other side

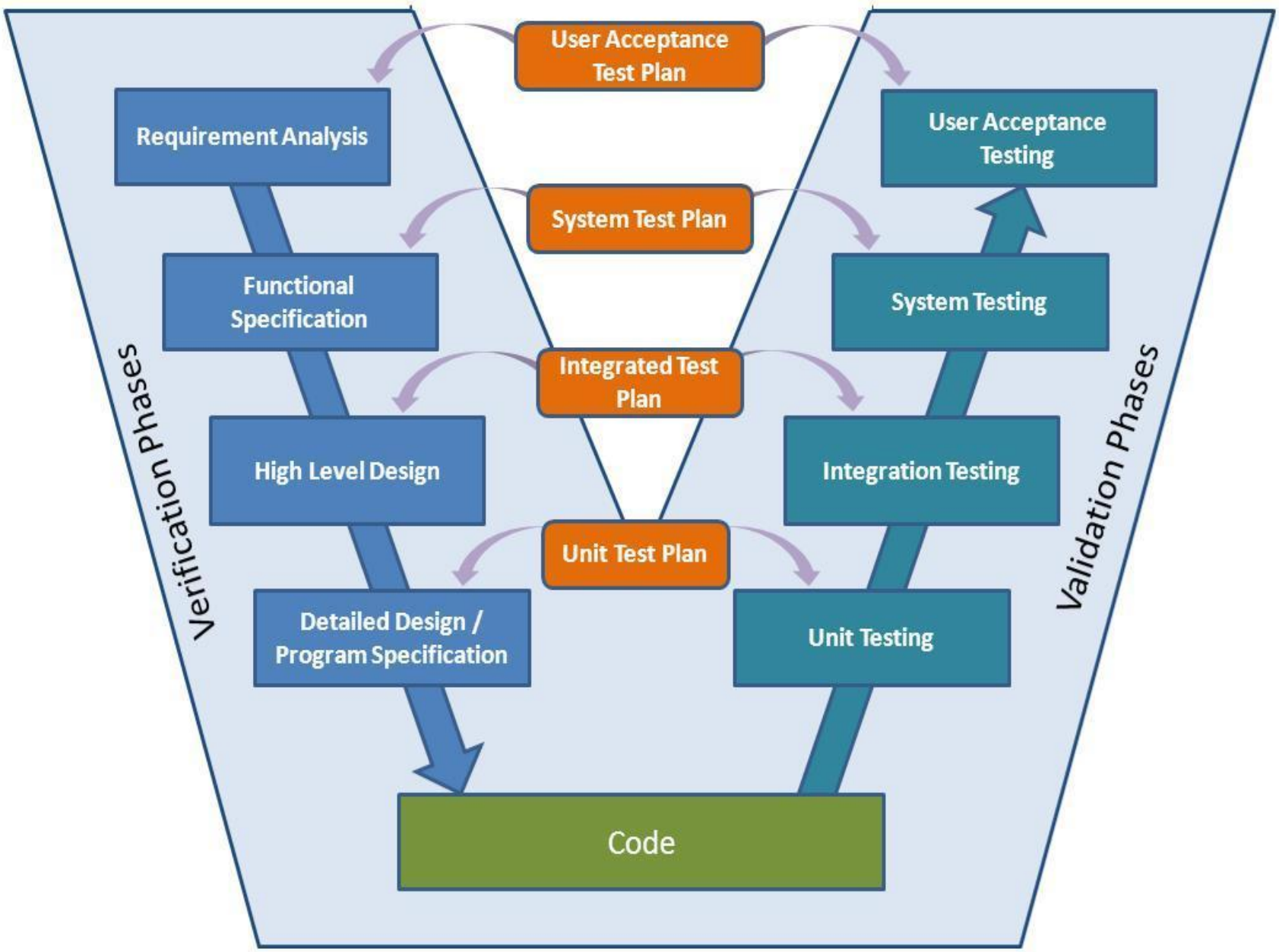
Verification Phases

- **Business Requirement Analysis**: This phase involves detailed communication with the customer to understand his expectations and exact requirement.
- **System Design :** System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development.
- **Architectural Design(Detail Design) :** The data transfer and communication between the internal modules and with the outside world other systems is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Verification Phases

- **Module Design : I**n this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design LLD.

V- Model


User Acceptance Test Plan
Requirement Analysis
User Acceptance Testing
System Test Plan
Functional Specification
System Testing
Integrated Test Plan
High Level Design
Integration Testing
Unit Test Plan
Detailed Design / Program Specification
Unit Testing
Code
Verification Phases
Validation Phases

Advantages of the V-Model

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Each phase has specific deliverables and a review process.

Disadvantages of the V-Model

- High 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.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

Kanban Model

- "Kanban" is a word of Japanese origin. It consists of two words: "Kan" – visible; and "Ban" – board.
- The Kanban board is a way of your project representation that allows tracking the current status.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner

Kanban Model

- 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.

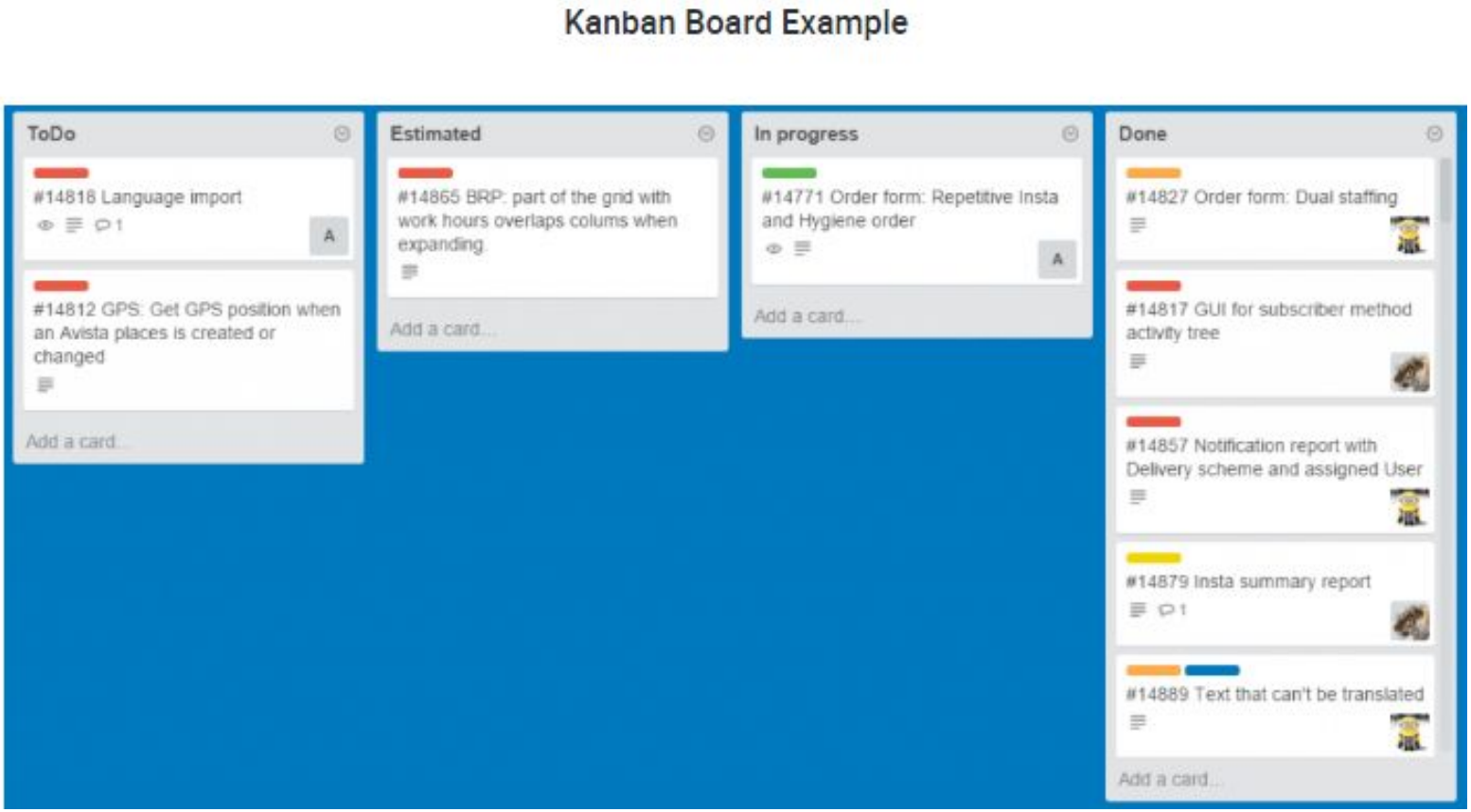
Kanban Board

- Kanban board 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.

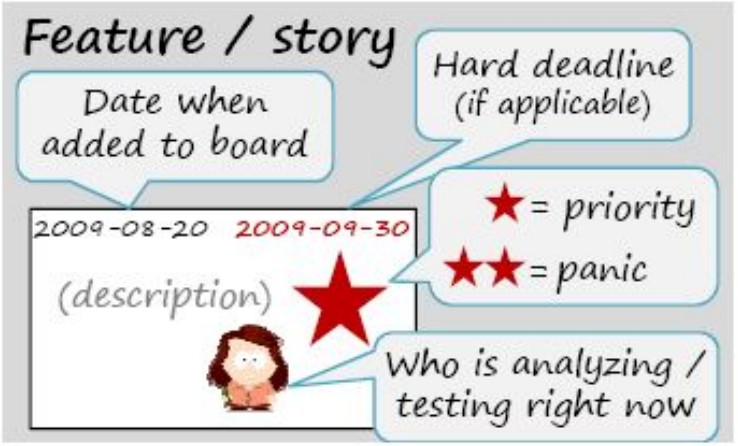
Kanban Board Example



Kanban (contd…)

Visualize the workflow

- Split the work into pieces, write each item on a card and put on the wall
- Use named columns to illustrate where each item is in the workflow



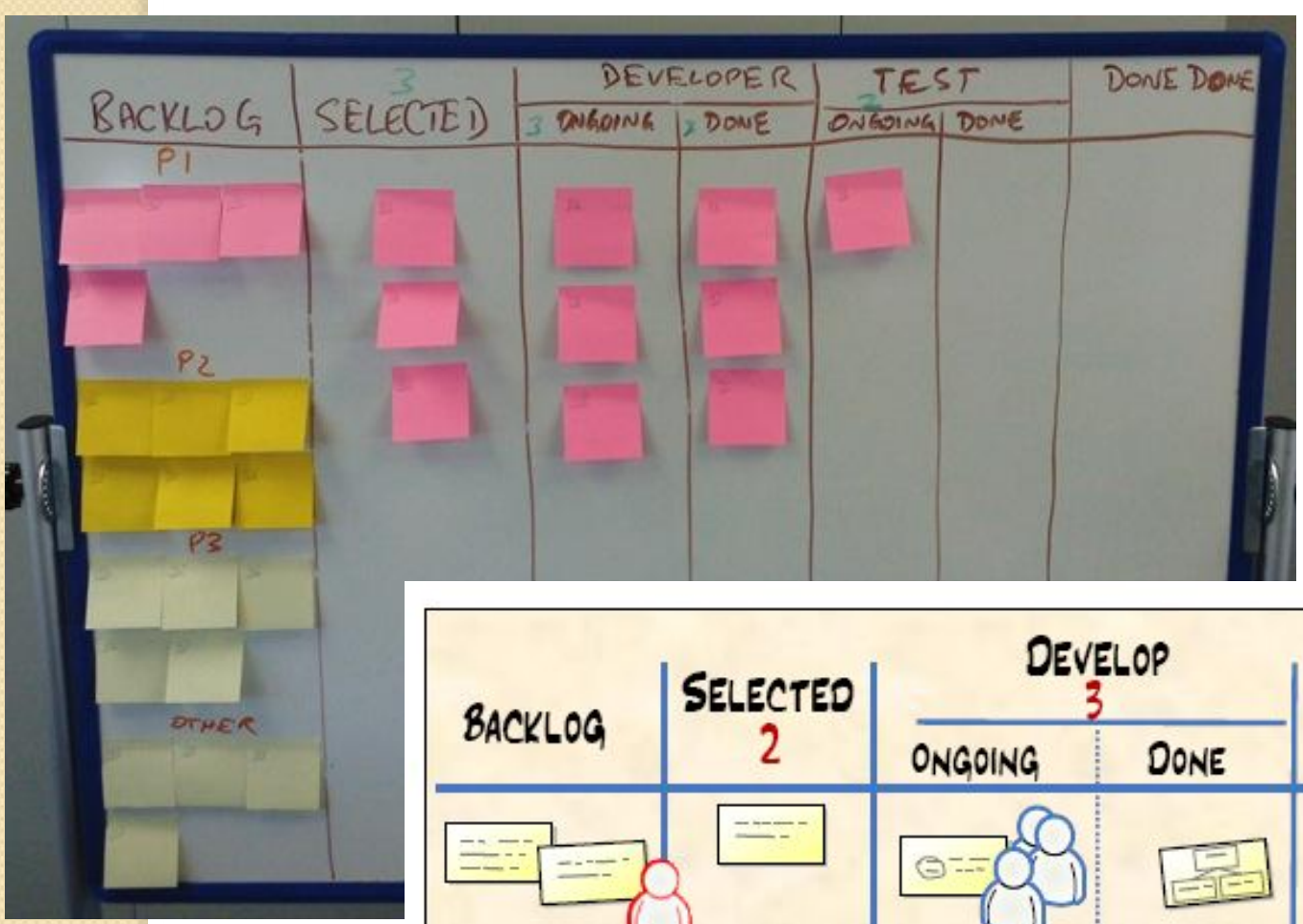
Limit WIP (work in progress)

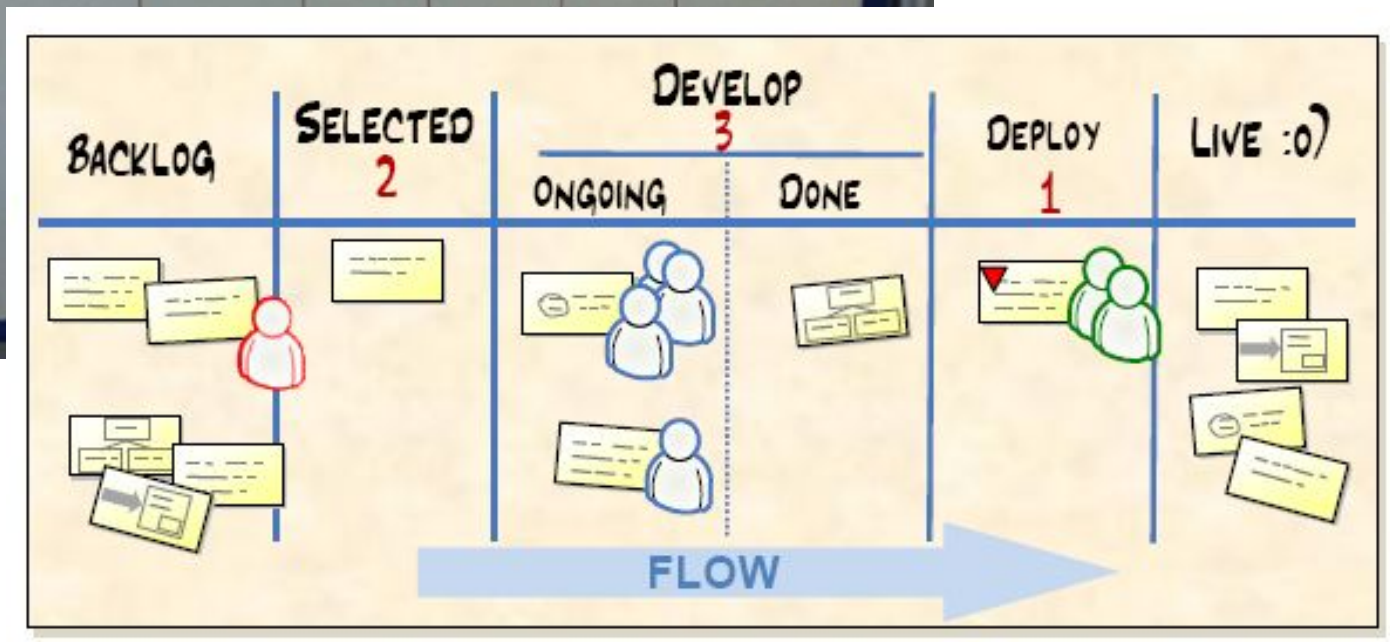
- Assign explicit limits to how many items may be in progress at each stage

Measure the lead time (average time to complete one item, sometimes called “cycle time”)

- Optimize the process to make lead time as small and predictable as possible

Kanban Board Illustration - I





Kanban Board

- every task has its card. In this case, the board was divided into four parts:
- ToDo section contains tasks that were received from the customer and required to be analyzed. Each task is marked with color according to its priority.
- When tasks from the first section have been analyzed and estimated by the Team, they are moved to the Estimated section.
- When the developer takes a task to be developed, he moves it from Estimated to In Progress section and marks it with his tag to show who handles each task.
- When a task is done, it's moved to the Done section.

Kanban Model

- WIP Limit
- The label in the Doing/In-process column also contains a number, which represents the maximum number of tasks that can be in that column at any point of time. i.e., the number associated with the **Doing** column is the WIP (Work-In-Progress) Limit.
- Pull Approach
- Pull approach is used as and when a task is completed in the Doing column. Another card is pulled from the To Do column.

CMM Models

- CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
- It is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.

CMM Models

- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

What is CMM?

- CMM: Capability Maturity Model
- Also called as SEI-CMM
- Developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University
- Framework that describes the key elements of an effective software process.

What is CMM?

- Describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one.
- Provides guidance on how to gain control of processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence.

Process Maturity Concepts

- Software Process
  - set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, user manuals)
- Software Process Capability
  - describes the range of expected results that can be achieved by following a software process
  - means of predicting the most likely outcomes to be expected from the next software project the organization undertakes

Process Maturity Concepts

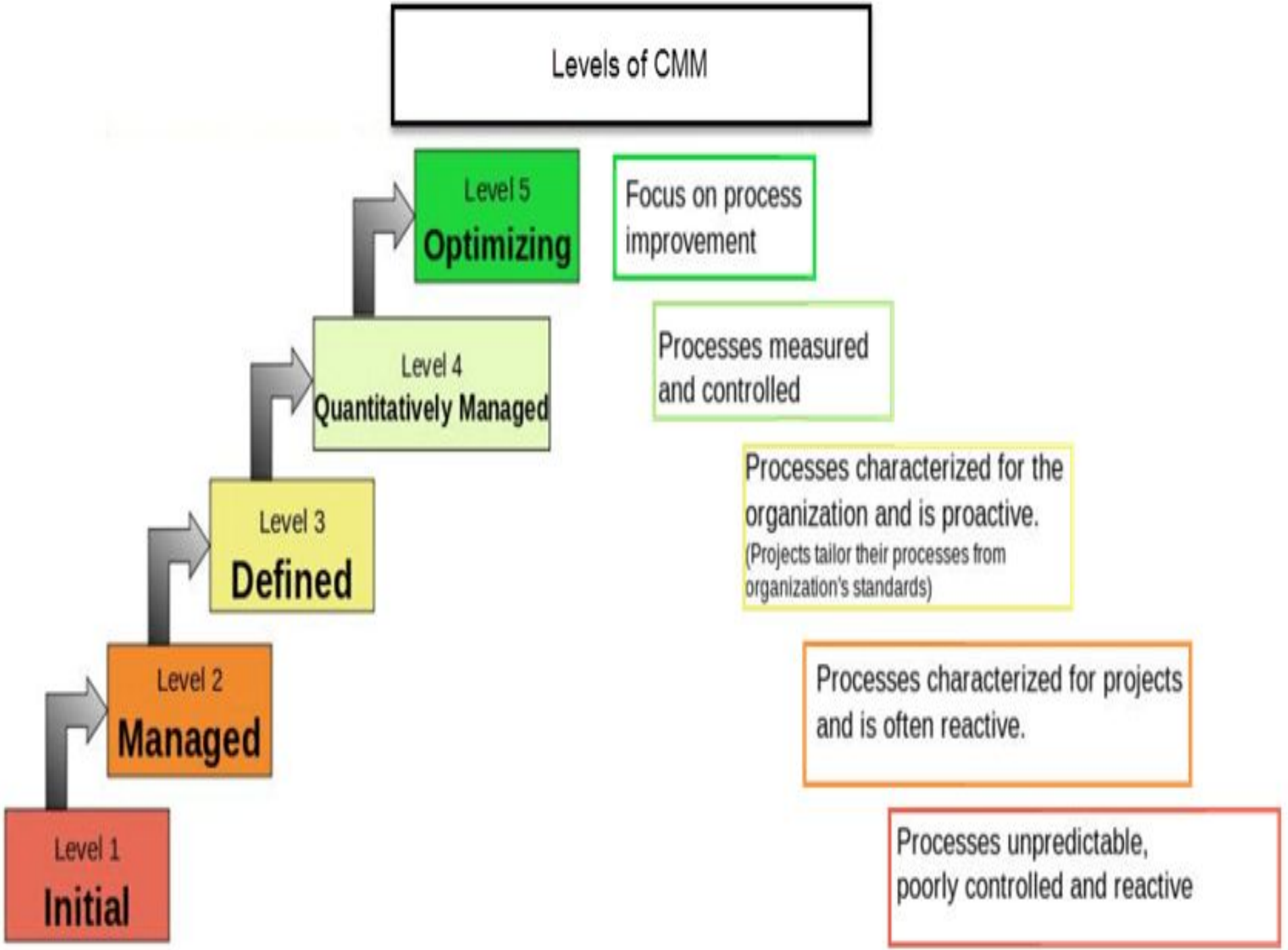
- Software Process Performance
  - actual results achieved by following a software process
- Software Process Maturity
  - extent to which a specific process is explicitly defined, managed, measured, controlled and effective
  - implies potential growth in capability
  - indicates richness of process and consistency with which it is applied in projects throughout the organization

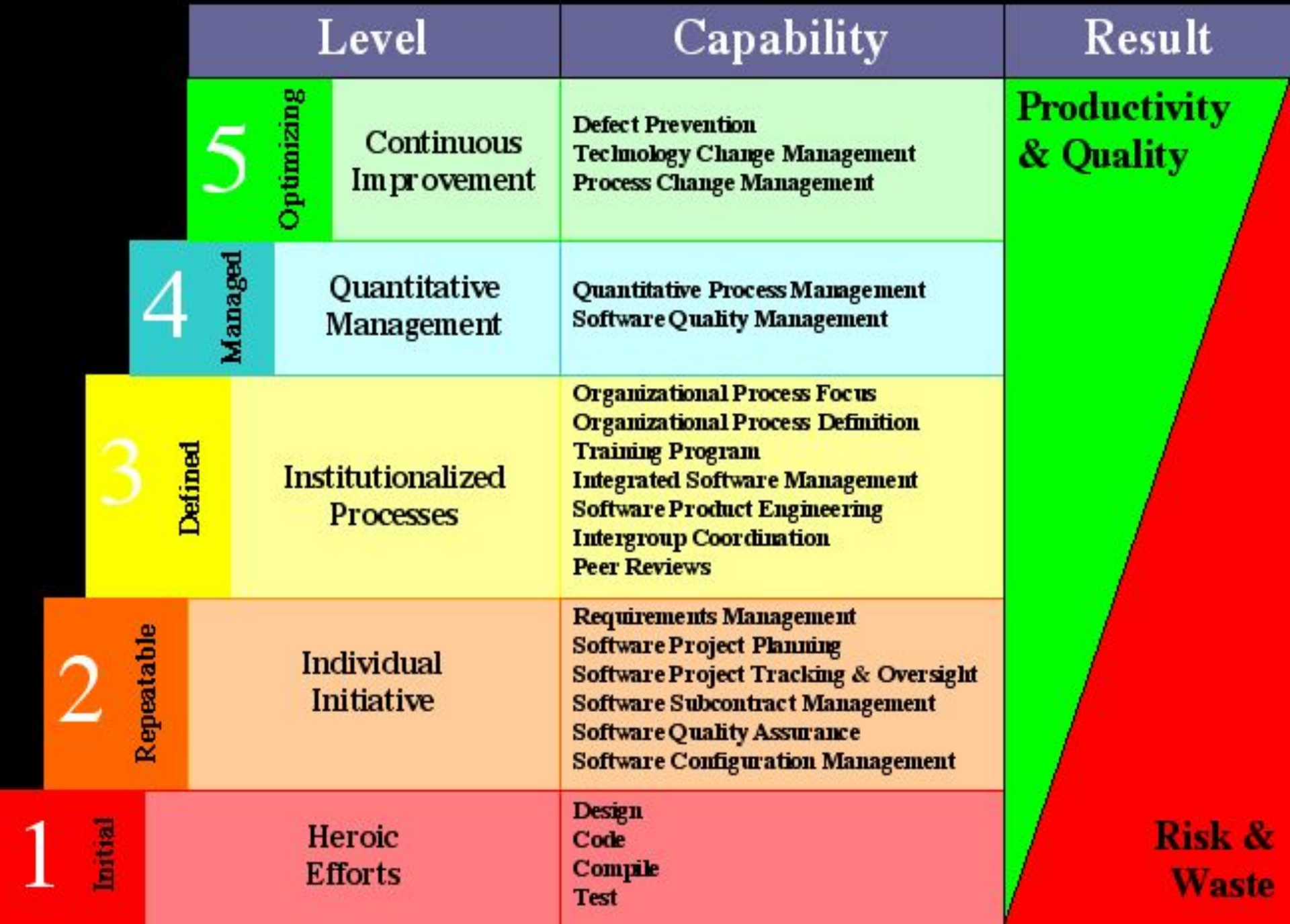
What are the CMM Levels?

(The five levels of software process maturity)

Maturity level indicates level of process capability:

- Initial
- Repeatable
- Defined
- Managed
- Optimizing


Levels of CMM
Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive


Level
Capability
Result
5 Optimizing
Continuous Improvement
Defect Prevention
Technology Change Management
Process Change Management
4 Managed
Quantitative Management
Quantitative Process Management
Software Quality Management
3 Defined
Institutionalized Processes
Organizational Process Focus
Organizational Process Definition
Training Program
Integrated Software Management
Software Product Engineering
Intergroup Coordination
Peer Reviews
2 Repeatable
Individual Initiative
Requirements Management
Software Project Planning
Software Project Tracking & Oversight
Software Subcontract Management
Software Quality Assurance
Software Configuration Management
1 Initial
Heroic Efforts
Design
Code
Compile
Test
Productivity & Quality
Risk & Waste

The 5 levels of CMM are as follows:

- **Level-1: Initial –**
- No KPA's defined
- Processes followed are adhoc and immature and are not well defined
- Unstable environment for software dvelopment
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies
- Experience with earlier projects is used for managing new similar natured projects

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well defined integrated set of project specific software engineering and management processes.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

Level-5: Optimizing –

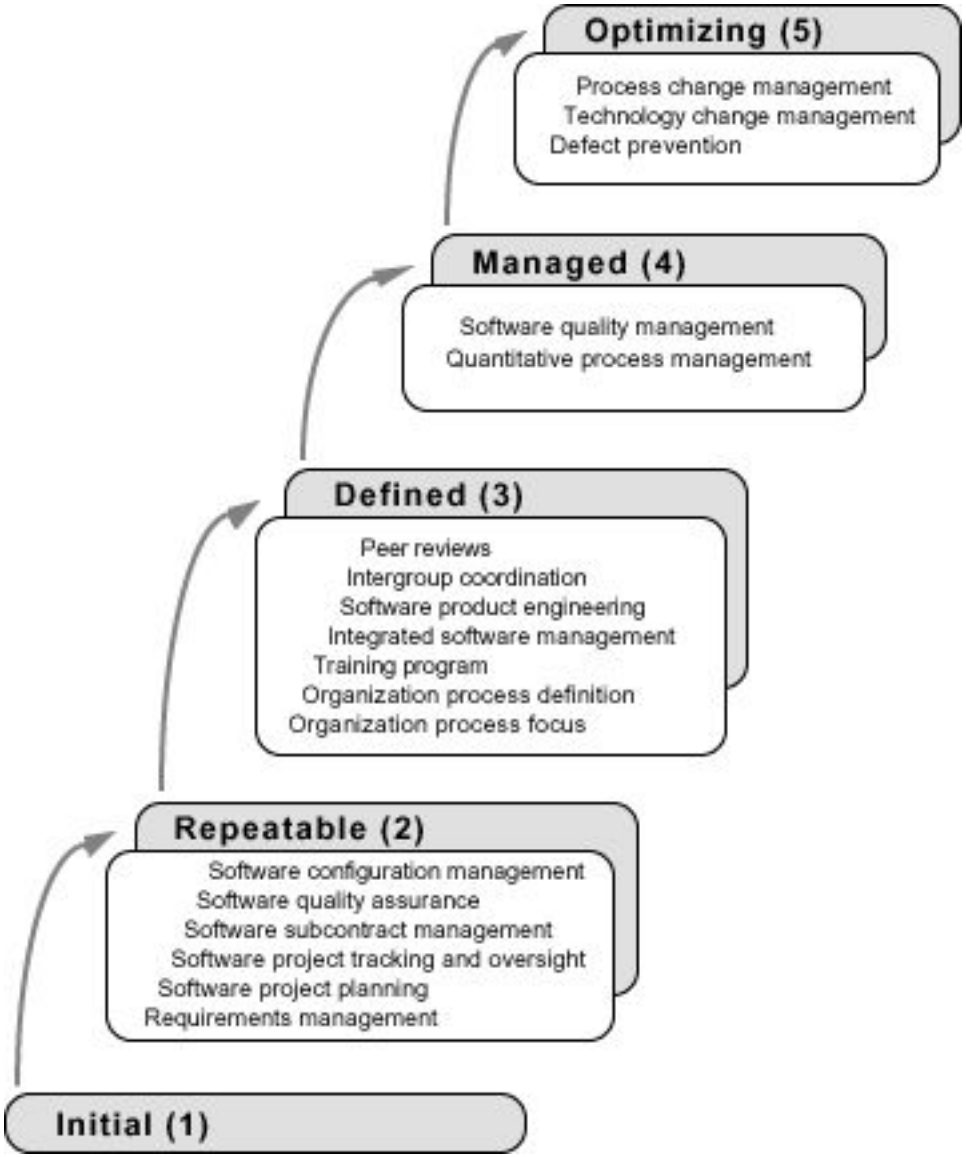
- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

Internal Structure to Maturity Levels

- Except for level 1, each level is decomposed into key process areas (KPA)
- Each KPA identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing software capability.
  - commitment
  - ability
  - activity
  - measurement
  - verification

Key Process Areas (KPA's):

- Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
- Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.



The Key Process Areas by Maturity Level

Repeatable

Level 2 KPAs

- Requirement Management
- Software Project Planning
- Software project tracking & Oversight
- Software Subcontract management
- Software Quality Assurance
- Software Configuration Management

Level 2 KPAs

- Requirements Management
  - Establish common understanding of customer requirements between the customer and the software project
  - Requirements is basis for planning and managing the software project
  - Not working backwards from a given release date!
- Software Project Planning
  - Establish reasonable plans for performing the software engineering activities and for managing the software project

Level 2 KPAs

- Software Project Tracking and Oversight
  - Establish adequate visibility into actual progress
  - Take effective actions when project's performance deviates significantly from planned
- Software Subcontract Management
  - Manage projects outsourced to subcontractors
- Software Quality Assurance
- **Software quality assurance** (SQA) is a means and practice of monitoring the **software** engineering processes and methods used in a project to ensure proper **quality** of the **software**.
  - Provide management with appropriate visibility into
    - process being used by the software projects
    - work products

Level 2 KPAs

- Software Configuration Management
- **software configuration management** (SCM or S/W **CM**) is the task of tracking and controlling changes in the **software**,
  - Establish and maintain the integrity of work products
  - Product baseline
  - Baseline authority

Defined

Level 3 KPAs

- Organization Process Focus
- Organization Process Definition
- Training Program
- Integrated Software Management
- Software Product Engineering
- Intergroup Coordination
- Peer Reviews

Level 3 KPAs

- Organization Process Focus
  - Establish organizational responsibility for software process activities that improve the organization's overall software process capability
- Organization Process Definition
  - Develop and maintain a usable set of software process assets
    - stable foundation that can be institutionalized
    - basis for defining meaningful data for quantitative process management

Level 3 KPAs

- Training Program
  - Develop skills and knowledge so that individual can perform their roles effectively and efficiently
  - Organizational responsibility
  - Needs identified by project
- Integrated Software Management
  - Integrated engineering and management activities
  - Engineering and management processes are tailored from the organizational standard processes
  - Tailoring based on business environment and project needs

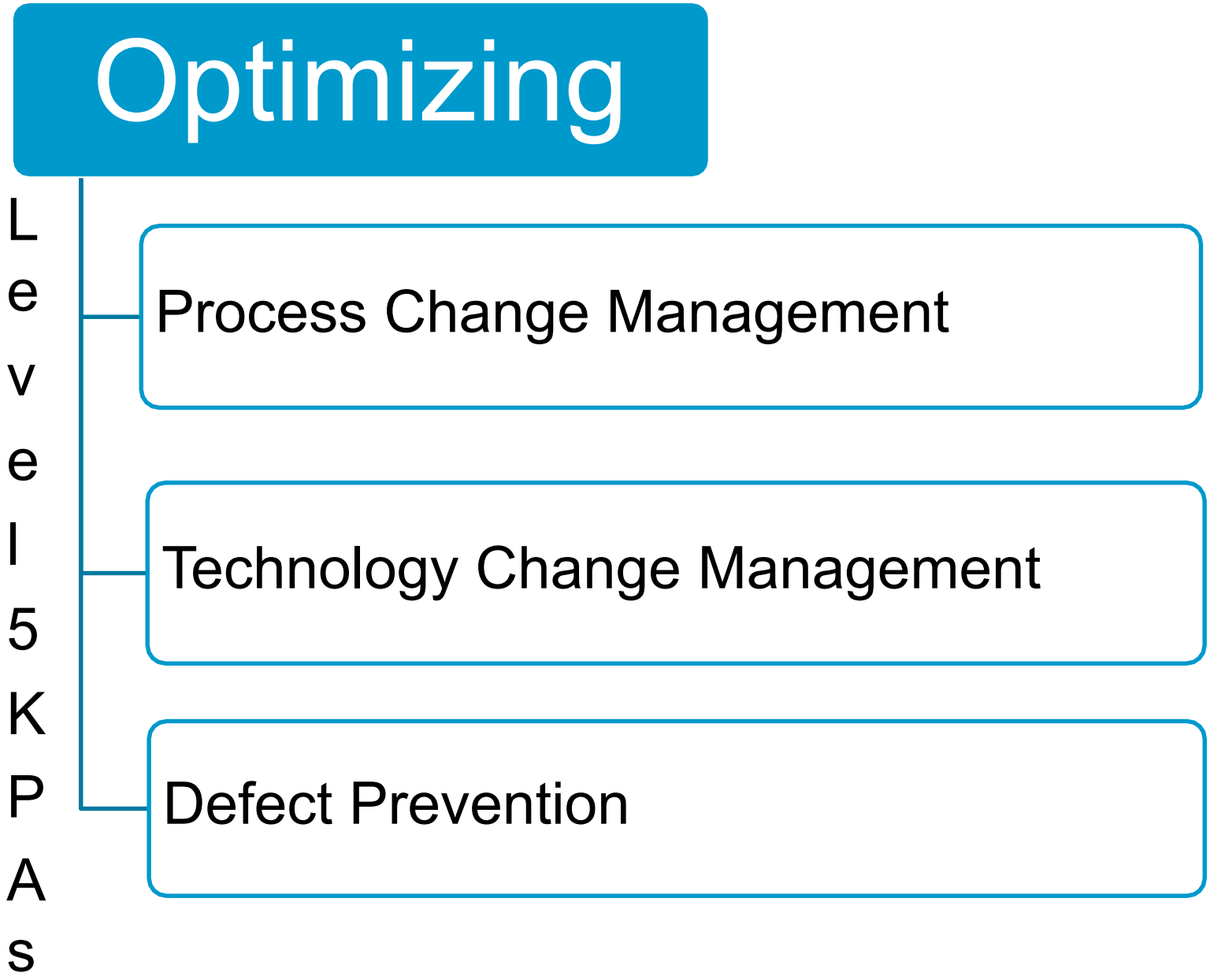
Level 3 KPAs

- Software Product Engineering
  - technical activities of the project are well defined (SDLC)
  - correct, consistent work products
- Intergroup Coordination
  - Software engineering groups participate actively with other groups
- Peer Reviews
  - early defect detection and removal
  - better understanding of the products
  - implemented with inspections, walkthroughs, etc


Managed
Level 4 KPAs
Quantitative Process Management
Software Quality Management

Level 4 KPAs

- Quantitative Process Management
  - control process performance quantitatively
  - actual results from following a software process
  - focus on identifying and correcting special causes of variation with respect to a baseline process
- Software Quality Management
  - quantitative understanding of software quality
    - products
    - process


Optimizing
Level 5 KPAs
Process Change Management
Technology Change Management
Defect Prevention

Level 5 KPAs

- Process Change Management
  - continuous process improvement to improve quality, increase productivity, decrease cycle time
- Technology Change Management
  - identify and transfer beneficial new technologies
    - tools
    - methods
    - processes
- Defect Prevention
  - causal analysis of defects to prevent recurrence