SOFTWARE PRODUCT AND PROCESS CHARACTERISTICS:

Software: -

Software is nothing but collection of computer programs and related documents that are planned to provide desired features, functionalities and better performance.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Characteristics of software: -

1. Software is developed or engineered; it is not manufactured in the classical sense:
   - Although some similarities exist between software development and hardware manufacturing, but few activities are fundamentally different.
   - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.
2. Software doesn't "wear out."
   - Hardware components suffer from the growing effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.
   - 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".
   - When a hardware component wears out, it is replaced by a spare part.
   - There are no software spare parts.
   - Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance.
   - However, the implication is clear—software doesn't wear out. But it does deteriorate.

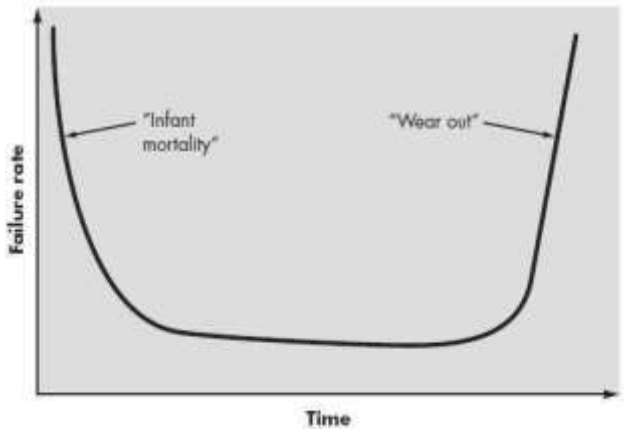


Figure 1.1 Hardware Failure Curve

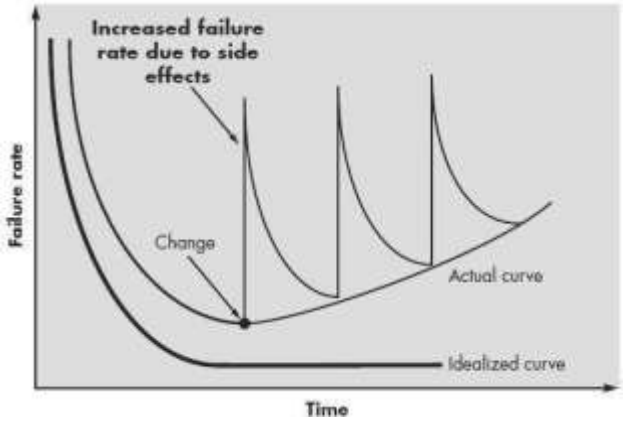


Figure 1.2 Software Failure Cure

3. Although the industry is moving toward component-based construction, most software continues to be custom built.

- A software component should be designed and implemented so that it can be reused.
- Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new application form reusable parts.
- In the hardware world, component reuse is a natural part of the engineering process

Good Software are-

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance
- Well-engineered and crafted software is expected to have the following characteristics:

Operational: -

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional: -

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance: -

This aspect briefs about how well software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

Software product classified in 2 classes:

2. **Generic software**: Developed to solution whose requirements are very common fairly stable and well understood by software engineer.
3. **Custom software**: Developed for a single customer according to their specification.

A Layered Technology:

Figure 1.3 Layed Architecture

A quality Focus:

- Every organization is rest on its commitment to quality.
- Total quality management, Six Sigma, or similar continuous improvement culture and it is this culture ultimately leads to development of increasingly more effective approaches to software engineering.
- The foundation that supports software engineering is a quality focus.

Process:

- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework that must be established for effective delivery of software engineering technology.
- The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products are produced, milestones are established, quality is ensured, and change is properly managed.

Methods:

- Software engineering methods provide the technical aspects for building software.
- Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
- Software engineering methods rely on the set of modeling activities and other descriptive techniques.

Tools:

- Software engineering tools provide automated or semi automated support for the process and the method.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called CASE (Computer- aided softeware engineering), is established.

SOFTWARE PROCESS MODEL:

Software process can be defining as the structured set of activates that are required to develop the software system.

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods, and tools layers. This strategy is often referred to as a process model or a software engineering paradigm.

A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Goal of Software Process Models: -

The goal of a software process model is to provide guidance for systematically coordinating and controlling the tasks that must be performed in order to achieve the end product and their project objectives. A process model defines the following:

- A set of tasks that need to be performed.

- The inputs to and output from each task.
- The preconditions and post-conditions for each task.
- The sequence and flow of these tasks.

Characteristics of Software Process: -

Software is often the single largest cost item in a computer-based application. Though software is a product, it is different from other physical products.

i. Software costs are concentrated in engineering (analysis and design) and not in production.
ii. Cost of software is not dependent on volume of production.
iii. Software does not wear out (in the physical sense).
iv. Software has no replacement (spare) parts.
v. Software maintenance is a difficult problem and is very different from hardware (physical product) maintenance.
vi. Most software is custom-built.
vii. Many legal issues are involved (e.g. inter-actual property rights, liability).

Software Product: -

A software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different. In case of a program, very little documentation is expected, but a software product must be well documented. A program can be developed according to the programmer's individual style of development, but a software product must be developed using the accepted software engineering principles.

Various Operational Characteristics of software are:

- **Correctness:** The software which we are making should meet all the specifications stated by the customer.
- **Usability/Learn-ability:** The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.
- **Integrity:** Just like medicines have side-effects, in the same way software may have aside-effect i.e. it may affect the working of another application. But quality software should not have side effects.
- **Reliability:** The software product should not have any defects. Not only this, it shouldn't fail while execution.
- **Efficiency:** This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.
- **Security:** With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data / hardware. Proper measures should be taken to keep data secure from external threats.
- **Safety:** The software should not be hazardous to the environment/life.

Difference between software process and software product: -

Table 1 Difference between software process and software product

Software Process	**Software Product**
Processes are developed by individual user and it is used for personal use.	It is developed by multiple users and it is used by large number of people or customers.
Process may be small in size and possessing limited functionality.	It consists of multiple program codes; relate documents such as SRS, designing documents, user manuals, test cases.
Process is generally developed by process engineers.	Process is generally developed by process engineers. Therefore systematic approach of developing software product must be applied.
Software product relies on software process for its stability quality and control Only one person uses the process, hence lack of user interface	It is important than software product. Multiuser no lack of user interface.

Software Development Life Cycle/Process model/ Software Development Life Cycle: -

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product. it is a team of engineers must incorporate a development strategy that encompasses the process, method and tools layers. Each phase has various activities to develop the software product. It also specifies the order in which each phase must be executed.

A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed.

Definition: Software Development Life Cycle (SDLC) is a process used by software industry to design, develop and test high quality software. The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is the acronym of Software Development Life Cycle. It is also called as Software development process. The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process.

LINEAR SEQUENTIAL MODEL:

A few of software development paradigms or process models are defined as follows:

Waterfall model or linear sequential model or classic life cycle model: -

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and maintenance.

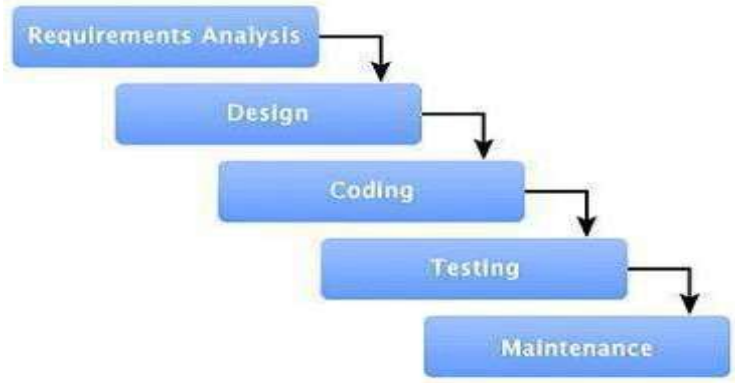


Figure 1.4 Waterfall model

Software requirements analysis: The requirements gathering process is focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design: Software design is actually a multi-step process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.

Coding : The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

Testing: Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

Maintenance: Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g., a change required

because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements. Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Advantages of waterfall model: -

- This model is simple and easy to understand and use.
- Waterfall model works well for smaller projects where requirements are very well understood.
- Each phase proceeds sequentially.
- Documentation is produced at every stage of the software's development. This makes understanding the product designing procedure, simpler.
- After every major stage of software coding, testing is done to check the correct running of the code. help us to control schedules and budgets.

Disadvantages of waterfall model: -

- 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.
- High amounts of risk and uncertainty.
- Customer can see working model of the project only at the end. after reviewing of the working model if the customer gets dissatisfied then it causes serious problem.
- You cannot go back a step if the design phase has gone wrong, things can get very complicated in the implementation phase.

PROTOTYPING MODEL:

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

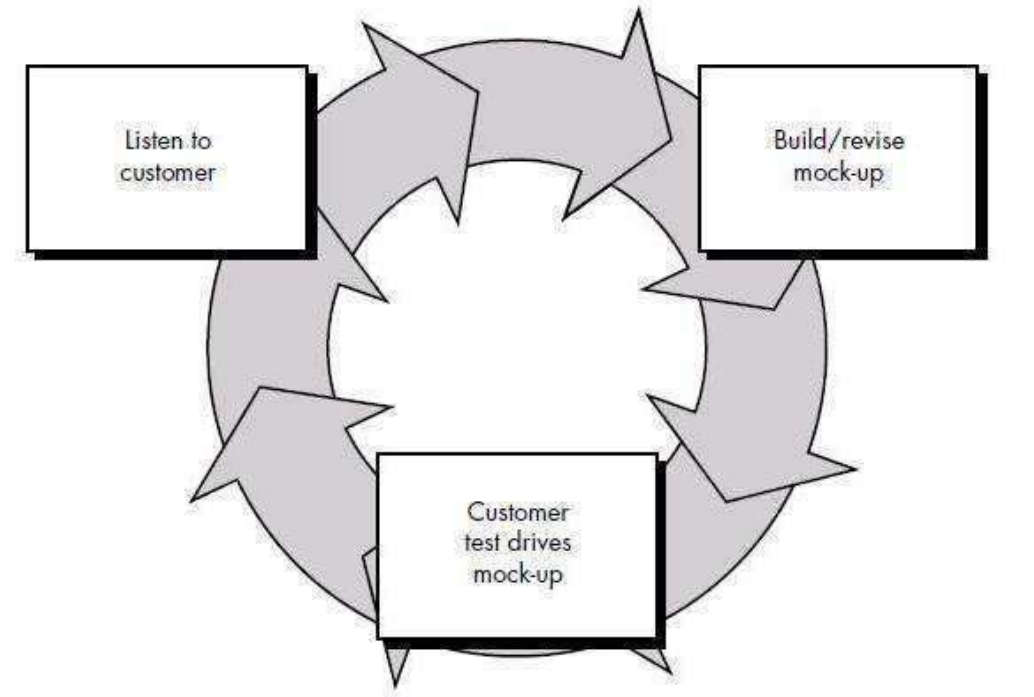


Figure 1.5 Prototype Model

Need for a prototype in software development: -

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- How the screens might look like
- How the user interface would behave
- How the system would produce outputs

A prototyping model can be used when technical solutions are unclear to the development team.

A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

A prototype of the actual product is preferred in situations such as:

- User requirements are not complete
- Technical issues are not clear

RAPID APPLICATION MODEL:

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed pre-planning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The 'AD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days). Used primarily for information systems applications, the RAD approach encompasses the following phases:

Business modeling: The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling: The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined.

Process modeling: The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation: RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover: Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all

interfaces must be fully exercised.

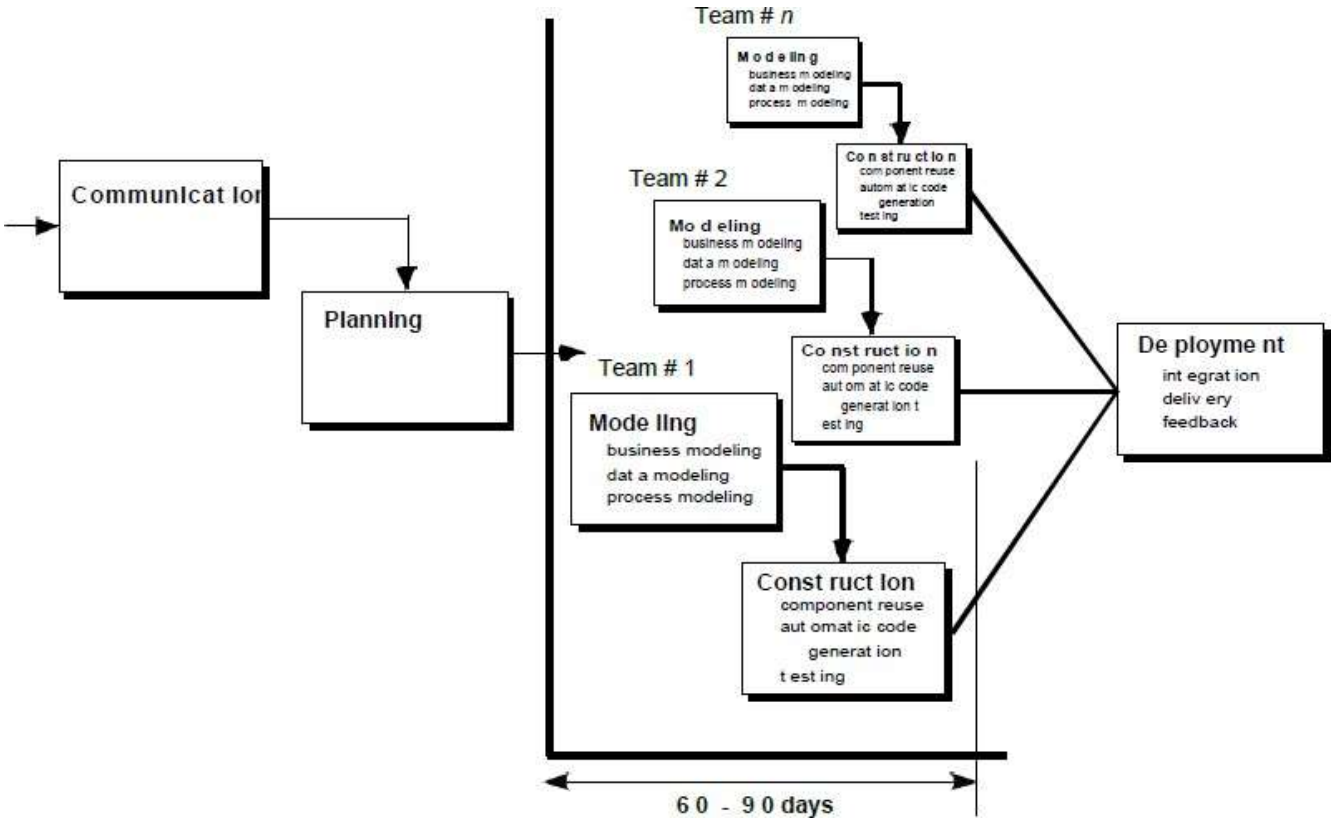


Figure 1.6.1:Rapid Application Model

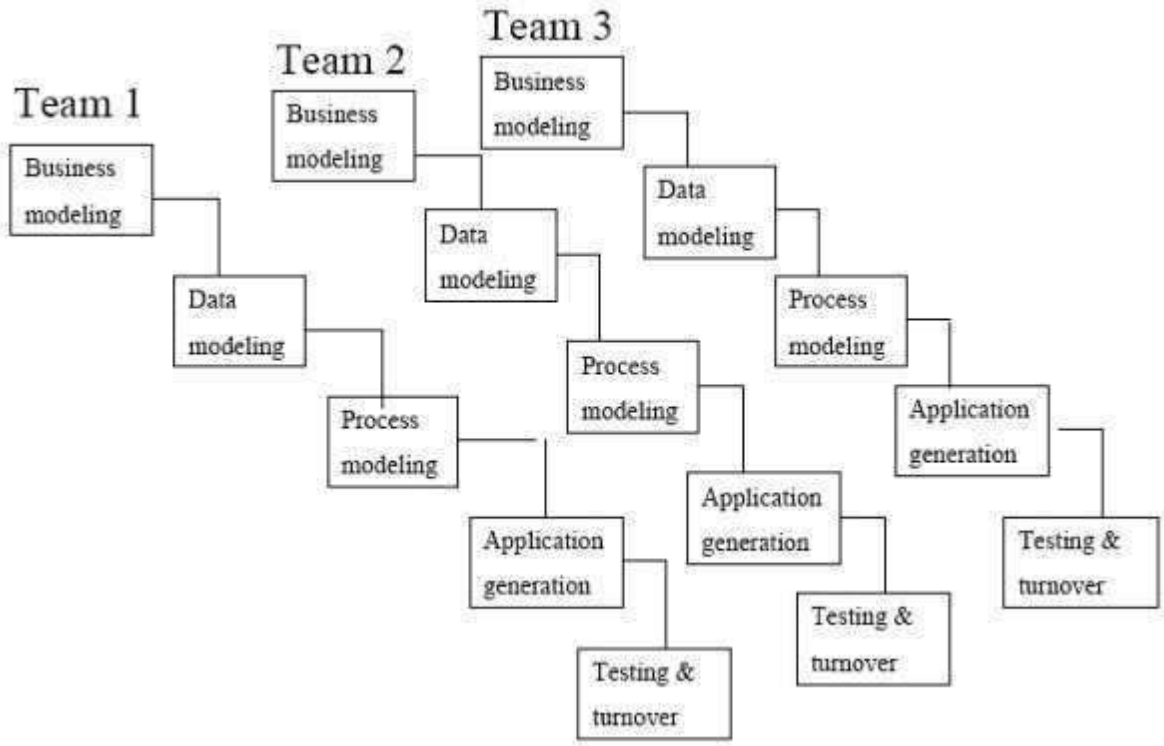


Figure 1.6.2: Rapid Application Model Teamwise

Advantages of the RAD model:

- Reduced development time.
- Increases re usability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

EVOLUTIONARY PROCESS MODEL:

Evolutionary Software Process Model Evolutionary software models are iterative. They are characterized in manner that enables the software engineers to develop increasingly more complete version of software. In programming "iteration" means sequential access to objects. It is typically a cycle. Software engineers can follow this process model that has been clearly designed to put up a product that regularly complete over time.

Iterative Model design:

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:

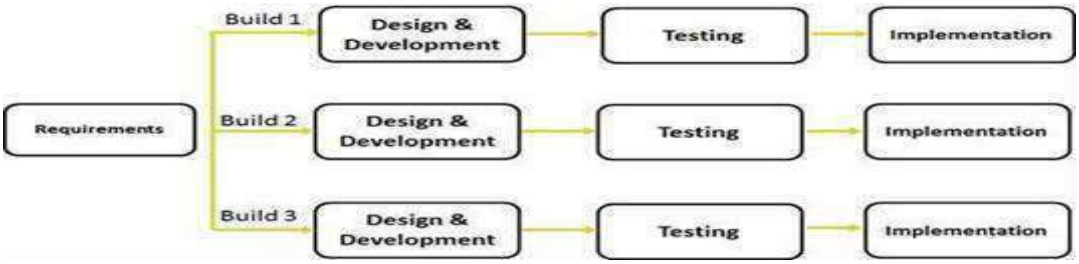


Figure 1.7 Iterative Model

Iterative Model Application:

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.

- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

Evolutionary Process Model is of 2 types

- Incremental model and
- Spiral model

INCREMENTAL MODEL:

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.

For example, the word-processing software is developed using the incremental model.

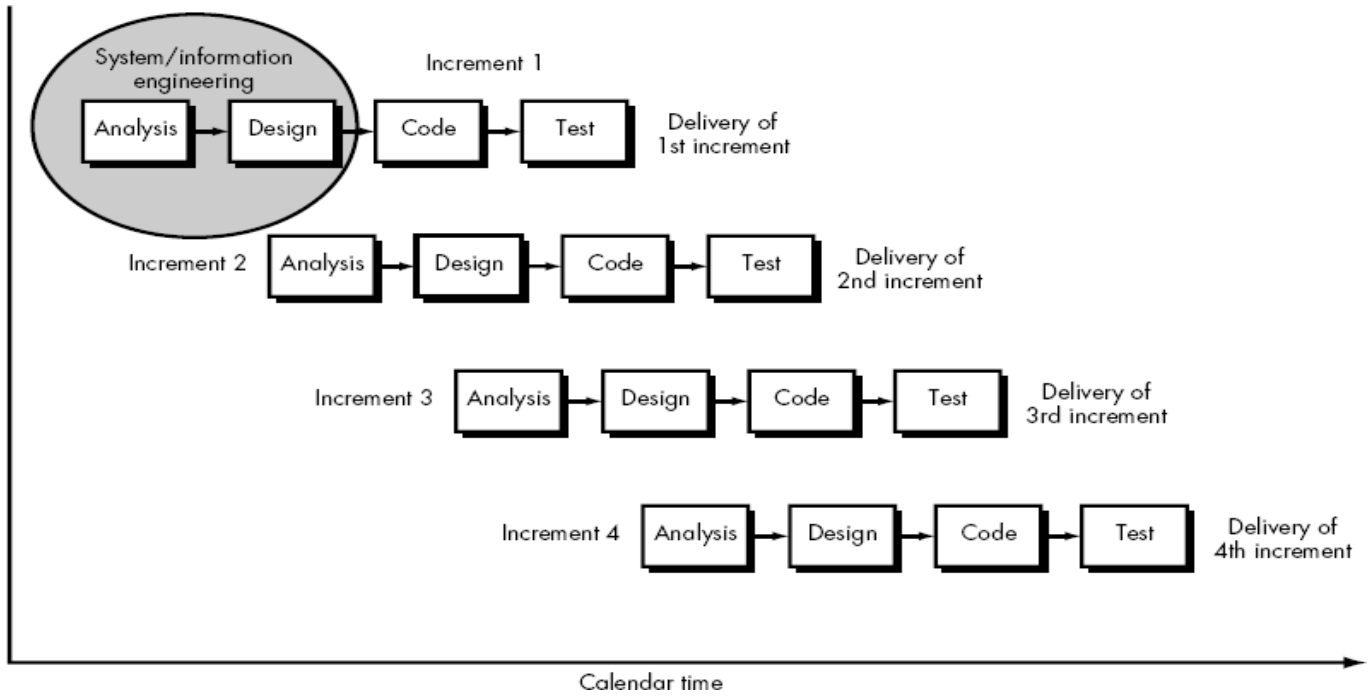


Figure 1.8 Incremental Model

Advantages of Incremental model: -

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.
- There is low risk for overall project failure.

- Customer does not have to wait until the entire system is delivered.

Disadvantages of Incremental model: -

- Needs good planning and design at the management a technical level.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.
- Time foundation create problem to complete the project.

When to use the Incremental model:

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high-risk features and goals.

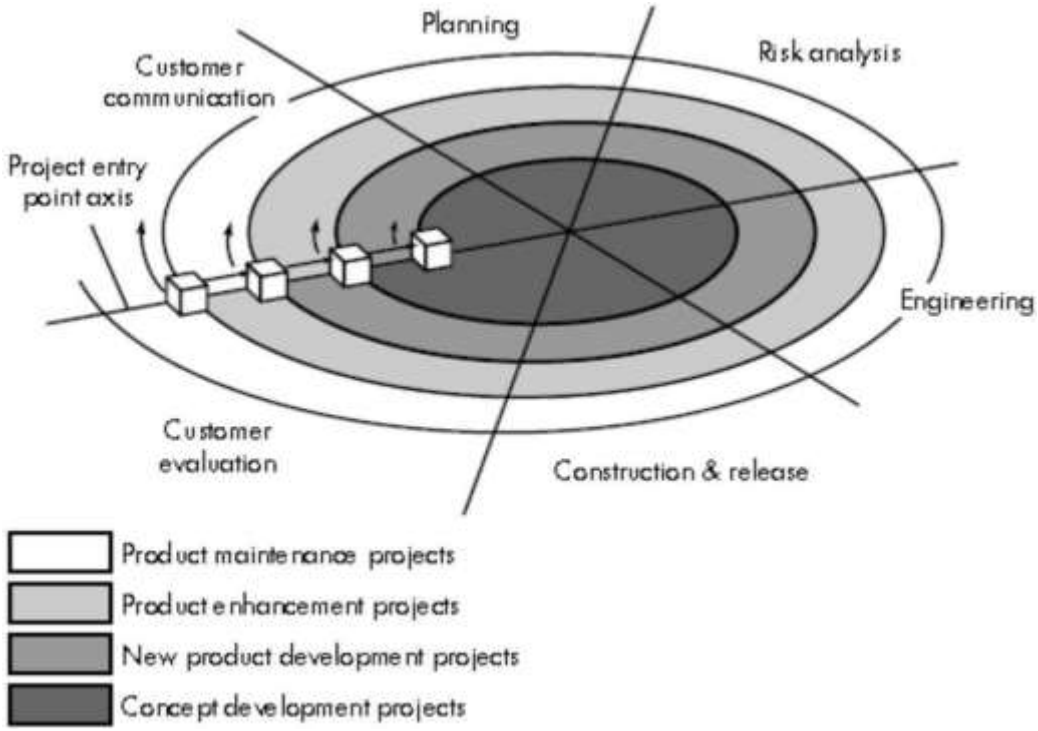
SPIRAL MODEL :

The spiral model, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Project entry point axis is defined this axis represents starting point for different types of project. Every framework activities represent one section of the spiral path. As the development process starts, the software team performs activities that are indirect by a path around the spiral model in a clockwise direction. It begins at the center of spiral model.

Typically, there are between three and six task regions. In blow Figure depicts a spiral model that contains six task regions:

- **Customer communication**—tasks required to establish effective communication between developer and customer.
- **Planning**—tasks required to define resources, time lines, and other project related information.
- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.
- **Construction and release**—tasks required to construct, test, install, and provide user support(e.g., documentation and training).
- **Customer evaluation**—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation



stage.

Figure 1.9 Spiral Model

Advantages of Spiral model: -

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model: -

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

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

COMPONENT ASSEMBLY MODEL:

Component Assembly Model is just like the Prototype model, in which first a prototype is created according to the requirements of the customer and sent to the user for evaluation to get the feedback for the modifications to be made and the same procedure is repeated until the software will cater the need of businesses and consumers is realized. Thus it is also an iterative development model.

This model work in following manner:

1. Identify all required candidate component i.e. classes with the help of application data and algorithm.

2. If these candidate components are used in previous software project then thy must be present in library.
3. Such preexisting component can be extracted from the library and used for further development.
4. But if required component is not presented in the library then build or create the component as per requirement.
5. Place the newly created component in library. This makes one iteration of the system.
6. ‘epeat step 1 to 5 for creating ‘n’ iteration. Where ‘n’ denotes the no of iterations required to develop complete application.

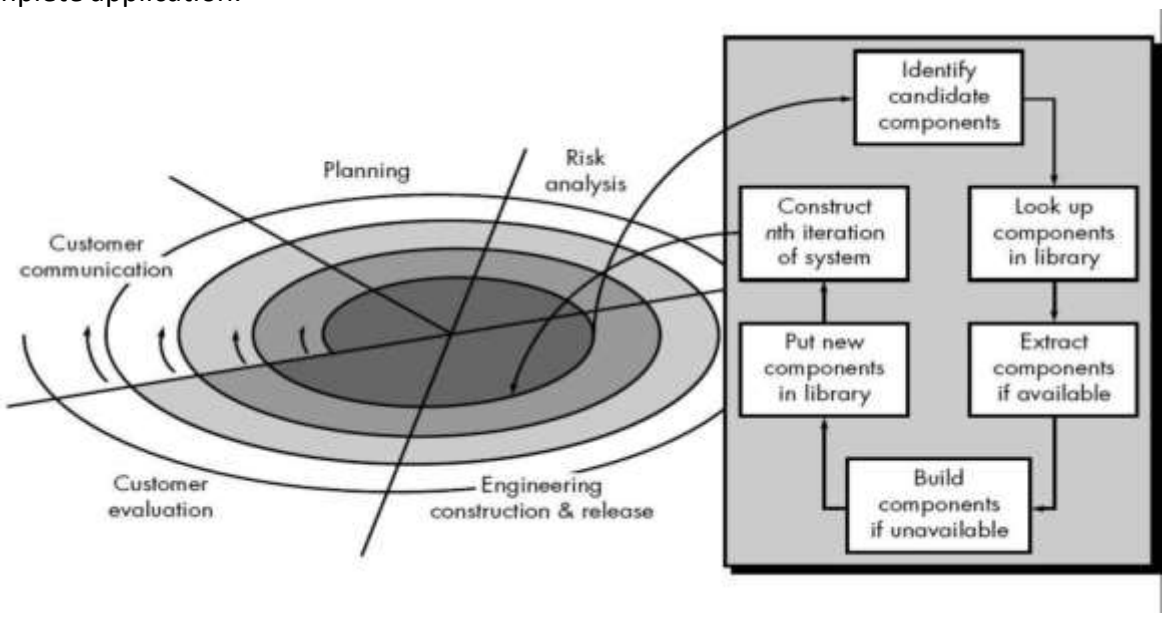


Figure 1.10: Component Assembly Model

Component Assembly Model Characteristics:

- Use of object-oriented technology.
- Components – classes that encapsulate both data and algorithms.
- Components developed to be reusable.
- Paradigm similar to spiral model, but engineering activity involves components.
- System produced by assembling the correct components.

RATIONAL UNIFIED PROCESS (RUP):

Rational Unified Process (RUP) is an object-oriented and Web-enabled program development methodology. RUP is a software application development technique with many tools to assist in coding the final product and tasks related to this goal. RUP is an object-oriented approach used to ensure effective project management and high-quality software production. It divides the development process into four distinct phases that each involves business modeling, analysis and design, implementation, testing, and deployment. The four phases are:

1. **Inception** - The idea for the project is stated. The development team determines if the project is worth pursuing and what resources will be needed.
2. **Elaboration** - The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development.
3. **Construction** - The project is developed and completed. The software is designed, written, and tested.
4. **Transition** - The software is released to the public. Final adjustments or updates are made based on feedback from end users.

The RUP development methodology provides a structured way for companies to envision create software programs. Since it provides a specific plan for each step of the development process, it helps prevent resources from being wasted and reduces unexpected development costs.

Advantages of RUP Software Development: -

- This is a complete methodology in itself with an emphasis on accurate documentation
- It is pro-actively able to resolve the project risks associated with the client's evolving requirements requiring careful change request management
- Less time is required for integration as the process of integration goes on throughout the software development life cycle.
- The development time required is less due to reuse of components.
- There is online training and tutorial available for this process.

Disadvantages of RUP Software Development: -

- The team members need to be expert in their field to develop a software under this methodology.
- The development process is too complex and disorganized.
- On cutting edge projects which utilize new technology, the reuse of components will not be possible. Hence the time saving one could have made will be impossible to fulfill.
- Integration throughout the process of software development, in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing.

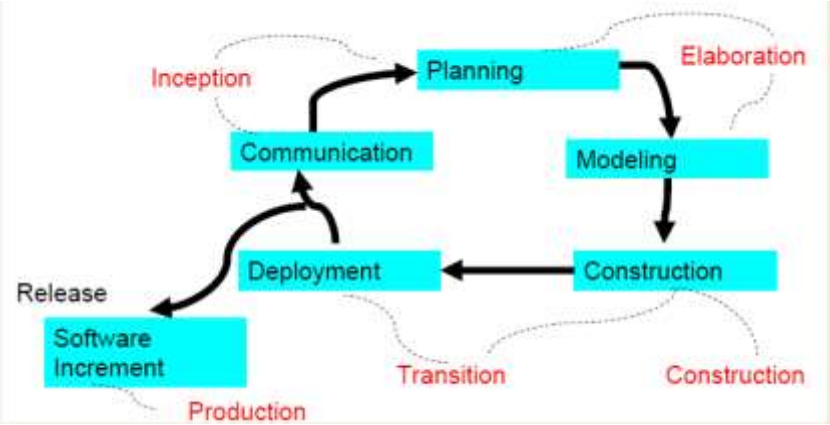


Figure 1.11 Rational Unified Process (RUP)

AGILE DEVELOPMENT MODEL

Agile Process:-The word 'agile' means able to think quickly and clearly. In business, 'agile' is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job.

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model.

Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for new programmers, unless combined with experienced resources.

Extreme Programming

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

Extreme Programming is based on the following values-

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.

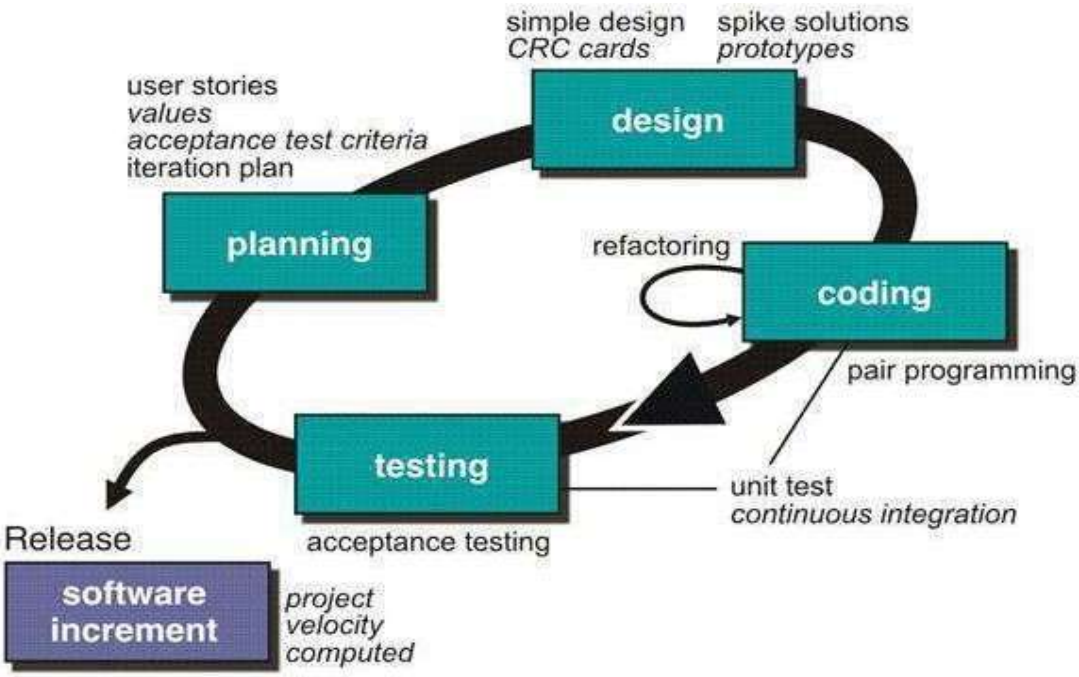


Figure 1.12 Extrem Programming

CAPABILITY MATURITY MODEL (CMM):

The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization's current state of process maturity, the

SEI uses an assessment that results in a five-point grading scheme. The grading scheme determines compliance

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

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

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

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

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

The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

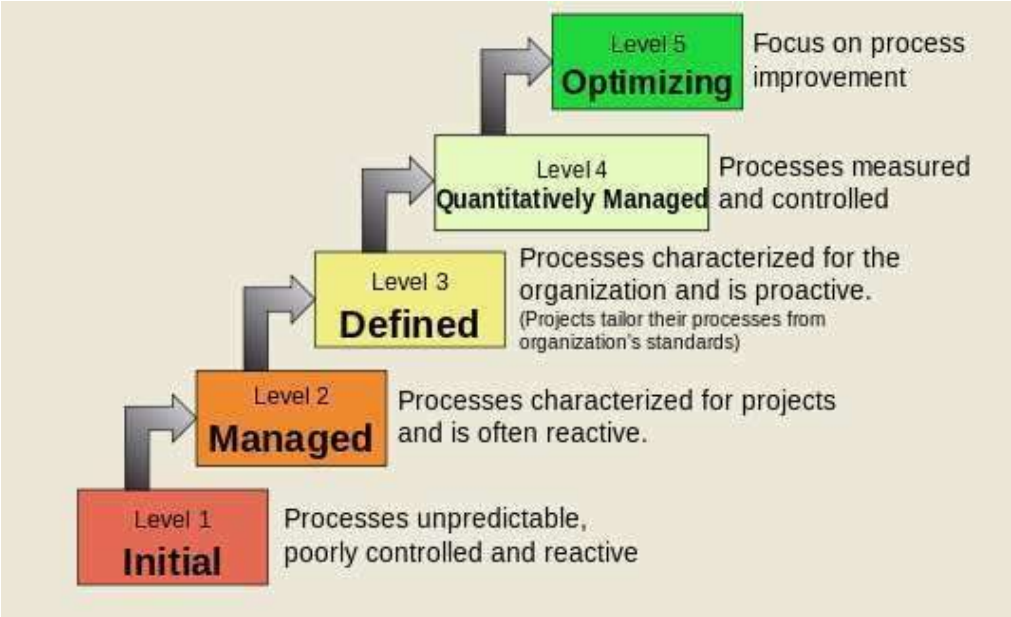


Figure 1.13 Capability Maturity Model

Process maturity level 2: -

- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management

Process maturity level 3: -

- Peer reviews
- Intergroup coordination

- Software product engineering
- Integrated software management
- Training program
- Organization process definition
- Organization process focus

Process maturity level 4: -

- Software quality management
- Quantitative process management

Process maturity level 5: -

- Process change management
- Technology change management
- Defect prevention

SOFTWARE PROCESS CUSTOMIZATION:

In software industry, most of the projects are customized software product 3 major factors that are involved in software process customization and those are:

- PEOPLE
- PRODUCT
- PROCESS

People: -

The primary element of any project is the people. People gather requirements, people interview users (people), people design software, and people write software for people. No people -- no software. I'll leave the discussion of people to the other articles in this special issue, except for one comment. The best thing that can happen to any software project is to have people who know what they are doing and have the courage and self-discipline to do it. Knowledgeable people do what is right and avoid what is wrong. Courageous people tell the truth when others want to hear something else. Disciplined people work through projects and don't cut corners. Find people who know the product and can work in the process.

Process: -

Process is how we go from the beginning to the end of a project. All projects use a process. Many project managers, however, do not choose a process based on the people and product at hand. They simply use the same process they've always used or misused. Let's focus on two points regarding process: (1) process improvement and (2) using the right process for the people and product at hand.

Product: -

The product is the result of a project. The desired product satisfies the customers and keeps them coming back for more. Sometimes, however, the actual product is something less. The product pays the bills and ultimately allows people to work together in a process and build software. Always keep the product in focus.

PRODUCT AND PROCESS METRICS:

Software process metrics measure the software development process and environment. Example productivity, effort estimates, efficiency and failure rate.

Software Product metrics measure the software product.

Example: - size, reliability, complexity and functionality.

Process Metrics and Software Process Improvement: -

The only rational way to improve any process is

- To measure specific attributes of the process
- Develop a set of meaningful metrics based on these attributes
- Use the metrics to provide indicators that will lead to a strategy for improvement

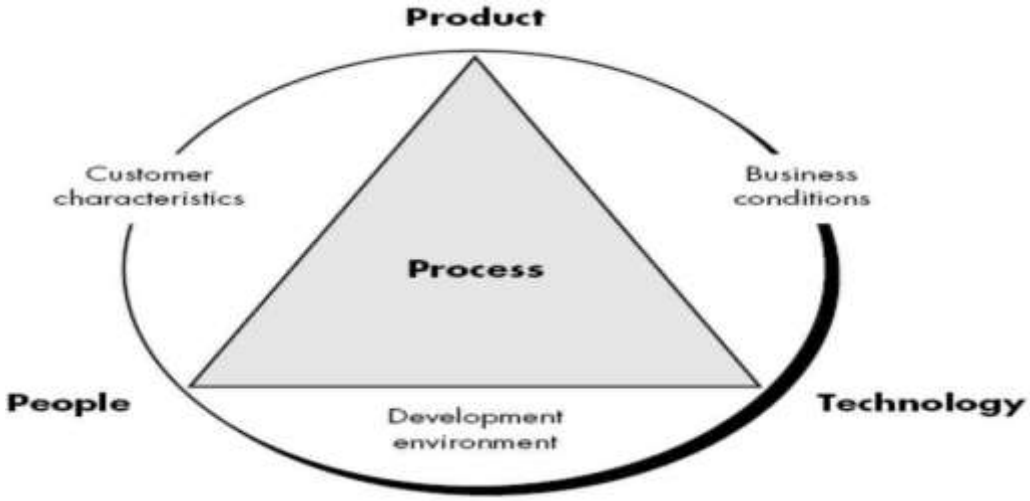


Figure 1.14 Product and Process Metrics

Processes it's at the center of a triangle connecting three factors that have profound influence on software quality and organizational performance

- The skill and motivation of **people** has most influential factor in quality and performance.
- The complexity of the **product** has impact on quality and team performance.
- The technology (the software engineering methods) the process triangle exists within a circle of environmental conditions that include the development environment, business conditions, customer characteristics.