SCS1305 Software Engineering

UNIT I INTRODUCTION S/W engineering paradigm –Life cycle models – Water fall – Incremental – Spiral – Evolutionary – Prototyping – Object oriented – System engineering – Computer based system – Verification – Validation – Life cycle process – Development process – System engineering hierarchy – Introduction to CMM – Levels of CMM.

1.1 Software engineering paradigm

Software engineering paradigm is refers to the method and steps, which are taken while design the software. It consists of three parts.

- Software development paradigm
- Software design paradigm
- Programming paradigm

Software development paradigm

Software development paradigm pertains all the engineering concepts which includes Requirements, software design, programming.

Software design paradigm

Software design paradigm is a part of software development . it includes design, programming, maintenance.

Programming paradigm

Programming paradigm which concerns about the programming aspect of software development. This includes coding, testing, integration.

Need for software engineering

Software engineering fulfill the higher rate of change in user requirement and environment. The user requirement lies on cost, scalability, dynamaic nature of working environments, quality management.

Characteristics of good software

The software product can be judged by what it offer and how well it can be used . furthermore, the software must satisfied on the following grounds :

- Operational
- Transitional
- Maintainance

a) Operational

Operational defines how well the software works in operations. It can be measure on budget, usability, efficiency, correctness, functionality, dependability, security, safety.

b) Transitional

This aspect is important if the software is moved from one platform to another. The key parameters that includes portability, Interoperability, Reusability, Adaptability.

c) Maintenance

Maintenance of the software defines the capabilities to maintain itself in the ever changing environment. It includes modularity, maintainability, flexibility etc.

Software is

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

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

(3) documents that describe the operation and use of the programs.

Software Applications is classified as follows:

- System software : System software is a collection of programs written to service other programs. Some system software e.g., compilers, editors, and file management utilities
- Real-time software: Software that monitors/analyzes/controls real-world events as they occur is called real time
- Business software : Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory) have evolved into management information

system (MIS) software that accesses one or more large databases containing business information.

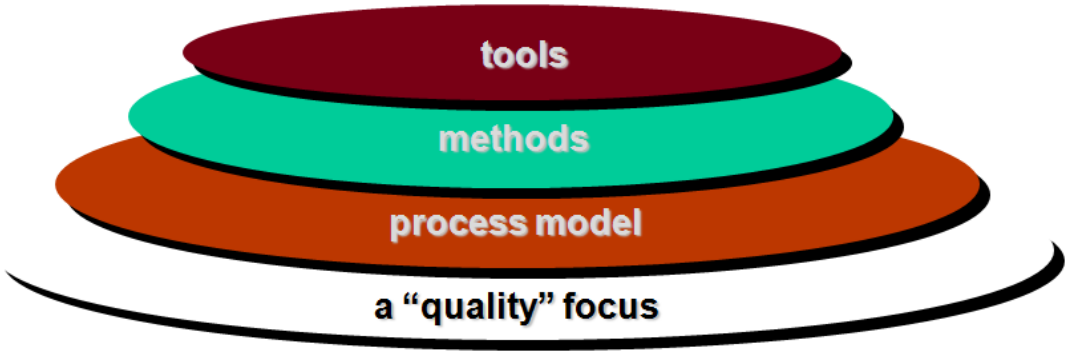
- Engineering and scientific software : Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
- Embedded software : Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
- Personal computer software : The personal computer software market has burgeoned over the past two decades. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.
- Web-based software : The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats).
- Artificial intelligence software : Artificial intelligence (AI) software makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

Software Engineering:

- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines
- Software Engineering is 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 engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the client’s needs.

Furthermore, the software must be easy to modify when the user's needs change.

A Layered Technology



Processes:

Systematic ways of organizing teams and tasks so that there is a clear, traceable path from customer requirements to the final product. (e.g.,:Waterfall, Prototyping, Spiral etc.) Processes help organize and co-ordinate teams, prepare documentation, reduce bugs, manage risk, increase productivity, etc.

Models:

Well-defined formal or informal languages and techniques for organizing and communicating arguments and decisions about software. e.g:

– specification languages

– design models (UML)

Models help stake-holders communicate: customers with developers, designers and developers, developers and testers etc. If they are formal, they also can help support automation

Tools:

Programs which automate or otherwise support software development tasks: e.g., – Eclipse etc. Tools increase productivity, quality and can reduce costs.

1.2 Life cycle models

Software Development Life-Cycle (SDLC) Models:

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process. The following figure is a graphical representation of the various stages of a typical SDLC. The SDLC consist of the following phases:

- Communication
- Requirement gathering
- Feasibility study
- System Analysis
- Software design
- Coding
- Testing and Integration
- Operations & maintenance
- Disposition

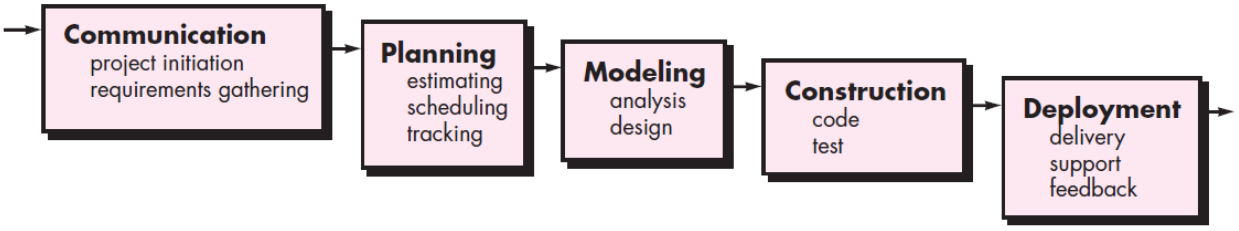
Process framework: activities

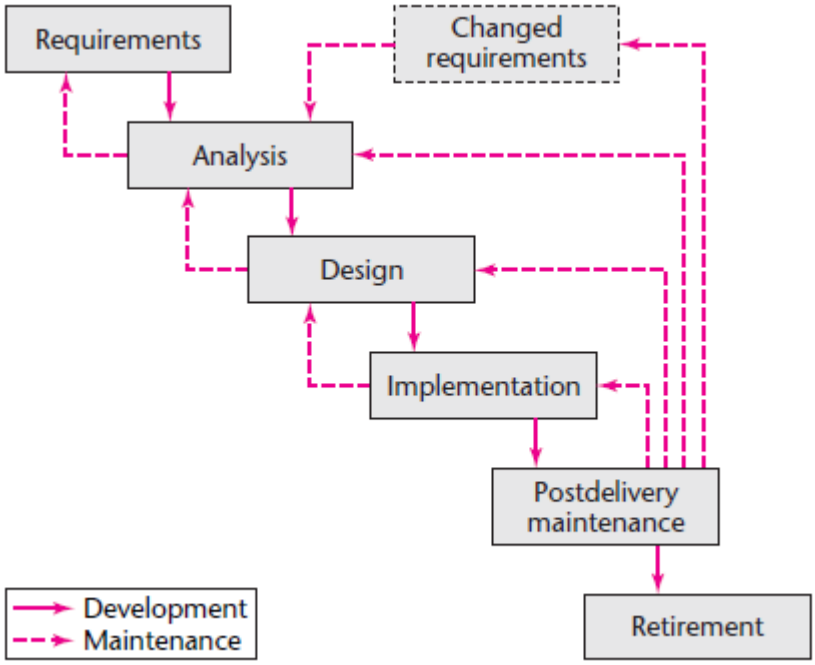
The followings are the key activities of process models:

- Communication: communication and collaboration between developer and customer or stakeholder.
- Planning: technical task to be conducted, risk analysed, resources required, work schedule.
- Modeling : creation of model for better understanding
- Construction : combination of code generation and testing.
- Deployment software is delivered to customers and getting feedback.

1.3 Waterfall model

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach5 to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.





- The classic life cycle suggests a systematic, sequential approach to software development.
- A critical point regarding the waterfall model is that no phase is complete until the documentation for that phase has been completed and the products of that phase have been approved by the software quality assurance (SQA) group.
- Inherent in every phase of the waterfall model is testing. Testing is not a separate phase to be performed only after the product has been constructed, nor is it to be performed only at the end of each phase. Instead, testing should proceed continually throughout the software process.

- In particular, during maintenance, it is necessary to ensure not only that the modified version of the product still does what the previous version did—and still does it correctly (regression testing)—but that it also satisfies any new requirements imposed by the client.
- The feedback loops permits modifications to be made to design documents, the software project management plan, and even the specification document, if necessary.
- The specification document, design document , code document and other documents such as database manual, user manual and operational manual are essential tool for maintaining the product.

Advantage

- Easy to understand and implement.
- Widely used and known
- Reinforces good habits: define-before- design, design-before-code
- Identifies deliverables and milestones
- Document driven
- Maintenance is easier

Disadvantage

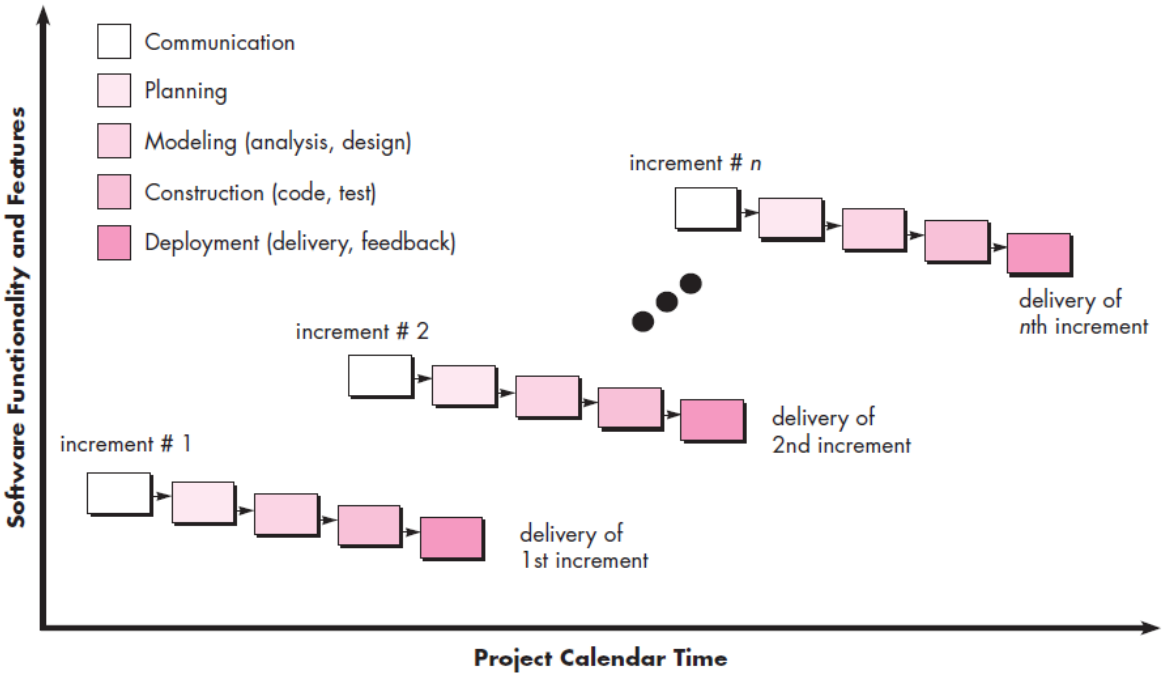
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Delivered product may not meet client needs

Evolutionary Software Process Models

Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software

1.4 Incremental Model

- Software is constructed step by step , in the same way that a building is constructed
- Incremental model in software engineering is a one which combines the elements of waterfall model which are then applied in an iterative manner. It basically delivers a series of releases called increments which provide progressively more functionality for the client as each increment is delivered.
- In incremental model of software engineering, waterfall model is repeatedly applied in each increment. The incremental model applies linear sequences in a required pattern as calendar time passes. Each linear sequence produces an increment in the work.
- When an incremental model is used, the first increment is often 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 review). 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.



Advantages Of Incremental Model

- Initial product delivery is faster.
- Lower initial delivery cost.
- Core product is developed first i.e main functionality is added in the first increment.
- After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
- It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration. This allows for more targeted and rigorous testing of each element within the overall product.
- With each release a new feature is added to the product.
- Customer can respond to feature and review the product.
- Risk of changing requirement is reduced
- Work load is less.

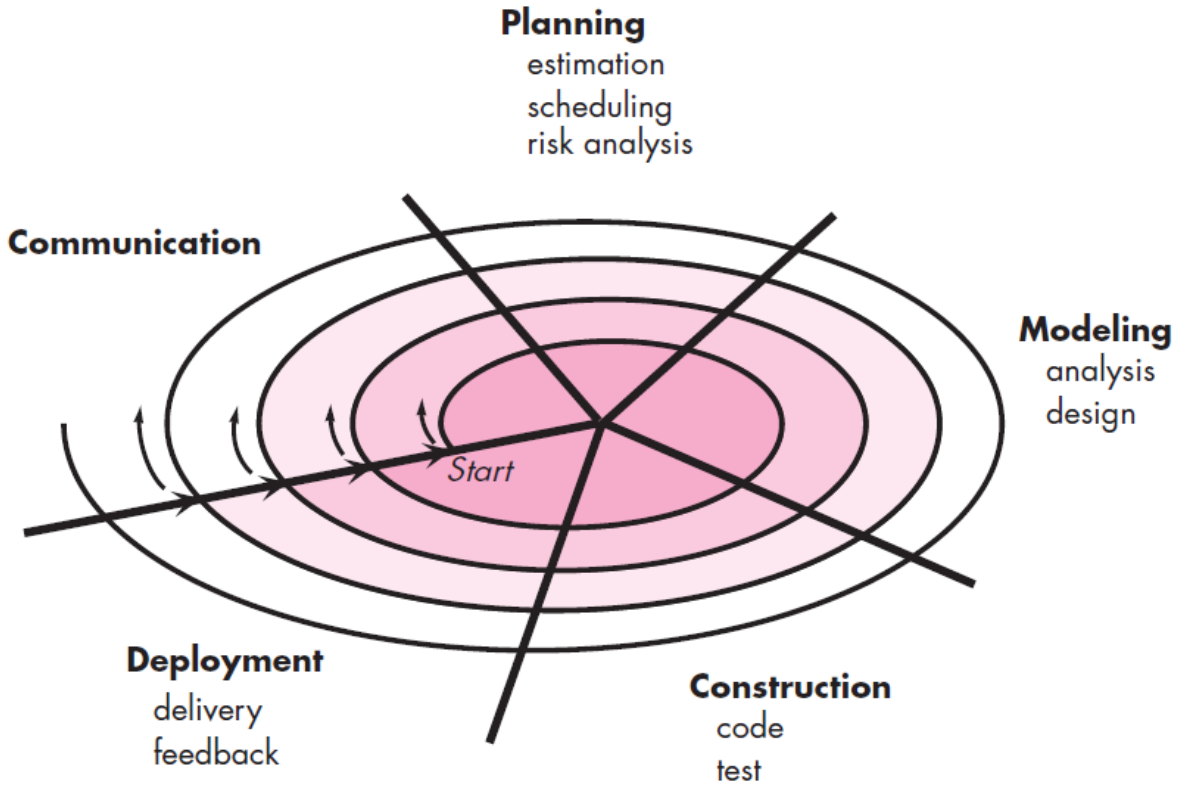
Disadvantages of Incremental Model

- Needs good planning and design.
- 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.
- Mostly such model is used in web applications and product based companie

1.5 Spiral Model

- An element of risk is always involved in the development of software.
- For example, key personnel can resign before the product has been adequately documented. Too much, or too little, can be invested in testing and quality assurance. After spending hundreds of thousands of dollars on developing a major software product, technological breakthroughs can render the entire product worthless.

- An organization may research and develop a database management system, but before the product can be marketed, a lower-priced, functionally equivalent package is announced by a competitor.
- For obvious reasons, software developers try to minimize such risks wherever possible. One way of minimizing certain types of risk is to construct a prototype.
- The idea of minimizing risk via the use of prototypes and other means is the idea underlying the spiral life-cycle model. A simplified way of looking at this lifecycle model is as a waterfall model with each phase preceded by risk analysis.
-



- In full spiral model the radial dimension represents cumulative cost to date, and the angular dimension represents progress through the spiral.
- Each cycle of the spiral corresponds to a phase. A phase begins (in the top left quadrant) by determining objectives of that phase, alternatives for achieving those objectives, and constraints imposed on those alternatives. This process results in a strategy for achieving those objectives.

- Attempts are made to mitigate every potential risk, in some cases by building a prototype.
- If certain risks cannot be mitigated, the project may be terminated immediately; under some circumstances, however, a decision could be made to continue the project but on a significantly smaller scale.
- If all risks are successfully mitigated, the next development step is started (bottom right quadrant). This quadrant of the spiral model corresponds to the classical waterfall model. Finally, the results of that phase are evaluated and the next phase is planned.

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.
- Project estimates in terms of schedule, cost etc become more and more realistic as the project moves forward and loops in spiral get completed.
- It is suitable for high risk projects, where business needs may be unstable. A highly customized product can be developed using this.

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.
- It is not suitable for low risk projects.
- May be hard to define objective, verifiable milestones.
- Spiral may continue indefinitely.

1.6 Evolutionary model

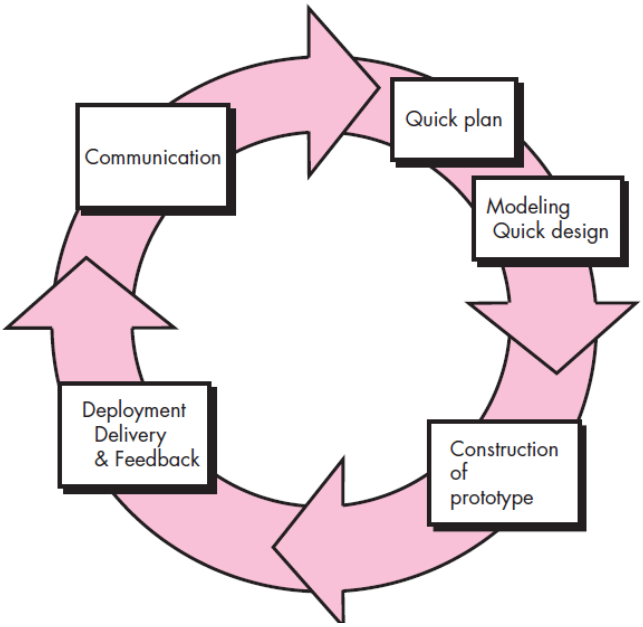
- Evolutionary models are iterative.
- Straight line to complete end product is not possible.
- Limited version of the software product must be delivered.
- Customer does not identify detailed input, processing or output requirements.
- The prototyping paradigm assists the software engineer and the customer to better understand what is to be built when the requirements are fuzzy.
- This model begins with communication.
- The software engineer and the customer meet and define the overall objectives for the software.
- Requirements often change as development proceeds.
- Set of core product or system requirements is well understood but the details and extensions have yet to be defined.

Prototyping

- When to use: customer defines a set of general objectives.
- This model follows an evolutionary and iterative approach.
- This model used when requirements are not well understood.
- It serves as a mechanism for identifying software requirements.
- Feedback is used to refine the prototype.

Steps

- Start with communication by meeting the stakeholders to define the objectives.
- Identify the requirements.
- Quick plan for prototyping and modeling (Quick design) occur.(interface and output)
- Construction of prototype.



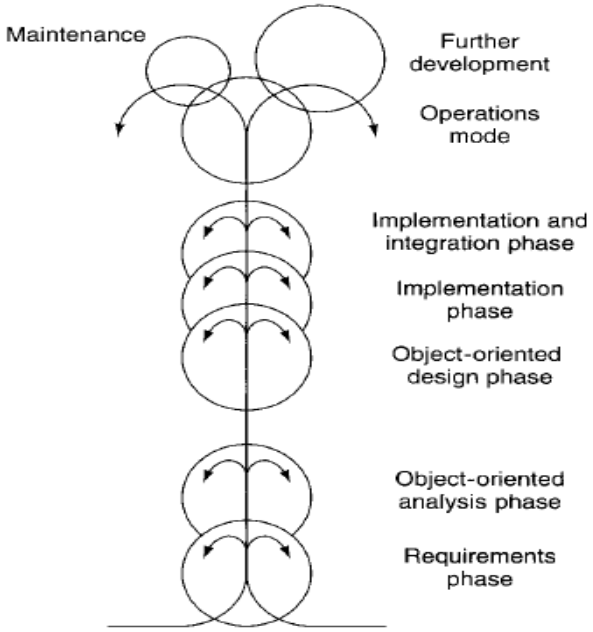
Problems

- Once the customer sees the “working version” of the software, they want to stop all developments.
- Implementation compromises to get the software running quickly. (language choice, user interface, OS choice, inefficient algorithm)

1.7 Object oriented system engineering

- Need for iteration within and between phases
  - Fountain model
  - Unified software development process
- All incorporate some form of
  - Iteration
  - Parallelism
  - Incremental development

Fountain Model



- Object oriented life cycle model have been proposed that explicitly reflect the need for iteration
- The circles representing the various phases overlap, explicitly reflecting an overlap between activities.
- The arrows within a phase represent iteration within that phase.
- The maintenance circle is smaller, to symbolize reduce maintenance effort when the object oriented paradigm is used.

Advantages

- Support Iteration within phases
- Parallelism between phases

Disadvantages

- It may be degraded in to CABTAB(code-a-bit test-a-bit) which reuires frequent iteration and refinements

Unified Process

- Unified process is a framework for OO software engineering using UML (Unified Modeling Language)

- Unified process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development

Inception phase

- Encompasses the customer communication and planning activities
- Rough architecture, plan, preliminary use-cases

Elaboration phase

- Encompasses the customer communication and modeling activities
- Refines and expands preliminary use-cases
- Expands architectural representation to include: use-case model, analysis model, design model, implementation model, and deployment model
- The plan is carefully reviewed and modified if needed

Construction phase

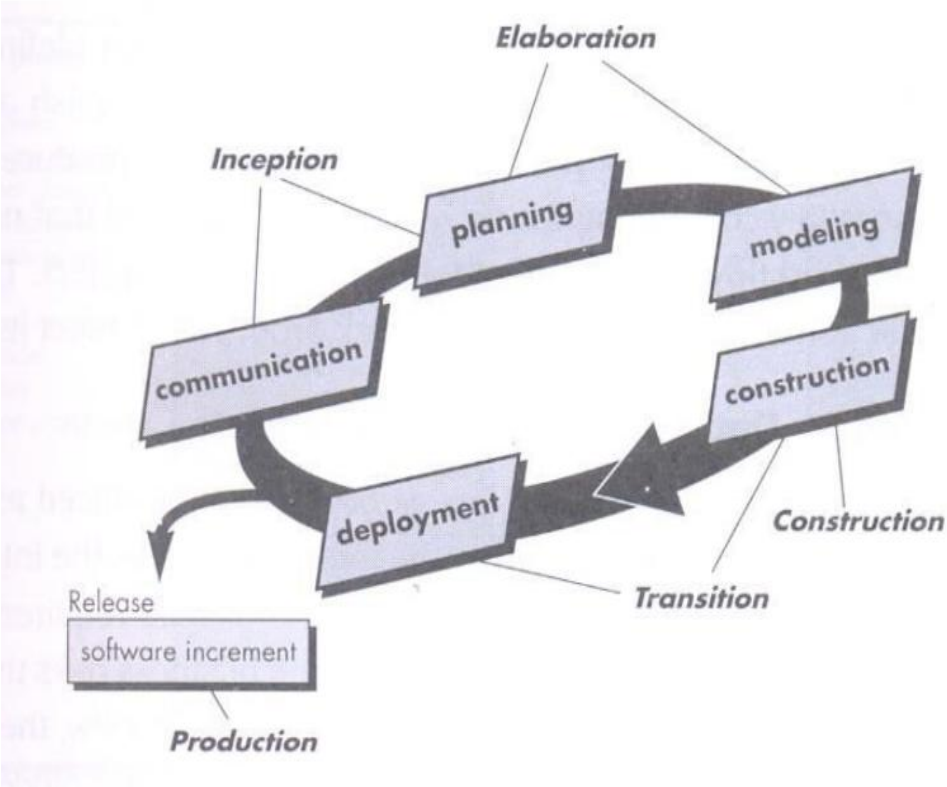
- Analysis and design models are completed to reflect the final version of the software increment
- Using the architectural model as an input develop or acquire the software components, unit tests are designed and executed, integration activities are conducted
- Use-cases are used to derive acceptance tests

Transition phase

- Software is given to end-users for beta testing
- User report both defects and necessary changes
- Support information is created (e.g., user manuals, installation procedures)
- Software increment becomes usable software release

Production phase

- Software use is monitored
- Defect reports and requests for changes are submitted and evaluated



UNIFIED PROCESS WORK PRODUCTS

Tasks which are required to be completed during different phases

Inception Phase

- Vision document
- Initial Use-Case model
- Initial Risk assessment
- Project Plan

Elaboration Phase

- Use-Case model
- Analysis model
- Software Architecture description
- Preliminary design model

Construction Phase

- Design model
- System components
- Test plan and procedure

- Test cases
- Manual

• **Transition Phase**

- Delivered software increment
- Beta test results
- General user feedback

Computer based system

1.8 Verification – Validation

Validation: Are we building the right system?

Verification: Are we building the system right?

- validation is concerned with checking that the system will meet the customer's actual needs, verification is concerned with whether the system is well-engineered, error-free, and so on.
- Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful.
- The distinction between the two terms is largely to do with the role of specifications. Validation is the process of checking whether the specification captures the customer's needs, while verification is the process of checking that the software meets the specification.
- Verification includes all the activities associated with the producing high quality software: testing, inspection, design analysis, specification analysis, and so on.
- Validation includes activities such as requirements modelling, prototyping and user evaluation.
- In a traditional phased software lifecycle, verification is often taken to mean checking that the products of each phase satisfy the requirements of the previous phase.
- Validation is relegated to just the begining and ending of the project: requirements analysis and acceptance testing. This view is common in many software engineering textbooks, and is misguided. It assumes that the customer's requirements can be captured completely at the start of a project, and

that those requirements will not change while the software is being developed. In practice, the requirements change throughout a project, partly in reaction to the project itself: the development of new software makes new things possible. Therefore both validation and verification are needed throughout the lifecycle.

1.9 System Engineering

- Before software can be engineered, the ”system” in which it resides must be understood.
- To accomplish this, the overall objective of the system must be determined; the role of hardware, software, people, database, procedures, and other system elements must be identified; and operational requirements must be elicited, analyzed, specified, modeled, validated, and managed. These activities are the foundation of system engineering.

1.10 Computer-based systems

- A set or arrangement of elements that are organized to accomplish some predefined goal by processing information.
- The goal may be to support some business function or to develop a product that can be sold to generate business revenue.
- To accomplish the goal, a computer-based system makes use of a variety of system elements:

a) **Software.** Computer programs, data structures, and related documentation that serve to effect the logical method, procedure, or control that is required.

b) **Hardware.** Electronic devices that provide computing capability, the interconnectivity devices (e.g., network switches, telecommunications devices) that enable the flow of data, and electromechanical devices (e.g., sensors, motors, pumps) that provide external world function.

c) **People.** Users and operators of hardware and software.

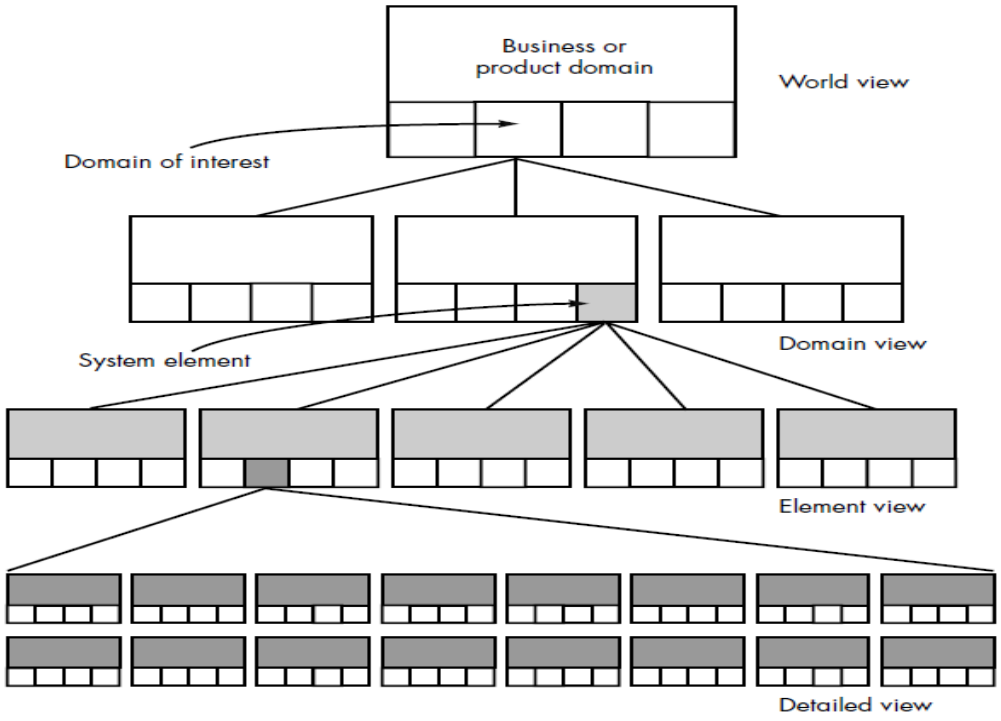
d) **Database.** A large, organized collection of information that is accessed via software.

e) **Documentation.** Descriptive information (e.g., hardcopy manuals, on-line help files, Web sites) that portrays the use and/or operation of the system.

f) Procedures. The steps that define the specific use of each system element or the procedural context in which the system resides.

1.11 The System Engineering Hierarchy

- The system engineering process usually begins with a “world view.” That is, the entire business or product domain is examined to ensure that the proper business or technology context can be established.
- The world view is refined to focus more fully on specific domain of interest.
- Within a specific domain, the need for targeted system elements (e.g., data, software, hardware, people) is analyzed.
- Finally, the analysis, design, and construction of a targeted system element is initiated.
- At the top of the hierarchy, a very broad context is established
- At the bottom, detailed technical activities, performed by the relevant engineering discipline (e.g., hardware or software engineering), are conducted.



The system engineering hierarchy

- The world view (WV) is composed of a set of domains (Di), which can each be a system or system of systems in its own right.

$$WV = \{D1, D2, D3, \ldots, Dn\}$$

- Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain or component:

$$Di = \{E1, E2, E3, \ldots, Em\}$$

- Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:

$$Ej = \{C1, C2, C3, \ldots, Ck\}$$

- In the software context, a component could be a computer program, a reusable program component, a module, a class or object, or even a programming language statement.

System Modeling

System engineering is a modeling process. Whether the focus is on the world view or the detailed view, the engineer creates models that

- Define the processes that serve the needs of the view under consideration.
- Represent the behavior of the processes and the assumptions on which the behavior is based.
- Explicitly define both exogenous and endogenous input3 to the model.
- Represent all linkages (including output) that will enable the engineer to better understand the view.

To construct a system model, the engineer should consider a number of restraining factors:

- Assumptions

  These reduce the number of possible variations, thus enabling a model to reflect the problem in a reasonable manner

- Simplifications

  These enable the model to be created in a timely manner

- Limitations

These help to bound the maximum and minimum values of the system

- Constraints

  These guide the manner in which the model is created and the approach taken when the model is implemented

- Preferences

  These indicate the preferred solution for all data, functions, and behavior

  They are driven by customer requirements

System Simulation

- Many computer-based systems interact with the real world in a reactive fashion. That is, real-world events are monitored by the hardware and software that form the computer-based system, and based on these events, the system imposes control on the machines, processes, and even people who cause the events to occur.
- Unfortunately, the developers of reactive systems sometimes struggle to make them perform properly. Until recently, it has been difficult to predict the performance, efficiency, and behavior of such systems prior to building them.
- If the system fails, significant economic or human loss could occur.
- software tools for system modeling and simulation are being used to help to eliminate surprises when reactive, computer-based systems are built. These tools are applied during the system engineering process, while the role of hardware and software, databases and people is being specified.
- Modeling and simulation tools enable a system engineer to "test drive" a specification of the system.

1.12 Introduction to CMM

- Not life-cycle models
- Rather, a set of strategies for improving the software process
  - SW–CMM for software
  - P–CMM for human resources (“people”)
  - SE–CMM for systems engineering

  - IPD–CMM for integrated product development
  - SA–CMM for software acquisition
- These strategies are unified into CMMI (capability maturity model integration)
- A strategy for improving the software process
- Put forward in 1986 by the SEI (Software Engineering Institute)

Fundamental ideas:

- Improving the software process leads to
  - Improved software quality
  - Delivery on time, within budget
- Improved management leads to
  - Improved techniques

1.13 Levels of CMM.

Five levels of maturity are defined

- Maturity is a measure of the goodness of the process itself
- An organization advances stepwise from level to level

Level 1. Initial Level(Ad hoc approach)

- The entire process is unpredictable
- Management consists of responses to crises
- Most organizations world-wide are at level 1

Level 2. Repeatable Level(Basic software management)

- Management decisions should be made on the basis of previous experience with similar products
- Measurements (“metrics”) are made
- These can be used for making cost and duration predictions in the next project
- Problems are identified, immediate corrective action is taken

Level 3. Defined Level (Process Definition)

- The software process is fully documented
- Managerial and technical aspects are clearly defined
- Continual efforts are made to improve quality and productivity
- Reviews are performed to improve software quality

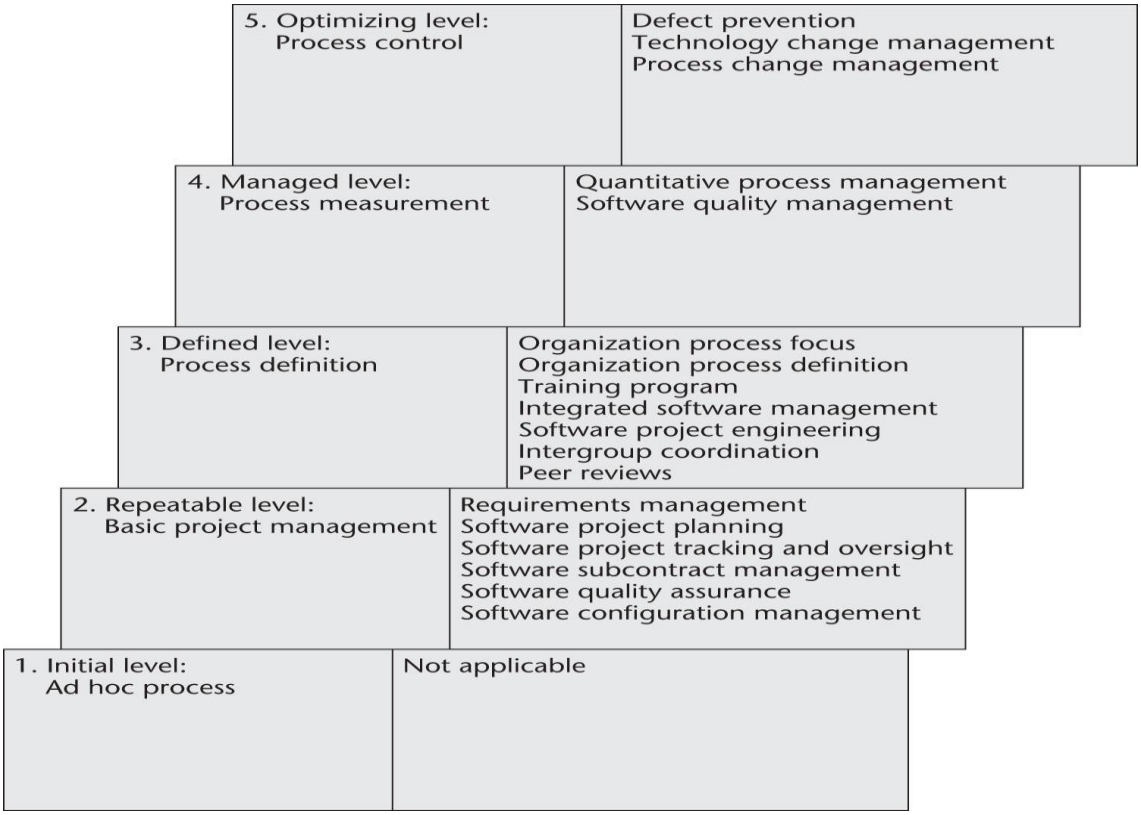
- CASE environments are applicable now (and not at levels 1 or 2)

Level 4. Managed Level(process measurement)

- Quality and productivity goals are set for each project
- Quality and productivity are continually monitored
- Statistical quality controls are in place

Level 5. Optimizing Level

- Continuous process improvement
- Statistical quality and process controls
- Feedback of knowledge from each project to the next



Experiences with SW–CMM

- It takes:
  - 3 to 5 years to get from level 1 to level 2
  - 1.5 to 3 years from level 2 to level 3
  - SEI questionnaires highlight shortcomings, suggest ways to improve the process

Key Process Areas

The KPAs describe those software engineering functions that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

- Goals—the overall objectives that the KPA must achieve.
- Commitments—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- Abilities—those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
- Activities—the specific tasks required to achieve the KPA function.
- Methods for monitoring implementation—the manner in which the activities are monitored as they are put into place.
- Methods for verifying implementation—the manner in which proper practice for the KPA can be verified.

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

Reference Books:

1. Pressman, “Software Engineering and Application”, 6th Edition, Mcgraw International Edition, 2005.

2. Sommerville, ”Software Engineering”, 6th Edition, Pearson Education, 2000.