Software Engineering

Purbanchal University

BCA 5th Semester

S@R0Z

 Downloaded from: sarojpandey.com.np

Chapter 1 - Introduction

Software is a computer programs and its associated document and configuration data which is needed to make these programs operate correctly. A software system usually consists of a number of separate programs, configuration files which are used to set up these programs, system documentation which describes the structure of the system and user documentation which explains how to use the system, and for software products, web sites for users to download recent information.

Software is........
Instructions (computer programs) that when executed provide desired function and performance.
Data structures that enables the programs to adequately manipulate information, and
Documents that describe the operation and use of the programs.

Evolving role of software

Software takes on a dual role. It is a product and the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, a network of computers that are accessible by local hardware. Software is an information transformer-producing, managing, acquiring, modifying, displaying, or transmitting information. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information(networks), and the creation and control of other programs (software tools and environments).

Software delivers the most important product of our time- information. Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (Internet) and provides the means for acquiring information in all of its form.

Software Characteristics

Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware. When hardware is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a physical form.

- Software is developed or engineered; it is not manufactured in the classical sense.
- Software does not "wear out" but it does deteriorate.
- Currently, most software is still custom-built.

Attributes of a good Software

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

 Downloaded from: sarojpandey.com.np

- Maintainability

  Software must evolve to meet changing needs of the customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.

- Dependability

  Software must be trustworthy. Dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economical damage in the event of system failure.

- Efficiency

  Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, and memory utilization, etc.

- Usability

  Software must be usable by the users for which it was designed, this means it should have an appropriate user interface and adequate documentation.

Software Applications

- System software
- Real-time software
- Business software
- Engineering and scientific software
- Embedded software
- Personal computer software
- Web-based software
- Artificial intelligence software

System software

- A collection of programs written to service other programs.
- Some systems software (e.g., compilers, editor, and file management utilities) process complex, but determinate, information structures.
- Other system applications (e.g. operating system components, drivers, telecommunications processors) process largely indeterminate data.

- It is characterized by heavy interaction with computer hardware, heavy usage by multiple users; concurrent operation that requires scheduling, resources sharing, and sophisticated process management, complex data structures, and multiple external interfaces.

 Downloaded from: sarojpandey.com.np

Real time Software

- It monitors/analyzes/ controls real-world events as they occur.
- Elements of real time software include
    - **Data gathering** component that collects and formats information from an external environment,
    - **Analysis component** that transforms information as required by the applications,
    - **Control/Output components** that responds to the external environment,
    - **Monitoring component** that coordinates all other components so that real time response can be maintained.

Business Software

- Business information processing is the largest single software application area.
- Discrete "system" (e.g., payroll, accounts receivable/payable, inventory) has evolved into MIS software that access database containing business information.
- Encompass interactive computing (e.g., point of sale transaction processing)

Engineering and Scientific Software

- 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.
- Ex- CAD, System simulation and other interactive applications.

Embedded Software

- Resides in ROM (Read only memory) and is used to control products and systems for the consumer and industrial markets.
- Perform very limited and esoteric functions (e.g., keypad control for the microwave oven)
- Provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems.)

Personal computer software

Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access.

Web-based Software

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

- It makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

 Downloaded from: sarojpandey.com.np

- Ex- Expert systems, also called knowledge based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Programs vs. Software products

Program	Software Product
Programs are developed by individuals for their personal use.	Software products are usually developed by a group of software engineers and have multiple users.
Small in size and have limited functionality	Medium or large size program and have complex functionality.
The author of a program himself uses and maintains his programs.	Software products are too large to be developed by any individual programmer. A group of software engineers are involved in the development.
Program usually do not have good user-interface and lack proper documentation	A software product not only consists of the program code but also of all the associated documents such as SRS document, design document, test document and users' manuals.

Software products

Software Products may be developed for a particular customer or may be developed for a general market. There are two types of software products:

- ☑ **Generic products**: These are stand-alone systems which are produced by a development organization and sold on the open market to a range of different customers. Sometimes they are referred to as *shrink-wrapped software*. Examples: databases, word processors, drawing packages and project management tools.

- ☑ **Bespoke (or customised) products** - These are systems developed for a single customer according to their specification. Examples: control systems for electronic devices, systems written to support a particular business process and air traffic control systems.

An important difference between these different types of software is that, in generic products, the organization, which develops the software, controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification.

Downloaded from: sarojpandey.com.np

Key Challenges facing software engineering

Software Engineering in the 21st century faces three key challenges, i.e., Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times.

- ☑ **Legacy systems**
  Old, valuable systems which hold the majority of larger software systems in use must be maintained and updated

- ☑ **Heterogeneity**
  Systems are distributed and include a mix of hardware and software. The heterogeneity challenge is the challenge of developing techniques to build dependable software, which is flexible enough to cope with this heterogeneity.

- ☑ **Delivery**
  There is increasing pressure for faster delivery of software unlike time-consuming traditional software engineering techniques. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

Software Myths

- ☑ Software standards provide software engineers with all the guidance they need. The reality is the standards may be outdated and rarely referred to.

- ☑ People with modern computers have all the software development tools. The reality is that CASE tools are more important than hardware to producing high quality software, yet they are rarely used effectively.

- ☑ Adding people is a good way to catch up when a project is behind schedule. The reality is that adding people only helps the project schedule when is it done in a planned, well-coordinated manner.

- ☑ Giving software projects to outside parties to develop solves software project management problems. The reality is people who can’t manage internal software development problems will struggle to manage or control the external development of software too.

- ☑ A general statement of objectives from the customer is all that is needed to begin a software project. The reality is without constant communication between the customer and the developers it is impossible to build a software product that meets the customer's real needs.

- ☑ Project requirements change continually and change is easy to accommodate in the software design. The reality is that every change has far-reaching and unexpected consequences. Changes to software requirements must be managed very carefully to keep a software project on time and under budget.

 Downloaded from: sarojpandey.com.np

- ☑ Once a program is written, the software engineer's work is finished. The reality is that maintaining a piece of software is never done, until the software product is retired from service.

- ☑ There is no way to assess the quality of a piece of software until it is actually running on some machine. The reality is that one of the most effective quality assurance practices (formal technical reviews) can be applied to any software design product and can serve as a quality filter very early in the product life cycle.

- ☑ The only deliverable from a successful software project is the working program. The reality is the working program is only one of several deliverables that arise from a well-managed software project. The documentation is also important since it provides a basis for software support after delivery.

- ☑ Software engineering is all about the creation of large and unnecessary documentation. The reality is that software engineering is concerned with creating quality. This means doing things right the first time and not having to create deliverables needed to complete or maintain a software product. This practice usually leads to faster delivery times and shorter development cycles.

Summary

Software has become a key element in the evolution of computer based systems and products. Over the past 50 years, software has evolved from a specialized problem solving and information analysis tool to an industry in itself. Since software is composed of programs, data and documents; each of these items comprises a configuration that is created as part of the software engineering process. The intent of software engineering is to provide a framework for building software with higher quality. Software engineering is the systematic collection of decades of programming experience together with the innovations made by researchers towards developing high-quality software in a cost effective manner.

Assignment Questions

Q1. Software is both a product and a vehicle for delivering a product. Explain

Q2. Software is engineered, not manufactured. Explain

Q3. “Software does not wear, but it does deteriorate”. Describe this statement with reference to software characteristics.

Q4. Define Software and explain its characteristics and also mention the various types of software product.

Q5. Explain some of the Software applications you have noticed.

Q6. Compare and contrast programs with Software products.

Q7. What are the various software myth and explain what should be the reality.

Q8. What are the key challenges facing software engineering?

 Downloaded from: sarojpandey.com.np

Chapter 2

Software engineering is an engineering discipline, which is concerned with all aspects of software production from the early stages of system specification through to maintaining the software.

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

Software engineering encompasses a process, management techniques, technical methods, and the use of tools to develop software. It provides the specification, development, management, and evolution of software systems, not constrained by materials governed by physical laws or manufacturing processes.

According to Stephen R. Schach, Richard D.Irwin, Inc. and Aksen Associates,

> *Software engineering* is a discipline whose aim is the production of quality software, delivered on time, within budget, and satisfying users' needs.

According to Shari Lawrence Pfleeger,

> *Software Engineering* is the designing and developing high-quality software. Application of computer science techniques to a variety of problems.

According to Fritz Bauer [NAU69].....

> *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.

According to Boehm [Boe81], from the economic and human perspective…

> *Software engineering* is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.

The IEEE [IEE87] defines Software engineering as..

> The systematic approach to the development, operation, maintenance, and retirement of software.

The IEEE [IEE93] defines Software engineering as..

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

 Downloaded from: sarojpandey.com.np

Difference between software engineering and computer science

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software

Computer science theories are currently insufficient to act as a complete underpinning for software engineering

Difference between software engineering and system engineering

System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process

System engineers are involved in system specification, architectural design, integration and deployment

Generic view of Software Engineering

Engineering is the analysis, design, construction, verification and management of technical (or social) entities. The questions being asked are:

- What is the problem to be solved?
- What characteristics of the entity are used to solve the problem?
- How will the entity (and the solution) be realized?
- How will the entity be constructed?
- What approach will be used to uncover errors that were made in the design and construction of the entity?
- How will the entity be supported over the long term, when corrections, adaptations and enhancement are requested by users of the entity?

Generic phases of Software Engineering

1. Definition phase
2. Development Phase
3. Support Phase

1. Definition phase

- It focus on **WHAT** i.e. to identify what is to be processed.
- What functions and performance are desired?
- What design constraints exist?
- What validation criteria are required to define a successful system?
- Identify the key requirements of the systems and the software?
- It perform three major tasks as
  - System or Information engineering
  - Software project planning
  - Requirement analysis

 Downloaded from: sarojpandey.com.np

2. Development phase

- It focus on **HOW** i.e. to define how data are to be structured.
- How function is to be implemented within a software architecture?
- How procedural details are to be implemented?
- How interface are to be characterized?
- How the design will be translated into a programming language (or nonprocedural language)
- How testing will be performed?
- It perform three specific technical tasks as
  - Software design
  - Code generation
  - Software testing

3. Support

It focus on change associated with *error correction, adaptations* required as the software environment evolves, and changes due to enhancements brought by the changing customer requirements.

Four types of changes are encountered during the support phases:

☑ Correction
- Use of *corrective maintenance* for any error encountered during support.

☑ Adaptation
- Use of *adaptive maintenance* to accommodate changes in the external environment.
- Example: Change in the CPU, OS, Business rules, external product characteristics)

☑ Enhancement
- Use of *perfective maintenance* that extends the software beyond its original functional requirements.

☑ Prevention
- Use of *preventive maintenance* by conducting software re-engineering to avoid software deterioration and to correct, adopt or enhance its features.

Software Engineering Umbrella activities
Software project tracking & control
Formal technical reviews
Software configuration management
Document preparation and production
Reusability management
Measurement
Risk management

Note: These umbrella activities overlay the process model, and are independent of any one framework activity (Software engineering work tasks, Project milestones, Work products, Quality assurance points) and occur throughout the process.

 Downloaded from: sarojpandey.com.np

Software Engineering: A layered technology

Software engineering is a ***layered technology*** as referred in fig shown below. It consists of four layers as

- A quality focus
- Process
- Methods
- Tools

A quality Focus
The bedrock that supports software engineering is a quality focus. Software engineering approach must rest on the organizational commitment to quality. Total Quality Management (TQM) philosophies that support continuous improvement culture leads to the development of increasingly more approaches to software engineering.

Process Layer
It is the foundation of software engineering that defines a framework for a set of *KPA* (*key process areas*) that must be established for effective delivery of software engineering technology. The KPA form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms etc) are produced, milestones are established, quality is ensured, and change is properly managed.

Methods
It provides technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support. Methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools
It provide automated or semi-automated support for the process and methods. Integrated tools establish computer-aided software engineering (CASE). CASE combines software, hardware and software engineering database to create a software engineering environment analogous to CAD/CAE (computer aided design/engineering) for hardware.

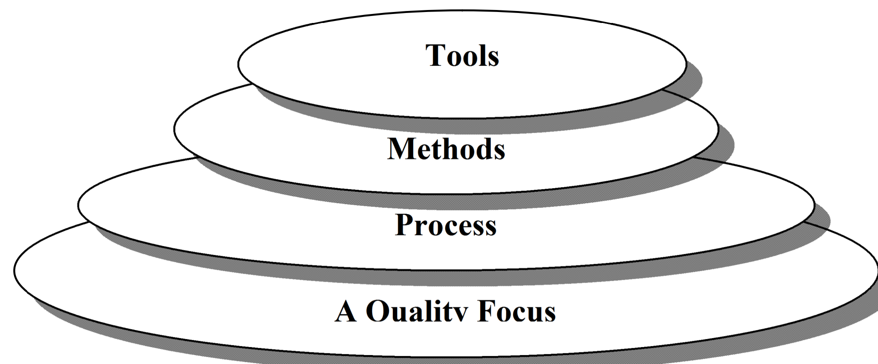


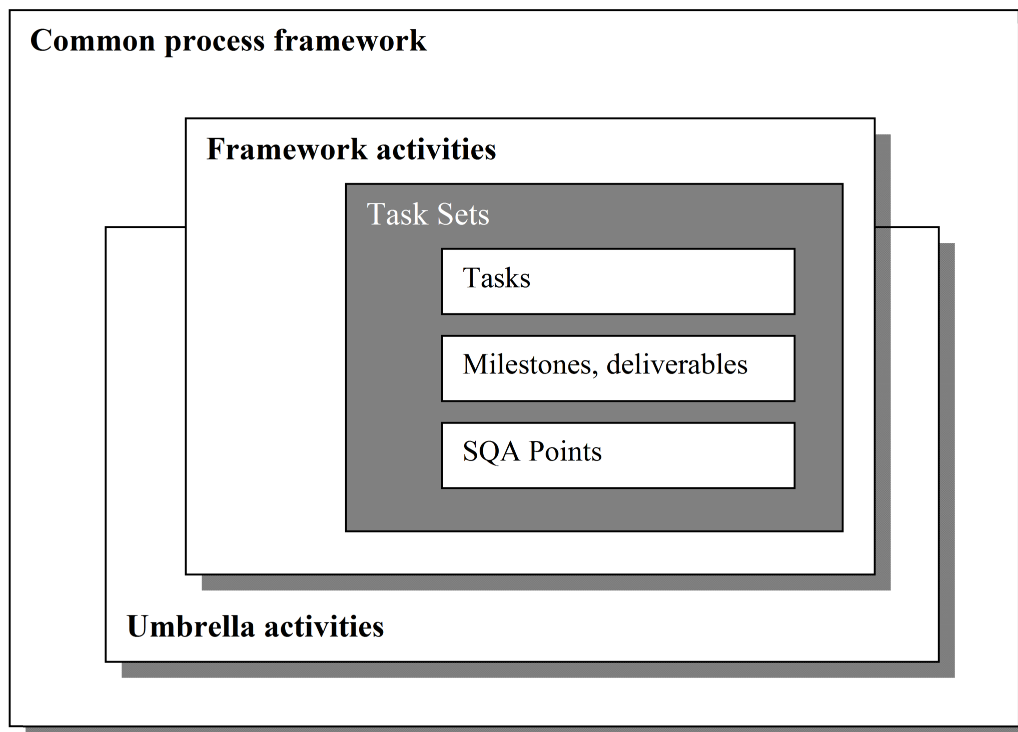
Fig: Software Engineering Layer

 Downloaded from: sarojpandey.com.np

Software Process

- The roadmap to building high quality software products is software process.
- Software processes are adapted to meet the needs of software engineers and managers as they undertake the development of a software product.
- A software process provides a framework for managing activities that can very easily get out of control.
- Different projects require different software processes.
- The software engineer's work products (programs, documentation, data) are produced as consequences of the activities defined by the software process.
- The best indicators of how well a software process has worked are the *quality*, *timeliness*, and long-*term viability* of the resulting software product.
- **A structured set of activities whose goal is the development or evolution of software**
- Generic activities in all software processes are:
    - Specification - what the system should do and its development constraints
    - Design/Development - production of the software system
    - Validation - checking that the software is what the customer wants
    - Evolution - changing the software in response to changing demands

Common Process Framework



A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

A number of task sets includes:

- Software engineering work tasks

 Downloaded from: sarojpandey.com.np

- Project milestones
- Work products
- Quality assurance points

These task sets enable the framework activities to be adopted to the characteristics of the software project and the requirements of the project team.

SEI-CMM (Capability Maturity Model)

To determine an organization's current state of process maturity, the SEI uses an assessment that re-suits in a five point grading scheme to determines compliance with a **CMM** (capability Maturity Model)

SEI approach provides a measures of the global effectiveness of a company's software engineering practices and establishes five process maturity levels. CMM defines key activities required at different levels of process maturity.

Level	Description
Level 1 : **Initial**	☑ Ad hoc and occasionally even chaotic software processes. ☑ Few processes are defined, and success depends on individual effort.
Level 2 : **Repeatable**	☑ Able to repeat earlier successes ☑ Establish basic project management to track cost, schedule & functionality
Level 3 : **Defined**	☑ Management and engineering processes documented, standardized, and integrated into organization-wide software process
Level 4 : **Managed**	☑ software process and products are quantitatively understood and controlled using detailed measures
Level 5 : **Optimizing**	☑ Continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas & technologies.

Key Process Areas (KPA)

Each maturity level is associated with KPA. KPA describe those software Engineering functions (e.g. software project planning, requirement management) that must be present to satisfy good practice at a particular level.

Characteristics of KPA includes:

- Goals : the overall objectives that the KPA must achieve.
- Commitments : requirements that must be met to achieve goals
- Abilities : those things(organizationally & technically) to meet the commitments
- Activities : specific tasks required to achieve KPA function.

 Downloaded from: sarojpandey.com.np

- Monitoring : manner in which the activities are monitored.
- Verification : manner in which proper practice for the KPA can be verified.

18 KPAs are defined across the maturity model & mapped into different levels of process maturity:

Process maturity level 2

Software configuration management
Software quality assurance
Software subcontract management
Software project planning
Requirements management

Process maturity level 3

Peer review
Intergroup co-ordination
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 Models

A software process model or *software engineering paradigm* is a development strategy that encompasses the process, methods and tools layers.

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

All software development process can be characterized as a **problem solving loop** as shown in figure that consists of four distinct stages such as:

- Status Quo : represents the current state of affairs.
- Problem definition : identifies the specific problem to be solved.

 Downloaded from: sarojpandey.com.np

- Technical development : Solve the problem through the application of some technology

- Solution Integration : delivers the results (e.g., document, programs, data, new business functions, new products)

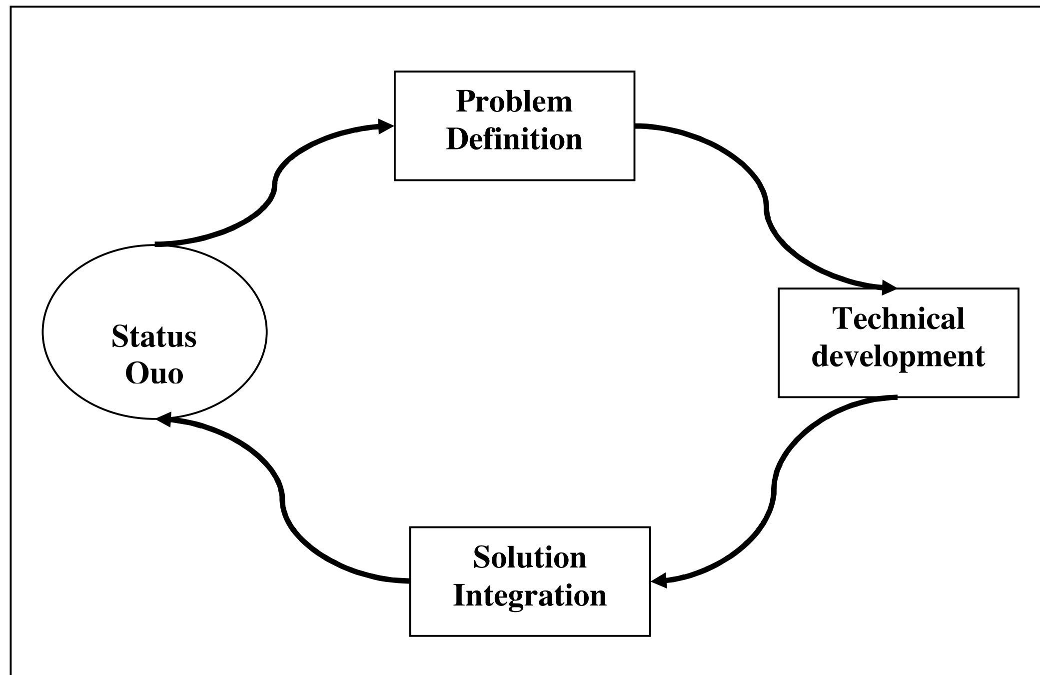


Fig : Problem solving loop

Types of Process Models

A process model is chosen based on the nature of the project and application, the methods and tools being used, and the controls and deliverables that are required.

The following types of process models are being used in the software development:

- **Linear Sequential Model**
  - old fashioned but reasonable approach when requirements are well understood
  - Separate and distinct phases of specification and development

- **Prototyping Model**
  - good first step when customer has a legitimate need, but is clueless about the details, developer needs to resist pressure to extend a rough prototype into a production product.

- **Rapid Application and Development (RAD) Model**
  - makes heavy use of reusable software components with an extremely short development cycle

 Downloaded from: sarojpandey.com.np

- **Evolutionary Software process Model**
    - **Incremental Model**
        - delivers software in small but usable pieces, each piece builds on pieces already delivered

    - **Spiral Model**
        - couples iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model

- **Component-Based Development**
    - spiral model variation in which applications are built from prepackaged software components called classes

1. Linear Sequential Model

- This model is also known as the Waterfall model or software life cycle.
- The principal stages of the model map onto fundamental development activities:
    - **Requirements analysis and definition**
        - system's services, constraints and goals are established.
        - details are served as system specifications.
    - **System and software design**
        - partitions the requirements to either hardware or software systems.
        - establishes an overall system architecture.
        - identifies and describes the fundamental software system abstractions and their relationships
    - **Implementation and unit testing**
        - software design is realized as a set of programs or programs units.
        - Unit test is performed to verify each unit meet its specification.
    - **Integration and system testing**
        - Individual program units or programs are integrated.
        - Perform test to ensure that the requirements are met.
    - **Operation and maintenance**
        - It is the longest life cycle phase.
        - The system is installed and put into practical use.
        - Maintenance involve error correction and improving the implementation of system units
        - Enhance the system service to meet new requirements.
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

 Downloaded from: sarojpandey.com.np

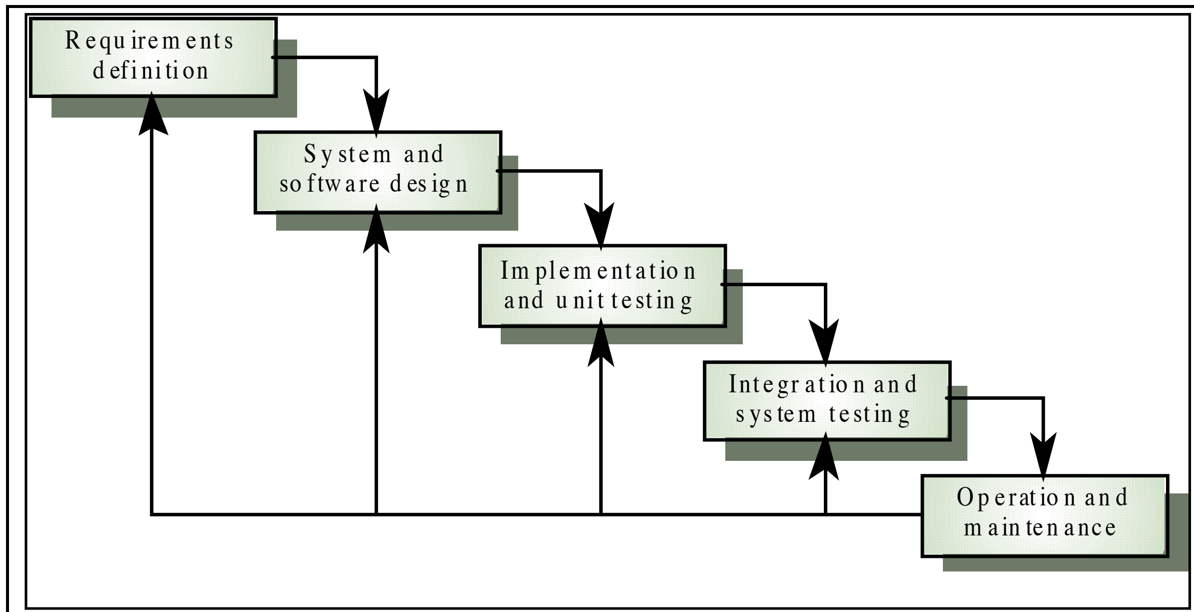


Fig: Waterfall Model

In principle, the result of each phase is one or more approved ("signed off") documents. The following phase should not start until the previous phase has finished. In practice, these stages overlap and feed information to each other. During design, problems with requirements are identified, during coding design problems are found and so on. The software process is not a simple linear model but involves a sequence of iterations of the development activities.

Because of the costs of producing and approved documents, iterations are costly and involve significant rework. Therefore, after a small number of iterations, it is normal to freeze parts of the development, such as the specification, and to continue with the later development stages. Problems are left for later resolution, ignored or are programmed around. This premature freezing of requirements may means that the system won't do what the user wants. It may also lead to badly structured systems.

Waterfall Model problems

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood

2. Prototyping Model

- A prototyping paradigm offer the best approach when.....
  - a customer defines a set of general activities for software but does not identify detailed input, processing, or output requirements.
  - the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system

 Downloaded from: sarojpandey.com.np

- It begins with requirement gathering where developer and customer, together, define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.

- Develop a " quick Design " that focus on input approaches and output formats that is visible to the customer/user. It leads to the construction of a prototype, that serves as a mechanism for identifying software requirements

- Prototype is evaluated by the customer/user and refine the requirements for the software to be developed.

- The prototype serves as " the first system" where users get a feel for the actual system and developers get to build something immediately.

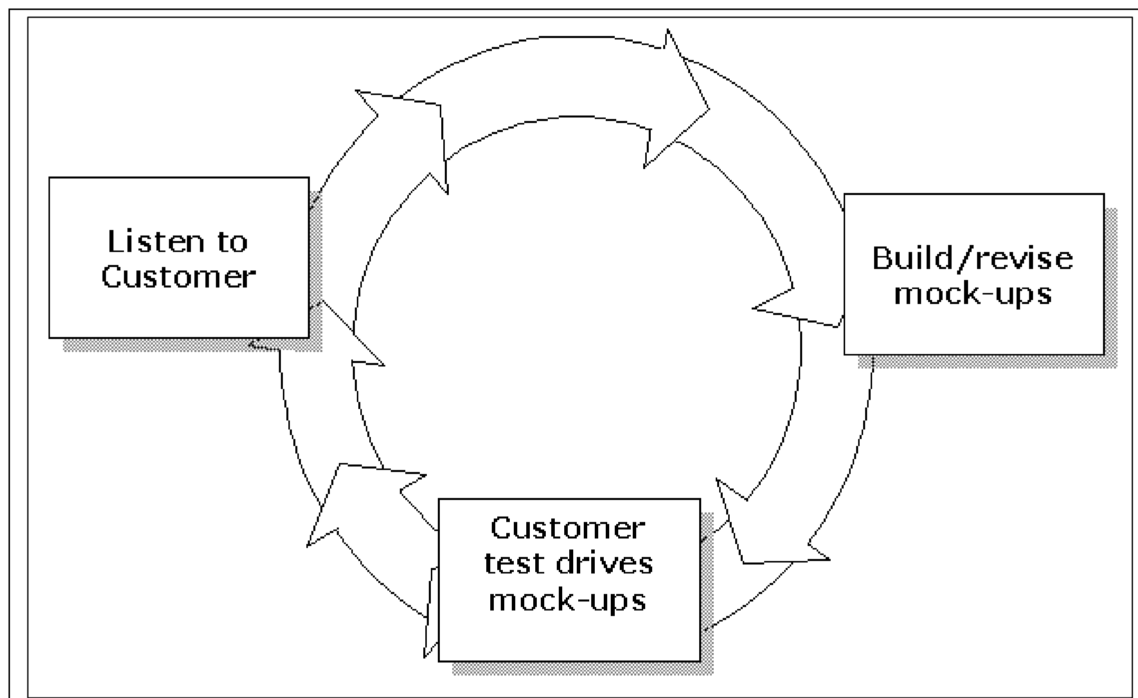


Fig: The prototyping paradigm

In this model, the key is to define the rules of the game at the beginning, that is, the customer and developer must agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded and the actual software is engineered considering quality and maintainability.

The prototype Model problem

- Prototype appears to be the working version of the software, but all software quality and long-term maintainability has not been considered. The final must be rebuilt.
- Implementation comprises for quick working of prototype.

 Downloaded from: sarojpandey.com.np

- Use of inappropriate Operating system or programming language
- Implementation of inefficient algorithm to demonstrate capability.

3. The RAD Model

RAD (Rapid application development) is an incremental software development process model with extremely short development cycle.

It is a "*high-speed*" adaptation of the linear sequential model in which rapid development by using component-based construction.

RAD process enable to create a "*fully functional system*" within very short time periods (e.g. 60-90 days) if requirements are well understood and project scope is constrained.

Used primarily for information systems applications, the RAD approach encompasses the following phases:

- **Business modeling**
    - The information flow among the business functions is modeled to answer the following questions:
        - What information drives the business process?
        - What information is generated?
        - Who generates it?
        - Where does the information go?
        - Who process it?

- **Data modeling**
    - The information defined in the business-modeling phase is refined into a set of data objects that are needed to support the business.
    - The characteristics (attributes) of each object are identified and the relationship between these objects is 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**
    - Use of forth generation techniques
    - Reuse existing program components or create reusable components.
    - Automated tools are used to facilitate construction of the software.

- **Testing and turnover**
    - Since RAD process emphasizes reuse, many of the program components have already been tested, that reduces overall testing time.

 Downloaded from: sarojpandey.com.np

- o Testing of new components and interfaces are done.

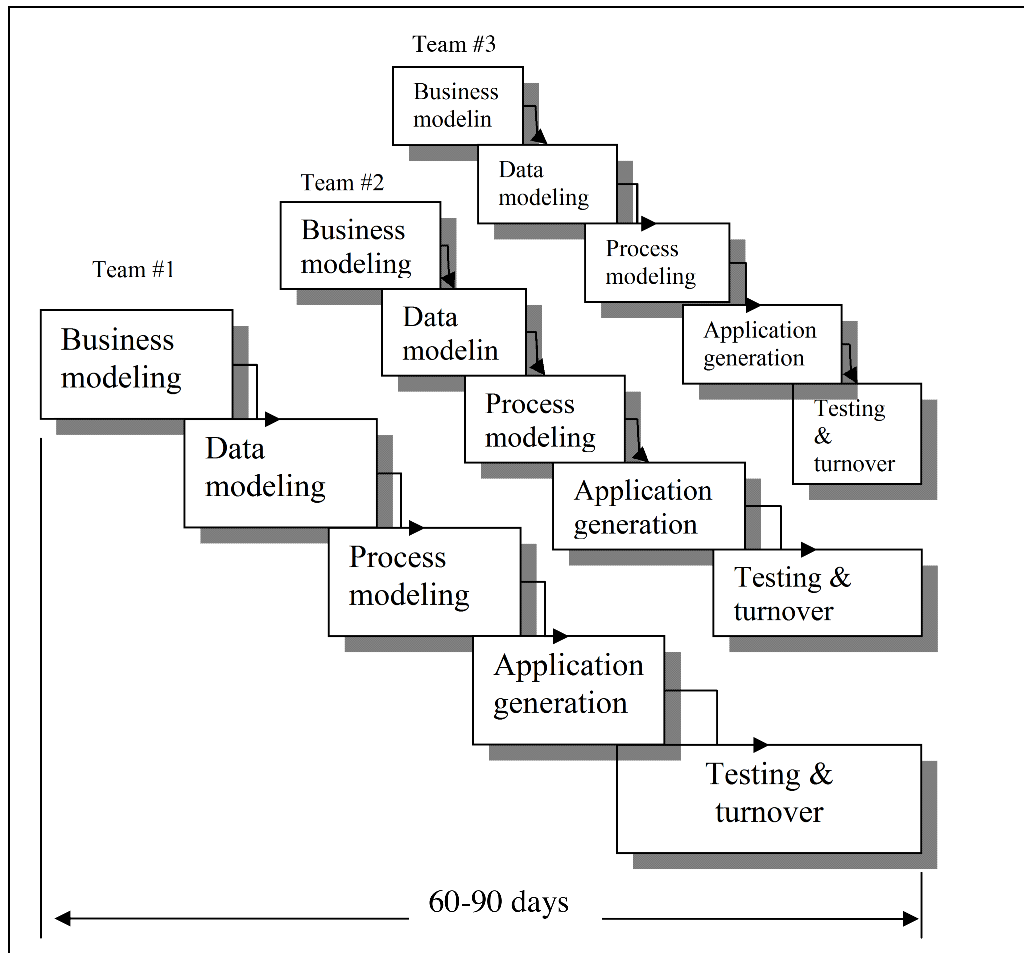


Fig: The RAD Model

Drawback of RAD Model

- For large and scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to the *rapid-fire* activities necessary to get a system complete in a much-abbreviated time frame. In the lack of commitment from either side, RAD project fails.
- If a system can not be properly modularized, building the components necessary for RAD will be problematic.
- RAD approach can not be used for high performance.
- RAD is not appropriate when technical risks are high.

4. Evolutionary Software Process Models

- The software system ALWAYS evolves over a period of time in the course of a project (due to change in the business and products requirements) so process iteration where earlier stages are reworked is always part of the process for large systems.

 Downloaded from: sarojpandey.com.np

The linear sequential model is designed for straight for straight line requirement. In essence, this waterfall approach assumes that a complete system will be delivered after the linear sequence is completed. The prototyping model is designed to assist the customer (or developer) in understanding requirements. In general, it is not designed to deliver a production system. The evolutionary nature of software is not considered in either case of these classic software engineering paradigms.

- Iteration can be applied to any of the generic process models
- Two (related) approaches
    - Incremental development model
    - Spiral model

4.1 Incremental Model

- It combines the elements of the linear sequential model with the iterative philosophy of prototyping.
- It applies the linear sequences in the staggered fashion as calendar time progresses.
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritised and the highest priority requirements are included in early increments
- Each linear sequence produces a deliverable "increment" of the software, where the process flow for any increment can incorporate the prototyping paradigm.
- The first increment results in a *core product* where the basic requirements are addressed, but many supplementary features remains undelivered.
- After the deployment of the core product and its evaluation, a plan is developed for the next increment, which addresses the modification of the core product to better meet the needs of the customer and delivery of additional features and functionality. This process is repeated following the delivery of each increment, unit the complete product is produced.

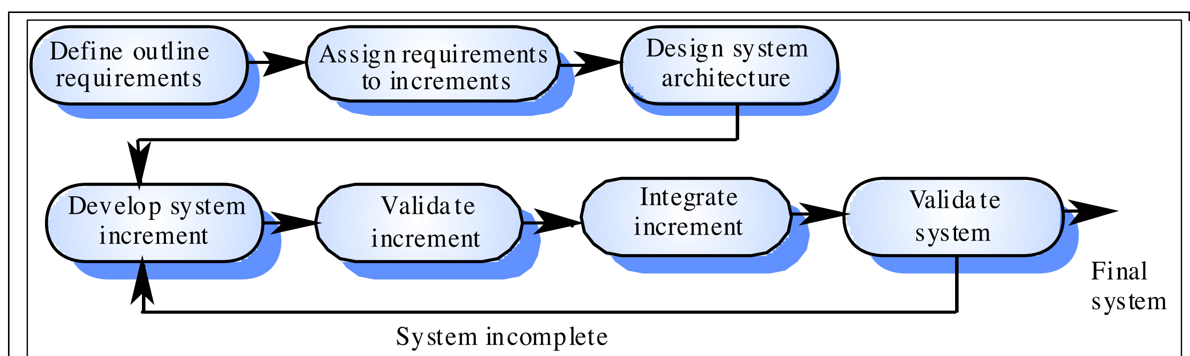


Fig: Incremental development

 Downloaded from: sarojpandey.com.np

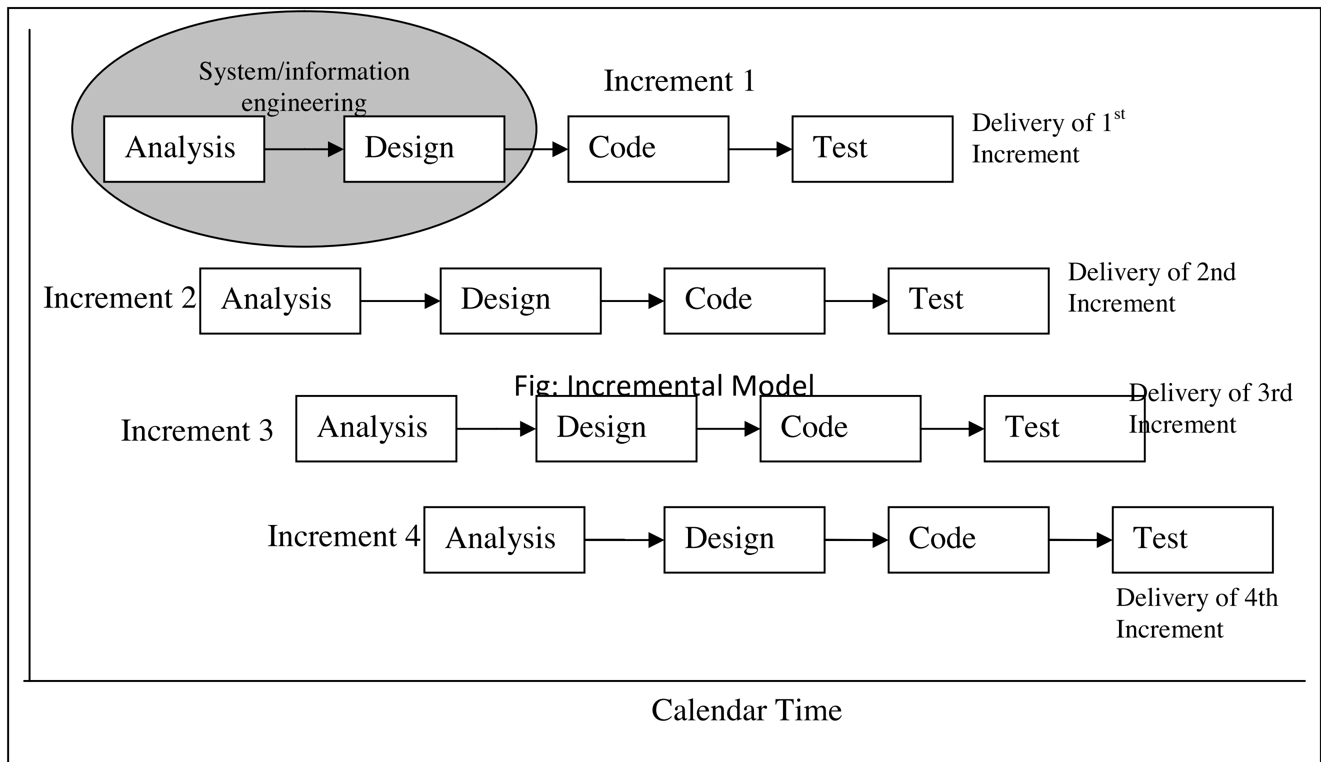


Fig: Incremental Model

Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

4.2 Spiral Model

- As proposed by Boehm, it is a risk-driven approach to software development that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. i.e. it uses prototyping as a risk reduction mechanism and retains the systematic step-wise approach of the waterfall model.
- Software is developed in a series of incremental releases.
- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process.
- Model is divided into a number of framework activities, called ***task regions*** that contains:
    - Customer communication: task required to establish effective communication between developer and customer.

    - Planning: task required to define resources, timeliness, and other project related information.

 Downloaded from: sarojpandey.com.np

- Risk analysis: task required to assess both technical and management risk.
- Engineering: task 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)

Spiral model of the software process

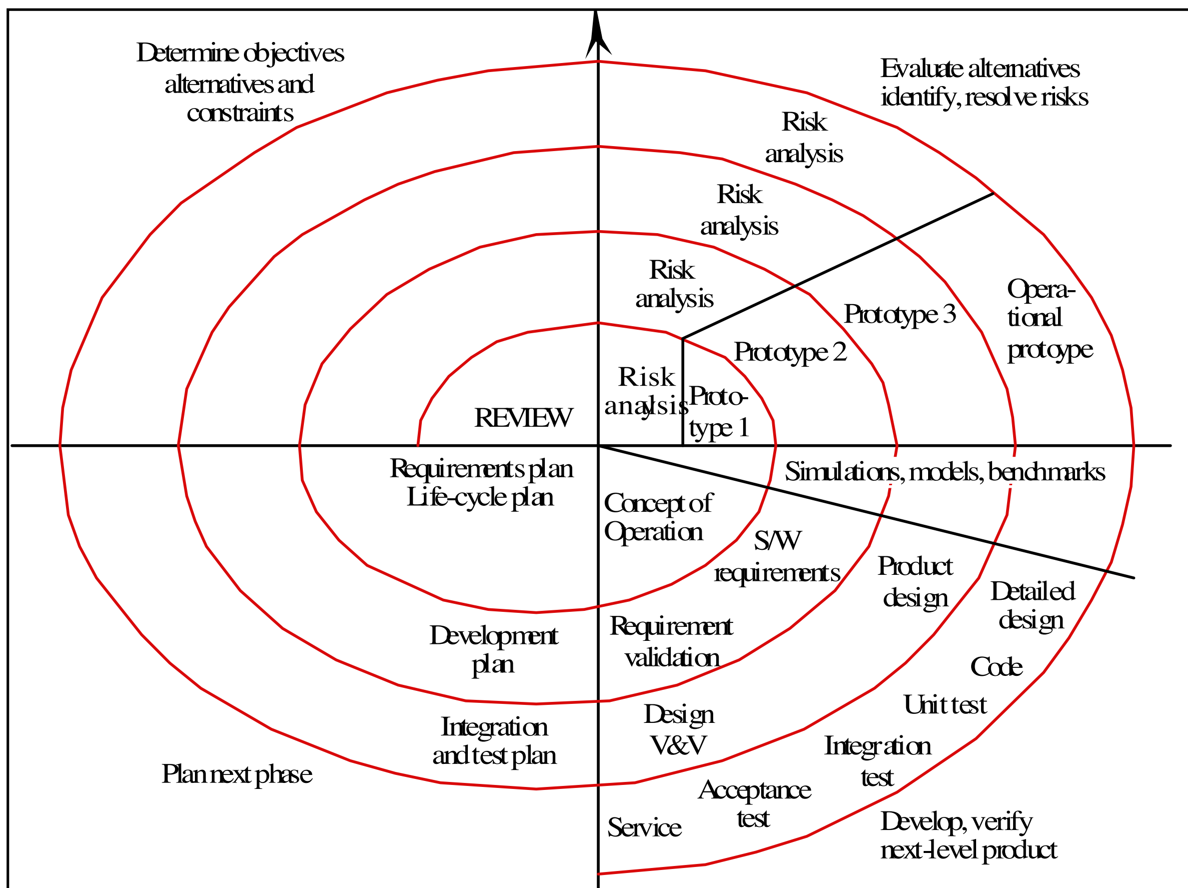


Fig: Boehm Spiral Model

Spiral model sectors

- **Objective setting**
  - Specific objectives for the phase are identified; alternative solutions and its constraints are defined.
  -
- **Risk assessment and reduction**
  - Alternative solutions are evaluated and the potential project risks are assessed and activities put in place to reduce the key risks by developing an appropriate prototype

 Downloaded from: sarojpandey.com.np

- **Development and validation**
  - A development model for the system is chosen which can be any of the generic models
- **Planning**
  - The project is reviewed and the next phase of the spiral is planned

Spiral Model as Meta Model

Spiral Model can be viewed as a *Meta model* since it encompasses all the previously discussed models, and uses prototyping as a risk reduction mechanism. For example, a single loop spiral actually represents the waterfall model. The spiral model uses an evolutionary approach and iterations along the spiral can be considered as evolutionary levels through which the complete system is built. The spiral model enables the developer to understand and react to the risks at each evolutionary level (i.e. iteration along the spiral). The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.

Advantages of Spiral Model

1. It could be used as effectively for system enhancement as for system development.
2. Most life cycle models can be considered as a special case of the spiral models, and
3. The embedded risk analysis built into the model avoids many of the difficulties that arise in other models.

5. Component-based development (CBD) model

Object-oriented technologies provide the technical framework for a component-based process model.
It emphasizes the creation of classes that encapsulate both data and the algorithms used to manipulate the data.
Properly designed and implemented object-oriented classes are reusable across different applications and computer-based system architectures.
It incorporates the characteristics of the spiral model and evolutionary in nature, demanding an iterative approach to the creation of software.
It composes applications from prepackaged software components (called classes)
It performs the following activities:

Identification of candidate classes which is accomplished by examining the data to be manipulated by the applications and the algorithms that will be applied to accomplish the manipulation.
Search for the class library from its library or repository of classes created earlier.
If available, *extract class components* for reuse, otherwise, develop using object-oriented methods.
Compose the first iteration of the application to be build, using the extracted classes from the library and any new classes built to meet the unique needs of the application.
Process flow returns to the spiral and re-enter the component assembly iteration during subsequent passes.

 Downloaded from: sarojpandey.com.np

Advantages of the CBD Model

- Reusability of the software.
- Reduction in the development cycle time
- Reduction in the project costs.
- Increase in the productivity

Comparison of different Life Cycle Models

The classical waterfall model with feedback is the most widely used software development model evolved so far. However, the iterative waterfall model is suitable only for well-understood problems. Further, the waterfall model is too simplistic. There are no milestones in the model, no mention of the documents generated, or the reviews conducted, and no indication of the quality assurance activities. Establishing milestones, review points, standardized documents, and management sign-offs improve product visibility. Without sufficient product visibility, it becomes very difficult to manage a software development project. If we cannot see something, we cannot hope to manage it.

The prototype model is suitable for projects whose user requirements or the underlying technical aspects are not well understood.

The evolutionary approach is suitable for large problems, which can be decomposed into a set of modules that can be implemented incrementally, and where incremental delivery of the system is acceptable to the customer.

The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.

Summary

- Software engineering is a discipline that integrates process, methods, and tools for the development of computer software.
- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Iterative process models describe the software process as a cycle of activities
- Requirements engineering is the process of developing a software specification
- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets to its specification and user needs
- Evolution is concerned with modifying the system after it is in use.
- Spiral model uses prototyping as a risk reduction mechanism and retains the systematic step-wise approach of the waterfall model.

 Downloaded from: sarojpandey.com.np

Assessment Questions

1. Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model which might be used as a basis for managing the development of the following systems:
   i. A system to control anti-lock braking in a car.
   ii. A virtual reality system to support software engineering maintenance.
   iii. A university accounting system that replace an existing system
   iv. An interactive system for railway passengers that might finds train times from terminals installed in stations.
   v. A well understood EDP applications
   vi. A new software product that would connect computers through satellite communication. Assume that your team has no previous experience in developing satellite communication software.
   vii. A software product that would function as the controller of a telephone switching system.
   viii. A new library automation software that would link various libraries in the city.
   ix. An extremely large software that would provide, monitor, and control cellular communication among its subscribers using a set of revolving satellites.
   x. A new text editor.
   xi. A compiler for a new language

2. Explain why programs, which are developed using evolutionary development, are likely to be difficult to maintain.

3. What do you understand by the common process framework of software development? List the various umbrella activities that are being complemented by generic view of software engineering.
4. Explain how both the waterfall model of the software process and the prototyping model can be accommodated in the spiral process model.

5. Explain why the spiral model is considered to be a Meta model. Compare the relative advantages of using the waterfall model and the spiral model of software development.

6. Explain why a software system that is used in a real-world environment must change or become progressively less useful.

7. IS there ever a case when the generic phases of the software engineering process don't apply? If so, describe.

8. The SEI's capability maturity model (CMM) is an evolving document. Explain the various KPAs mapped into each level.

9. Which of the software engineering paradigms do you think would be most effective? Why?

10. As you moved outward along the process flow path of the spiral model, what can you say about the software that is being developed or maintained?

11. Which is more important- the product or the process?
12. "Software Engineering is a layered technology". Justify your answer.
13. Explain the term Software and Software Engineering in your own words.

14. It is true to say that information is going to be seized during waterfall model? Explain

15. What do you mean by the term life cycle model of software development? Why is it important to adhere to a life cycle model while developing a large software product?

16. What is prototype model? Under what circumstances it beneficial to construct a prototype model? Does the construction of a prototype model always increase the overall cost of software development?

 Downloaded from: sarojpandey.com.np

Chapter 3 - System Engineering - Overview

Software engineering occurs as a consequence of a process called *system engineering.* Instead of concentrating solely on the software, system engineering focuses on a variety of elements, analyzing, designing, and organizing those elements into a system that can be a product, a service, or a technology for the transformation of information or control.

Before software can be engineered, the system it is part of must be understood. The overall objective of the system must be determined, the role of the system elements (hardware, software, people, data, etc.) must be identified, and the operational requirements must be created. A representation (i.e. prototype, specification, symbolic model) of the system is produced as the result of the system engineering process. It is important that system engineering work products be managed using the quality assurance techniques because problems of software engineering are often a result of system engineering decisions.

The system engineering process is called *business process engineering* when the context of the engineering work focuses on a business enterprise. When a product (e.g., air traffic control) is to be built, the process is called *product engineering*. The term software engineering is generic and is used to encompass both business process engineering and product engineering.

System engineering is the activity of specifying, designing, implementing, validating, deploying, and maintaining systems as a whole.

Systems

- Don't take a "*software-centric*" view of the system; consider all system elements before focusing on software.
- Good system engineering begins with a clear understanding of the "*world view*" and progressively narrows until technical detail is understood.
- Complex systems are actually a hierarchy of macro-elements that are themselves systems.

A system is a purposeful collection of interrelated components that work together to achieve some objectives.

Computer-Based System

System is Defined as ...

1. a set or arrangement of things so related as to form a unity or organic whole;
2. a set of facts, principles, rules etc., classified and arranged in an orderly form so as to show a logical plan thinking the various parts;
3. a method or plan of classification or arrangement;
4. an established way of doing something; method; procedure...

Computer-Based System is defined as a set or arrangement of elements that are organized to accomplish some predefined goal by processing information.

 Downloaded from: sarojpandey.com.np

Computer-Based System Elements

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

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

- People: Users and operator of hardware and software.

- Database: A large, organized collection of information that is accessed via software.

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

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

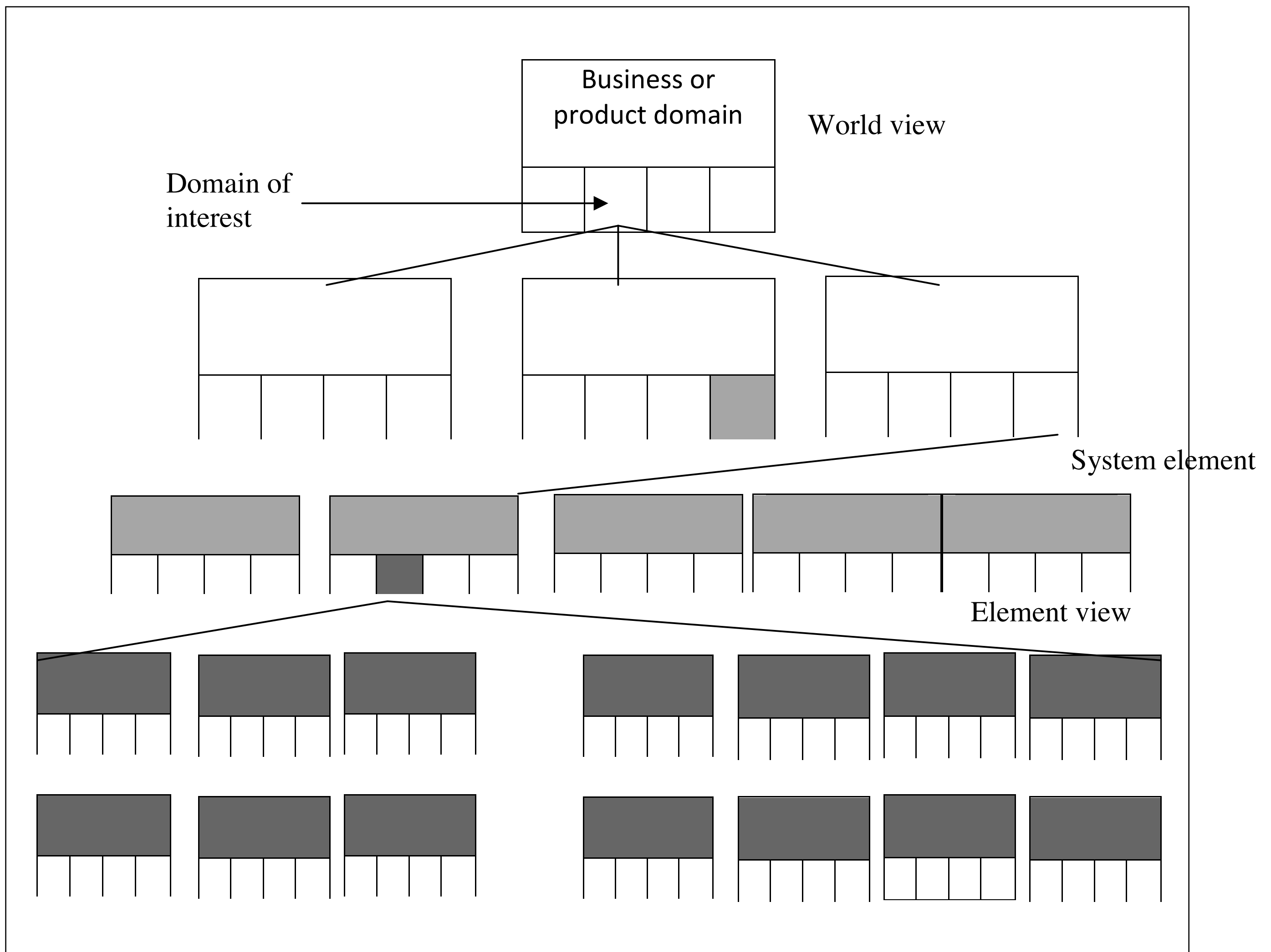
System Engineering Hierarchy

4. World view
5. Domain view
6. Element view
7. Detailed view

System Engineering encompasses a collection of top-down and bottom-up methods to navigate the hierarchy as shown in the fig below. The system engineering process 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 systems (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 and, at the bottom, detailed technical activities are performed.

 Downloaded from: sarojpandey.com.np



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

WV=[D1,D2,D3,........,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,............,En]

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

Ej=[C1,C2,C3,............,Ck]

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

 Downloaded from: sarojpandey.com.np

System Modeling

System Engineering is a modeling process. Whether the focus is on the world view or the detailed view, the model.....

- Define the processes that serve the needs of the view under consideration
- Represent the process behavior and the assumptions on which the behavior is modeled
- Explicitly define the exogenous (links between constituents) and endogenous (links between constituent components) input to the model
- Represent all linkages (including outputs) required to understand the view

System Model Restraining Factors

- **Assumptions**: It reduces the number of possible permutations and variations, enabling a model to reflect the problem in a reasonable manner.
- **Simplifications** : It enable the model to be created in a timely manner. e.g., generation of service order by understanding the internal demand and external request.
- **Limitations**: It helps to bound the system
- **Constraints**: It will guide the manner in which model is created and the approach taken when the model is implemented. e.g., computational complexity of problems imposed by the processor.
- **Preferences** : It indicates the preferred architecture for all data, functions, and technology

System Model Template

- User interface
- Input
- Process and control functions
- Output
- Maintenance and self test

Systems Modeling Process

- **System Context Diagram (SCD)** - top level node in system hierarchy used to establish the boundary between the system being implemented (system model template serves as its basis)
- **System Flow Diagram (SFD)** - refinement of the process and control functions from SCD, derived by identifying the major subsystems and lines of information flow. Initial SFD is becomes the top level node of a hierarchy of more successively more detailed SFD's
- **System Specification** - developed by writing narrative description for each subsystem and definitions for all data that flow between subsystems

 Downloaded from: sarojpandey.com.np

System Simulation

If simulation capability is not available for a reactive system, project risk increases.
Consider using an iterative process model that will allow the delivery and testing of incrementally more complete products.

Business Process Engineering- Overview

Goal of BPE is to define the architectures that will enable a business to use information effectively. BPE is one approach for creating an overall plan for implementing the computing architecture.

Business Process Engineering Architectures

- Data architecture - provides framework for information needs of a business or business function

- Applications architecture - those system elements that transform objects within the data architecture for some business purpose

- Technology infrastructure- provides foundation for the data and application architectures

Data Architecture

The individual building blocks of the architecture are the data objects (also called entities) that are used by the business. At the business level, typical data objects include: producers and consumers of information (e.g., a customer), things (e.g., a report), occurrences of events (e.g., a sales conference), organizational roles (e.g., a Vice President of Engineering); organizational units (e.g., Sales and Marketing), places (e.g., manufacturing cell), or information structures (e.g., an employee file). A data object contains a set of attributes that define some aspect, quality, characteristic or descriptor of the data that is being described. For example, during enterprise modeling an information engineer might define the data object: customer. To more fully describe customer, the following attributes are defined:
Object: Customer
Attributes:

name
company name
job classification and purchase authority
business address and contact information
product interest(s)
past purchase(s)
date of last contact
status of contact

Once a set of data objects is defined, their relationships are identified. A *relationship* indicates how objects are connected to one another. As an example, consider the objects: customer, product A, and salesperson. An information engineer creates a entity relationship diagram (E-R diagram) to depict these relationships. Relationships imply a connection between data objects. In general,

 Downloaded from: sarojpandey.com.np

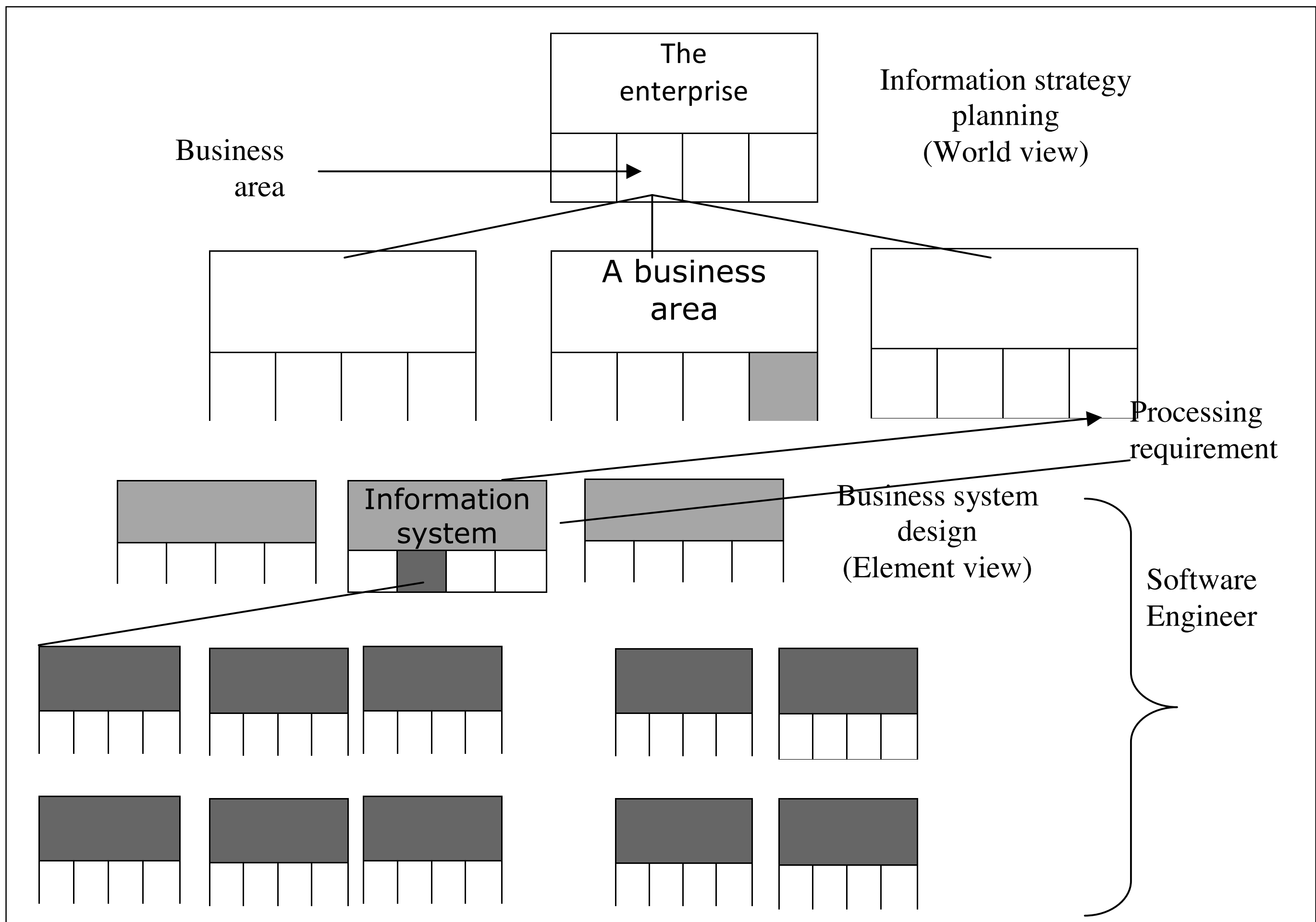
relationship can be read in either direction; hence, a customer *purchases* product A and product A *is purchased by* a customer.

Application Architecture

It encompasses those elements of a system that transform objects within the data architecture for some business purpose. It incorporates the role of people and business procedures that have not been automated.

Technology Infrastructure

It provides the foundation for the data and application architectures. The infrastructure encompasses the hardware and software that are used to support the application and data. This includes computers, operating systems, networks and telecommunication links, storage technologies, and the architecture (e.g., client/server) that has been designed to implement these technologies.



Business Process Engineering Hierarchy

- Information Strategy Planning (world view)
- Business Area Analysis (domain view)
- Business System Design (element view - software engineers)

 Downloaded from: sarojpandey.com.np

- Construction and Integration (detailed view - software engineers)

As shown in fig, the world view is achieved through information strategy planning (ISP). ISP view the entire business as an entity and isolates the domain of the business (e.g., engineering, manufacturing, marketing, finance, sales). ISP defines the data objects that are visible at the enterprise level, their relationships, and how they flow between the business domains.

The domain view is addressed with a BPE activity called *Business Area Analysis (BAA).*

Martin [MAR90] describes business area analysis in the following manner:

Business areas analysis (BAA) establishes a detailed framework for building an information-based enterprise. It takes one business area at a time and analyzes it in detail. It uses diagrams and matrices to model and record the data and activities in the enterprise and to give a clear understanding of the elaborate and subtle ways in which the information aspects of the enterprise interrelate.

Hares [HAR93] describes BAA in the following manner:

BAA is concerned with identifying in detail data (in the form of entity or data objects) and function requirements (in the form of processes) of selected business areas (domains) identified during ISP and ascertaining their interactions (in the form of metrics). It is only concerned with specifying what is required in a business area.

During BAA, focus shifts from the world view to the specific business domain view. To model "the elaborate and subtle ways in which the information aspects of the enterprise interrelate," the information engineer must depict how data objects are used and transformed within each business area and how the business functions and processes within each business area transform these data objects. To accomplish this work, BAA makes use of a number of different models:

• data models (now refined to the business area level)
• process flow models
• process decomposition diagrams
• a variety of cross-reference matrices

The data objects defined during ISP are refined for use within each business area. For example, the data object **customer** described in the preceding section is used by the sales department. After evaluation of the needs of the sales department (an analysis of the sales domain), the original definition of **customer** is further refined to meet the needs of sales:

Object: **Customer**
Attributes:

name
company name →**Object: Company**
job classification and purchase authority
business address and contact information

 Downloaded from: sarojpandey.com.np

product interest(s)
past purchase(s)
date of last contact ➝record of contacts
status of contact ➝status of last contact
➝next contact date
➝recommended nature of contact

The attribute *company name* has been modified to point to another object called **Company.** This object contains not only the company name but additional information about the size of the company, its purchasing requirements, the name of other contacts, etc. This information will be useful in the sales domain. Other attributes have been modified and added as noted above.

Process Modeling

The work performed within a business area encompasses a set of business functions that are further refined into business processes. To illustrate, consider a simplified version of the sales function discussed earlier. The processes the occur to accomplish sales are:

Sales function:

• Establish customer contact
• Provide product literature and related information
• Address questions and concerns
• Provide evaluation product
• Accept sales order
• Check availability of configuration ordered
• Prepare delivery order
• Confirm configuration, pricing, ship date with customer
• Transmit delivery order to order fulfillment
• Follow-up with customer

A process flow diagram can be developed for this sequence of processing. It should be noted that each business function relevant to the business area can be refined in a similar manner.

Information Flow Modeling

The process flow model is integrated with the data model to provide an indication of how information flows through a business area. Input and output data objects are shown for each process, providing an indication of how the process transforms information to accomplish a business function.

Once a complete set of process flow models have been created, the information engineer (along with others) examines how the existing process can be reengineered (e.g., [HAM93], [JAY94]) and where existing information systems or applications might be modified or replaced by more efficient information technologies. The revised process model is used as a basis for the specification of new or revised software to support the business function.

 Downloaded from: sarojpandey.com.np

The domain view established during BAA serves as the basis for *business system design* (BSD) and *construction and integration* (C&I). By invoking a business system design (BSD) step, the basic requirements of a specific information system are modeled and these requirements are translated into a data architecture, applications architecture, and technology infrastructure.

The final BPE step- construction and integration focuses on implementation detail. The architecture and infrastructure are implemented by constructing an appropriate database and internal data structures, by building applications using software components, and by selecting appropriate elements of a technology infrastructure to support the design created during BSD. Each of these system components must then be integrated to form a complete information system or application. The integration activity also places the new information system into the business area context, performing all user training and logistics support to achieve a smooth transition.

Product Engineering Hierarchy

- Requirements engineering (world view)
- Component engineering (domain view)
- Analysis and Design modeling (element view - software engineers)
- Construction and Integration (detailed view - software engineers)

(Note: For detail on Product Engineering Hierarchy, refer text book: A Software Engineering- A practitioner's approach by Roger S Pressman, page no. 254.)

Summary

A high-technology system encompasses a number of elements: Software, hardware, people, database, documentation and procedures. System engineering helps to translate a customer's needs into a model of a system that makes use of one or more of these elements.

System engineering begins by taking a "world view". A business domain or product is analyzed to establish all basic business requirements. Focus is then narrowed to a "domain view", where each of the system elements is analyzed individually. Each element is allocated to one or more engineering components, which are then addressed by the relevant engineering discipline.

Business process engineering is a system engineering approach that is used to define architecture that enable a business to use information effectively. The intent of business process engineering is to derive comprehensive data architecture, application architecture, and technology infrastructure that will meet the needs of the business strategy and the objectives and goals of each business area. Business process engineering encompasses Information Strategy Planning (world view), Business Area Analysis (domain view), Business System Design (element view), Construction and Integration (detailed view).

Product engineering is a system engineering approach that begins with system analysis. The system engineer identifies the customer's needs, determines economic and technical feasibility, and allocates function and performance to software, hardware, people and databases- the key engineering components.

 Downloaded from: sarojpandey.com.np

Assessment Questions

Q1. Business profess engineering strives to define data and application architecture as well as technology infrastructure. Describe what each of these terms means and provide an example.

Q2. Develop an abbreviated System Specification for the following computer-based systems:

- a University registration system
- an interactive hotel reservation system
- an interactive online learning system

Q3. Write short notes on

- ☑ System Modeling
- ☑ System Engineering Process (refer. Software Engineering by Ian Sommerville)
- ☑ Business Process Engineering (BPE) and/or Business process re-engineering
- ☑ Product Engineering

 Downloaded from: sarojpandey.com.np

Chapter 4 - Project Management Concepts

- *Project management involves the planning, monitoring, and control of people, process, and events that occur during software development.*
- *It is concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.*
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software
- Everyone manages, but the scope of each person's management activities varies according his or her role in the project.
- Software needs to be managed because it is a complex undertaking with a long duration time.
- Managers must focus on the fours P's to be successful (people, product, process, and project) i.e., it defines the process and task to be conducted, the people who will do the work, and the mechanisms for assessing risks, controlling change and evaluating quality.
- A project plan is a document that defines the four P's in such a way as to ensure a cost effective, high quality software product. i.e., it defines the process and tasks to be conducted, the people who will do the work, and the mechanisms for assessing risks, controlling change and evaluating quality.
- The only way to be sure that a project plan worked correctly is by observing that a high quality product was delivered on time and under budget.

Project management is the Organising, planning and scheduling software projects

Management Spectrum

17. People (The PM-CMM's Key practice areas includes recruiting, selection, performance management, training, compensation, career development, organization, work design, team/culture development)
18. Product (product objectives, scope, alternative solutions, constraint tradeoffs)
19. Process (framework activities populated with tasks, milestones, work products, and QA points)
20. Project (planning, monitoring, controlling)

People

- Players (senior managers, technical managers, practitioners, customers, end-users)
- Team leadership model (motivation, organization, skills)
- Characteristics of effective project managers (problem solving, managerial identity, achievement, influence and team building)

Note: *People Management- Capability Maturity Model (PM-CMM) enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability. PM-CMM is a companion to the software capability maturity model that guides organizations in the creation of a mature software process.*

Downloaded from: sarojpandey.com.np

Software Team Organization

- Democratic decentralized (rotating task coordinators and group consensus)
- Controlled decentralized (permanent leader, group problem solving, subgroup implementation of solutions)
- Controlled centralized (top level problem solving and internal coordination managed by team leader)

Factors Affecting Team Organization

- Difficulty of problem to be solved
- Size of resulting program
- Team lifetime
- Degree to which problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of communication required for the project
- Coordination and Communication Issues
- Formal, impersonal approaches (e.g. documents, milestones, memos)
- Formal interpersonal approaches (e.g. review meetings, inspections)
- Informal interpersonal approaches (e.g. information meetings, problem solving)
- Electronic communication (e.g. e-mail, bulletin boards, video conferencing)
- Interpersonal network (e.g. informal discussion with people other than project team members)

The Product

- Software scope (context, information objectives, function, performance)
- Problem decomposition (partitioning or problem elaboration - focus is on functionality to be delivered and the process used to deliver it)

The Process

- Process model chosen must be appropriate for the: customers and developers, characteristics of the product, and project development environment
- Project planning begins with melding the product and the process
- Each function to be engineered must pass though the set of framework activities defined for a software organization
- Work tasks may vary but the common process framework (CPF) is invariant (project size does not change the CPF)
- The job of the software engineer is to estimate the resources required to move each function though the framework activities to produce each work product
- Project decomposition begins when the project manager tries to determine how to accomplish each CPF activity

 Downloaded from: sarojpandey.com.np

The Project

Five part common sense approach to software project includes:

- Start on the right foot
- Maintain momentum
- Track progress
- Make smart decisions
- Conduct a postmortem analysis

W5HH Principle

- Why is the system being developed?
- What will be done by When?
- Who is responsible for a function?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource is needed?

Critical Practices

- Formal risk management
- Empirical cost and schedule estimation
- Metric-based project management
- Earned value tracking
- Defect tracking against quality targets
- People-aware program management

Summary

Software project management is an umbrella activities within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development, and support of computer software.

Four P's have a substantial influence on the software project management-people, product, process, and project. People must be organized into effective teams, motivated to do high-quality software work, and coordinated to achieve effective communication. The product requirements must be communicated from customer to developer, partitioned(decomposed) into their constituent parts, and positioned for work by the software team. The process must be adopted to the people and the problem. A common process framework (CPF) is selected, an appropriate software engineering paradigm is applied, and a set of work tasks is chosen to get the job done. Finally, the project must be organized in a manner that enables the software team to succeed. The project management activity encompasses measurement and metrics, estimation, risk analysis, schedules, tracking, and control.

Software Project Planning-Overview

Software planning involves estimating how much time, effort, money, and resources will be required to build a specific software system. After the project scope is determined and the problem is

 Downloaded from: sarojpandey.com.np

decomposed into smaller problems, software managers use historical project data (as well as personal experience and intuition) to determine estimates for each. The final estimates are typically adjusted by taking project complexity and risk into account. The resulting work product is called a project management plan.

- Probably the most time-consuming project management activity
- Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available
- Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget

Project Planning Objectives

- To provide a framework that enables software manager to make a reasonable estimate of resources, cost, and schedule.
- Project outcomes should be bounded by 'best case' and 'worst case' scenarios.
- Estimates should be updated as the project progresses.

Types of Project Plan

Plan	Description
Quality Plan	Describes the quality procedures and standards that will be used in a project.
Validation Plan	Describes the approach, resources and schedules used for system validation.
Confirmation management Plan	describes the configuration management procedures and structures and structures to be used.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance cost, and effort required.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

Project Planning Process

Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled **loop**
    Draw up project schedule
    Initiate activities according to schedule
    Wait (for a while)
    Review project progress
    Revise estimates of project parameters
    Update the project schedule
    Re-negotiate project constraints and deliverables

Downloaded from: sarojpandey.com.np

```
    if (problems arise) then
        Initiate technical review and possible revision
    end if
end loop
```

Project Plan Structure

- Introduction
- Project organisation
- Risk analysis
- Hardware and software resource requirements
- Work breakdown
- Project schedule
- Monitoring and reporting mechanisms

Estimation Reliability Factors

5. Project complexity
6. Project size
7. Degree of structural uncertainty (degree to which requirements have solidified, the ease with which functions can be compartmentalized, and the hierarchical nature of the information processed)
8. Availability of historical information

Software Project Planning activities

Step1: Determination of software scope and feasibility study
Step2: Estimation of the resources

Step1: Determination of software scope

- Describes the data to be processed and produced, control parameters, function, performance (wrt processing and response time requirement), constraints (wrt limits placed on the software by external hardware, available memory, or other existing system), external interfaces, and reliability.
- Often functions described in the software scope statement are refined to allow for better estimates of cost and schedule.

Customer Communication and Scope

Necessary information is obtained by conducting a preliminary meeting or interview with customer, where Analyst prepare a set of *context free questions* in order to....

- Determine the customer's overall goals for the proposed system and any expected benefits.
- Determine the customer's perceptions concerning the nature if a good solution to the problem.
- Evaluate the effectiveness of the customer meeting.

 Downloaded from: sarojpandey.com.np

FAST (Facilitated Application Specification Techniques)

It is a team-oriented approach to requirement gathering that can be applied to help establish the scope of a project. It encourages the creation of a joint team of customer and development who work together to identify the problem, propose elements of the solution, negotiate different approaches & specify a preliminary set of requirements.

Feasibility Study

- Technical feasibility is not a good enough reason to build a product.
- The product must meet the customer's needs and not be available as an off-the-shelf purchase.

Step 2: Estimation of Resources

- Human Resources (number of people required and skills needed to complete the development project)
- Reusable Software Resources (off-the-shelf components, full-experience components, partial-experience components, new components)
- Development Environment (hardware and software required to be accessible by software team during the development process)

 Downloaded from: sarojpandey.com.np

Each resource is specified with four characteristics:

- Description of the resource
- statement of availability
- time when the resource will be required
- duration of time that resource will be applied

} Time Window

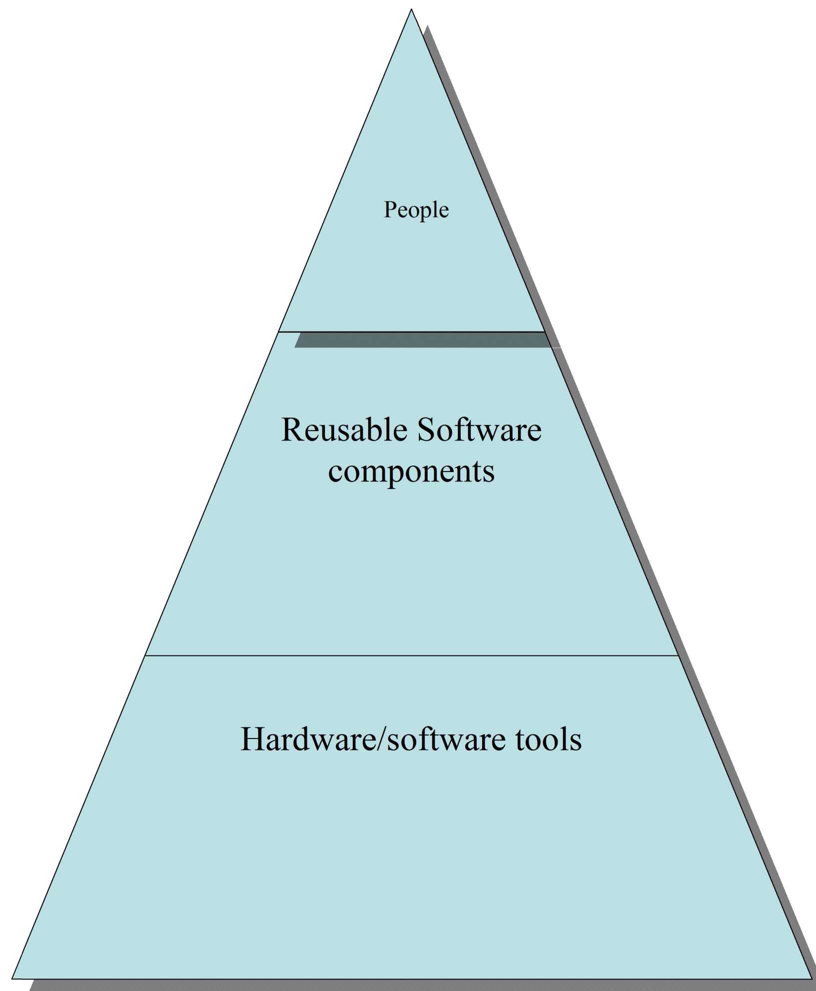


Fig: Project Resources

Software Project Estimation Options

- Delay estimation until late in the project.
- Base estimates on similar projects already completed.
- Use simple *decomposition techniques* to estimate project cost and effort.
- Use *empirical models* for software cost and effort estimation.
- *Automated tools* may assist with project decomposition and estimation.

Note: The accuracy of a software project estimate is predicted based on the product size; human effort, calendar time, cost; software team; product requirement and environment.

 Downloaded from: sarojpandey.com.np

Decomposition Techniques

- It is "divide and conquer approach" to software project estimation.
- Cost and effort estimation can be performed in a stepwise fashion by decomposing a project into major functions and activities.
- It should generates an estimates of *Software sizing* (fuzzy logic, function point, standard component, change)
- This techniques use either decomposition of the problem or the process.
  - *Problem-based estimation* (using LOC decomposition focuses on software functions, using FP decomposition focuses on information domain characteristics)
  - *Process-based estimation* (decomposition based on tasks required to complete the software process framework)

Further Reading ....

Software Sizing refers to a quantifiable outcome of the software project. Size can be measured in LOC, for direct approach or FP, for indirect measure.
Four different approach to the sizing problem are:

1. **"Fuzzy logic" sizing**:
   Approximate reasoning techniques to identify the type of application, establishing its magnitude on a qualitative scale, and refine the magnitude within the original range.

2. **Function point sizing**:
   Estimation of the information domain characteristics

3. **Standard component sizing**:
   Estimation of the number of occurrence of each standard components (subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, object-level instructions) and use of historical project data.

4. **Change sizing**:
   Use of existing software for modification

Summary

The software project planner must estimate three things before a project begins: how long it will take, how much effort will be required, and how many people will be involved. In addition, the planner must predict the resources (hardware and software) that will be required and the risk involved.

Each step in the software engineering process should produce a deliverable that can be reviewed and that can act as a foundation for the steps that follow. The software project plan is produced at the culmination of the planning tasks. It provides baseline costs and scheduling information that will be used throughout the software process.

 Downloaded from: sarojpandey.com.np

The software project plan is a relatively brief document that is addressed to a diverse audience. It must (1) communicate scope and resources to software management, (2) define risks and suggest risk aversion techniques; (3) define cost and schedule for management review; (4) provide an overall approach to software development for all people associated with the project; and (5) outline how quality will be ensured and change will be managed.

It is important to note that the *Software Project Plan* is not a static document. That is, the project team revisits the plan repeatedly- updating risks, estimates, schedules and related information- as the project proceeds.

The estimation of scope helps the planner to develop estimates using one or more techniques that fall into two broad categories: *decomposition* and *empirical model*. Decomposition techniques requires a delineation of a major software functions, followed by estimation of either (1) the number of LOC (2) selected values within the information domain (3) the number of person-months required to implement each function, or (4) the number of person-month required for each software engineering activity. Empirical techniques use empirically derived expressions for effort and time to predict these project quantities. Automated tools can be used to implement a specific empirical model. Software project estimation can never be an exact science, but a combination of good historical data and systematic techniques can improve estimation accuracy.

- Good project management is essential for project success
- The intangible nature of software causes problems for management Managers have diverse roles but their most significant activities are planning, estimating and scheduling
- Planning and estimating are iterative processes which continue throughout the course of a project
- A project milestone is a predictable state where some formal report of progress is presented to management.

 Downloaded from: sarojpandey.com.np

Project Scheduling

This chapter describes the process of building and monitoring schedules for software development projects. To build complex software systems, many engineering tasks need to occur in parallel with one another to complete the project on time. The output from one task often determines when another may begin. It is difficult to ensure that a team is working on the most appropriate tasks without building a detailed schedule and sticking to it.

Things done before project scheduling.....

- *Selection of an appropriate process model*
- *Identification of the software engineering tasks*
- *Estimation of Work size, people*
- *Calculation of deadline*
- *Risk analysis*

Things to do during scheduling....

- *Creation of activity network (Effort and duration allocated to each task and a task network)*
- *Assign responsibility for each task*

Project Scheduling

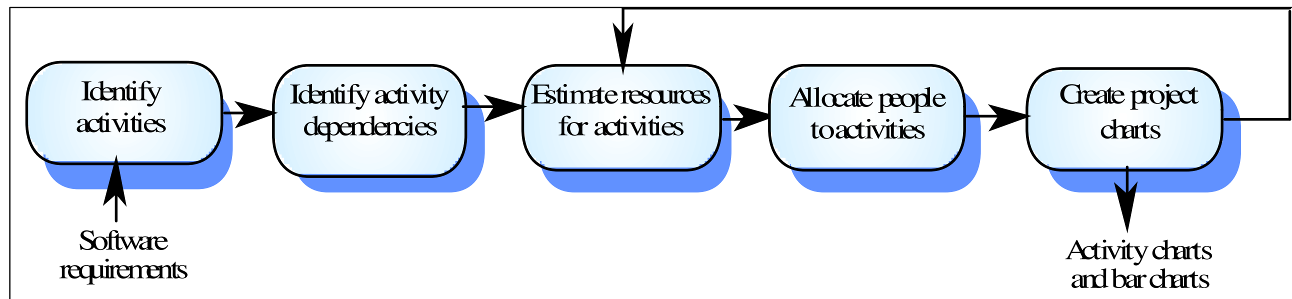
- Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.
- Split project into tasks and estimate time and resources required to complete each task
- Organize tasks concurrently to make optimal use of workforce
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete

Software Project Scheduling Principles

- **Compartmentalization** - the product and process must be decomposed into a manageable number of activities and tasks
- **Interdependency** - tasks that can be completed in parallel must be separated from those that must completed serially
- **Time allocation** - every task has start and completion dates that take the task interdependencies into account
- **Effort validation** - project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan
- **Defined Responsibilities** - every scheduled task needs to be assigned to a specific team member
- **Defined outcomes** - every task in the schedule needs to have a defined outcome (usually a work product or deliverable)
- **Defined milestones** - a milestone is accomplished when one or more work products from an engineering task have passed quality review

 Downloaded from: sarojpandey.com.np

Project Scheduling Process



Scheduling Problem

☑ Estimating the difficulty of problems and hence the cost of developing a solution is hard
☑ Productivity is not proportional to the number of people working on a task
☑ Adding people to a late project makes it later because of communication overheads
☑ The unexpected always happens. Always allow contingency in planning

Relationship Between People and Effort

- Adding people to a project after it is behind schedule often causes the schedule to slip further
- The relationship between the number of people on a project and overall productivity is not linear (e.g. 3 people do not produce 3 times the work of 1 person, if the people have to work in cooperation with one another)
- The main reasons for using more than 1 person on a project are to get the job done more rapidly and to improve software quality.

Project Effort Distribution

40-20-40 rule recommends that 40 % of all effort is allocated to front-end analysis and design, and a similar percentage is applied to back-end testing while remaining 20 % is allocated for coding part only.

Generally accepted guidelines are:

02-03 % planning
10-25 % requirements analysis
20-25 % design
15-20 % coding
30-40 % testing and debugging

Defining task set for the software projects

The software process model is populated by a set of tasks that enable a software team to define, develop and ultimately support computer software.

 Downloaded from: sarojpandey.com.np

A *task sets* is a collection of software engineering work tasks, milestones, and deliverables that must be accomplished to complete a particular projects.

Task sets are designed to accommodate different types of projects and different degrees of rigor.

Software Project Types

Concept development - initiated to explore new business concept or new application of technology
New application development - new product requested by customer
Application enhancement - major modifications to function, performance, or interfaces (observable to user)
Application maintenance - correcting, adapting, or extending existing software (not immediately obvious to user)
Reengineering - rebuilding all (or part) of a legacy system

Software Process Degree of Rigor

It is a function of many software project characteristics. Four different degrees of rigor can be defined as:

- **Casual** - all framework activities applied, only minimum task set required (umbrella activities minimized and documentation reduced)
- **Structured** - all framework and umbrella activities applied (SQA, SCM, documentation, and measurement tasks are streamlined)
- **Strict** - full process and umbrella activities applied (high quality products and robust documentation produced)
- **Quick reaction** - emergency situation, process framework used, but only tasks essential to good quality are applied (back filling used to develop documentation and conduct additional reviews after product is delivered)

Rigor Adaptation Criteria

- Size of project
- Number of potential users
- Mission criticality
- Application longevity
- Requirement stability
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

Task Set Selector Value

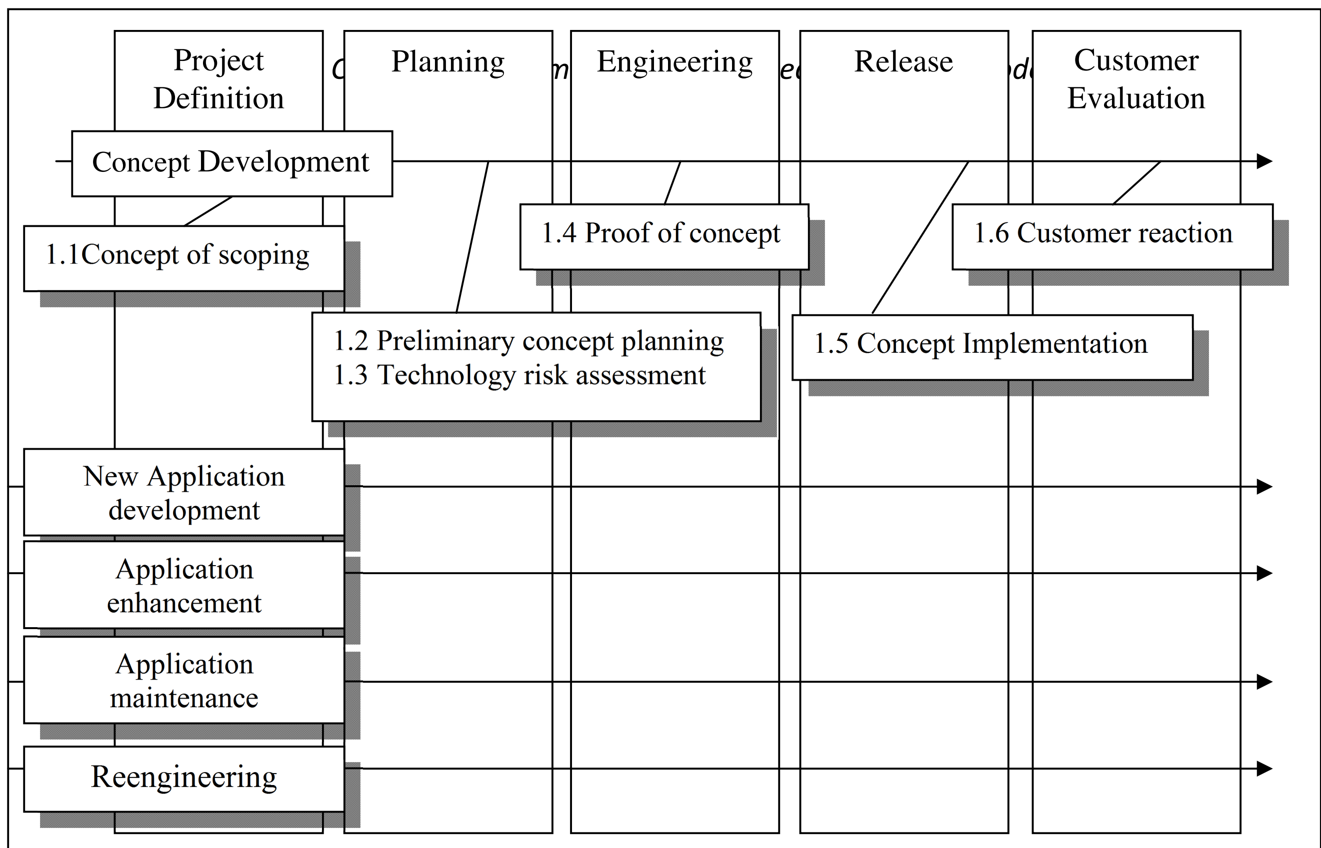
- Computed by scoring rigor adaptation criteria and adjusting the scores using differential weighting based on project characteristics.
- Once computed the task selector value can be used to select the appropriate task set (casual, structured, strict) for the project.

 Downloaded from: sarojpandey.com.np

- It is OK to choose a less formal degree of rigor when the task selector value falls in the overlap area between two levels of rigor, unless project risk is high.

Concept Development Tasks

4. **Concept scoping** - determine overall project scope
5. **Preliminary concept planning** - establishes development team's ability to undertake the proposed work
6. **Technology risk assessment** - evaluates the risk associated with the technology implied by the software scope
7. **Proof of concept** - demonstrates the feasibility of the technology in the software context
8. **Concept implementation** - concept represented in a form that can be used to sell it to the customer
9. **Customer reaction to concept** - solicits feedback on new technology from customer

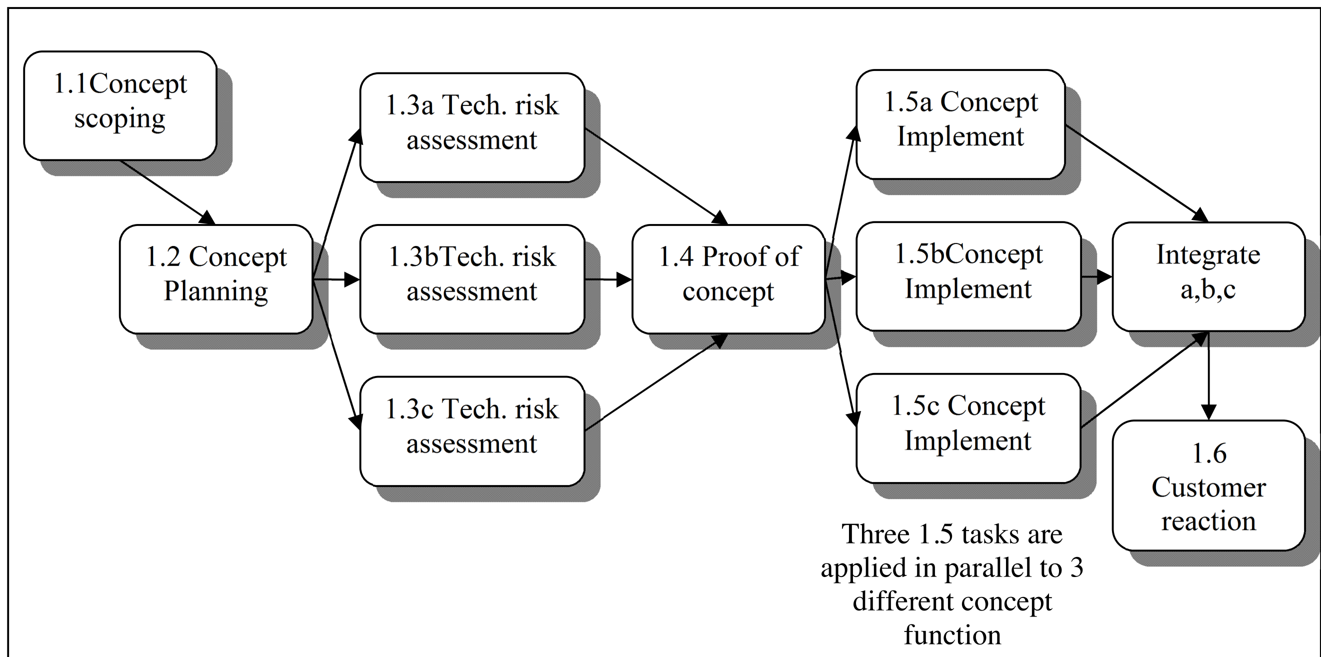


Note: Concept development projects are initiated when the potential for some new technology must be explored.

Task Network or Activity Network

A *task network*, also called an *activity network,* is a graphical representation of the task flow for a project. It is a useful mechanism for depicting intertask dependencies and determining the critical path. Critical path task must be completed on schedule if the as a whole is to be completed on schedule.

 Downloaded from: sarojpandey.com.np



Project Scheduling Methods

Scheduling tools should be used to schedule any non-trivial project. **PERT** (*program evaluation and review technique*) and **CPM** (*critical path method*) are quantitative techniques that allow software planners to identify the chain of dependent tasks in the project *work breakdown structure (WBS)* that determine the project duration time.

Timeline (Gantt) charts or **Bar Chat** is an alternative way of representing project schedule information that enable software planners to determine what tasks will be need to be conducted at a given point in time (based on estimates for effort, start time, and duration for each task). Bar charts show schedule against calendar time. The best indicator of progress is the completion and successful review of a defined software work product.

Note: Activity Network and Bar Chart is the graphical notations used to illustrate the project schedule. It show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.

Time-boxing is the practice of deciding a priori the fixed amount of time that can be spent on each task. When the task's time limit is exceeded, development moves on to the next task (with the hope that a majority of the critical work was completed before time ran out).

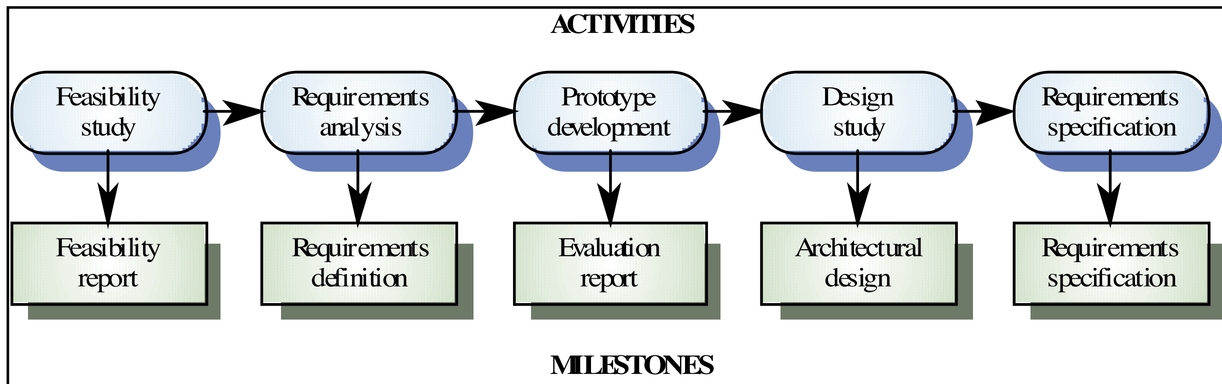
Further Reading......

Milestones and Deliverables

- *Milestones* are the end-point of a process activity. At each milestone, there should be a formal output (such as report). It should represent the end of a distinct, logical stage in the project.
- *Deliverables* are project results delivered to customer. It is usually delivered at the end of some major project phase such as specification, design etc.

 Downloaded from: sarojpandey.com.np

- Deliverables are usually milestones but milestones may not be deliverables. Milestones may be internal project results that are used by the project manager to check project progress but which are not delivered to the customer.
- To establish milestones, the software process must be broken down into basic activities with associated outputs. Fig below shows activities involved in requirements specification when prototyping is used to help validate requirements. The principle outputs for each activity (the project milestones) are shown. The project deliverables are the requirements definition and the requirements specification.



Task duration and dependencies

Task	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2,T4 (M2)
T6	5	T1,T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3,T6 (M4)
T10	15	T5,T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

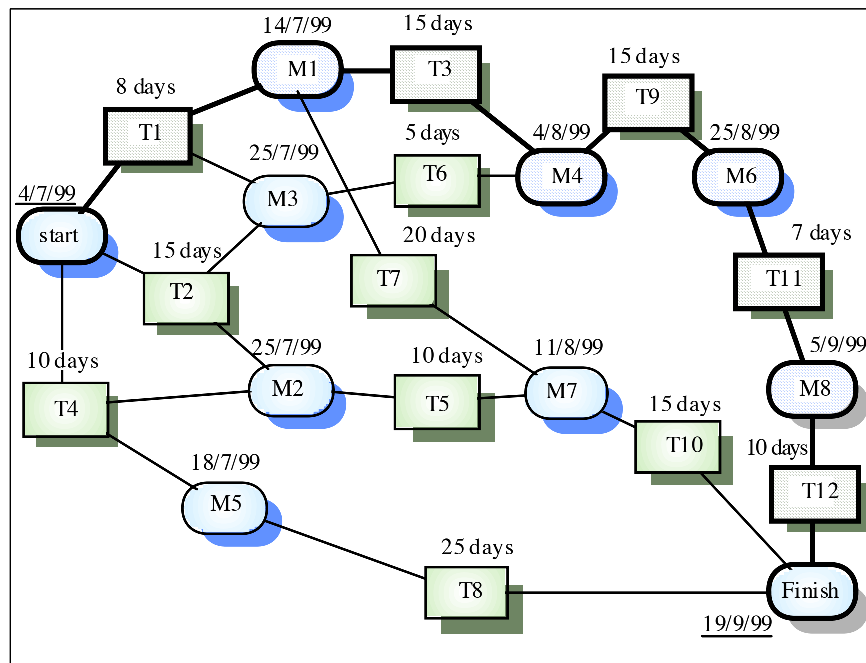
The above table shows activities, their duration, and activity interdependencies. It is seen that T3 (implementation of design) is dependent on Task T1 (task-preparation of a component design) , means T1 must be completed before T3 starts.
Given dependency and estimated duration of activities, an activity network which shows activity sequences may be generated. It shows which activities can be carried out in parallel and which must be executed in sequence because of a dependency on an earlier activity. Activities are represented as rectangles. Milestones and deliverables are shown with rounded corners. Dates in the diagram shows the start date of the activity and all activities must end in milestones.

 Downloaded from: sarojpandey.com.np

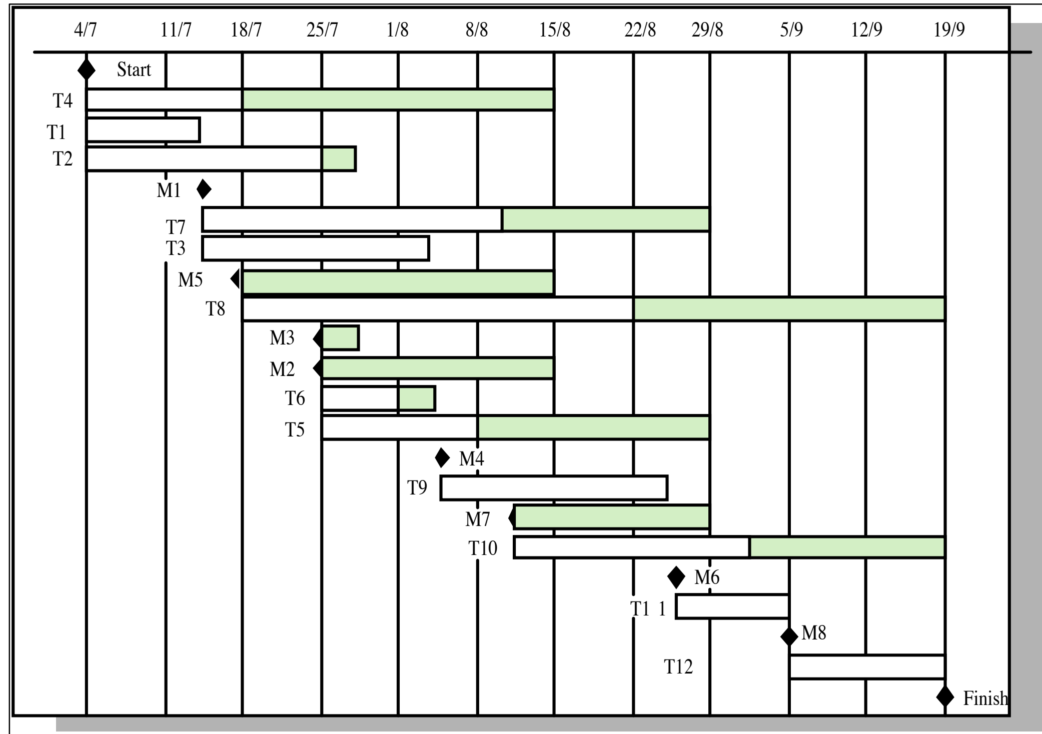
Before progress can be made from one milestone to another, all paths leading to it must be complete. For example, task T9 can not be started until tasks T3 and T6 are finished. The arrival at milestone M4 shows that these tasks have been completed.

The minimum time required to finish the project can be estimated by considering the longest path in the activity graph (the critical path). Here, it is 11 weeks or elapsed time or 55 working days. In the activity network, the critical path is shown as a sequence of emboldened boxes. The overall schedule of the project depends on the critical path. Any slippage in the completion of any critical activity causes project delays.

Activity Network



 Downloaded from: sarojpandey.com.np



Activity Timeline or Gantt Chart or Bar Chart

Some of the activities in the above figure are followed by a shaded bar whose length is computed by the scheduling tool. This shows that there is some flexibility in the completion date of these activities. If an activity does not complete on time, the critical path will not be affected until the end of the period marked by the shaded bar. Activities which lies on the critical path have no margin of error and they can be identified because they have no associated shaded bar.

Staff Allocation

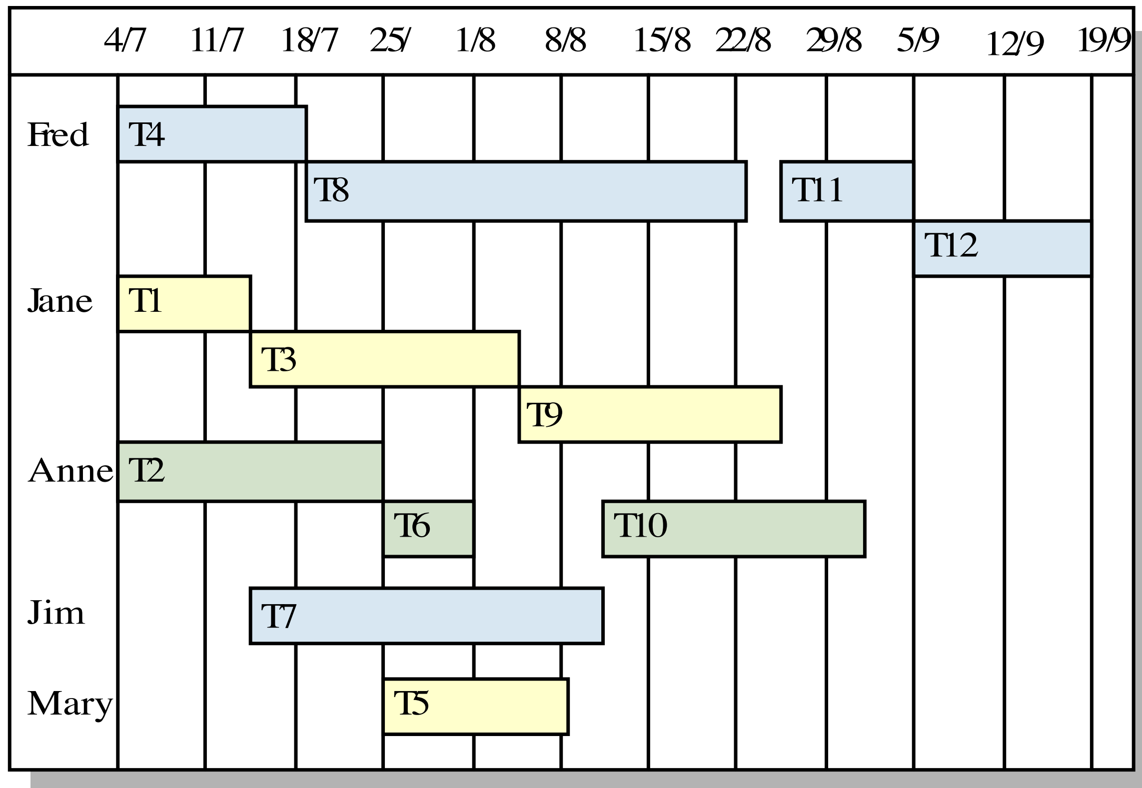
While scheduling the project, after the completion of the activity network, the allocation of staff to project activities must be performed. Allocation of people to activities is shown in the table below.

Task	Engineer
T1	Jane
T2	Anne
T3	Jane
T4	Fred
T5	Mary
T6	Anne
T7	Jim
T8	Fred
T9	Jane
T10	Anne

 Downloaded from: sarojpandey.com.np

T11	Fred
T12	Fred

Staff allocation vs time chart



- The total hours to complete the entire project are estimated and each task is given an earned value based on its estimated percentage contribution to the total.

Error Tracking

- ☑ Allows comparison of current work to past projects and provides a quantitative indication of the quality of the work being conducted.
- ☑ The more quantitative the approach to project tracking and control, the more likely problems can be anticipated and dealt with in a proactive manner.

Summary

Scheduling is a culmination of a planning activity that is a primary component of software project management. When combined with estimation methods and risk analysis, scheduling establishes a road map for the project manager.
Scheduling begins with process decomposition. The characteristics of the project are used to adopt an appropriate task set for the work to be done. A task network depicts each engineering task, its dependency on other tasks, and its projected duration. The task network is used to compute the critical path, a timeline chart (Gantt Chart) and a variety of project information. Using the schedule as a guide, the project manager can track and control each step in the software process.

 Downloaded from: sarojpandey.com.np

A project milestone is a predictable outcome of an activity where formal report of progress should be presented to management. Milestones should occur regularly throughout a software project. A deliverable is a milestone which is delivered to the project customer.
Project Scheduling involves the creation of various graphical representations of part of the project plan. These include activity charts showing the interrelationships of project activities and bar charts showing activity durations.

Assessment Questions on Project Planning

Q1. Write a statement of scope that describes the software for a home security system. Be sure your statement of scope is bounded.

Q2. Performance is an important consideration during planning. Discuss how performance can be interpreted differently depending upon the software application area.

Q3. What do you mean by project planning and project scheduling? Explain the different activities which are performed during project planning.

Q4. Explain why the intangibility of software system poses special problems for software project management ?

Q5. Explain why the process of project planning is an iterative one and why a plan must be continually reviewed during a software projects ?

Q6. Briefly explain the purpose of each of the sections in a software plan.
Q7. Develop a list of software characteristics that affect the complexity of a projects. Prioritize the list.

Q8. It seems odd that cost and schedule estimates are developed during software project planning- before detailed software requirements analysis or design has been conducted. Why do you think this is done? Are there circumstances when it should not be done?

Assessment Questions on Project Scheduling

Q9. What is the difference between a macroscopic schedule and a detailed schedule. Is it possible to manage a project if only a macroscopic schedule is developed? Why?

Q10. What is the critical distinction between a milestone and a deliverable ?

 Downloaded from: sarojpandey.com.np

Chapter 5-10 - Analysis Concepts and Principles

After system engineering is completed, software developers need to look at the role of software in the proposed system. Software requirements analysis is necessary to avoid creating a software product that fails to meet the customer's needs. Data, functional, and behavioral requirements are elicited from the customer and refined to create a specification that can be used to design the system. Software requirements work products must be reviewed for clarity, completeness, and consistency.

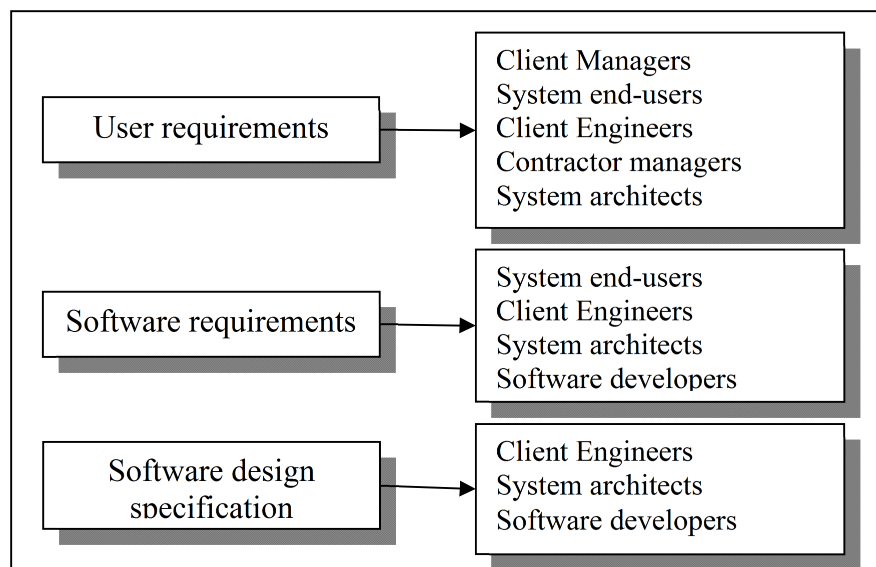
Definition of requirement

- Requirements set out what the system should do and define constraints on its operation and implementation
- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation
  - May be the basis for the contract itself - therefore must be defined in detail
  - Both these statements may be called requirements

"Software requirement is the descriptions and specifications of a system"

Types of requirements

- **User requirements**
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- **System requirements**
  - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor.
- **Software specification**
  - A detailed software description which can serve as a basis for a design or implementation. Written for developers.



 Downloaded from: sarojpandey.com.np

Functional and Non-functional requirements

Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Describe functionality or system services
- Depend on the type of software, expected users and the type of system where the software is used
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail

Non-functional requirements

- constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular CASE system, programming language or development method
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

Non-functional classifications

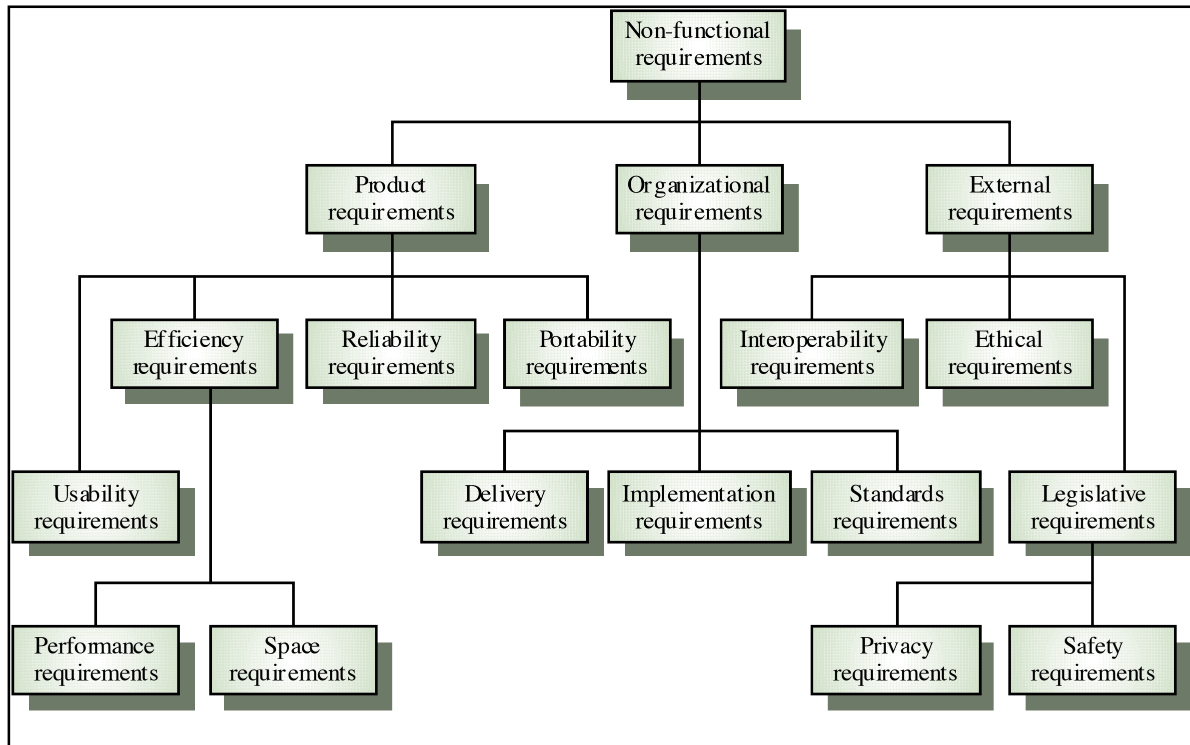
- **Product requirements**
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- **Organisational requirements**
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- **External requirements**
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Domain requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain
- Derived from the application domain and describe system characterisics and features that reflect the domain
- May be new functional requirements, constraints on existing requirements or define specific computations
- If domain requirements are not satisfied, the system may be unworkable

 Downloaded from: sarojpandey.com.np

Non-functional requirement types



Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Numbers of RAM chips
Ease of Use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

 Downloaded from: sarojpandey.com.np

Requirements Engineering

Requirements Engineering provides the appropriate mechanism for understanding what the customer wants, analyzing the need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into a operational system.

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process

Requirement Engineering Process

It is the processes used to discover, analyse and validate system requirements. The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements. However, there are a number of generic activities common to all processes can be described in six distinct steps:

Step 1: Requirements elicitation
Step 2: Requirements analysis and negotiation
Step 3: Requirements specification
Step 4: System modeling
Step 5: Requirements validation
Step 6: Requirements management

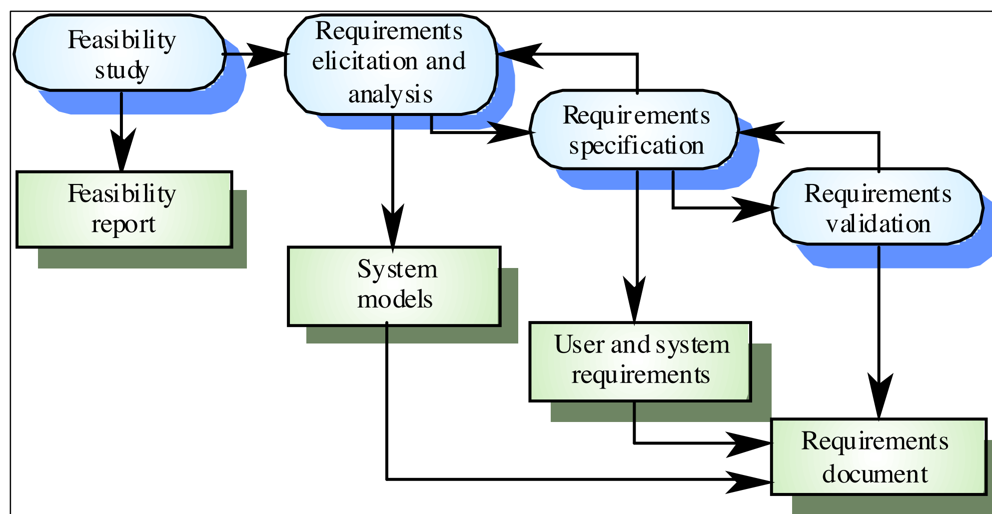


Fig: Requirement Engineering Process

Downloaded from: sarojpandey.com.np

Step1: Requirements elicitation (find out from customers what the product objectives are, what is to be done, how the product fits into business needs, and how the product is used on a day to day basis)

Software Requirements Elicitation

- Customer meetings are the most commonly used technique.
- Use *context free questions* to find out customer's goals and benefits, identify stakeholders, gain understanding of problem, determine customer reactions to proposed solutions, and assess meeting effectiveness.
- If many users are involved, be certain that a representative cross section of users is interviewed.

Facilitated Action Specification Techniques (FAST)

- Meeting held at neutral site, attended by both software engineers and customers.
- Rules established for preparation and participation.
- Agenda suggested to cover important points and to allow for brainstorming.
- Meeting controlled by facilitator (customer, developer, or outsider).
- Definition mechanism (flip charts, stickers, electronic device, etc.) is used.
- Goal is to identify problem, propose elements of solution, negotiate different approaches, and specify a preliminary set of solution requirements.

Step2: Requirements analysis and negotiation (requirements are categorized and organized into subsets, relations among requirements identified, requirements reviewed for correctness, requirements prioritized based on customer needs)

Step3: Requirements specification (work product produced describing the function, performance, and development constraints for a computer-based system)

Step4: System modeling (system representation that shows relationships among the system components)

Step5: Requirements validation (examines the specification to ensure requirement quality and that work products conform to agreed upon standards)

Step6: Requirements management (set of activities that help project team to identify, control, and track requirements and changes as project proceeds)

Traceability Tables

Traceability is concerned with the relationships between requirements, their sources and the system design

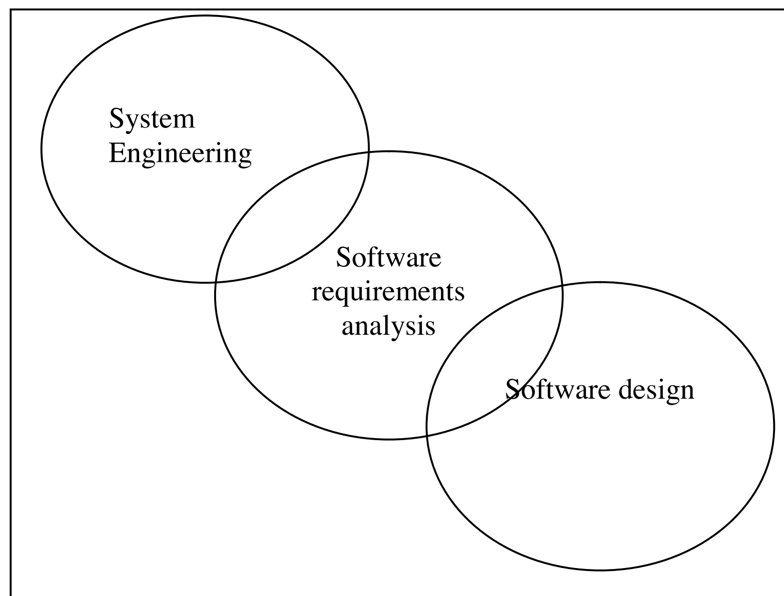
- Features traceability table : Shows how requirements relate to important customer observable system/product features.
- Source traceability table: Identifies the source of each requirements.
- Dependency traceability table: Indicates how requirements are related to one another.
- Subsystem traceability table: Categorizes requirements by the subsystems that they govern.

 Downloaded from: sarojpandey.com.np

- Interface traceability table: Show how requirements relate to both internal and external system interfaces.

Step2: Requirements Analysis

- Software engineering task that bridges the gap between system level requirements engineering and software design.
- Provides software designer with a representation of system information, function, and behavior that can be translated to data, architectural, and component-level designs.
- Expect to do a little bit of design during analysis and a little bit of analysis during design.



Problems of requirements analysis

- Stakeholders don't know what they really want
- Stakeholders express requirements in their own terms
- Different stakeholders may have conflicting requirements
- Organisational and political factors may influence the system requirements
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change

Analysis Principles

- The *information domain* of the problem must be represented and understood.
- The functions that the software is to perform must be defined.
- Software behavior must be represented.
- *Models* depicting information, function, and behavior must be *partitioned* in a hierarchical manner that uncovers detail.
- The analysis process should move from the essential information toward implementation detail.

 Downloaded from: sarojpandey.com.np

Further Readings.....

Information Domain

- Encompasses all data objects that contain numbers, text, images, audio, or video.
- Information content or data model (shows the relationships among the data and control objects that make up the system)
- Information flow (represents the manner in which data and control objects change as each moves through the system)
- Information structure (representations of the internal organizations of various data and control items)

Modeling

- Data model (shows relationships among system objects)
- Functional model (description of the functions that enable the transformations of system objects)
- Behavioral model (manner in which software responds to events from the outside world)

Partitioning

- Process that results in the elaboration of data, function, or behavior.
- Horizontal partitioning is a breadth-first decomposition of the system function, behavior, or information, one level at a time.
- Vertical partitioning is a depth-first elaboration of the system function, behavior, or information, one subsystem at a time.

Software Requirements Views

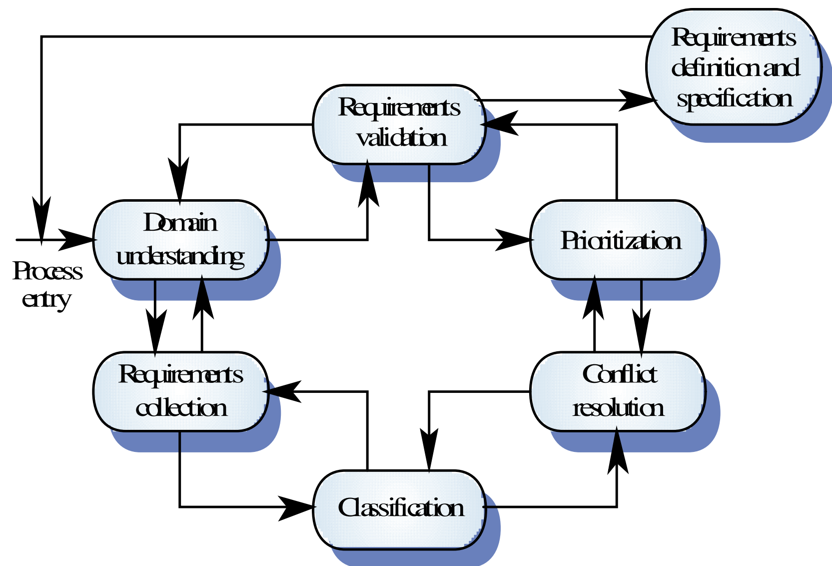
- **Essential view** - presents the functions to be accomplished and the information to be processed without regard to implementation.
- **Implementation view** - presents the real world manifestation of processing functions and information structures.
- Avoid the temptation to move directly to the implementation view, assuming that the essence of the problem is obvious.

Requirements elicitation and analysis process

- Sometimes called requirements elicitation or requirements discovery
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*
- The process activities include:

 Downloaded from: sarojpandey.com.np

- Domain understanding
- Requirements collection
- Classification
- Conflict resolution
- Prioritisation
- Requirements checking



View-point oriented methods

- Stakeholders represent different ways of looking at a problem or problem viewpoints
- This multi-perspective analysis is important as there is no single correct way to analyse system requirements

Types of viewpoints

- Data sources or sinks
  - Viewpoints are responsible for producing or consuming data. Analysis involves checking that data is produced and consumed and that assumptions about the source and sink of data are valid
- Representation frameworks
  - Viewpoints represent particular types of system model. These may be compared to discover requirements that would be missed using a single representation. Particularly suitable for real-time systems
- Receivers of services
  - Viewpoints are *external* to the system and receive services from it. Most suited to interactive systems

External viewpoints

- Natural to think of end-users as receivers of system services
- Viewpoints are a natural way to structure requirements elicitation
- It is relatively easy to decide if a viewpoint is valid
- Viewpoints and services may be sued to structure non-functional requirements

 Downloaded from: sarojpandey.com.np

Methods based analysis

- Widely used approach to requirements analysis. Depends on the application of a structured method to understand the system
- Methods have different emphases. Some are designed for requirements elicitation, others are close to design methods
- A *viewpoint-oriented method (VORD)* is used as an example here. It also illustrates the use of viewpoints

VORD Method

The principle stages of the VORD method includes

- Viewpoint identification
  - Discover viewpoints which receive system services and identify the services provided to each viewpoint
- Viewpoint structuring
  - Group related viewpoints into a hierarchy. Common services are provided at higher-levels in the hierarchy
- Viewpoint documentation
  - Refine the description of the identified viewpoints and services
- Viewpoint-system mapping
  - Transform the analysis to an object-oriented design

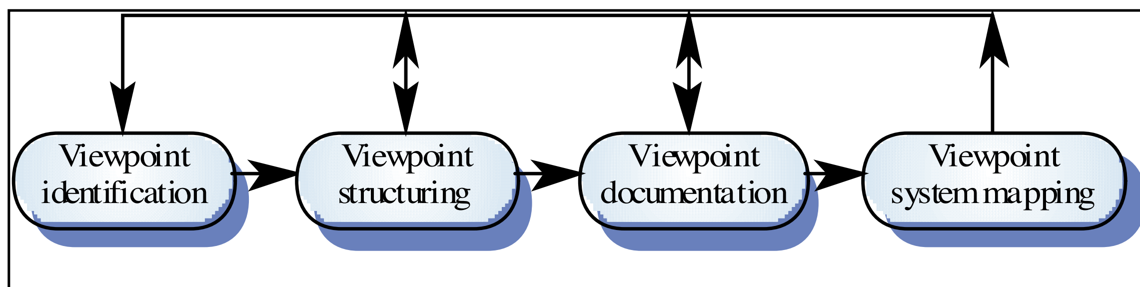


Fig: The VORD method

Step3: Software Requirement Specification

Specification Principles

- Separate functionality from implementation.
- Develop a behavioral model that describes functional responses to all system stimuli.
- Define the environment in which the system operates and indicate how the collection of agents will interact with it.
- Create a cognitive model rather than an implementation model.
- Recognize that the specification must be extensible and tolerant of incompleteness.
- Establish the content and structure of a specification so that it can be changed easily.

 Downloaded from: sarojpandey.com.np

Specification Representation

- Representation format and content should be relevant to the problem.
- Information contained within the specification should be nested.
- Diagrams and other notational forms should be restricted in number and consistent in use.
- Representations should be revisable.

Software Requirement specification

- Produced at the culmination of the analysis task.
- Functions and Performance are refined by establishing a complete information description, a representation of system behavior, and indication of performance requirements and design constraints, appropriate validation criteria, and other relevant information
- Candidate format for software requirement specification:
    - Introduction
    - Information Description
    - Functional Description
    - Behavioral Description
    - Validation Criteria
    - Bibliography and Appendix

Specification Review

- Conducted by customer and software developer.
- Once approved, the specification becomes a contract for software development.
- The specification is difficult to test in a meaningful way.
- Assessing the impact of specification changes is hard to do.

Step5: Requirements validation

- It shows the requirements actually define the system which the customer wants.
- It checks for validity, consistency, completeness, realism, verifiability.
- Requirements Validation techniques

    1. **Requirements reviews**
        - Systematic manual analysis of the requirements
    2. **Prototyping**
        - Using an executable model of the system to check requirements.
    3. **Test-case generation**
        - Developing tests for requirements to check testability
    4. **Automated consistency analysis**
    - Checking the consistency of a structured requirements description

1. Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated
- Both client and contractor staff should be involved in reviews
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage

 Downloaded from: sarojpandey.com.np

- It checks for Verifiability, Comprehensibility, Traceability and Adaptability.

2. Software Prototyping

☑ Throwaway prototyping (prototype only used as a demonstration of product requirements, finished software is engineered using another paradigm)
☑ Evolutionary prototyping (prototype is refined to build the finished system)
☑ Customer resources must be committed to evaluation and refinement of the prototype.
☑ Customer must be capable of making requirements decisions in a timely manner.

Prototyping Methods and Tools

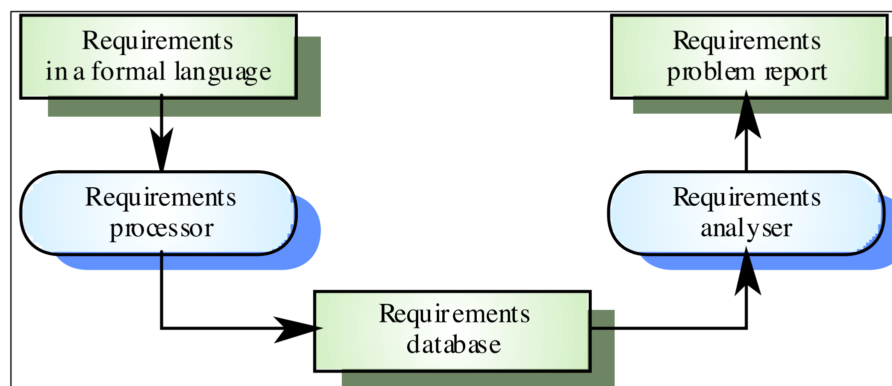
- Fourth generation techniques (4GT tools allow software engineer to generate executable code quickly)
- Reusable software components (assembling prototype from a set of existing software components)
- Formal specification and prototyping environments (can interactively create executable programs from software specification models)

3. Test case generation

Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirement problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered.

4. Automated consistency Analysis

If the requirements are expressed as a system model in a structured or formal notation the CASE tools may be used to check the consistency of the model. To check the consistency, the CASE tool must build a requirements database and then, using the rules of the method or notation, check all of the requirements in this database.



 Downloaded from: sarojpandey.com.np

Step6: Requirement Management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development
- Requirements are inevitably incomplete and inconsistent
    - New requirements emerge during the process as business needs change and a better understanding of the system is developed
    - Different viewpoints have different requirements and these are often contradictory
- Requirement change because:
    - The priority of requirements from different viewpoints changes during the development process
    - System customers may specify requirements from a business perspective that conflict with end-user requirements
    - The business and technical environment of the system changes during its development
- Principal stages consists of:
    - *Problem analysis*: Discuss requirements problem and propose change
    - *Change analysis and costing*: Assess effects of change on other requirements
    - *Change implementation*: Modify requirements document and other documents to reflect change

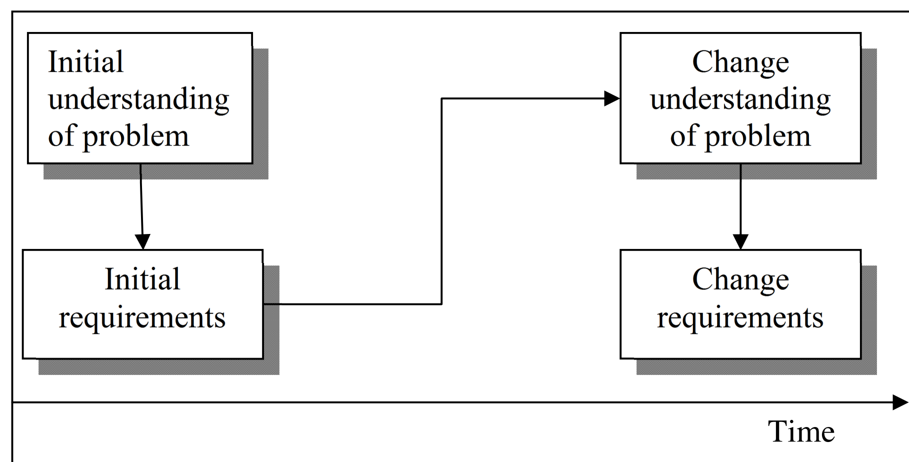


Fig: Requirement evolution

Enduring requirements: Stable requirements derived from the core activity of the customer organisation. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models

 Downloaded from: sarojpandey.com.np

Volatile requirements: Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy

Further Reading....

Q. Why is it so difficult to gain a clear understanding of what the customer wants?

A. It is because of the following problems that makes difficult to understand the customers wants:-

1. **Problem of scope**: The boundary of the system is ill-defined or the customer/users specify unnecessary technical detail that may confuse, rather than clarify, overall objectives.

2. **Problem of understanding**: The customers/users are not sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, do not have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "obvious", specify requirements that are ambiguous or untestable.

3. **Problem of volatility**: The requirements change over time

Guideline for requirements elicitation

1. Access the business and technical feasibility for the proposed system.
2. Identify the people who will help specify requirements and understand their organizational bias.
3. Define the technical environment (e.g., computer architecture, operating system, telecommunications needs) into which the system or product will be placed.
4. Identify "domain constraints" (i.e., characteristics of the business environment specific to the application domain) that limit the functionality or performance of the system or product to be built.
5. Define one or more requirements elicitation methods (e.g., interviews, focus groups, team meetings).
6. Solicit participation from many people so that requirements are defined from different points of view; be sure to identify the rationale for each requirement that is recorded.
7. Identify ambiguous requirements as candidates for prototyping.

creates usage scenario to help customers/users better identify key requirements.

The work products produced as a consequence of the requirements elicitation activity should include:

1. A statement of need and feasibility
2. A bounded statement of scope for the system or product
3. A list of customers, users, and other stakeholders who participated in the requirements elicitation activity.
4. A description of the system's technical environment.
5. A list of requirements (preferably organized by function) and the domain constraints that apply to each.

 Downloaded from: sarojpandey.com.np

6. A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
7. Any prototypes developed to better define requirements.

Requirements Analysis Checklist

1. Is each requirement consistent with the overall objective for the system/product?
2. Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
4. Is each requirement bounded and unambiguous?
5. Does each requirement have attribution? That is, a source (generally, a specific individual noted for each requirements?
6. Do any requirements conflict with other requirements?
7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?

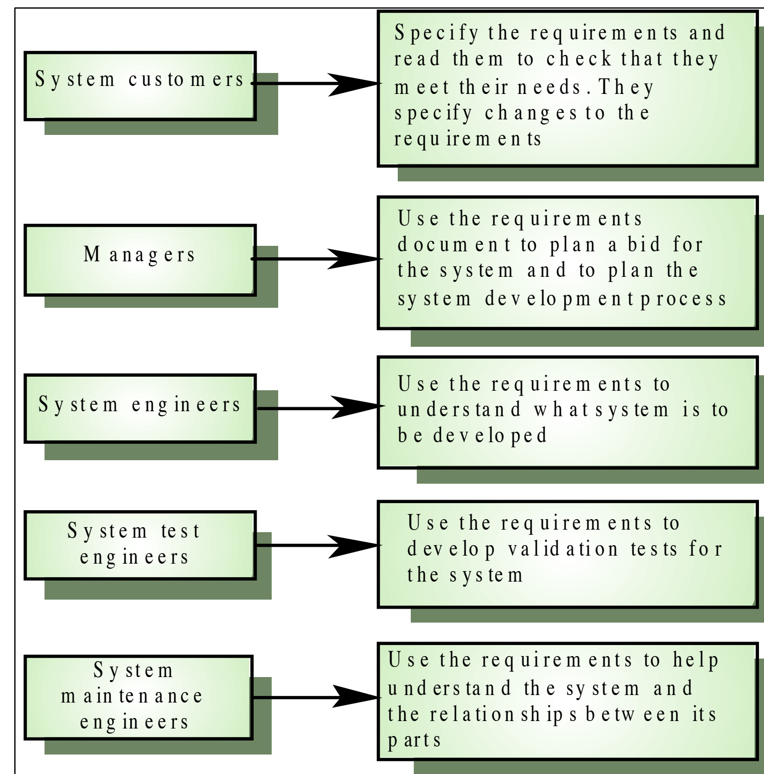
Requirements validation checklist

- Are requirements stated clearly? Can they be misinterpreted?
- Is the source (e.g., a person, a regulation, a document) of the requirements
- identified? Has the final statement of the requirements been examined by or against the original source?
- Is the requirement bounded in quantitative terms?
- What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?
- Does the requirement violate any domain constraints?
- Is the requirement testable? If so, can we specify tests, (validation criteria) to exercise the requirement?
- Is the requirement traceable to any system model that has been created?
- Is the requirement traceable to overall system/product objectives?
- Is the system specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?
- Has an index for the specification been created?
- Have requirements associated with system performance, behavior and operational characteristics been clearly stated? What requirements appear to be implicit?

The requirements document

- The requirements document is the official statement of what is required of the system developers
- Should include both a definition and a specification of requirements
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

 Downloaded from: sarojpandey.com.np



IEEE requirements standard

This is a generic structure that must be instantiated for specific systems that includes:

- Introduction
- General description
- Specific requirements
- Appendices
- Index

Requirements document structure

- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

Summary

- Requirements analysis is the first technical step in the software process. In this phase, a general statement of software scope is refined into a concrete specification that becomes the foundation for all software engineering activities that follow.
- Requirements set out what the system should do and define constraints on its operation and implementation
- Functional requirements set out services the system should provide
- Non-functional requirements constrain the system being developed or the development process

 Downloaded from: sarojpandey.com.np

- User requirements are high-level statements of what the system should do
- User requirements should be written in natural language, tables and diagrams
- System requirements are intended to communicate the functions that the system should provide
- System requirements may be written in structured natural language, a PDL or in a formal language
- A software requirements document is an agreed statement of the system requirements
- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management
- Requirements analysis is iterative involving domain understanding, requirements collection, classification, structuring, prioritisation and validation
- Requirement analysis must focus on the information, functional, and behavioral domains of a problem. To better understand what is required, models are created, the problem is partitioned, and representations that depict the essence of requirements and, later, implementation details, are developed.
- Systems have multiple stakeholders with different requirements
- Social and organisation factors influence system requirements
- In many cases, it is not possible to completely specify a problem at an early stage. Prototyping offers an alternative approach that results in an executable model of the software from which requirements can be refined.
- Software requirement specification is developed as a consequence of analysis. Review is essential to ensure that the developer and the customer have the same perception of the system.
- Requirements validation is concerned with checks for validity, consistency, completeness, realism and verifiability
- Business changes inevitably lead to changing requirements
- Requirements management includes planning and change management

Assignments

Q1. Is it fair that a Preliminary user manual is a form of prototype? Explain.

Q2. Who should be involved in a requirements review? Draw a process model showing how a requirements review might be organized?

Q3. When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications that ensure that the requirements document and the system implementation do not become inconsistent.

Q4. Explain the term verification and validation

Q5. What do you mean by term Data-Dictionary in context of structured analysis? How is the Data Dictionary useful in different phases of life cycle of a software product?

 Downloaded from: sarojpandey.com.np

Chapter 11-13 - Design Concepts and Principles

A software design is a meaningful engineering representation of some software product that is to be built. A design can be traced to the customer's requirements and can be assessed for quality against predefined criteria. During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components. Each design product is reviewed for quality before moving to the next phase of software development.

Software design sits at the technical kernel of software engineering and is applied regardless of the software process model being used. Beginning once software requirements have been analyzed and specified, software design is the first of three technical activities-designs, code generation, and test - that are required to build and verify the software. Each activity transforms information in a manner that ultimately results in validated computer software.

Design Specification Models

- *Data design* - created by transforming the analysis information model (data dictionary and ERD) into data structures required to implement the software
- *Architectural design* - defines the relationships among the major structural elements of the software, it is derived from the system specification, the analysis model, and the subsystem interactions defined in the analysis model (DFD)
- *Interface design* - describes how the software elements communicate with each other, with other systems, and with human users; the data flow and control flow diagrams provide much the necessary information
- *Component-level design* - created by transforming the structural elements defined by the software architecture into procedural descriptions of software components using information obtained from the PSPEC, CSPEC, and STD

Design Guidelines

A design should

- exhibit good architectural structure
- be modular
- contain distinct representations of data, architecture, interfaces, and components (modules)
- lead to data structures that are appropriate for the objects to be implemented and be drawn from recognizable design patterns
- lead to components that exhibit independent functional characteristics
- lead to interfaces that reduce the complexity of connections between modules and with the external environment
- be derived using a reputable method that is driven by information obtained during software requirements analysis

Downloaded from: sarojpandey.com.np

Design Principles

Software design is both a process and a model. The *design process* is a sequence of steps that enable the designer to describe all aspects of the software to be built. The design model is the equivalent of an architect's plan for a house. It begins by representing the totality of the thing to be built and slowly refines the thing to provide guidance for constructing each detail.

Basic design principles, as suggested by Davis, enable to navigate the design process.

The design

- process should not suffer from "*tunnel vision*"
- should be traceable to the analysis model
- should not reinvent the wheel
- should "*minimize intellectual distance*" between the software and the problem as it exists in the real world
- should exhibit uniformity and integration
- should be structured to accommodate change
- should be structured to degrade gently, even with bad data, events, or operating conditions are encountered
- should be assessed for quality as it is being created
- should be reviewed to minimize conceptual (semantic) errors

Fundamental Software Design Concepts

- *Abstraction* - allows designers to focus on solving a problem without being concerned about irrelevant lower level details (procedural abstraction - named sequence of events, data abstraction - named collection of data objects)
- *Refinement* - process of elaboration where the designer provides successively more detail for each design component
- *Modularity* - the degree to which software can be understood by examining its components independently of one another
- *Software architecture* - overall structure of the software components and the ways in which that structure provides conceptual integrity for a system
- *Control hierarchy or program* structure - represents the module organization and implies a control hierarchy, but does not represent the procedural aspects of the software (e.g. event sequences)
- *Structural partitioning* - horizontal partitioning defines three partitions (input, data transformations, and output); vertical partitioning (factoring) distributes control in a top-down manner (control decisions in top level modules and processing work in the lower level modules)
- *Data structure* - representation of the logical relationship among individual data elements (requires at least as much attention as algorithm design)
- *Software procedure* - precise specification of processing (event sequences, decision points, repetitive operations, data organization/structure)
- *Information hiding* - information (data and procedure) contained within a module is inaccessible to modules that have no need for such information

 Downloaded from: sarojpandey.com.np

Modular Design Method Evaluation Criteria

- Modular decomposability - provides systematic means for breaking problem into sub problems
- Modular composability - supports reuse of existing modules in new systems
- Modular understandability - module can be understood as a stand-alone unit
- Modular continuity - side-effects due to module changes minimized
- Modular protection - side-effects due to processing errors minimized

Control Terminology

- Span of control (number of levels of control within a software product)
- Depth (distance between the top and bottom modules in program control structure)
- Fan-out or width (number of modules directly controlled by a particular module)
- Fan-in (number of modules that control a particular module)
- Visibility (set of program components that may be called or used as data by a given component)
- Connectivity (set of components that are called directly or are used as data by a given component)

Effective Modular Design

- Functional independence - modules have high cohesion and low coupling
- Cohesion - qualitative indication of the degree to which a module focuses on just one thing
- Coupling - qualitative indication of the degree to which a module is connected to other modules and to the outside world

Design Heuristics for Effective Modularity

- Evaluate the first iteration of the program structure to reduce coupling and improve cohesion.
- Attempt to minimize structures with high fan-out; strive for fan-in as structure depth increases.
- Keep the scope of effect of a module within the scope of control for that module.
- Evaluate module interfaces to reduce complexity, reduce redundancy, and improve consistency.
- Define modules whose function is predictable and not overly restrictive (e.g. a module that only implements a single sub function).
- Strive for controlled entry modules, avoid pathological connection (e.g. branches into the middle of another module)

Assignments

- Do you design software when you "write" a program? What makes software design different from coding?
- Discuss the relationship between the concept of information hiding as an attribute of effective modularity and the concept of module independence.
- Discuss how structural partitioning can help to make software more maintainable.
- What is the purpose of developing a program structure that is factored?
- Explain the different design principles.
- Explain Top-down Vs. Bottom-up design technique
- Write short notes on Fan-in and Fan-out

Downloaded from: sarojpandey.com.np

Chapter 14-17 - Software Testing Techniques

The importance of software testing to software quality cannot be overemphasized. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation. Once source code has been generated, software must be tested to allow errors to be identified and removed before delivery to the customer. While it is not possible to remove every error in a large software package, the software engineer's goal is to remove as many as possible early in the software development cycle. It is important to remember that testing can only find errors; it cannot prove that a program is bug free. . The greater visibility of software systems and the cost associated with software failure are motivating factors for planning, through testing. It is not uncommon for a software organization to spent 40% of its effort on testing. Two basic test techniques involve testing module input/output (*black-box*) and exercising internal logic of software components (*white-box*). Formal technical reviews by themselves can not find allow software defects, test data must also be used. For large software projects, separate test teams may be used to develop and execute the set of test cases used in testing. Testing must be planned and designed.

The following are some commonly used terms associated with testing:

A **failure** is a manifestation of an error (or defect or bug). But , the mere presence of an error may not necessarily lead to a failure.

A **fault** is an incorrect intermediate state that may have been entered during program execution, e.g., a variable value is different from what it should be. A fault may or may not lead to a failure.

A **test data** is the inputs which have been devised to test the system.

A **test case** is the triplet [I,S,O], where I is the data input to the system, S is the state of the system at which data is input, and O is the expected output of the system or Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

During testing the software engineering produces a series of test cases that are used to "rip apart" the software they have produced. Testing is the one step in the software process that can be seen by the developer as destructive instead of constructive. Software engineers are typically constructive people and testing requires them to overcome preconceived concepts of correctness and deal with conflicts when errors are identified.

No single type of test provides enough information to quantify the quality of a system. Instead, multiple types of testing must be undertaken to fully ascertain a software application's quality. However, it says that " You can't test in quality. If its not there before you begin testing, it won't be there when you're finished testing."

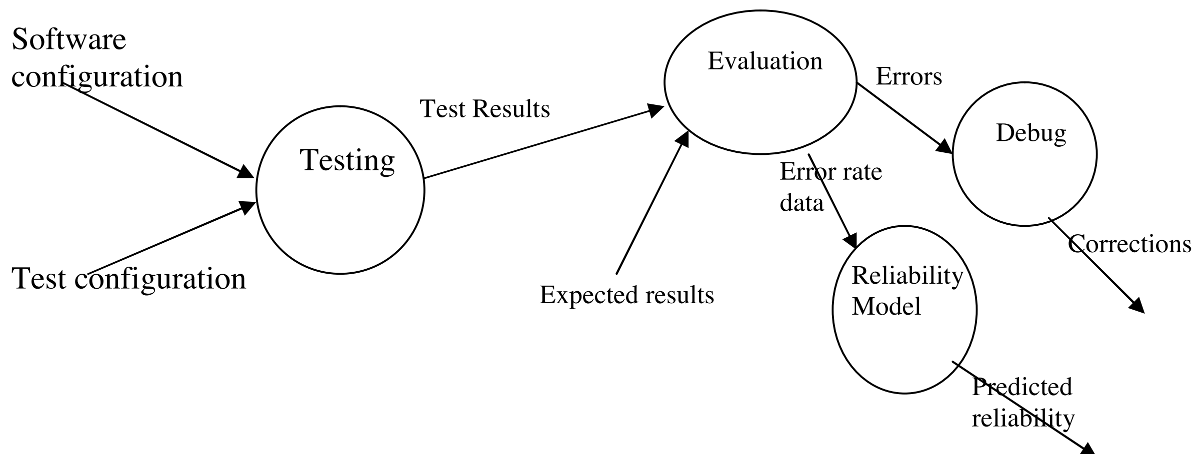
Software Testing Objectives

- Testing is the process of executing a program with the intent of finding errors.
- A good test case is one with a high probability of finding an as-yet undiscovered error.
- A successful test is one that discovers an as-yet-undiscovered error.

 Downloaded from: sarojpandey.com.np

Test information flow

Information flow for testing follows the pattern shown in the figure below. Two types of input are given to the test process: (1) a software configuration; (2) a test configuration. Tests are performed and all outcomes considered, test results are compared with expected results. When erroneous data is identified error is implied and debugging begins. The debugging procedure is the most unpredictable element of the testing procedure. An "error" that indicates a discrepancy of 0.01 percent between the expected and the actual results can take hours, days or months to identify and correct. It is the uncertainty in debugging that causes testing to be difficult to schedule reliability.



Test information flow

Software Testing Principles

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- The *Pareto principle* (80% of all errors will likely be found in 20% of the code) applies to software testing.
- Testing should begin in the small and progress to the large.
- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party.

Software Testability Checklist

- *Operability*- the better it works the more efficiently it can be tested
- *Observability*- what you see is what you test
- *Controllability*- the better software can be controlled the more testing can be automated and optimized
- *Decomposability*- by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently
- *Simplicity*- the less there is to test, the more quickly we can test
- *Stability*- the fewer the changes, the fewer the disruptions to testing

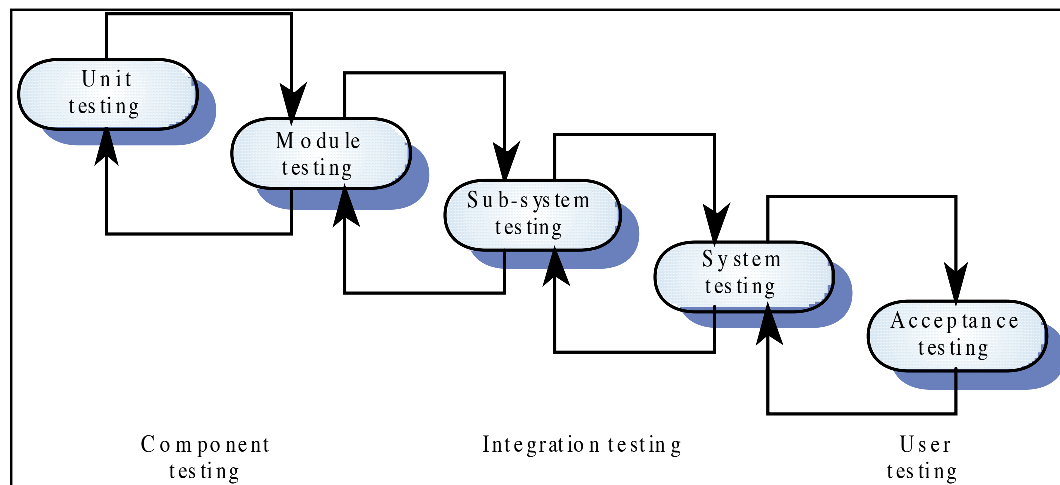
 Downloaded from: sarojpandey.com.np

- *Understandability*-the more information known, the smarter the testing

Good Test Attributes

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be best of breed.
- A good test should not be too simple or too complex.

Software Testing Process



- Unit testing - Individual components are tested independently, without other system components
- Module testing - Related collections of dependent components(class, ADT, procedures & functions) are tested, without other system module.
- Sub-system testing-Modules are integrated into sub-systems and tested. The focus here should be on interface testing to detect module interface errors or mismatches.
- System testing - Testing of the system as a whole. Validating functional and non-functional requirements & Testing of emergent system properties.
- Acceptance testing-Testing with customer data to check that it is acceptable. Also called *Alpha Testing.*
  - Making sure the software works correctly for intended user in his or her normal work environment.
  - Alpha test-version of the complete software is tested by customer under the supervision of the developer at the developer's site.
  - Beta test-version of the complete software is tested by customer at his or her own site without the developer being present

Component testing

- Testing of individual program components.
- Usually the responsibility of the component developer (except sometimes for critical systems).
- Tests are derived from the developer's experience.

 Downloaded from: sarojpandey.com.np

Integration testing

- Testing of groups of components integrated to create a system or sub-system.
- The responsibility of an independent testing team.
- Tests are based on a system specification.

Testing Phases

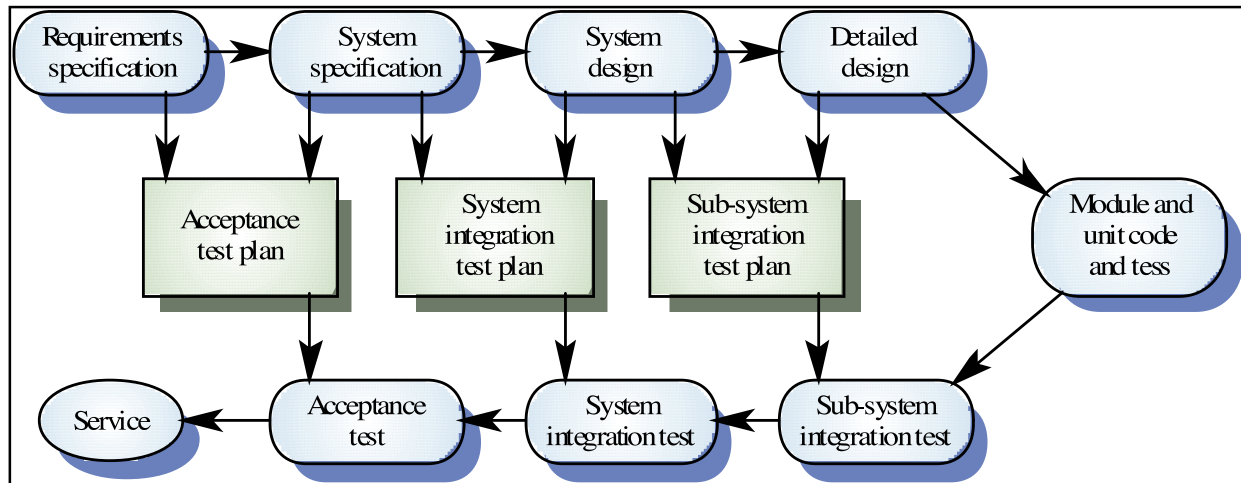


Fig: Software Testing phases

Test case design

The design of software testing can be a challenging process. However software engineers often see testing as an afterthought, producing test cases that feel right but have little assurance that they are complete. *The primary objective of testing is to have the highest likelihood of finding the most errors with a minimum amount of timing and effort*. A large number of test case design methods have been developed that offer the developer with a systematic approach to testing. Methods offer an approach that can ensure the completeness of tests and offer the highest likelihood for uncovering errors in software.

Any engineering product can be tested in two ways: (1) Knowing the specified functions that the product has been designed to perform, tests can be performed that show that each function is fully operational (2) knowing the internal workings of a product, tests can be performed to see if they jell. The first test approach is known as a **black box testing** and the second ***white box testing***.

Black box testing relates to the tests that are performed at the software interface. Although they are designed identify errors, black box tests are used to demonstrate that software functions are operational; that inputs are correctly accepted and the output is correctly produced. A black box test considers elements of the system with little interest in the internal logical arrangement of the software. White box testing of software involves a closer examination of procedural detail. Logical paths through the software are considered by providing test cases that exercise particular sets of conditions and/or loops. The status of the system can be identified at diverse points to establish if the expected status matches the actual status.

 Downloaded from: sarojpandey.com.np

Test Case Design Strategies

- Black-box or behavioral testing
- White-box or glass-box testing

Black Box testing

- Also known as ***Behavioral*** or ***Functional*** testing.
- The system is a " *Black-box*" whose behavior is determined by studying its input and related the outputs.
- It is techniques of knowing the specified function a product is to perform and demonstrating correct operation based solely on its specification without regard for its internal logic.
- Focus on the functional requirements of the software i.e., information domain not the implementation part of the software.
- Different categories of errors include incorrect or missing functions, interface error, errors in data structures or external database access, behavior or performance errors, and initiation and termination errors.
- It is performed during later stages of testing process, like in the acceptance testing or beta testing.
- Test are designed to answer the following questions:
    - How is functional validity tested?
    - How is system behavior and performance tested?
    - What classes of input behavior will make good test case?
    - Is the system particularly sensitive to certain input values?
    - How are the boundaries of data class isolated?
    - What data rates and data volume can the system tolerate?
    - What effect will specific combinations of data have on system operations?
- Techniques includes Equivalence Partitioning, Boundary Value Analysis, Comparison Testing, Orthogonal Array Testing.
- Advantages:
    - Validates whether or not a given system conforms to its software specification
    - Introduce a series of inputs to a system and compare the outputs to a pre-defined test specification.
    - Test integration between individual system components.
    - Tests are architecture independent — they do not concern themselves with how a given output is produced, only with whether that output is the desired and expected output.
    - Require no knowledge of the underlying system, one need not be a software engineer to design black box tests.
- Disadvantages:
    - Offer no guarantee that every line of code has been tested.
    - Being architecture independent, it cannot determine the efficiency of the code.
    - Will not find any errors, such as memory leaks, that are not explicitly and instantly exposed by the application.

 Downloaded from: sarojpandey.com.np

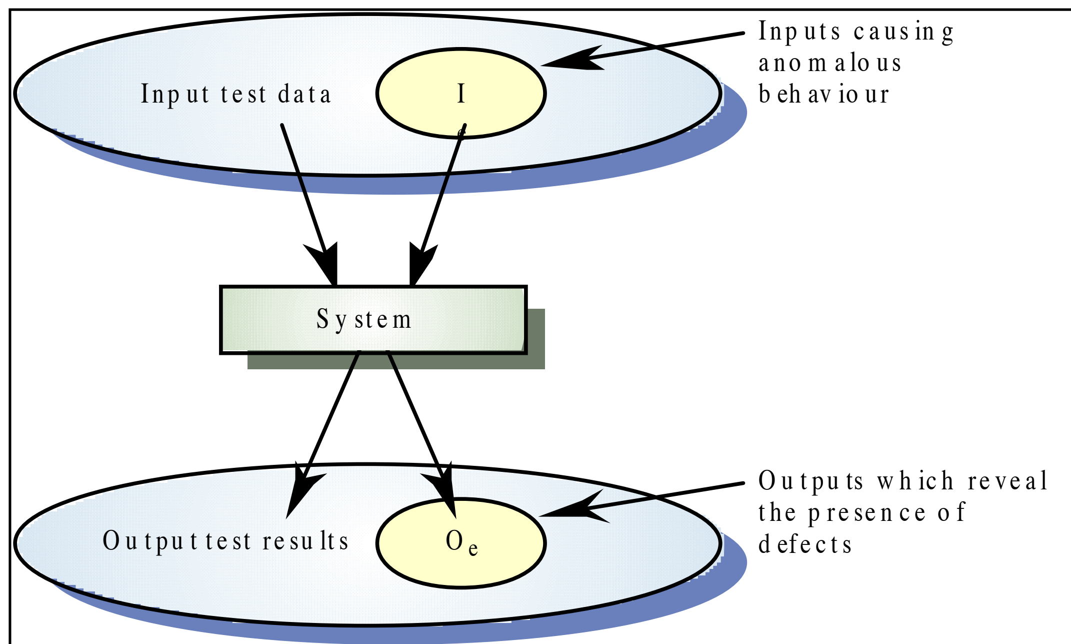


Fig: Black box testing

White-box testing

- Also known as ***Structural*** or ***Glass-box*** testing.
- Knowing the internal workings of a product, tests are performed to check the workings of all independent logic paths.
- It uses the control structure of the procedural design to derive test cases that:
    - Guarantee that all independent paths within a module have been exercised at least once.
    - Exercise all logical decisions on their true and false sides.
    - Execute all loops at their boundaries and within their operational bounds
    - Exercise internal data structures to ensure their validity
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)
- Techniques being used: *basic path* and *control structure testing.*

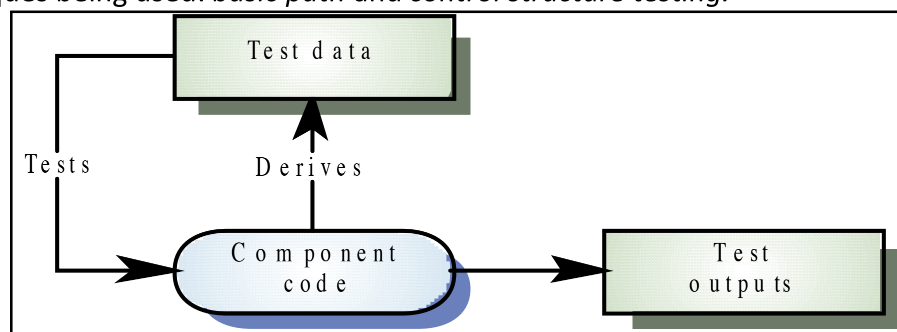


Fig: Structural testing or while box testing

 Downloaded from: sarojpandey.com.np

Difference between Black-box testing and White-box testing

Black-box testing	White-box testing
Also referred as Function testing as it test the functionality of the system	Also referred as Structural testing as it checks the workings of all independent logic paths.
Also known as behavioral testing as system behavior is determined by studying its input and related the outputs.	Also known as glass-box testing as its internal coding part should be tested and verified.
Called " testing in the large"	Called “testing in the small" as applied to small program components.
Knowledge of internal program logic is not important.	Knowledge of internal program logic is an essential
Objective is to check it meets the user requirements as per the specification documents.	Objective is to check all program statement as per the design specification.
Discover ***faults of omission***, indicating that part of the specification has not been fulfilled.	Discover ***faults of commission***, indicating that part of the implementation is faulty.
Mostly performed by the user's side.	Mostly performed by the programmer's side
Back-box testing is done at the later stage of the SWSDLC.	While-box testing is done at the earlier stage of the SWDLC.
Performed as per the acceptance test plan.	Performed as per the Component and Integration test plan.
Techniques include Equivalence Partitioning, Boundary Value Analysis, Comparison Testing, and Orthogonal Array Testing.	Techniques include basic path and control structure testing
Errors include incorrect or missing functions, interface error, errors in data structures or external database access, behavior or performance errors, and initiation and termination errors.	Errors include logical errors in the coding statement and typing error.

Basis Path Testing

- White-box technique usually based on the program flow graph. It derives logical complexity measures of a procedural design for defining a basis set of execution paths.
- Test cases produced to exercise each statement in the program at least one time during testing.
- The *cyclomatic complexity* of the program computed from its flow graph using the formula $V(G) = E - N + 2$ or by counting the conditional statements in the PDL representation and adding 1
- Determine the basis set of linearly independent paths (the cardinality of this set id the program cyclomatic complexity)
- Prepare test cases that will force the execution of each path in the basis set.

Flow graph notation

The flow graph can be used to represent the logical flow control and therefore all the execution paths that need testing. To illustrate the use of flow graphs consider the procedural design depicted in the flow chart below. This is mapped into the flow graph below where the circles are nodes that represent one or more procedural statements and the arrow in the flow graph called edges

 Downloaded from: sarojpandey.com.np

represent the flow control. Each node that includes a condition is known as a predicate node, and has two or more edges coming from it.

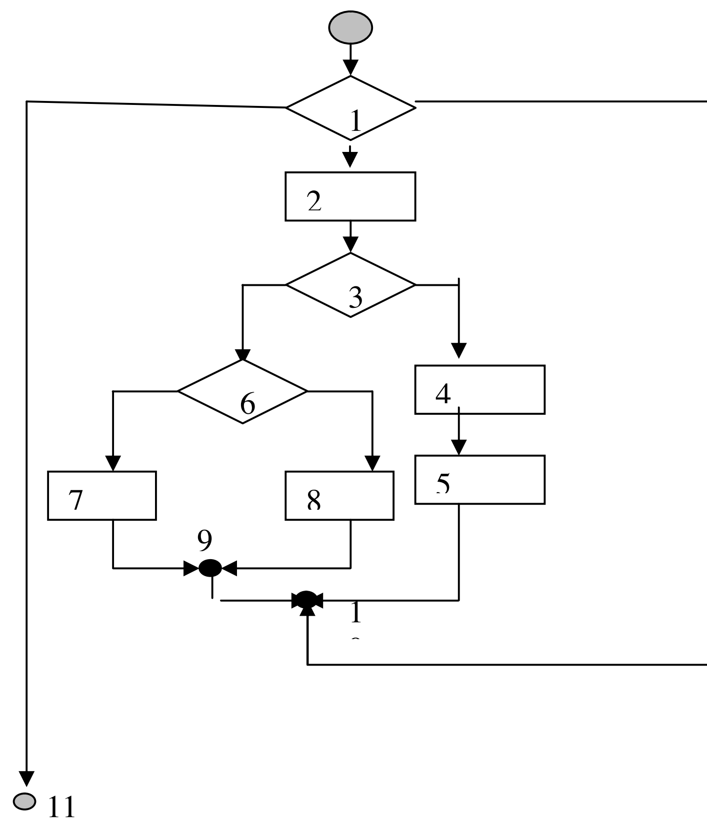


Fig: Flow Chart

Downloaded from: sarojpandey.com.np

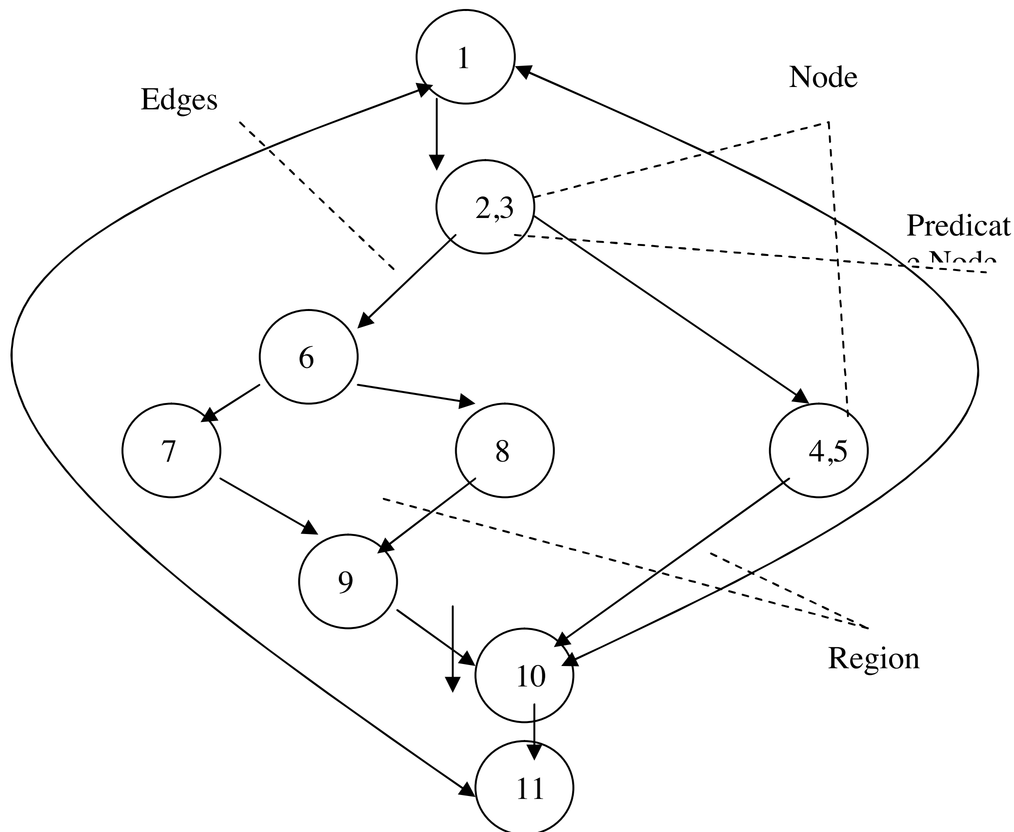


Fig: Flow graph

Control Structure Testing

- White-box techniques focusing on control structures present in the software
- Condition testing (e.g. branch testing) focuses on testing each decision statement in a software module, it is important to ensure coverage of all logical combinations of data that may be processed by the module (a truth table may be helpful)
- Data flow testing selects test paths based according to the locations of variable definitions and uses in the program (e.g. definition use chains)
- Loop testing focuses on the validity of the program loop constructs (i.e. simple loops, concatenated loops, nested loops, unstructured loops), involves checking to ensure loops start and stop when they are supposed to (unstructured loops should be redesigned whenever possible)

Graph-based Testing Methods

- Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph
- Transaction flow testing (nodes represent steps in some transaction and links represent logical connections between steps that need to be validated)
- Finite state modeling (nodes represent user observable states of the software and links represent transitions between states)

 Downloaded from: sarojpandey.com.np

- Data flow modeling (nodes are data objects and links are transformations from one data object to another)
- Timing modeling (nodes are program objects and links are sequential connections between these objects, link weights are required execution times)

Equivalence Partitioning

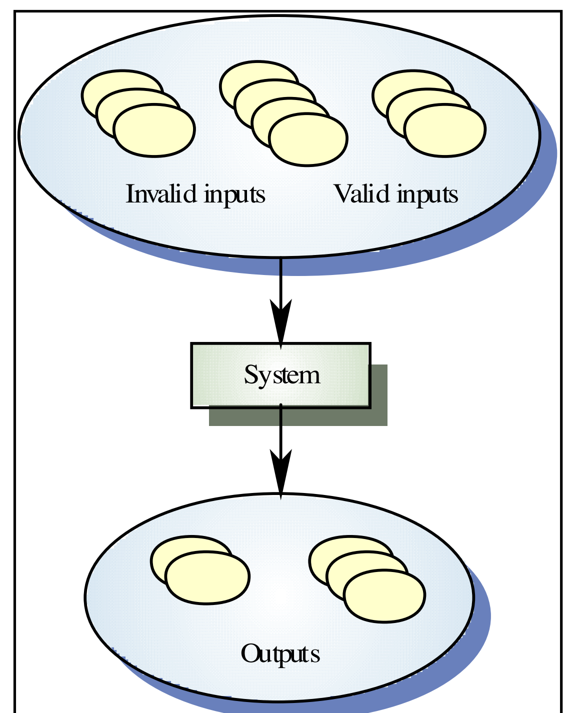
- Black-box technique that divides the input domain into classes of data from which test cases can be derived
- Input data and output results often fall into different classes where all members of a class are related
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition
- Equivalence class guidelines:
  1. If input condition specifies a range, one valid and two invalid equivalence classes are defined
  2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined
  3. If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined
  4. If an input condition is Boolean, one valid and one invalid equivalence class is defined

Partition system inputs and outputs into 'equivalence sets'

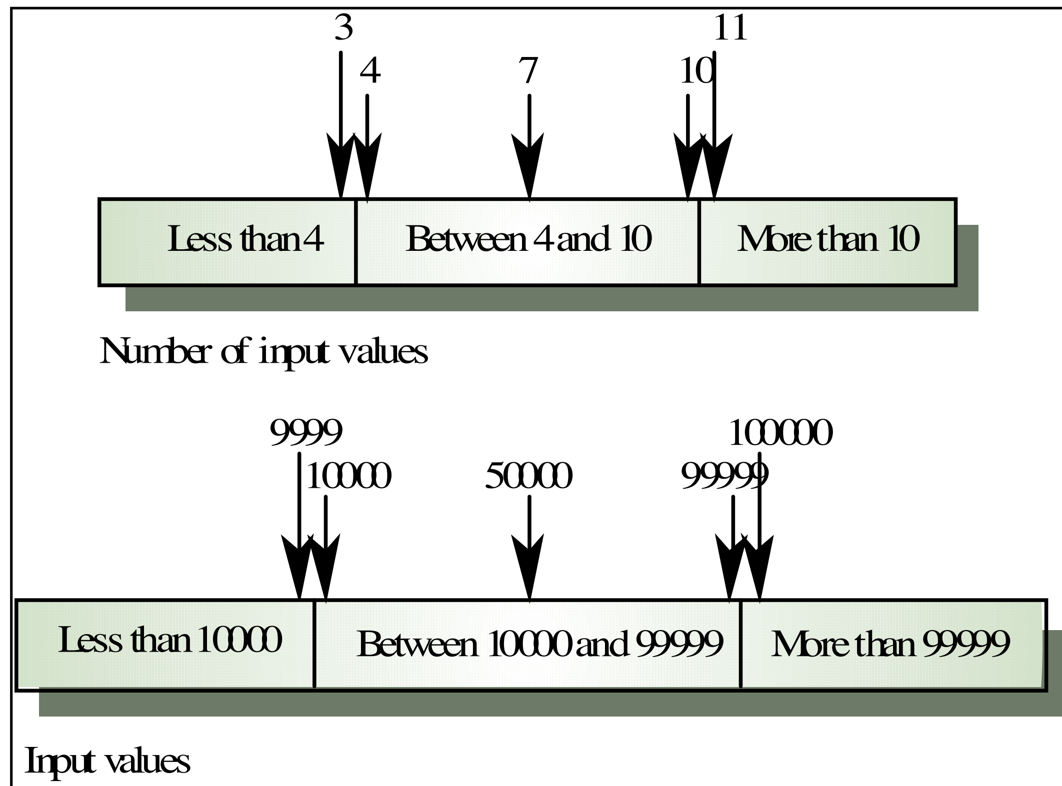
If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are <10,000, 10,000-99, 999 and > 10, 000

Choose test cases at the boundary of these sets

00000, 09999, 10000, 99999, 10001



Downloaded from: sarojpandey.com.np



Boundary Value Analysis

- Black-box technique that focuses on the boundaries of the input domain rather than its center
- BVA guidelines:
  1. If input condition specifies a range bounded by values a and b, test cases should include a and b, values just above and just below a and b
  2. If an input condition specifies and number of values, test cases should be exercise the minimum and maximum numbers, as well as values just above and just below the minimum and maximum values
  3. Apply guidelines 1 and 2 to output conditions, test cases should be designed to produce the minimum and maxim output reports
  4. If internal program data structures have boundaries (e.g. size limitations), be certain to test the boundaries

Comparison Testing

- Black-box testing for safety critical systems in which independently developed implementations of redundant systems are tested for conformance to specifications
- Often equivalence class partitioning is used to develop a common set of test cases for each implementation.

 Downloaded from: sarojpandey.com.np

Orthogonal Array Testing

- Black-box technique that enables the design of a reasonably small set of test cases that provide maximum test coverage
- Focus is on categories of faulty logic likely to be present in the software component (without examining the code)
- Priorities for assessing tests using an orthogonal array
  - Detect and isolate all single mode faults
  - Detect all double mode faults
  - Mutimode faults

Specialized Testing

- It encompass a broad array of software capabilities and application areas.
- Graphical user interfaces
- Client/server architectures
- Documentation and help facilities
- Real-time systems
  - Task testing (test each time dependent task independently)
  - Behavioral testing (simulate system response to external events)
  - Intertask testing (check communications errors among tasks)
  - System testing (check interaction of integrated system software and hardware)

Software Testing Strategies-Overview

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Software testing must be planned carefully to avoid wasting development time and resources. Testing begins "in the small" and progresses "to the large". Initially individual components are tested using white box and black box techniques. After the individual components have been tested and added to the system, integration testing takes place. Once the full software product is completed, system testing is performed. The Test Specification document should be reviewed like all other software engineering work products.

Strategic Approach to Software Testing

Testing is a set of activities that can be planned in advance and conducted systematically. Software testing strategy should have the following characteristics:

- Testing begins at the component level and works outward toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software conducts testing and may be assisted by independent test groups for large projects.
- The role of the independent tester is to remove the conflict of interest inherent when the builder is testing his or her own product.
- Testing and debugging are different activities.

 Downloaded from: sarojpandey.com.np

- Debugging must be accommodated in any testing strategy.
- Make a distinction between verification and validation.

Distinction between Verification and Validation

Software testing is one type of a broader domain that is known as verification and validation (V&V). Verification and validation is intended to show that a system conforms to its specification and meets the requirements of the system customer. It involves checking and review processes and system testing. The following table shows the distinction between verification and validation.

Verification	Validation
A set of operations that the software correctly implements a particular function.	a different set of activities that ensures that the software that has been produced is traceable to customer real needs.
Software verification is achieved through a series of while box tests that shows conformity to its design and statement of codes.	Software validation is achieved through a series of black box tests that show conformity with requirements.
Ensure the correct implementation of a specific function.	Ensure that the software that has been built is traceable to customer requirements.
It focuses on "are we building the product right?"	It focuses on "are we building the right product?"
It is performed at the each and every stages of the software coding like unit, module and subsystem testing.	It is performed at the later stages of the software development like system testing, acceptance testing.

Strategic Testing Issues

- Specify product requirements in a quantifiable manner before testing starts.
- Specify testing objectives explicitly.
- Identify the user classes of the software and develop a profile for each.
- Develop a test plan that emphasizes rapid cycle testing.
- Build robust software that is designed to test itself (e.g. uses anitbugging).
- Use effective formal reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases.

Software Testing Plan

Preparation and planning for the test phase should begin early in the development cycle and be a constant concern throughout the development process.

Downloaded from: sarojpandey.com.np

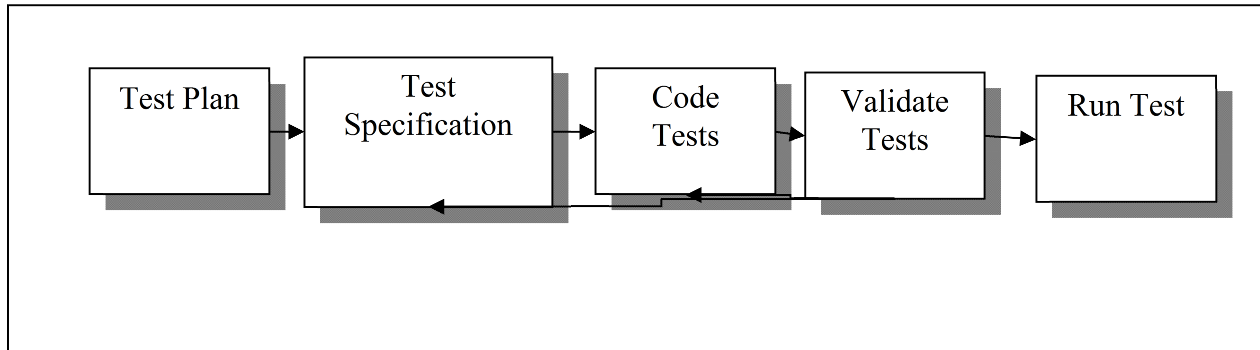


Fig: Test development cycle

The structure of a software testing plan includes:

The testing process: A description of the major phases of the testing process.

Requirements traceability: It describes the system meeting users requirements and testing should be planned so that all requirements are individually tested.

Tested items: Specify the product of the software process to be tested.

Test recording procedures: Recording the tests results systematically and auditing the testing process to check that it has been carried out correctly.

Hardware and Software requirements: Set out software tools required and estimated hardware utilization.

Constraints: Specify the constraints affecting the testing process such as staff shortages.

Unit Testing

- Black box and white box testing.
- Unit testing concentrates verification on the smallest element of the– the module
- Module interfaces are tested for proper information flow.
- Local data are examined to ensure that integrity is maintained.
- Boundary conditions are tested.
- Basis path testing should be used.
- All error handling paths should be tested.
- Drivers and/or stubs need to be developed to test incomplete software.

Unit test considerations

The tests that are performed as part of unit testing are shown in the figure below. The module interface is tested to ensure that information properly flows into and out of the program unit being tested. The local data structure is considered to ensure that data stored temporarily maintains its integrity for all stages in an algorithm's execution. Boundary conditions are tested to ensure that the modules perform correctly at boundaries created to limit or restrict processing. All independent

 Downloaded from: sarojpandey.com.np

paths through the control structure are exercised to ensure that all statements in been executed once. Finally, all error-handling paths are examined.

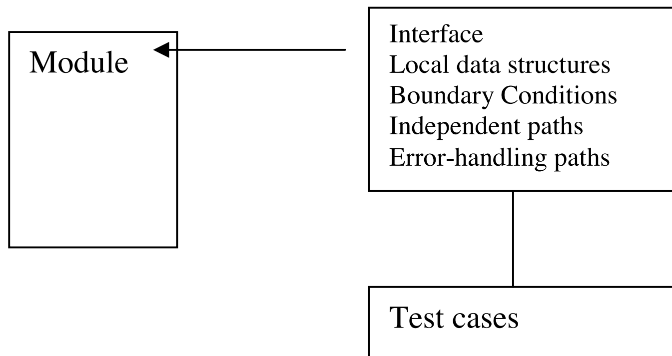


Fig: Unit test

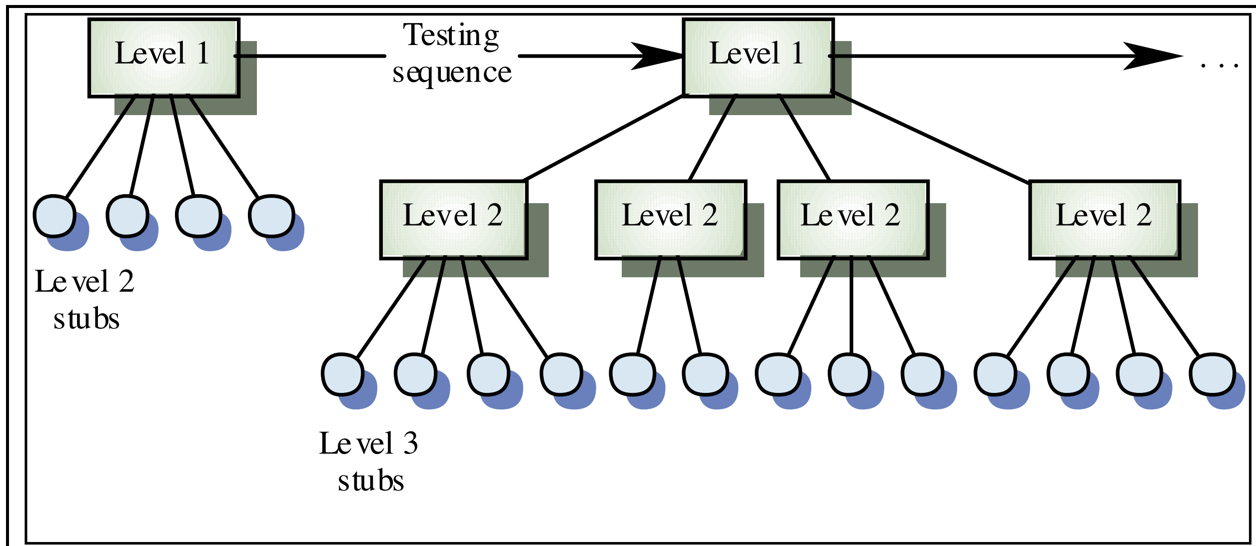
Unit test procedures

Unit testing is typically seen as an adjunct to the coding step. Once source code has been produced, reviewed, and verified for correct syntax, unit test case design can start. A review of design information offers assistance for determining test cases that should uncover errors. Each test case should be linked with a set of anticipated results. As a module is not a stand-alone program, driver and/stub software must be produced for each test units. In most situations a driver is a "main program" that receives test case data, passes this to the module being tested and prints the results. Stubs act as the sub-modules called by the test modules. Unit testing is made easy if a module has cohesion.

Integration Testing

- **Top-down integration testing**
    - Start with high-level system and integrate from the top-down replacing individual components by stubs where appropriate
    - Main control module used as a test driver and stubs are substitutes for components directly subordinate to it.
    - Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).
    - Tests are conducted as each component is integrated.
    - On completion of each set of tests and other stub is replaced with a real component.
    - Regression testing may be used to ensure that new errors not introduced.

 Downloaded from: sarojpandey.com.np



The integration process is performed in a series of five stages:

- The main control module is used as a test driver and stubs are substituted for all modules directly subordinate to the main control module.
- Depending on the integration technique chosen, subordinate stubs are replaced one at a time with actual modules.
- Tests are conducted as each module is integrated.
- On the completion of each group of tests, another stub is replaced with the real module.
- Regression testing may be performed to ensure that new errors have been introduced.

- **Bottom-up integration testing**
  - Integrate individual components in levels until the complete system is created
  - Low level components are combined in clusters that perform a specific software function.
  - A driver (control program) is written to coordinate test case input and output.
  - The cluster is tested.
  - Drivers are removed and clusters are combined moving upward in the program structure.
  - Regression testing (check for defects propagated to other modules by changes made to existing program)
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.

 Downloaded from: sarojpandey.com.np

A bottom-up integration strategy may be implemented with the following steps:

1. Low-level modules are combined into clusters that perform a particular software subfunction.
2. A driver is written to coordinate test cases input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

Comments on Integration Testing

There has been much discussion on the advantages and disadvantages of bottom-up and top-down integration testing. Typically a disadvantage of one is an advantage of the other approach. The major disadvantage of top-down approaches is the need for stubs and the difficulties that are linked with them. Problems linked with stubs may be offset by the advantage of testing major control functions early. The major drawback of bottom-up integration is that the program does not exist until the last module is included.

Smoke testing

- Software components already translated into code are integrated into a build.
- A series of tests designed to expose errors that will keep the build from performing its functions are created.
- The build is integrated with the other builds and the entire product is smoke tested daily (either top-down or bottom integration may be used).

General Software Test Criteria

- Interface integrity (internal and external module interfaces are tested as each module or cluster is added to the software)
- Functional validity (test to uncover functional defects in the software)
- Information content (test for errors in local or global data structures)
- Performance (verify specified performance bounds are tested)

Validation Testing

- Ensure that each function or performance characteristic conforms to its specification.
- Deviations (deficiencies) must be negotiated with the customer to establish a means for resolving the errors.
- Configuration review or audit is used to ensure that all elements of the software configuration have been properly developed, cataloged, and documented to allow its support during its maintenance phase.
- Validation test criteria- Software validation is achieved through a series of black box tests that show conformity with requirements. A test plan provides the classes of tests to be performed and a test procedure sets out particular test cases that are to be used to show conformity with requirements.

 Downloaded from: sarojpandey.com.np

Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
- Alpha test (version of the complete software is tested by customer under the supervision of the developer at the developer's site)
- Beta test (version of the complete software is tested by customer at his or her own site without the developer being present)

System Testing

- Recovery testing (checks the system's ability to recover from failures)
- Security testing (verifies that system protection mechanism prevent improper penetration or data alteration)
- Stress testing (program is checked to see how well it deals with abnormal resource demands – quantity, frequency, or volume)
- Performance testing (designed to test the run-time performance of software, especially real-time software)

Debugging

- Debugging (removal of a defect) occurs as a consequence of successful testing.
- Testing establishes the existence of defects where as debugging is concerned with locating and correcting these defects
- Common approaches:
  1. *Brute force* (memory dumps and run-time traces are examined for clues to error causes)
  2. *Backtracking* (source code is examined by looking backwards from symptom to potential causes of errors)
  3. *Cause elimination* (uses binary partitioning to reduce the number of locations potential where errors can exist)

Debugging Process

Figure illustrates a possible debugging process. Defects in the code must be located and the program modified to meet its requirements. Testing must then be repeated to ensure that the change has been made correctly. Thus debugging process is a part of both software development and software testing.

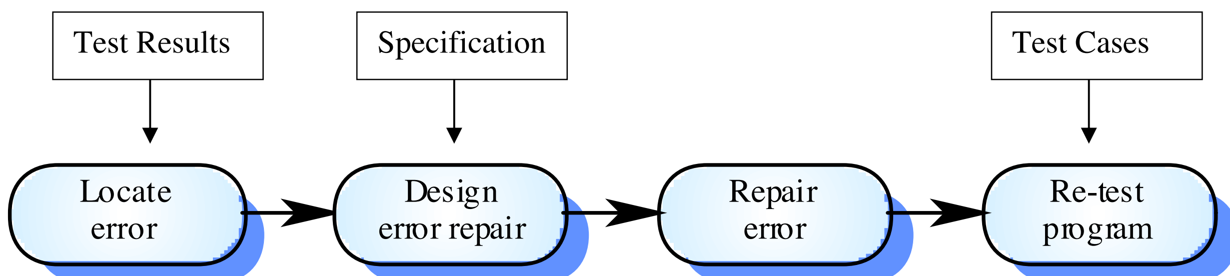


Fig: debugging process

 Downloaded from: sarojpandey.com.np

Bug Removal Considerations

1. Is the cause of the bug reproduced in another part of the program?
2. What "next bug" might be introduced by the fix that is being proposed?
3. What could have been done to prevent this bug in the first place?

TEST SPECIFICATION DOCUMENT

1.0 Introduction

This section provides an overview of the entire test document. This document describes both the test plan and the test procedure.

1.1 Goals and objectives

Overall goals and objectives of the test process are described.

1.2 Statement of scope

A description of the scope of software testing is developed. functionality/features/behavior to be tested is noted. In addition any functionality/features/behavior that is not to be tested is also noted.

1.3 Major constraints

Any business, product line or technical constraints that will impact the manner in which the software is to be tested are noted here.

2.0 Test Plan

This section describes the overall testing strategy and the project management issues that are required to properly execute effective tests.

2.1 Software (SCIís) to be tested

The software to be tested is identified by name. Exclusions are noted explicitly.

2.2 Testing strategy

The overall strategy for software testing is described.

2.2.1 Unit testing

The strategy for unit tested is described. This includes an indication of the components that will undergo unit tests or the criteria to be used to select components for unit test. Test cases are NOT included here.

2.2.2 Integration testing

The integration testing strategy is specified. This section includes a discussion of the order of integration by software function. Test cases are NOT included here.

2.2.3 Validation testing

The validation testing strategy is specified. This section includes a discussion of the order of validation by software function. Test cases are NOT included here.

2.2.4 High-order testing

The high-order testing strategy is specified. This section includes a discussion of the types of high order tests to be conducted, the responsibility for those tests. Test cases are NOT included here.

2.3 Testing resources and staffing

Specialized testing resources are described and staffing is defined. The role of any ITG is also defined.

 Downloaded from: sarojpandey.com.np

2.4 Test work products

The work products produced as a consequence of the testing strategy are identified.

2.5 Test record keeping

Mechanisms for storing and evaluating test results are specified.

2.6 Test metrics

A description of all test metrics to be used during the testing activity is noted here.

2.7 Testing tools and environment

A description of the test environment, including tools, simulators, specialized hardware, test files, and other resources is presented here.

2.8 Test schedule

A detailed schedule for unit, integration, and validation testing as well as high order tests is described.

3.0 Test Procedure

This section describes as detailed test procedure including test tactics and test cases for the software.

3.1 Software (SCIís) to be tested

The software to be tested is identified by name. Exclusions are noted explicitly.

3.2 Testing procedure

The overall procedure for software testing is described.

3.2.1 Unit test cases

The procedure for unit testing is described for each software component (that will be unit tested) is presented. This section is repeated for all components).

3.2.1.2 Stubs and/or drivers for component i
3.2.1.3 Test cases component i
3.2.1.4 Purpose of tests for component i
3.2.1.5 Expected results for component i
3.2.2 Integration testing: The integration testing procedure is specified.
3.2.2.1 Testing procedure for integration
3.2.2.2 Stubs and drivers required
3.2.2.3 Test cases and their purpose
3.2.2.4 Expected results
3.2.3 Validation testing: The validation testing procedure is specified.
3.2.3.1 Testing procedure for validation
3.2.3.3 Expected results
3.2.3.4 Pass/fail criterion for all validation tests
3.2.4 High-order testing (a.k.a. System Testing)

The high-order testing procedure is specified. For each of the high order tests specified below, the test procedure, test cases, purpose, specialized requirements and pass/fail criteria are specified. It should be noted that not all high-order test methods noted in Sections 3.2.4.n will be conducted for every project.

3.2.4.1 Recovery testing
3.2.4.2 Security testing
3.2.4.3 Stress testing
3.2.4.4 Performance testing
3.2.4.5 Alpha/beta testing
3.2.4.6 Pass/fail criterion for all validation tests

 Downloaded from: sarojpandey.com.np

3.3 Testing resources and staffing

Specialized testing resources are described and staffing is defined. The role of any ITG is also defined.

3.4 Test work products

The work products produced as a consequence of the testing procedure are identified.

3.5 Test record keeping and test log

Mechanisms for storing and evaluating test results are specified. The test log is used to maintain a chronological record of all tests and their results.

Summary

- It is more important to test the parts of the system which are commonly used rather than parts which are only rarely exercised.

- Equivalent partitioning is way of deriving test cases. It depends on finding partitions in the input and output data sets and exercising the program with values from these partitions. Often, the value which is most likely to lead to a successful test is a value at the boundary of a partition.

- Black-box testing does not need access to source code. Test cases are derived from the program specification.
- Structural testing relies on a analyzing a program to determine paths through it and using this analysis to assist with the selection of test cases.

- Integration testing should focus on testing the interactions between the components in a system and component interfaces.
- Interface defects may arise because of errors made in reading the specification, specification misunderstandings or errors or invalid timing assumptions. Interface testing is intended to discover defects in the interfaces of objects or modules.

- When testing object classes, you should design tests that exercise all of the operations associated with a class, assign and evaluate all object attributes and test the object in all possible states.

- Object oriented systems should be integrated around natural clusters of objects such as those associated with a particular use-cases or with object threads.

Assignments Questions

- Discuss the difference between black-box and structural testing and suggest how they can be used together in the defect testing process.
- Show, using a small example, why it is practically impossible to exhaustively test a program.
- Explain why interface testing is necessary given that individual units have been extensively validated through unit testing and program inspections.
- Explain why bottom-up and top-down testing may be inappropriate testing strategies for object-oriented systems.
- What is black-box testing? Explain the advantages and disadvantages of back-box testing.
- Why White-box testing is called Glass-box testing? Explain the flow graph-notation

 Downloaded from: sarojpandey.com.np

Chapter 18 - Software Quality Assurance

This chapter provides an introduction for software quality assurance (SQA). SQA is the concern of every software engineer to reduce cost and improve product time-to-market. A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are included in an SQA plan. SQA activities are performed on every software project. Use of metrics is an important part of developing a strategy to improve the quality of both software processes and work products.

Quality Concepts

- Quality as “a characteristics or attribute of something.”
- Quality, simplistically, means that a product should meet its specification
- **Quality of design**
  - It refers to characteristics of an end item as specified by designer.
  - In software development, it encompasses requirements, specifications, and the design of the system.
- **Quality of conformance**
  - Degree to which the design specifications are followed during manufacturing.
  - In software, Quality of conformance focuses on the implementation part.

User satisfaction= compliant product+ good quality+ delivery within budget and schedule.

Software quality defined...

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

The above definition serves three important points:

- Software requirement forms a foundation for quality being measured. Lack of conformance to requirements is lack of quality.
- If specified standards, that define a set of development criteria, is not followed; lack of quality will almost surely result.
- If software conforms to its explicit requirements but fails to meet implicit requirements (ease of use and maintainability), software quality is suspect.

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product
- Involves defining appropriate quality standards and procedures and ensuring that these are followed
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility

Quality Management activities

- **Quality assurance**
  - Establish a framework of organizational procedures and standards for high-quality software.

 Downloaded from: sarojpandey.com.np

- Consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions.
- If data provided through QA identifies problems, management is responsible to address the problems and apply the necessary resources to resolve quality issues.

❖ **Quality planning**

- Selection of appropriate procedures and standards from this framework and adaptation of these for specific software projects and modify these as required.
- A quality plan sets out the desired product qualities and how these are assessed and define the most significant quality attributes
- It should define the quality assessment process.
- It should set out which organisational standards should be applied and, if necessary, define new standards.

❖ **Quality control**

- Definition and enactment of processes which ensure that the project quality procedures and standards are followed by the software development team.
- Variation control is the heart of quality control (software engineers strive to control the process applied, resources expended, and end product quality attributes).
- QC involves the series of inspection, reviews, and test used throughout the software process to ensure conformance of a work product to its specifications.
- Feedback loop to the process that created the work product, minimize the defects produced.
- Key concept of QC- all work products have been defined, measurable specifications to compare the output of each process.
- Two approaches to quality control
    - Quality reviews
    - Automated software assessment and software measurement

Quality management provides an independent check on the software development process. The deliverables from the software process are input to the quality management process and are checked to ensure that they are consistent with organizational standards and goals. As the quality assurance and control team should be independent, they can rake an objective view of the process and can report problems and difficulties to senior management in the organization.

Downloaded from: sarojpandey.com.np

Quality management should be separate from project management to ensure independence without compromising on quality for project budget and schedule.

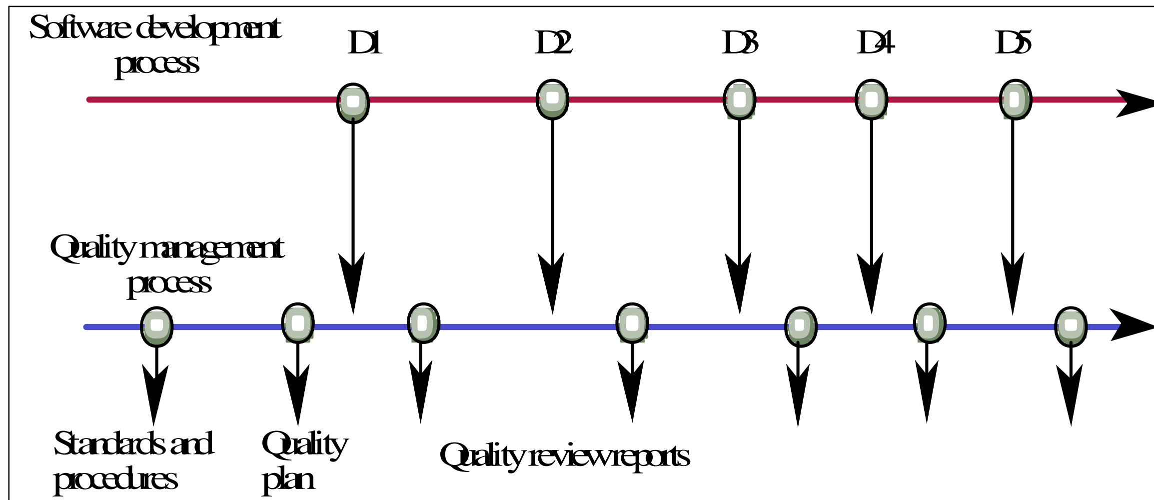


Fig: Quality management and software development

Cost of Quality

It includes all costs incurred in the pursuit of quality or in performing quality related activities. Aim is to provide a baseline for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison.
Quality costs may be divided into costs associated with prevention, appraisal, and failure as mentioned below:

- **Prevention costs** - quality planning, formal technical reviews (FTR), test equipment, training.
- **Appraisal costs** includes activities to gain insight into product condition the "first time through" each process. Example include - in-process and inter-process inspection, equipment calibration and maintenance, testing.
- **Failure costs**
    - *Internal failure costs* are incurred when a defect in a product is detected prior to shipment and include - rework, repair, failure mode analysis
    - *External failure costs* are associated with defects found after the product has been shipped to the customer which include- complaint resolution, product return and replacement, help line support, warranty work

Total Quality Management (TQM)

- **Kaizen** - develop a process that is visible, repeatable, and measurable.
- **Atarimae hinshitsu** - examine the intangibles that affect the process and work to optimize their impact on the process.
- **Kansei** - examine the way the product is used by the customer with an eye to improving both the product and the development process
- **Miryokuteki hinshitsu** - observe product use in the market place to uncover new product applications and identify new products to develop.

 Downloaded from: sarojpandey.com.np

Software Quality Assurance (SQA)

- The aim of SQA is to help an organization develop high quality software in a repeatable manner. A repeatable development software organization is the one where the software development process is person-independent.
- Conformance to software requirements is the foundation from which software quality is measured.
- Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.
- Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

SQA Group Activities

- Prepare SQA plan for the project.
- Participate in the development of the project's software process description.
- Review software engineering activities to verify compliance with the defined software process.
- Audit designated software work products to verify compliance with those defined as part of the software process.
- Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
- Record any evidence of noncompliance and reports them to management.

Software Reviews

- Purpose is to find defects (errors) before they are passed on to another software engineering activity or released to the customer.
- Software engineers (and others) conduct formal technical reviews (FTR) for software engineers.
- Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

Formal Technical Reviews

- Involves 3 to 5 people (including reviewers)
- Advance preparation (no more than 2 hours per person) required
- Duration of review meeting should be less than 2 hours
- Focus of review is on a discrete work product
- Review leader organizes the review meeting at the producer's request
- Reviewers ask questions that enable the producer to discover his or her own error (the product is under review not the producer)
- Producer of the work product walks the reviewers through the product
- Recorder writes down any significant issues raised during the review
- Reviewers decide to accept or reject the work product and whether to require additional reviews of product or not

Statistical Quality Assurance

- Information about software defects is collected and categorized
- Each defect is traced back to its cause

 Downloaded from: sarojpandey.com.np

- Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the "vital few" defect causes
- Move to correct the problems that caused the defects

Software Reliability

- Defined as the probability of failure free operation of a computer program in a specified environment for a specified time period
- Can be measured directly and estimated using historical and developmental data (unlike many other software quality factors)
- Software reliability problems can usually be traced back to errors in design or implementation.

Software Safety

- Defined as a software quality assurance activity that focuses on identifying potential hazards that may cause a software system to fail.
- Early identification of software hazards allows developers to specify design features to can eliminate or at least control the impact of potential hazards.
- Software reliability involves determining the likelihood that a failure will occur, while software safety examines the ways in which failures may result in conditions that can lead to a mishap.

Mistake-Proofing Software

- Poka-yoke devices are mechanisms that lead to the prevention of a potential quality problem before it occurs or to the rapid detection of a quality problem if one is introduced
- Poka-yoke devices are simple, cheap, part of the engineering process, and are located near the process task where the mistakes occur

ISO Quality Standards

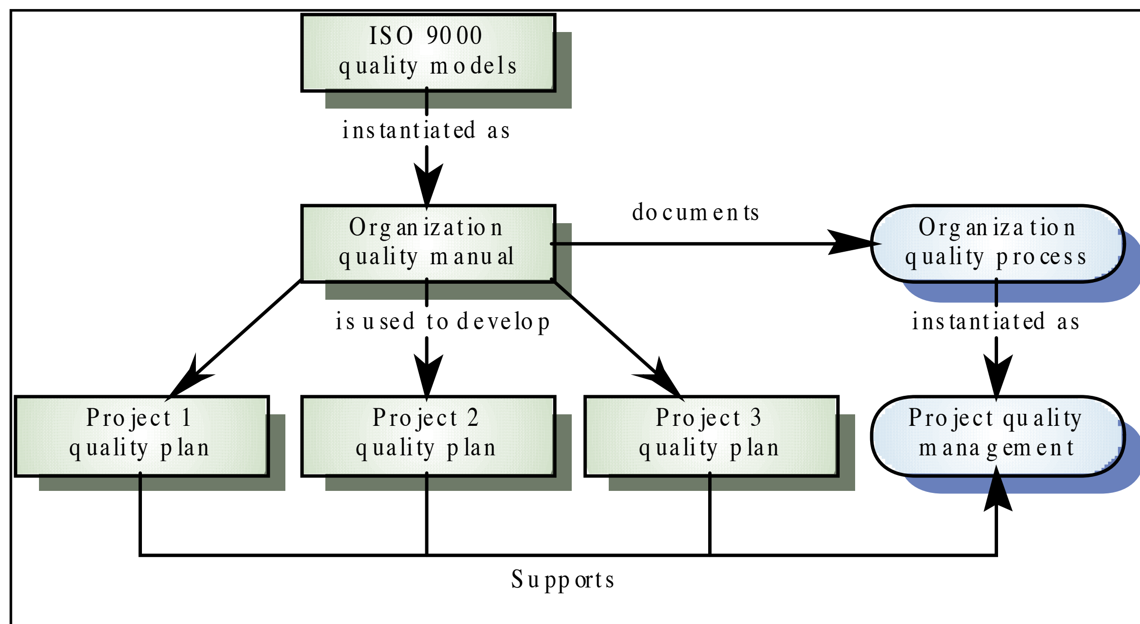
- International set of standards for quality management
- Applicable to a range of organisations from manufacturing to service industries
- Quality assurance systems are defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
- ISO 9000 describes the quality elements that must be present for a quality assurance system to be compliant with the standard, but it does not describe how an organization should implement these elements.
- ISO 9001 is the quality standard that contains 20 requirements that must be present in an effective software quality assurance system.
- ISO 9001 applicable to organisations which design, develop and maintain products
- ISO 9001 is a generic model of the quality process Must be instantiated for each organisation

 Downloaded from: sarojpandey.com.np

ISO 9001

Management responsibility	Quality system
Control of non-conforming products	Design control
Handling, storage, packaging and delivery	Purchasing
Purchaser-supplied products	Product identification and traceability
Process control	Inspection and testing
Inspection and test equipment	Inspection and test status
Contract review	Corrective action
Document control	Quality records
Internal quality audits	Training
Servicing	Statistical techniques

- ☑ Quality standards and procedures should be documented in an organisational quality manual
- ☑ External body may certify that an organisation's quality manual conforms to ISO 9000 standards
- ☑ Customers are, increasingly, demanding that suppliers are ISO 9000 certified



SQA Plan

- ❖ Management section - describes the place of SQA in the structure of the organization
- ❖ Documentation section - describes each work product produced as part of the software process
- ❖ Standards, practices, and conventions section - lists all applicable standards/practices applied during the software process and any metrics to be collected as part of the software engineering work
- ❖ Reviews and audits section - provides an overview of the approach used in the reviews and audits to be conducted during the project
- ❖ Test section - references the test plan and procedure document and defines test record keeping requirements

 Downloaded from: sarojpandey.com.np

- Problem reporting and corrective action section - defines procedures for reporting, tracking, and resolving errors or defects, identifies organizational responsibilities for these activities
- Other - tools, SQA methods, change control, record keeping, training, and risk management

Assignments

- Discuss the difference between black-box and structural testing and suggest how they can be used together in the defect testing process.

- Show, using a small example, why it is practically impossible to exhaustively test a program.

- Explain why interface testing is necessary given that individual units have been extensively validated through unit testing and program inspections.

- Explain why bottom-up and top-down testing may be inappropriate testing strategies for object-oriented systems.

- What is black-box testing? Explain the advantages and disadvantages of back-box testing.

- Why White-box testing is called Glass-box testing ? Explain the flow graph-notation

 Downloaded from: sarojpandey.com.np

ANNEXURE-I

SOFTWARE DESIGN SPECIFICATION TEMPLATES

1.0 Introduction

This section provides an overview of the entire design document. This document describes all data, architectural, interface and component-level design for the software.

1.1 Goals and objectives

Overall goals and software objectives are described.

1.2 Statement of scope

A description of the software is presented. Major inputs, processing functionality, and outputs are described without regard to implementation detail.

1.3 Software context

The software is placed in a business or product line context. Strategic issues relevant to context are discussed. The intent is for the reader to understand the 'big picture'.

1.4 Major constraints

Any business or product line constraints that will impact he manner in which the software is to be specified, designed, implemented or tested are noted here.

2.0 Data design

A description of all data structures including internal, global, and temporary data structures.

2.1 Internal software data structure

Data structures that are passed among components the software are described.

2.2 Global data structure

Data structured that are available to major portions of the architecture are described.

2.3 Temporary data structure

Files created for interim use are described.

2.4 Database description

Database(s) created as part of the application is(are) described.

3.0 Architectural and component-level design

A description of the program architecture is presented.

3.1 Program Structure

A detailed description the program structure chosen for the application is presented.

3.1.1 Architecture diagram

A pictorial representation of the architecture is presented.

3.1.2 Alternatives

A discussion of other architectural styles considered is presented. Reasons for the selection of the style presented in Section3.1.1 are provided.

3.2 Description for Component n

A detailed description of each software component contained within the architecture is presented. Section 3.2 is repeated for each of n components.

3.2.1 Processing narrative (PSPEC) for component n

A processing narrative for component n is presented.

3.2.2 Component n interface description.

A detailed description of the input and output interfaces for the component is presented.

3.2.3 Component n processing detail

Downloaded from: sarojpandey.com.np

A detailed algorithmic description for each component is presented. Section 3.2.3 is repeated for each of n components.

3.2.3.1 Interface description
3.2.3.2 Algorithmic model (e.g., PDL)
3.2.3.3 Restrictions/limitations
3.2.3.4 Local data structures
3.2.3.5 Performance issues3.2.3.6 Design constraints

3.3 Software Interface Description

The software's interface(s) to the outside world are described.

3.3.1 External machine interfaces
Interfaces to other machines (computers or devices) are described.
3.3.2 External system interfaces
Interfaces to other systems, products, or networks are described.
3.3.3 Human interface
An overview of any human interfaces to be designed for the software is presented. See Section 4.0 for additional detail.

4.0 User interface design

A description of the user interface design of the software is presented.

4.1 Description of the user interface

A detailed description of user interface including screen images or prototype is presented.

4.1.1 Screen images
Representations of the interface from the user's point of view.
4.1.2 Objects and actions
All screen objects and actions are identified.

4.2 Interface design rules

Conventions and standards used for designing/implementing the user interface are stated.

4.3 Components available

GUI components available for implementation are noted.

4.4 UIDS description

The user interface development system is described.

5.0 Restrictions, limitations, and constraints

Special design issues which impact the design or implementation of the software are noted here.

6.0 Testing Issues

Test strategy and preliminary test case specification are presented in this section.

6.1 Classes of tests

The types of tests to be conducted are specified, including as much detail as is possible at this stage. Emphasis here is on black-box and white-box testing.

6.2 Expected software response

The expected results from testing are specified.

6.3 Performance bounds

Special performance requirements are specified.

6.4 Identification of critical components

Downloaded from: sarojpandey.com.np

Those components that are critical and demand particular attention during testing are identified.

7.0 Appendices

Presents information that supplements the design specification.

7.1 Requirements traceability matrix

A matrix that traces stated components and data structures to software requirements is developed.

7.2 Packaging and installation issues

Special considerations for software packaging and installation are presented.

7.3 Design metrics to be used

A description of all design metrics to be used during the design activity is noted here.

7.4 Supplementary information (as required)

Downloaded from: sarojpandey.com.np

Chapter 19 - Software Measurement

Software measurement is a challenging but essential component of a healthy and highly capable software engineering culture. The four reasons for measuring software processes, products, and resources are:

- ✓ *To characterize* means to gain understanding of processes, products, and resources and environments, and to establish baselines for comparisons with future assessments.

- ✓ *To evaluate* means to determine the status with respect to plans. To assess achievement of quality goals and to assess the impacts of technology and process improvements on products and processes.

- ✓ *To predict* means to build a model of relationship between processes and products, to establish achievable goals for cost, schedule and quality- so that appropriate resources can be applied.

- ✓ *To improve* means to identify the inefficiencies, and other opportunities for improving product quality and process performance.

Measurement can be applied to the software process with the intent of improving it on a continuous basis. Measurement can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control. Finally, measurement can be used to help assess the quality of technical work products and to assist in tactical decision as a project proceeds.

- ✓ Direct measures of software engineering process include cost and effort.
- ✓ Direct measures of the product include lines of code (LOC), execution speed, memory size, and defects per reporting time period.
- ✓ Indirect measures examine the quality of the software product itself (e.g. functionality, complexity, efficiency, reliability, maintainability).

What to measure?

- *Product size:* count lines of code, function points, object classes, number of requirements, or GUI elements
- *Estimated and actual duration (calendar time) and effort (labor hours):* track for individual tasks, project milestones, and overall product development
- *Work effort distribution:* record the time spent in development activities (project management, requirements specification, design, coding, testing) and maintenance activities (adaptive, perfective, corrective)
- *Defects:* count the number found by testing and by customers and their type, severity, and status (open or closed)

 Downloaded from: sarojpandey.com.np

Measures, Metrics, and Indicators

- **Measure** - provides a quantitative indication of the extent, amount, dimension, capacity, or size of some product or process attribute of a product or process.
- **Measurement** -
    - Act of determining/obtaining a measure.
    - Software measurement is concerned with deriving a numeric value for an attribute of a software product or process
    - This allows for objective comparisons between techniques and processes
- **Metric** -
    - IEEE standard [IEE93] defines metric as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute to gain insight into the efficiency of the software process and the projects conducted using the process framework”.
    - Any type of measurement, which relates to a software system, process or related documentation. Ex- Lines of code in a program, the Fog index (a measure of the readability of a passage of written text), the number of reported faults in a delivered software product, the number of person-days required to develop a component.
    - Allow the software and the software process to be quantified.
    - May be used to *predict product attributes* or to *control the software process.*
- **Indicator**-
    - A metric or combination of metrics that provide insight into the software process, a software project, or the product itself to adjust the process, project for better decision making.

Process and Project Indicators

- Metrics should be collected so that process and product indicators can be ascertained
- *Process indicators* enable software project managers to: assess project status, track potential risks, detect problem area early, adjust workflow or tasks, and evaluate team ability to control product quality.
- *Project indicator* enables a software project manager to (1) assess the status of an ongoing project. (2) track potential risks (3) uncover problem areas before they go "critical" (4) adjust work flows or tasks (5) evaluate the project team's ability to control quality of software work products.

Process Metrics

- Process is only one of a number of “controllable factors in improving software quality and organizational performance".
- *Private process metrics* (e.g. defect rates by individual or module and errors found during development) are known only to the individual or team concerned.
- *Public process metrics* enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.

 Downloaded from: sarojpandey.com.np

- Statistical software process improvement (SSPI) helps and organization to discover where they are strong and where they are week. It uses software failure analysis to collect information about all errors and defects encountered as an application, system, or product is developed and used.

Project Metrics

- Software project metrics are used by the software team to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables).

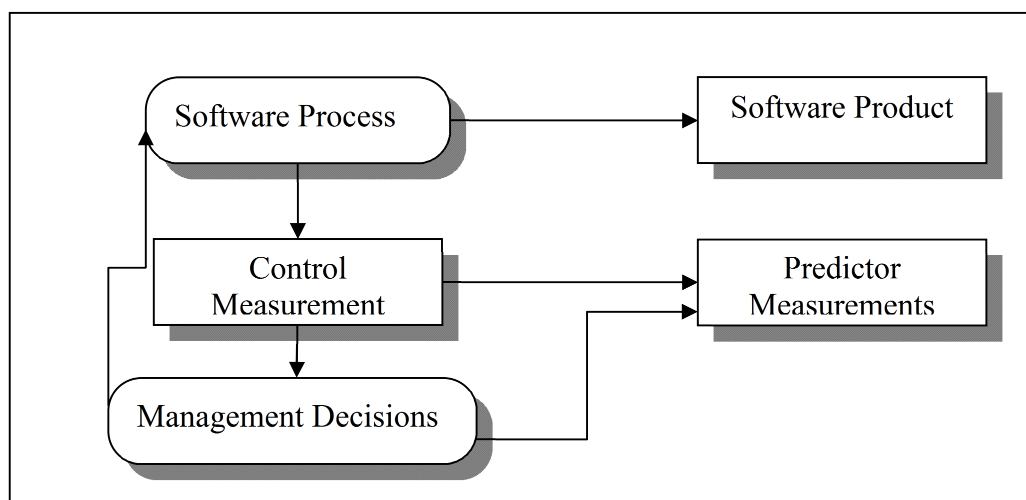


Fig: Predictor and control metrics

Product Metrics

- A quality metric should be a predictor of product quality
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution
  - Dynamic metrics are closely related to software quality attributes
    - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute)
  - Static metrics which are collected by measurements made of the system representations
  - Static metrics have an indirect relationship with quality attributes
    - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability
  - Dynamic metrics help assess efficiency and reliability; static metrics help assess complexity, understandability and maintainability

 Downloaded from: sarojpandey.com.np

Table: Software product metrics

Software Metric	**Description**
Fan-in/Fan-out	Fan in is a measure pf the number of functions that call some other function (say x). Fan-out is the number of functions which are called by function x. A high-value for fan-in means that x is tightly coupled to the rest of the design and changes to x will have extensive knock-on effects. A high-value for fan-out suggests that the overall complexity of x may be be high because of the complexity of the control logic needed to coordinate the called components.
Length of code (LOC)	This the measure of the size of a program. Generally, the larger the size of the code of a program component, the more complex and error-prone that component is likely to be.
Cyclomatic complexity	This is the measure of the control complexity of a program. This control complexity may be related to program understandability.
Length of identifiers	This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and are more potential error-prone.
Fog index	This is a measure of the average length of words and sentence in documents. The higher the value for fog index, the more difficult the document may be to understand.

 Downloaded from: sarojpandey.com.np

Table: Object-oriented metrics

Object-oriented metric	Description
Depth of inheritance tree	This represents the number of discrete levels in the inheritance tree where sub-classes inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design as, potentially, many different object classes have to be understood to understand the object classes at the leaves of the tree.
Method fan-in/fan-out	This is directly related to fan-in and fan-out as described above and means essentially the same thing. However, it may be appropriate to make a distinction between calls from other methods within the object and calls from external methods.
Weighted methods per class	This is the number of methods included in a class weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1 and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand. They may not be logically cohesive so cannot be reused effectively as super-classes in an inheritance tree.
Number of overriding operations	These are the number of operations in a super-class which are over-ridden in a sub-class. A high value for this metric indicates that the super-class used may not be an appropriate parent for the sub-class.

Size-Oriented Metrics

- Derived by normalizing (dividing) any direct measure (e.g. defects or human effort) associated with the product or project by LOC.
- Size oriented metrics are widely used but their validity and applicability is widely debated.

Example

Project	LOC	Effort	$ (000)	Pp.doc	Errors	Defects	People
Alpha	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1224	321	86	5

The above table shows a size-oriented measure, which list two projects, alpha and beta. Project alpha has developed with 12,100 LOC with 24 person-months of effort at a cost of $168,000. An effort and cost recorded in the table represent all software engineering activities (analysis, design, and test), not just coding. Further information for project alpha indicates that 365 pages of documents were developed, 134 errors were recorded before the software was released, and 29 defects were encountered after the release to the customer with the first year of operation. Three people worked on the development of software for project alpha.

- Size-oriented metrics include:
  - Errors per KLOC

Downloaded from: sarojpandey.com.np

- Defects per KLOC
- $ per LOC
- Page of documentation per KLOC
- Additional metrics includes
  - Errors per person-month
  - LOC per person-month
  - $ per page of documentation

Advantages of LOC

- Simple to use metric as it measures the number of source instructions required to solve a problem.
- LOC is an "artifact" of all software development projects that can be easily counted, and many estimation models use LOC or KLOC as key input.

Disadvantages of LOC

- LOC measures are programming language dependent.
- Nonprocedural language is not considered for measures.
- Detailed information which is required for estimation is hard to achieve (i.e., the planner must estimate the LOC to be produced long before analysis and design have been completed.)
- Ignores the relative complexity of design, testing
- LOC measure correlates poorly with the quality and efficiency of the code.
- LOC metric penalizes the use of high-level programming languages, code, and reuse.
- It is very difficult to arrive at an accurate LOC estimation from a problem specification. LOC metric can be computed only after the code has been fully developed, and is of little use at the beginning of a project.

Function-Oriented Metrics

- Function points are computed from direct measures of the information domain of a business software application and assessment of its complexity as the size of a software product is directly dependent on the number and type of different functions it performs.
- Once computed function points are used like LOC to normalize measures for software productivity, quality, and other attributes.
- Feature points and 3D function points provide a means of extending the function point concept to allow its use with real-time and other engineering applications.
- The relationship of LOC and function points depends on the language used to implement the software.
- Advantages:
  - can be used to estimate the size of software directly from the problem specification.
- It computes the size of a software product (in units of function points or FPs) using five different characteristics of the product.

 Downloaded from: sarojpandey.com.np

Measurement parameter	Count	Weighing Factor			Count
		Simple	Average	Complex	
No of user inputs		3	4	6	
No of user outputs		4	5	7	
No of user inquires		3	4	6	
No of files		7	10	15	
Number of external interfaces		5	7	10	
				Total Count	

Taxonomy

- *Number of inputs*: Each user data input is counted. Data inputs should be distinguished from user inquiries which are counted separately.
- *Number of outputs*: Output refers to reports, screens, and error messages. Individual data items within a report are not counted separately.
- *Number of user inquires*: It is the number of distinct interactive queries made by the user which requires specific action by the system.
- Number of files: Each logical file, e.g, groups of logically related data, is counted as a file. Thus, the files can be either data structures or physical files.
- *Number of external interfaces*: Data files on tape, disk etc and other interfaces that are used to transmit information to other systems are counted.

1. Backup and recovery
2. Data communication
3. Distributed processing
4. Critical performance
5. Heavily used configuration
6. On-line data entry
7. Transaction complexity
8. Online master file update
9. Complex external processing
10. Complex internal processing
11. Reusability
12. Installation factor
13. Multiple sites
14. Change facilitation

To each of these factors, a complexity adjustment value of 0-5 are assigned. A value of 0 means the factor has no effect on the product, and 5 means that it is essential for software design.

$$AFP = TFP \times [0.65+0.01 \times \sum_{i=1}^{i=14}(F_i)$$

Software Quality Metrics

- Factors assessing software quality come from three distinct points of view (product operation, product revision, product modification).
- Software quality factors requiring measures include correctness (defects per KLOC), maintainability (mean time to change), integrity (threat and security), and usability (easy to learn, easy to use, productivity increase, user attitude).

 Downloaded from: sarojpandey.com.np

- Defect removal efficiency (DRE) is a measure of the filtering ability of the quality assurance and control activities as they are applied throughout the process framework.

Relationship between internal and external attributes

It is often impossible to measure software quality attributes directly. Attributes such as maintainability, complexity, and understanding are affected by many different factors and there are no straightforward metrics for them. Rather, we have to measure some internal attributes of the software

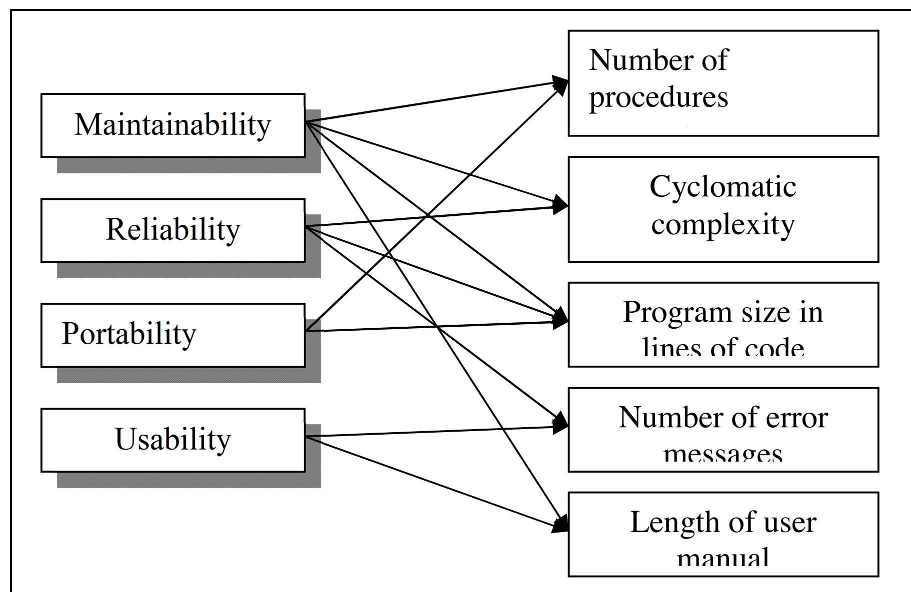


Fig: Relationship between internal and external attributes

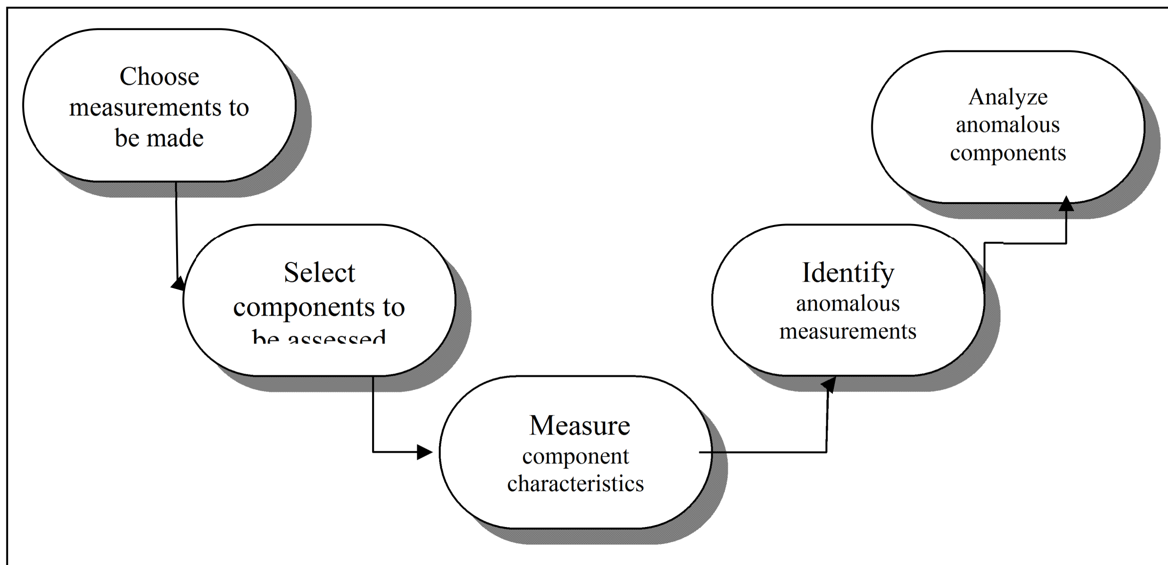
The measurement Process

- A software measurement process may be part of a quality control process
- Data collected during this process should be maintained as an organisational resource
- Once a measurement database has been established, comparisons across projects become possible.
- The key stages in this process include:
    - Choose measurements to be made
        - formulation of questions and defining measurements as per GQM (Goal-Question-Metric) paradigm for data collection.
    - Select components to be assessed
        - critical core components which are in constant use are chosen.
    - Measure component characteristics
        - selected components are measured and metric values computed by processing the component representation (design, code) using an automated data collection tool such as CASE.
    - Identify anomalous measurements

 Downloaded from: sarojpandey.com.np

- Measured components are compared to each other and to previous measurements from database to check for unusually high or low values (anomalous values) for each metrics.
- Analyze anomalous components
  - Analyzing the anomalous value components for quality problems.

Fig: Process of product measurement



Integrating Metrics with Software Process

- Many software developers do not collect measures.
- Without measurement it is impossible to determine whether a process is improving or not.
- Baseline metrics data should be collected from a large, representative sampling of past software projects.
- Getting this historic project data is very difficult, if the previous developers did not collect data in an on-going manner.

Statistical Process Control (SPC)

- It is important to determine whether the metrics collected are statistically valid and not the result of noise in the data.
- Control charts provide a means for determining whether changes in the metrics data are meaningful or not.

 Downloaded from: sarojpandey.com.np

- Zone rules identify conditions that indicate out of control processes (expressed as distance from mean in standard deviation units).

Establishing a Software Metrics Program

- Identify business goal
- Identify what you want to know
- Identify subgoals
- Identify subgoal entities and attributes
- Formalize measurement goals
- Identify quantifiable questions and indicators related to subgoals
- Identify data elements needed to be collected to construct the indicators
- Define measures to be used and create operational definitions for them
- Identify actions needed to implement the measures
- Prepare a plan to implement the measures

Software Cost Estimation

In the project planning, the estimation of time effort and time with the identified project activities need to be done. Project managers normally ask the following questions in order to estimate all the software project activities.

- ✓ How much effort is required to complete an activity?
- ✓ How much calendar time is needed to complete an activity?
- ✓ What is the total cost of an activity?

Project estimation and scheduling are interleaved management activities. However, some cost

estimation may be required to establish a budget for the project or to set a price for the software.

Software cost components

- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - Social and insurance costs
- Effort costs must take overheads into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g library, staff restaurant, etc.)

 Downloaded from: sarojpandey.com.np

Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system
- There is not a simple relationship between the development cost and the price charged to the customer
- Broader organisational, economic, political and business considerations influence the price charged

Table: Factor affecting software pricing

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system
  - Initial estimates are based on inadequate information in a user requirements definition
  - The software may run on unfamiliar computers or use new technology
  - The people in the project may be unknown
- Project cost estimates may be self-fulfilling
  - The estimate defines the budget and the product is adjusted to meet the budget

Types of Estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy

 Downloaded from: sarojpandey.com.np

- Parkinson's Law
- Pricing to win

Summary

- Software measurement can be used to gathers quantitative data about both the software process and the software product. The values of the software metrics which are collected may be used to make inferences about product and process quality.
- Product quality metrics should be used to identify potentially problematical components

Downloaded from: sarojpandey.com.np