Unit-1

Fundamentals of SPM

Essential Elements of Software Project Management:

 With the significant expense and business impact of most software projects, the stakes are high for them to be done right. The more you understand the essential elements of software project management, the more likely your project will be successful.

Choose the Project Methodology

 a key decision is the methodology that will be used for your project. Although this will be driven mainly by your organization's accepted standards, understanding the strengths and weaknesses of different approaches will help you plan your project.

A traditional waterfall methodology is highly structured to deliver good results, but can result in longer projects based on its sequential set of tasks. Agile methodologies deliver quicker results, but require managing cross-functional teams and allowing them more freedom to create prototypes in several iterations as requirements are refined.

Identify Requirements

 a solid understanding of user requirements forms the foundation for your software, yet there's often a rush to skip this and move to the coding phase. This may cause you to miss

necessary requirements or try to meet an ever-changing target as new requirements are uncovered.

Understand the Technology

 It’s crucial that the project manager understands the maturity level of the technology used for the project, since technology changes at a rapid pace. If it's a well-understood technology, the chance of meeting the project schedule is high. However, if your project depends on a less proven technology, it's critical that the project manager and business stakeholders understand and accept the risk that more problems may occur or the team will need more time to get up to speed on the technology.

Communicate with Business Stakeholders

 Technical jargon is a foreign language to most business stakeholders, so communication between the project manager, the technical team and other stakeholders can be a challenge. Project managers and technical team members need to talk about the requirements and project risks using business terminology. If users can't understand the explanation, they can't make informed decisions about the level of risk they're willing to accept.

Deliver Phased Results

 Many software projects are large, expensive and lengthy efforts. Often the new software isn't delivered until months or years after the requirements were originally documented. However, the business may have changed by then--so even software that exactly matches the requirements is no longer what

the business or the users need. The team can deliver faster results that provide more value to the business by structuring large software projects into several shorter efforts.

Understand the Culture

 An organization's culture significantly impacts project planning. If the organization sees technology as peripheral to its core mission, the project may not get the management support needed for success. If the culture emphasizes strict adherence to meeting budget and schedule rather than emphasizing the business' core values such as safety or customer service excellence, decision makers may choose short cuts that negatively impact the quality of the software delivered.

Gantt chart: A **Gantt chart** is a type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.

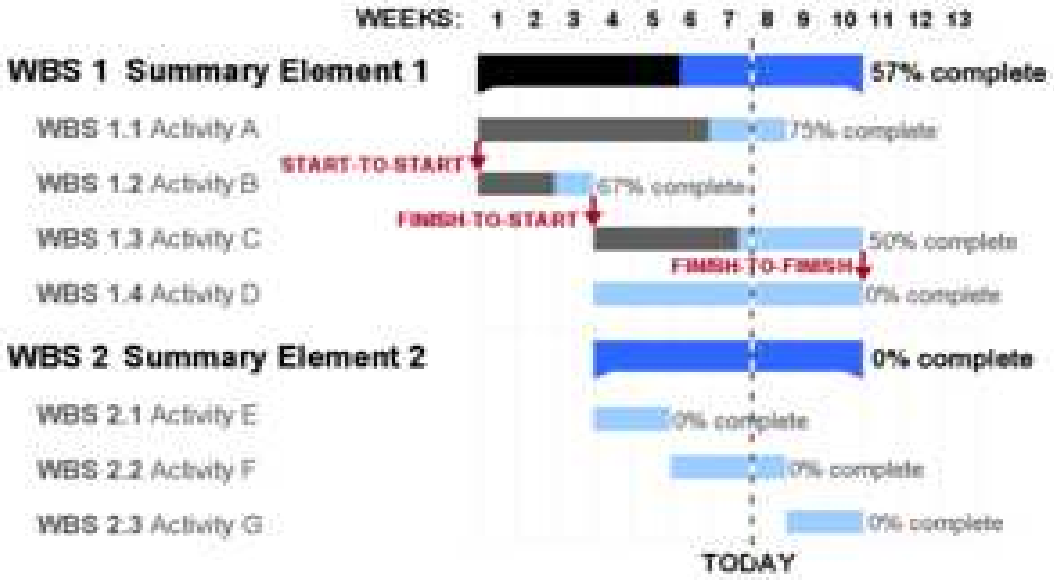
Although now regarded as a common charting technique, Gantt charts were considered revolutionary when they were introduced. In recognition of Henry Gantt's contributions, the Henry Laurence Gantt Medal is awarded for distinguished achievement in management and in community service. This

chart is used also in Information Technology to represent data that have been collected

Historical development

The first known tool of this type was reportedly developed in 1896 by Karol Adamiecki, who called it a *harmonogram*. Adamiecki did not publish his chart until 1931, however, and then only in Polish. The chart is commonly known after Henry Gantt (1861–1919), who designed his chart around the years 1910–1915.

In the 1980s, personal computers allowed for widespread creation of complex and elaborate Gantt charts. The first desktop applications were intended mainly for project managers and project schedulers. With the advent of the internet and increased collaboration over networks at the end of the 1990s, Gantt charts became a common feature of web-based applications, including collaborative groupware.



A Gantt chart showing three kinds of schedule dependencies (in red) and percent complete indications

Advantages and limitations

Gantt charts have become a common technique for representing the phases and activities of a project work breakdown structure (WBS), so they can be understood by a wide audience all over the world.

A common error made by those who equate Gantt chart design with project design is that they attempt to define the project work breakdown structure at the same time that they define schedule activities. This practice makes it very difficult to follow the 100% Rule. Instead the WBS should be fully defined to follow the 100% Rule, then the project schedule can be designed.

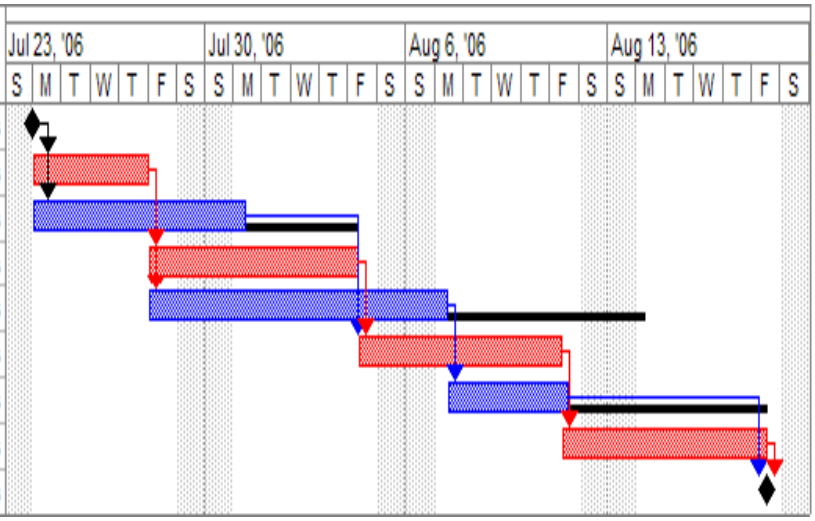
Although a Gantt chart is useful and valuable for small projects that fit on a single sheet or screen, they can become quite unwieldy for projects with more than about 30 activities. Larger Gantt charts may not be suitable for most computer displays. A related criticism is that Gantt charts communicate relatively little information per unit area of display. That is, projects are often considerably more complex than can be communicated effectively with a Gantt chart.

Gantt charts only represent part of the triple constraints (cost, time and scope) of projects, because they focus primarily on schedule management. Moreover, Gantt charts do not represent the size of a project or the relative size of work elements, therefore the magnitude of a behind-schedule condition is easily mis communicated. If two projects are the same number of days behind schedule, the larger project has a larger impact on resource utilization, yet the Gantt does not represent this difference.

Example

In the following example there are seven tasks, labeled *A* through *G*. Some tasks can be done concurrently (*A* and *B*) while others cannot be done until their predecessor task is complete (*C* cannot begin until *A* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*O*), the most likely or normal time estimate (*M*), and the pessimistic time estimate (*P*). The expected time (T_E) is computed using the formula $(O + 4M + P) \div 6$.

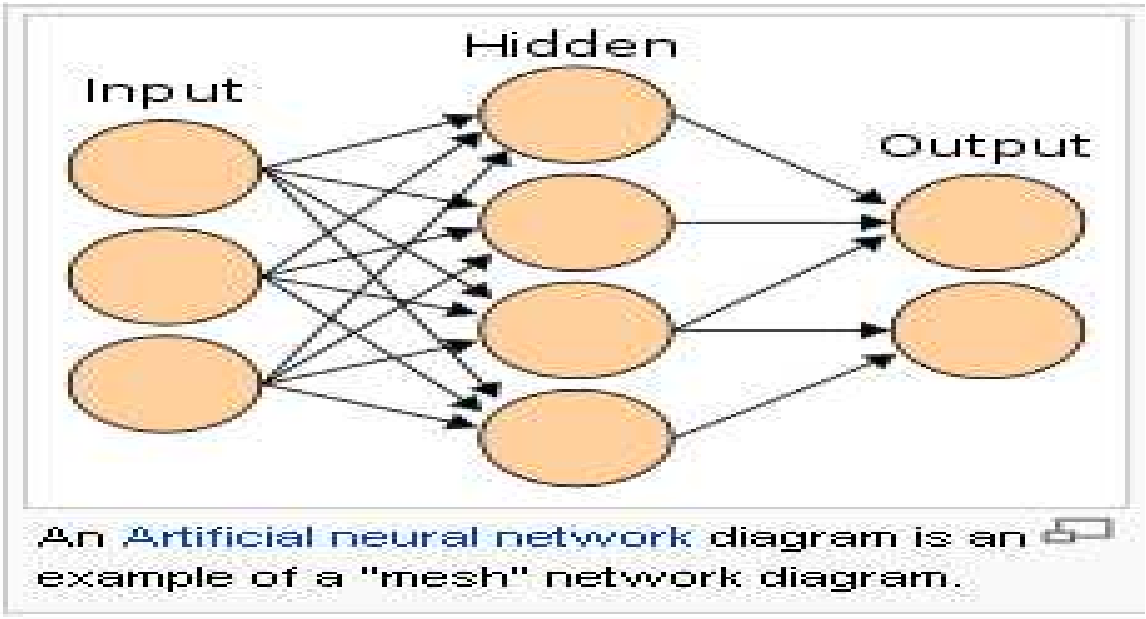
ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days



Network diagram: A network diagram is a type of network. A network in general is an interconnected group or system, or a fabric or structure of fibrous elements attached to each other at regular intervals, or formally: a graph.

A network diagram is a special kind of cluster diagram, which even more general represents any cluster or small group or bunch of something, structured or not. Both the flow diagram and the tree diagram can be seen as a specific type of network

diagram.



Types of network diagrams

There are different type's network diagrams:

- Artificial neural network or "neural network" (NN), is a mathematical model or computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation.
- Computer network diagram is a schematic depicting the nodes and connections amongst nodes in a computer network or, more generally, any telecommunications network.

- In project management according to Baker et al. (2003), a "network diagram is the logical representation of activities, that defines the sequence or the work of a project. It shows the path of a project, lists starting and completion dates, and names the responsibilities for each task. At a glance it explains how the work of the project goes together... A network for a simple project might consist one or two pages, and on a larger project several network diagrams may exist". Specific diagrams here are
  - Project network: a general flow chart depicting the sequence in which a project's terminal elements are to be completed by showing terminal elements and their dependencies.
  - PERT network

- Neural network diagram: is a network or circuit of biological neurons or artificial neural networks, which are composed of artificial neurons or nodes.

- A semantic network is a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes.

- A sociogram is a graphic representation of social links that a person has. It is a sociometric chart that plots the structure of interpersonal relations in a group situation

What's a project?, A project, by definition, is a temporary activity with a starting date, specific goals and conditions, defined responsibilities, a budget, a planning, a fixed end date and multiple parties involved. You know what you have to do,

do it, once, and that's the end of it. That's a project. However, being hip has it's disadvantages. Our local housing office has a permanent project manager for financial controlling. This is a on going activity for the office, and the project manager does it as his full time, never ending job. What ever he's managing, it surely isn’t a project.

There is no real harm in naming a tiger an elephant or vice versa. No one gets hurt by naming his job a project, even if it doesn't fit the definition by a mile. But don't be surprised if techniques for projects don't work on 'projects-just-by-name'. Don't be amazed when applying the techniques may seem like killing a bumble-bee with an machine gun. Don't get mad when other people don't know why you're doing an easy job the hard way, by making a project out if it.

Start a project, but only for the right reasons. Not for the heck if it. Not because of the cool acronym. Or the status.

For a project definition, look for the following aspects:

- Starting date
- Specific goals and conditions
- Defined responsibilities
- A budget
- A planning
- A fixed end date
- Parties involved

And then, if it looks like a duck, walks like a duck and quacks like a duck, it probably is a duck. However, if it doesn't come close, raising the question of treating it like a project is the right thing to do. There may be an easier and much simpler solution.

Often (…) top management has no project; it merely has a problem and does not know how to, or does not want to, find a solution. So, it expects the players to find the way. Much social and industrial unrest can be caused by top management not having real projects, but only having the desire to get rid of a problem. (Managing Sensitive Projects)

If you don't find all the aspects named above at this moment in time, it is possible. Wait, and make the final judgment at the end of the intake. Working on one of the subjects, might cause one of the aspects to popup.

Project Manager, You talked with all those people. You asked "what do you want?" "What can I do?" Now it's time for your gut feeling.

Do you trust the statements made by the stakeholders? Do you think top management will stick with you? Do you honestly believe the project has a chance to succeed?

Now is the time to know. It is late, but you can still get off the train. After this, it's your ass that's on the line. When this project is mentioned, it's your face the people see.

Unit-2

Project integration management

Project portfolio management: **Project Portfolio Management** (**PPM**) is a term used by project managers and project management (PM) organizations, (or PMO's), to describe methods for analyzing and collectively managing a group of current or proposed projects based on numerous key characteristics. The fundamental objective of PPM is to determine the optimal mix and sequencing of proposed projects to best achieve the organization's overall goals - typically expressed in terms of hard economic measures, business strategy goals, or technical strategy goals - while honoring constraints imposed by management or external real-world factors. Typical attributes of projects being analyzed in a PPM process include each project's total expected cost, consumption of scarce resources (human or otherwise) expected timeline and schedule of investment, expected nature, magnitude and timing of benefits to be realized, and relationship or inter-dependencies with other projects in the portfolio.

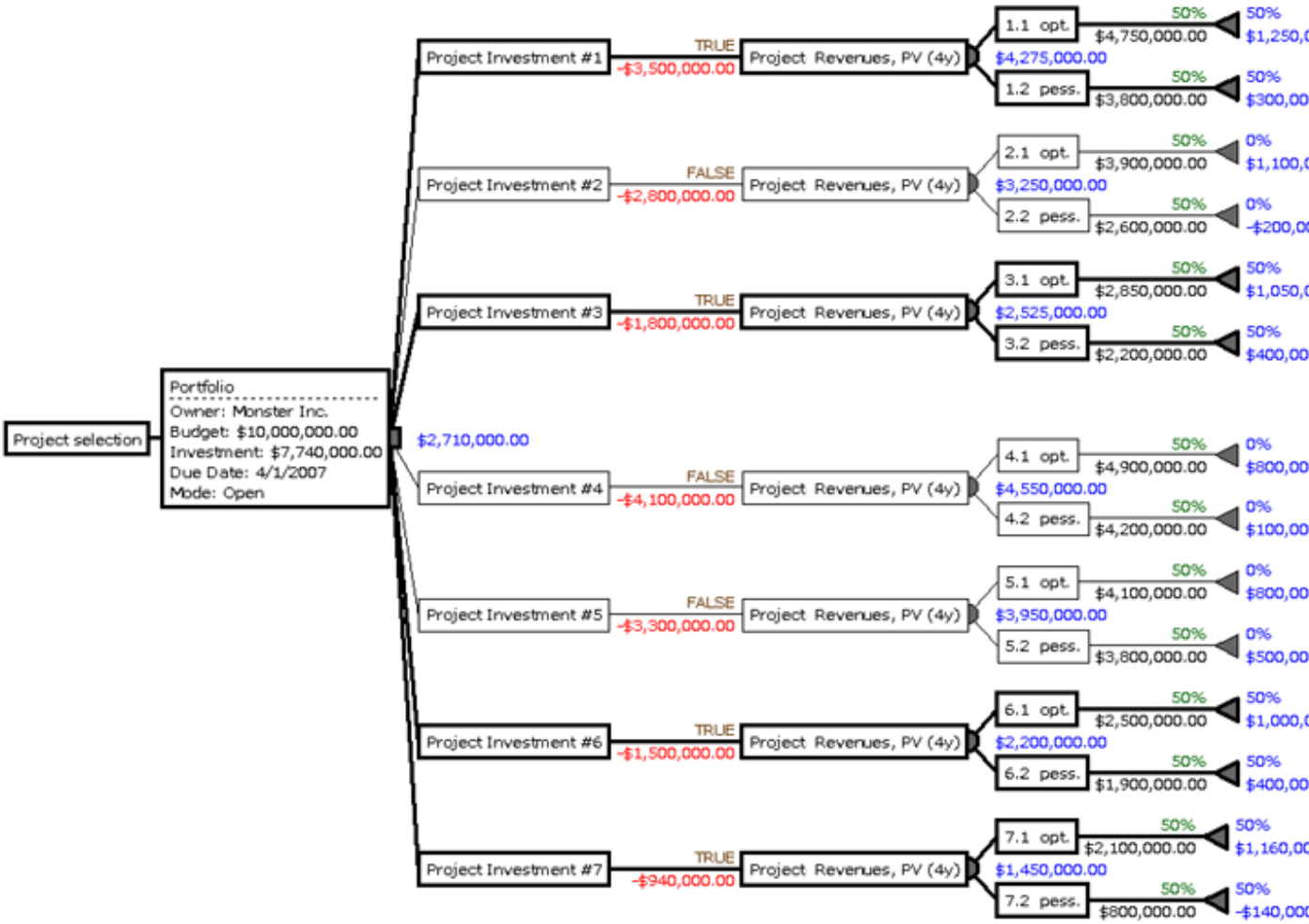
The key challenge to implementing an effective PPM process is typically securing the mandate to do so. Many organizations are culturally inured to an informal method of making project investment decisions, which can be compared to political processes observable in the U.S. legislature. However this approach to making project investment decisions has led many organizations to unsatisfactory results, and created demand for a more methodical and transparent decision making process. That

demand has in turn created a commercial marketplace for tools and systems which facilitate such a process.

Some commercial vendors of PPM software emphasize their products' ability to treat projects as part of an overall investment portfolio. PPM advocates see it as a shift away from one-off, ad hoc approaches to project investment decision making. Most PPM tools and methods attempt to establish a set of values, techniques and technologies that enable visibility, standardization, measurement and process improvement. PPM tools attempt to enable organizations to manage the continuous flow of projects from concept to completion.

Treating a set of projects as a portfolio would be, in most cases, an improvement on the ad hoc, one-off analysis of individual project proposals. The relationship between PPM techniques and existing investment analysis methods is a matter of debate. While many are represented as "rigorous" and "quantitative", few PPM tools attempt to incorporate established financial portfolio optimization methods like modern portfolio theory or Applied Information Economics, which have been applied to project portfolios, including even non-financial issues

Optimizing for payoff: One method PPM tools or consultants might use is the use of decision trees with decision nodes that allow for multiple options and optimize against a constraint. The organization in the following example has options for 7 projects but the portfolio budget is limited to $10,000,000. The selection made are the projects 1, 3, 6 and 7 with a total investment of $7,740,000 - the optimum under these conditions. The portfolio's payoff is $2,710,000.



Presumably, all other combinations of projects would either exceed the budget or yield a lower payoff. However, this is an extremely simplified representation of risk and is unlikely to be realistic. Risk is usually a major differentiator among projects but it is difficult to quantify risk in a statistically and actuarially meaningful manner (with probability theory, Monte Carlo Method, statistical analysis, etc.). This places limits on the deterministic nature of the results of a tool such as a decision tree (as predicted by modern portfolio theory).

An approach to include uncertainty and risk in portfolio optimization takes a decision centric view of the project portfolio optimization process [5] . In this view there are five key decisions that must be made:

- Decision D1: Decide what strategic initiatives, benefits, and resource limitation criteria (i.e. measures) to use for project filtering and portfolio ranking.
- Decision D2: Decide which criteria are most important to achieve.
- Decision D3: Decide which project ideas or needs are worth developing into business cases?
- Decision D4: Decide which Business Cases should be considered as part of the portfolio
- Decision D5: Decide which projects to fund

For each decision is imperative to fuse both qualitative and quantitative evaluations of alternatives where each measure includes uncertainty. It is the uncertainty in the strategic alignment, value benefits and resources demand that cause risk and thus drives all five decisions.

Statement of work. A **statement of work** (SOW) is a formal document that captures and defines the work activities, deliverables and timeline a vendor will execute against in performance of specified work for a client. Detailed requirements and pricing are usually included in the Statement of Work, along with standard regulatory and governance terms and conditions

Overview

There are many formats and styles of Statement of Work document templates that have been specialized for the Hardware or Software solutions being described in the Request for Proposal. Many companies create their own customized version of SOWs for use within their industry or vertical that have been either specialized or generalized to accommodate the typical request and proposals they receive.

It is important to note that in most cases the Statement of Work being agreed upon is a binding contract. Master Service Agreements or Consultant/Training Service agreements postpone certain work specific contractual components that are addressed in individual Statement(s) of Work.

Areas addressed

Areas that are typically addressed by a SOW are as follows:

Purpose, why are we doing this project? This is the question that the purpose statement attempts to answer.

Scope of Work, This describes roughly the work to be done in detail and specifies the hardware and software involved and the exact nature of the work to be done.

Location of Work, This describes where the work is to be performed. This also specifies the location of hardware and software and where people will meet to perform the work.

Period of Performance, This specifies the allowable time for projects, such as start and finish time, number of hours that can

be billed per week or month, where work is to be performed and anything else that relates to scheduling.

Deliverables Schedule, This part lists the specific deliverables, describing what is due and when.

Applicable Standards, This describes any industry specific standards that need to be adhered to in fulfilling the contract.

Acceptance Criteria, This specifies how the buyer or receiver of goods will determine if the product or service is acceptable, what objective criteria will be used to state the work is acceptable.

Special Requirements, This specifies any special hardware or software, specialized workforce requirements, such as degrees or certifications for personnel, travel requirements, and anything else not covered in the contract specifics.

Type of Contract/Payment Schedule, The project acceptance will depend on if the budget available will be enough to cover the work required. Therefore payments breakdown whether up front or phased will be negotiated very early at this stage.

Miscellaneous, there are many items that do not form part of the main negotiations but are nonetheless very important to the project. They seem minor but being overlooked or forgotten could pose problems for the project.

Project charter: In project management, a **project charter** or **project definition** (sometimes called the terms of reference) is a statement of the scope, objectives and participants in a project. It provides a preliminary delineation of roles and responsibilities,

outlines the project objectives, identifies the main stakeholders, and defines the authority of the project manager. It serves as a reference of authority for the future of the project.

The project charter is usually a short document that refers to more detailed documents such as a new offering request or a request for proposal. In Initiative for Policy Dialogue (IPD), this document is known as the project charter. In customer relationship management (CRM), it is known as the project definition report. Both IPD and CRM require this document as part of the project management process.

The project charter establishes the authority assigned to the project manager, especially in a matrix management environment. It is considered industry best practice.

The purpose of the project charter is to document:

- Reasons for undertaking the project
- Objectives and constraints of the project
- Directions concerning the solution
- Identities of the main stakeholders

The three main uses of the project charter:

- To authorize the project - using a comparable format, projects can be ranked and authorized by Return on Investment.
- Serves as the primary sales document for the project - ranking stakeholders have a 1-2 page summary to distribute, present, and keep handy for fending off other project or operations runs at project resources.

- As a focus point throughout the project - for example: project as people walk in to team meetings and use in change control meetings to ensure tight scope management.

Project Organization A project organization is a temporary thing. It will only exist from the projects start until its end. All the project team members are coming from different organizations of part of the organization. They will all have a temporary assignment to the project. So, they have not only a project boss (the project manager, that might be you), but also their 'normal' boss, who orders him around when the employee is not in the project. These 'normal bosses' are an important group of stakeholders.

The project organization should be a result from the project strategy; it should be constructed in such a way that the strategy can be implemented within the environment of the project ("look what the dog brought in, a presumptuous sentence"). A very obvious example: if the strategy contains an aspect of having independent reviews, the organization should support its independence, by creating a separate working group with no ties to the other team members, e.g. But, I'm a little too far now mentioning working groups and the like.

The project team that does the work should be as small as possible. Small is beautiful, and effective. Don't start inviting everyone to the organization. Only people who have an added value and will spend a significant amount of time to the project can be in the core organization. Try to avoid going overboard on working groups. Working groups can drown a project in communication overhead. If there should be that much

discussion anyway, postpone the project and first make up the minds.

Next to the people who do the work, are the people that have some influence on it, but do nothing; a large part of the stakeholders. The project organization can be used to satisfy some wishes of stakeholders to create the much needed win-win situations. In its most simple form, you can create a project trashcan ("The Project Tactical Non-Binding Advisory Committee") where you put in the people who just want to be involved in the project (to save their territory), but which you have no use for.

Be creative while constructing the project organization. Take the Bob Ross way to paint your organization: "This is a sweet little project staff. I put it here next to the tracking and control group, so it has a friend."

Project Management Institute: The **Project Management Institute** (PMI) is a non-profit professional organization for the project management profession with the purpose of advancing project management

Overview

The Project Management Institute (PMI) offers a range of services to the Project Management profession such as the development of standards, research, education, publication, networking-opportunities in local chapters, hosting conferences and training seminars, and maintaining multiple credentials in project management. These credentials are:[3]

- Certified Associate in Project Management (CAPM)
- Project Management Professional (PMP)
- PMI Scheduling Professional (PMI-SP)
- PMI Risk Management Professional (PMI-RMP)
- Program Management Professional (PgMP)

In addition to career development credentials, PMI offers one certification:

- Organizational Project Management Maturity Model Certified Consultant (OPM3-CC)

PMI has recruited volunteers to create industry standards, such as "A Guide to the Project Management Body of Knowledge", which has been recognized by the American National Standards Institute (ANSI).

Project phases: Comprises of

Phase1:- Project initiation

Phase2:- Project planning

Phase3:- Project Execution

Phase4:- Project closure

Phase1: Project initiation

(a)Develop a business case

- Identify the objectives/alternate solutions

- Researches

(b) **Feasibility study**. Feasibility is defined as the practical extent to which a project can be performed successfully

Types-Technical Feasibility, operational Feasibility, economical feasibility, managerial Feasibility

© **Establish the project charter.** The project charter helps us to

- Identify the project vision and objectives.
- Define complete scope of the project.
- List all the critical project deliverables.
- List any risk, issues and assumption.

(d) **Appointing the project team**. It defines how the targets are to be measured and how the performance of the role will be assessed.

(e)**Performing the phase review**. Phase reviews are conducted at the end of initiation. Planning and execution phases with in a project. This form helps us to complete a phase review for the project initiation phase with in the project.

Phase2:- Project planning. The main tasks that are involved

(a) **Project plan.** A project plan is created to identify all of the phase's activities and tasks, a project plan helps a project manager to understand, monitor and control the development of software.

(b) **Create a resource plan.** There are three types of resources involve during any project work:-People, equipment and

materials, the resource plan helps to identify this 3 resources that are needed to deliver the product.

© **Create a financial plan.** Financial plans enable us to set a budget to deliver a project with in a budget we need to produce the project deliverables at a total cost which is minimal and optimal.

(d)**Creating a quality plan.** Involves setting up of the quality targets for the project to ensure that the deliverables produced meet the needs of the customer.

Phase3:- Project Execution. The project is executed. Project execution involves

(a) Building deliverables

(b) Performing time management cost management, quality or risk management.

Phase4:- Project closure.

(a) **Performing project closure.** All of the task to complete project are documented and deliverables are handled over to the customer

(b) **Reviewing project completion**. By reviewing, the project successes, deliverables, achievements and lessons are learned; this is the last critical step.

Project dimensions:

1) **People:** Every software project is populated by five types of people

a) Senior manager:- who defines the business issues
b) Project manager:-who must plan motivate
c) Practioners:- who delivers the technical skills that are necessary
d) Customers:-who specify the requirements of the software.
e) End users.

2) **Product:** it involves

a)Software scope- it must be unambiguous and understandable at the management and technical level.
b) Problem decomposition: - It is also called as portioning or problem elaboration, decomposition is applied in two major areas.

1. The functionality that must be delivered.

2. The process that will be used to deliver it.

3) **Process:** Since we have many process models like waterfall prototype model etc, the project manager must decide which process model to choose for the customers who have requested the product who will do the work.

4) **Project**: In order to manage successful software project there are 10 signs that indicate that a project is in problem

a) Software people do not understand the customer needs.
b) The project scope is poorly defined
c) Changes are management poorly.
d) Chosen technology changes
e) Business needs changes
f) Unrealistic deadlines

g) Resistant users
h) Sponsorship is lost
i) Project team with un appropriate skills.
j) Poor management

Procurement management:

- It is also known as purchasing or outsourcing.
- Procurement involves acquiring goods or services from a source outside the organization.
- Typically organizations outsource products or services for the following reason- reduce cost, focus on core business, increased flexibility, and increased accountability.

Need for procurement: a) procurement means acquiring goods or services from an outside source.

b) An organization outsources product and services because- it reduces its fixed and recurring cost enables it to focus on its own core competencies.

c) Typical outsourcing in IT function- operations, IT projects, hardware, software, manpower services

Procurement Management Process. This Procurement Management process will help you to purchase goods and services from external suppliers. It gives you a complete

procurement process and *procurement procedures*, which explain step-by-step, how to purchase from suppliers**.**

This procurement process will also help you to:

- Identify the goods and services to procure
- Complete Purchase Orders and issue to suppliers
- Agree on delivery timeframes and methods
- Receive goods and services from suppliers
- Review and accept the items procured
- Approve supplier payments

This Procurement Management Process will enable you to:

- Identify supplier contract milestones
- Review supplier performance against contract
- Identify and resolve supplier performance issues
- Communicate the status to management

Procuring goods and services from external suppliers can be a critical path for many projects. Often, the performance of the supplier will reflect on the performance of the overall project team. It's therefore crucial that you manage your supplier's performance carefully, to ensure that they produce deliverables which meet your expectations.

What is a Procurement Management Process?

A Procurement Management Process, or *Procurement Process*, is a method by which items are purchased from external suppliers. The procurement management process involves managing the ordering, receipt, review and approval of items

from suppliers. A procurement process also specifies how the supplier relationships will be managed, to ensure a high level of service is received. This is a critical task in Procurement Management.

When do I use a Procurement Management Process?

You need to implement a Procurement Process any time you want to buy items from external suppliers. By using this Procurement Management Process, you can ensure that the items provided meet your need. It also helps you manage the supplier relationship, ensuring that any issues are resolved quickly. By implementing a Procurement Process, you can ensure you get the maximum value from your supplier relationship.

Project Scope:

- Features to be delivered to the end users.
- What type of data will be?
- Project scope is achieved by (a) customer needs, (b) project boundaries

A narrative description of software scope based on communication with stake holders.

Main process: 1) project initiation

2) Scope definition

3) Scope change control

4) Scope planning

5) Scope visioning

Project cost: project manager can be estimated by budgeting, people, money, equipments, and materials

Resource planning

Cost estimation
- Assumption
- Scope
- Budget breakdown
- Cost/benefit analysis
- Cash flow analysis

Project time: 1) main process

- Activity definition
- Activity duration estimation
- Schedule control
- Activity sequencing
- Schedule development

Activity definition- specific requirements are specified, estimates the amount of time received

Activity sequencing- determines the various relations between other activities.

2) Network diagrams are used: it shows the relationship between tasks basically there are 4 types of relationship

a) Start to finish
b) Finish to finish
c) Finish to start
d) Start to start

Two methods are used

a) Arrow diagram method
b) Precedence diagram method

Project quality: Aim to satisfy the needs for which it was taken. It is achieved by

Proper planning → Quality Planning
→ Quality insurance
→ Quality control

Factors of quality planning are

a) Cost of quality
b) Cost of benefit analysis
c) Benchmarking
d) Design of experiments

Quality insurance: 1) application of land

2) Systematic activities to employ all process need to meet the requirements

Quality control:

a) Flow charts
b) Testing
c) Reviews
d) Control charts
e) histogram

Unit-3

Planning

Work Breakdown Structure (WBS): A **work breakdown structure (WBS)** in project management and systems engineering, is a tool used to define and group a

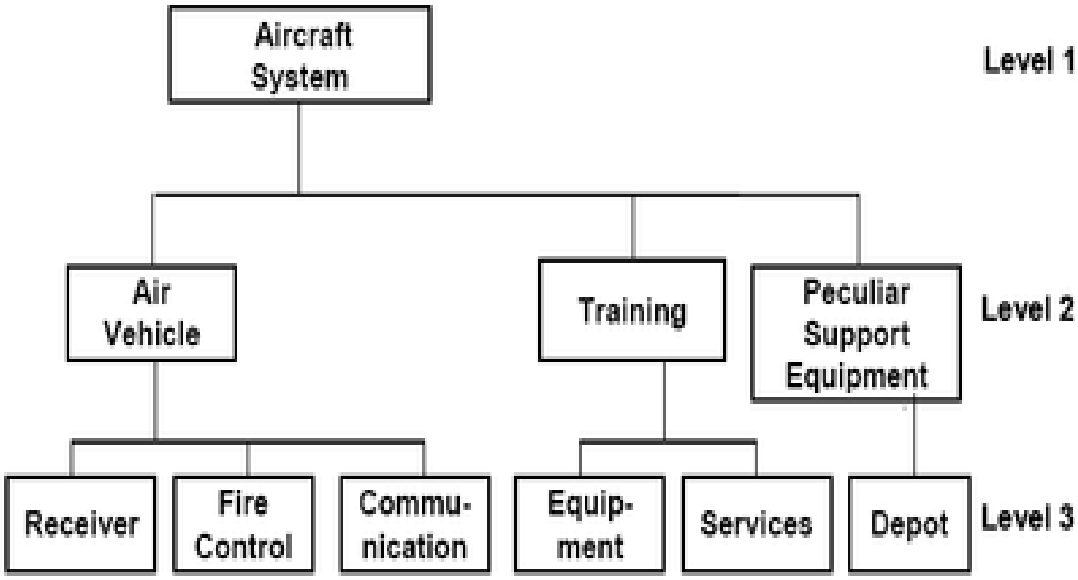
project's discrete work elements in a way that helps organize and define the total work scope of the project.

A work breakdown structure element may be a product, data, a service, or any combination. A WBS also provides the necessary framework for detailed cost estimating and control along with providing guidance for schedule development and control. Additionally the WBS is a dynamic tool and can be revised and updated as needed by the project manager.

Overview

The Work Breakdown Structure is a tree structure, which shows a subdivision of effort required to achieve an objective; for example a program, project, and contract. In a project or contract, the WBS is developed by starting with the end objective and successively subdividing it into manageable components in terms of size, duration, and responsibility (e.g., systems, subsystems, components, tasks, subtasks, and work packages) which include all steps necessary to achieve the objective.

The Work Breakdown Structure provides a common framework for the natural development of the overall planning and control of a contract and is the basis for dividing work into definable increments from which the statement of work can be developed and technical, schedule, cost, and labor hour reporting can be established.



The Work Breakdown Structure provides a common framework for the natural development of the overall planning and control of a contract and is the basis for dividing work into definable increments from which the statement of work can be developed and technical, schedule, cost, and labor hour reporting can be established.

History

The concept of Work Breakdown Structure developed with the Program Evaluation and Review Technique (PERT) in the United States Department of Defense (DoD). PERT was introduced by the U.S. Navy in 1957 to support the development of its Polaris missile program. While the term "work breakdown structure" was not used, this first implementation of PERT did organize the tasks into product-oriented categories.

Example

The WBS Construction Technique employing the 100% Rule during WBS construction.

The figure on the right shows a Work Breakdown Structure construction technique that demonstrates the 100% Rule and the "progressive elaboration" technique. At WBS Level 1 it shows 100 units of work as the total scope of a project to design and build a custom bicycle. At WBS Level 2, the 100 units are divided into seven elements. The number of units allocated to each element of work can be based on effort or cost; it is not an estimate of task duration.

The three largest elements of WBS Level 2 are further subdivided at Level 3. The two largest elements at Level 3 each represent only 17% of the total scope of the project. These larger elements could be further subdivided using the *progressive elaboration* technique described above.

Dynamic Systems Development Method: Dynamic Systems Development Method (DSDM) is a software development methodology originally based upon the Rapid Application Development methodology. DSDM is an iterative and incremental approach that emphasizes continuous user involvement.

Its goal is to deliver software systems on time and on budget while adjusting for changing requirements along the development process. DSDM is one of a number of Agile methods for developing software, and it forms a part of the Agile Alliance.

Overview

As an extension of rapid application development, DSDM focuses on Information Systems projects that are characterized by tight schedules and budgets. DSDM addresses the most common failures of information systems projects, including exceeding budgets, missing deadlines, and lack of user involvement and top-management commitment. By encouraging the use of RAD, however, careless adoption of DSDM may increase the risk of cutting too many corners. DSDM consists of

- Three phases: pre-project phase, project life-cycle phase, and post-project phase.

- A project life-cycle phase is subdivided into 5 stages: feasibility study, business study, functional model iteration, design and build iteration, and implementation.

In some circumstances, there are possibilities to integrate practices from other methodologies, such as Rational Unified Process (RUP), Extreme Programming (XP), and PRINCE2, as complements to DSDM. Another agile method that has some similarity in process and concept to DSDM is Scrum.

DSDM was developed in the United Kingdom in the 1990s by the DSDM Consortium, an association of vendors and experts in the field of software engineering created with the objective of "jointly developing and promoting an independent RAD framework" by combining their best practice experiences. The DSDM Consortium is a not-for-profit, vendor-independent organization which owns and administers the DSDM framework. The first version was completed in January 1995 and was published in February 1995. The current version in use (as of April 2006) is *Version 4.2: Framework for Business Centered Development*, and was released in May 2003. In July 2006, DSDM Public Version 4.2[1] was made available for individuals to view and use; however, anyone reselling DSDM must still be a member of the not-for-profit consortium.

The DSDM approach

Principles

There are nine underlying principles consisting of four foundations and five starting-points.

- User involvement is the main key in running an efficient and effective project, where both users and developers share a workplace, so that the decisions can be made accurately.
- The project team must be empowered to make decisions that are important to the progress of the project without waiting for higher-level approval.
- A focus on frequent delivery of products, with assumption that to deliver something "good enough" earlier is always better than to deliver everything "perfectly" in the end. By delivering product frequently from an early stage of the project, the product can be tested and reviewed where the test record and review document can be taken into account at the next iteration or phase.
- The main criteria for acceptance of a “deliverable” are delivering a system that addresses the current business needs. Delivering a perfect system which addresses all possible business needs is less important than focusing on critical functionalities.
- Development is iterative and incremental and driven by users’ feedback to converge on an effective business solution.
- All changes during the development are reversible.
- The high level scope and requirements should be base-lined before the project starts.
- Testing is carried out throughout the project life-cycle. (See Test-driven development for comparison).
- Communication and cooperation among all project stakeholders is required to be efficient and effective.

Prerequisites for using DSDM

In order for DSDM to be a success, a number of prerequisites need to be realized. First, there needs to be interactivity between the project team, future end users and higher management. This addresses well known failures of IS development projects due to lack of top management motivation and/or user involvement. The second prerequisite for DSDM projects is that the project can be decomposed in to smaller parts enabling the use of an iterative approach.

Two examples of types of projects for which DSDM is not considered well-suited are:

- Safety-critical projects - the extensive testing and validation found in safety-critical projects conflict with DSDM goals of being on time and on budget.
- Projects that aim to produce re-usable components - the demands on perfection are often too high and conflict with the 80%/20% principle described earlier.

DSDM Project Life-cycle

Overview: three phases of DSDM

The DSDM framework consists of three sequential phases, namely the pre-project, project life-cycle and post-project phases. The project phase of DSDM is the most elaborate of the three phases. The project life-cycle phase consists of 5 stages that form an iterative step-by-step approach in developing an IS. The three phases and corresponding stages are explained extensively in the subsequent sections. For each stage/phase, the

most important activities are addressed and the deliverables are mentioned.

Phase 1 - The Pre-Project

In the pre-project phase candidate projects are identified, project funding is realized and project commitment is ensured. Handling these issues at an early stage avoids problems at later stages of the project like cows.

Phase 2 - The Project life-cycle

The process overview in the figure below shows the project life-cycle of this phase of DSDM. It depicts the 5 stages a project will have to go through to create an IS. The first two stages, the Feasibility Study and Business Study are sequential phases that complement to each other. After these phases have been concluded, the system is developed iteratively and incrementally in the Functional Model Iteration, Design & Build Iteration and Implementation stages. The iterative and incremental nature of DSDM will be addressed further in a later section.

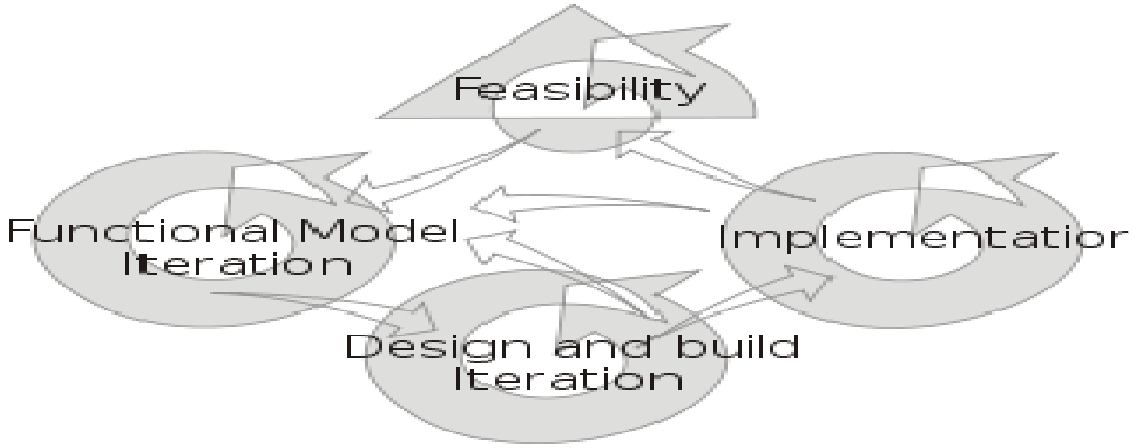
Phase 3 - Post-project

The post-project phase ensures the system operating effectively and efficiently. This is realized by maintenance, enhancements and fixes according to DSDM principles. The maintenance can be viewed as continuing development based on the iterative and incremental nature of DSDM.

Instead of finishing the project in one cycle usually the project can return to the previous phases or stages so that the previous step and the deliverable products can be refined.

Below is the process-data diagram of DSDM as a whole Project life-cycle with all of its four steps. This diagram depicts the DSDM iterative development, started on functional model iteration, design and build iteration, and implementation phase. The description of each stage will be explained later in this entry.

Four stages of the Project life-cycle



Model of the DSDM software development process.

Stage 1A: The Feasibility Study

During this stage of the project, the feasibility of the project for the use of DSDM is examined. Prerequisites for the use of DSDM are addressed by answering questions like: 'Can this project meet the required business needs?', 'Is this project suited

for the use of DSDM?' and 'What are the most important risks involved?'. The most important techniques used in this phase are the Workshops.

The deliverables for this stage are the Feasibility Report and the Feasibility Prototype that address the feasibility of the project at hand. It is extended with a Global Outline Plan for the rest of the project and a Risk Log that identifies the most important risks for the project.

Stage 1B: The Business Study

The business study extends the feasibility study. After the project has been deemed feasible for the use of DSDM, this stage examines the influenced business processes, user groups involved and their respective needs and wishes. Again the workshops are one of the most valuable techniques, workshops in which the different stakeholders come together to discuss the proposed system. The information from these sessions is combined into a requirements list. An important property of the requirements list is the fact that the requirements are (can be) prioritized. These requirements are prioritized using the Moscow approach. Based on this prioritization, a development plan is constructed as a guideline for the rest of the **project.**

An important project technique used in the development of this plan is time boxing. This technique is essential in realizing the goals of DSDM, namely being on time and on budget, guaranteeing the desired quality. A system architecture is another aid to guide the development of the IS. The deliverables for this stage are a business area definition that describes the context of the project within the company, a system architecture

definition that provides an initial global architecture of the IS under development together with a development plan that outlines the most important steps in the development process. At the base of these last two documents there is the prioritized requirements list. This list states all the requirements for the system, organized according to the Moscow principle. And last the Risk Log is updated with the facts that have been identified during this phase of DSDM.

Stage 2: Functional Model Iteration

The requirements that have been identified in the previous stages are converted to a functional model. This model consists of both a functioning prototype and models. Prototyping is one of the key project techniques within this stage that helps to realize good user involvement throughout the project. The developed prototype is reviewed by different user groups. In order to assure quality, testing is implemented throughout every iteration of DSDM. An important part of testing is realized in the Functional Model Iteration. The Functional Model can be subdivided into four sub-stages:

- Identify Functional Prototype: Determine the functionalities to be implemented in the prototype that results from this iteration.
- Agree Schedule: Agree on how and when to develop these functionalities.
- Create Functional Prototype: Develop the prototype. Investigate, refine, and consolidate it with the combined Functional prototype of previous iterations.

- Review Prototype: Check the correctness of the developed prototype. This can be done via testing by end-user, then use the test records and user's feedbacks to generate the functional prototyping review document.

The deliverables for this stage are a Functional Model and a Functional Prototype that together represent the functionalities that could be realized in this iteration, ready for testing by users. Next to this, the Requirements List is updated, deleting the items that have been realized and rethinking the prioritization of the remaining requirements. The Risk Log is also updated by having risk analysis of further development after reviewing the prototyping document.

Stage 3: Design and Build Iteration

The main focus of this DSDM iteration is to integrate the functional components from the previous phase into one system that satisfies user needs. It also addresses the non-functional requirements that have been set for the IS. Again testing is an important ongoing activity in this stage. The Design and Build Iteration can be subdivided into four sub-stages:

- Identify Design Prototype: Identify functional and non-functional requirements that need to be in the tested system.
- Agree Schedule: Agree on how and when to realize these requirements.
- Create Design Prototype: Create a system that can safely be handed to end-users for daily use. They investigate, refine, and consolidate the prototype of current iteration within prototyping process are also important in this sub-stage.

- Review Design Prototype: Check the correctness of the designed system. Again testing and reviewing are the main techniques used, since the test records and user's feedbacks are important to generate the user documentation.

The deliverables for this stage are a Design Prototype during the phase that end users get to test and at the end of the Design and Build Iteration the Tested System is handed over to the next phase. In this stage, the system is mainly built where the design and functions are consolidated and integrated in a prototype. Another deliverable for this stage is a User Documentation.

Stage 4: Implementation

In the Implementation stage, the tested system including user documentation is delivered to the users and training of future users is realized. The system to be delivered has been reviewed to include the requirements that have been set in the beginning stages of the project. The Implementation stage can be subdivided into four sub-stages:

- User Approval and Guidelines: End users approve the tested system for implementation and guidelines with respect to the implementation and use of the system are created.
- Train Users: Train future end user in the use of the system.
- Implement: Implement the tested system at the location of the end users.

Review Business: Review the impact of the implemented system on the business, a central issue will be whether the system meets the goals set at the beginning of the project. Depending on this

the project goes to the next phase, the post-project or loops back to one of the preceding phases

Gantt chart: A **Gantt chart** is a type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.

Although now regarded as a common charting technique, Gantt charts were considered revolutionary when they were introduced. In recognition of Henry Gantt's contributions, the Henry Laurence Gantt Medal is awarded for distinguished achievement in management and in community service. This chart is used also in Information Technology to represent data that have been collected

Historical development

The first known tool of this type was reportedly developed in 1896 by Karol Adamiecki, who called it a *harmonogram*. Adamiecki did not publish his chart until 1931, however, and then only in Polish. The chart is commonly known after Henry Gantt (1861–1919), who designed his chart around the years 1910–1915.

In the 1980s, personal computers allowed for widespread creation of complex and elaborate Gantt charts. The first desktop

applications were intended mainly for project managers and project schedulers. With the advent of the internet and increased collaboration over networks at the end of the 1990s, Gantt charts became a common feature of web-based applications, including collaborative groupware.

A Gantt chart showing three kinds of schedule dependencies (in red) and percent complete indications

Advantages and limitations

Gantt charts have become a common technique for representing the phases and activities of a project work breakdown structure (WBS), so they can be understood by a wide audience all over the world.

A common error made by those who equate Gantt chart design with project design is that they attempt to define the project work breakdown structure at the same time that they define schedule activities. This practice makes it very difficult to follow the 100% Rule. Instead the WBS should be fully defined to follow the 100% Rule, then the project schedule can be designed.

Although a Gantt chart is useful and valuable for small projects that fit on a single sheet or screen, they can become quite unwieldy for projects with more than about 30 activities. Larger Gantt charts may not be suitable for most computer displays. A related criticism is that Gantt charts communicate relatively little information per unit area of display. That is, projects are often considerably more complex than can be communicated effectively with a Gantt chart.

Gantt charts only represent part of the triple constraints (cost, time and scope) of projects, because they focus primarily on schedule management. Moreover, Gantt charts do not represent the size of a project or the relative size of work elements, therefore the magnitude of a behind-schedule condition is easily mis communicated. If two projects are the same number of days behind schedule, the larger project has a larger impact on resource utilization, yet the Gantt does not represent this difference.

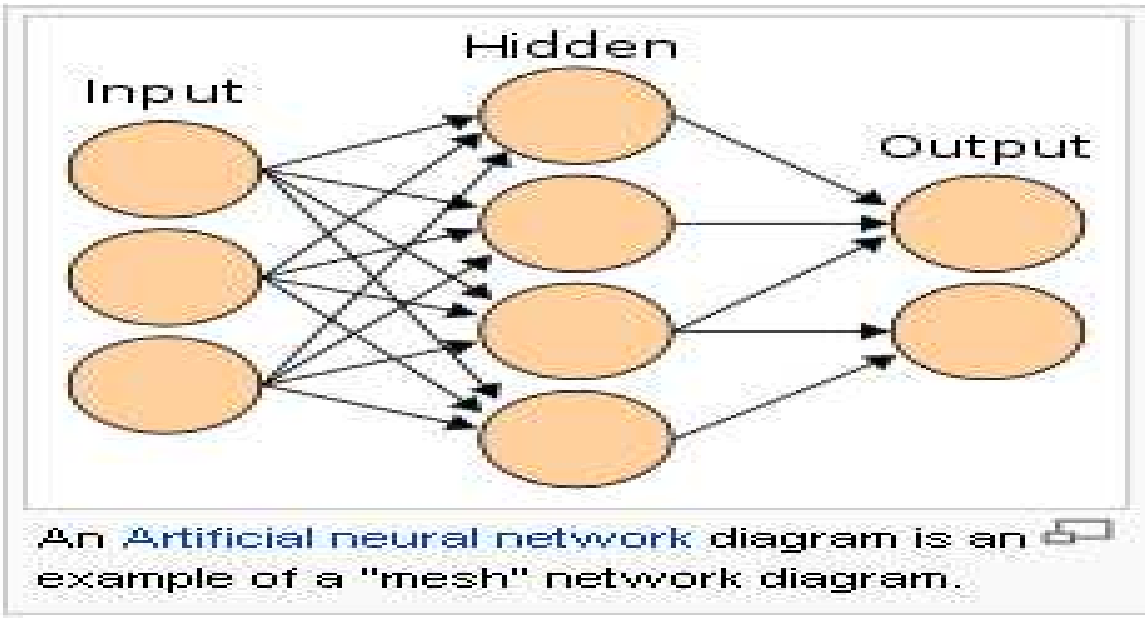
Example

- In the following example there are seven tasks, labeled *A* through *G*. Some tasks can be done concurrently (*A* and *B*) while others cannot be done until their predecessor task is

complete (*C* cannot begin until *A* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*O*), the most likely or normal time estimate (*M*), and the pessimistic time estimate (*P*). The expected time (T_E) is computed using the formula ($O + 4M + P$) ÷ 6for further development.

Network diagram: A network diagram is a type of network. A network in general is an interconnected group or system, or a fabric or structure of fibrous elements attached to each other at regular intervals, or formally: a graph.

A network diagram is a special kind of cluster diagram, which even more general represents any cluster or small group or bunch of something, structured or not. Both the flow diagram and the tree diagram can be seen as a specific type of network diagram.



An Artificial neural network diagram is an example of a "mesh" network diagram.

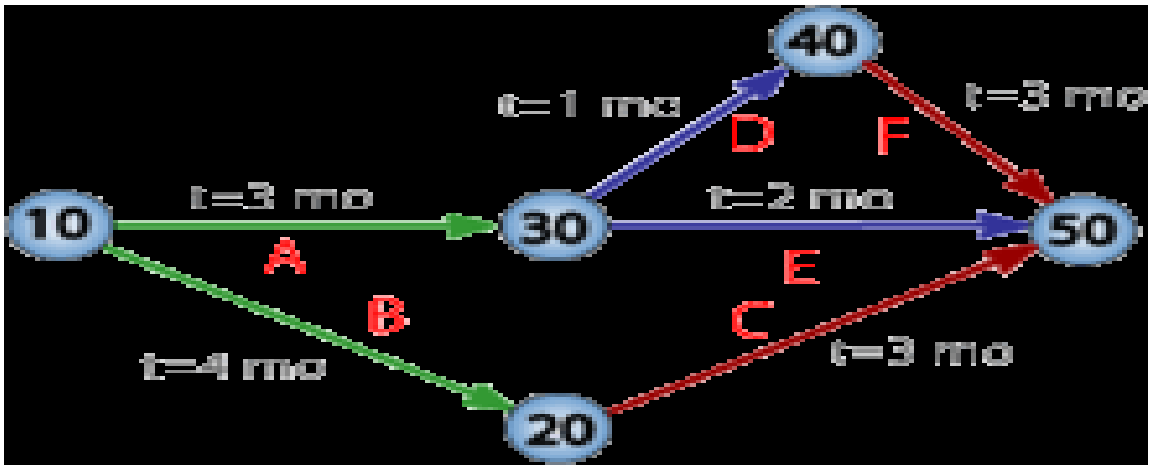
Types of network diagrams

There are different type's network diagrams:

- Artificial neural network or "neural network" (NN), is a mathematical model or computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation.
- Computer network diagram is a schematic depicting the nodes and connections amongst nodes in a computer network or, more generally, any telecommunications network.
- In project management according to Baker et al. (2003), a "network diagram is the logical representation of activities, that defines the sequence or the work of a project. It shows the path of a project, lists starting and completion dates, and names the responsibilities for each task. At a glance it explains how the work of the project goes together... A network for a simple project might consist one or two pages, and on a larger project several network diagrams may exist". Specific diagrams here are
  - Project network: a general flow chart depicting the sequence in which a project's terminal elements are to be completed by showing terminal elements and their dependencies.
  - PERT network

- Neural network diagram: is a network or circuit of biological neurons or artificial neural networks, which are composed of artificial neurons or nodes.
- A semantic network is a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes.
- A socio gram is a graphic representation of social links that a person has. It is a socio metric chart that plots the structure of interpersonal relations in a group situation

Critical path method (CPM) : The **critical path method (CPM)** is an algorithm for scheduling a set of project activities. It is an important tool for effective project management.



History

The Critical Path Method (CPM) is a project modeling technique developed in the late 1950s by Morgan R. Walker of DuPont and James E. Kelley, Jr. of Remington Rand.Kelley and Walker related their memories of the development of CPM in

1989.Kelley attributed the term "critical path" to the developers of the Program Evaluation and Review Technique which was developed at about the same time by Booz Allen Hamilton and the US Navy. The precursors of what came to be known as Critical Path were developed and put into practice by DuPont between 1940 and 1943 and contributed to the success of the Manhattan Project.

CPM is commonly used with all forms of projects, including construction, aerospace and defense, software development, research projects, product development, engineering, and plant maintenance, among others. Any project with interdependent activities can apply this method of mathematical analysis. Although the original CPM program and approach is no longer used, the term is generally applied to any approach used to analyze a project network logic diagram.

Basic technique

The essential technique for using CPM [6] is to construct a model of the project that includes the following:

1. A list of all activities required to complete the project (typically categorized within a work breakdown structure),
2. The time (duration) that each activity will take to completion, and
3. The dependencies between the activities

Using these values, CPM calculates the longest path of planned activities to the end of the project, and the earliest and latest that each activity can start and finish without making the project longer. This process determines which activities are "critical"

(i.e., on the longest path) and which have "total float" (i.e., can be delayed without making the project longer). In project management, a **critical path** is the sequence of project network activities which add up to the longest overall duration. This determines the shortest time possible to complete the project. Any delay of an activity on the critical path directly impacts the planned project completion date (i.e. there is no float on the critical path). A project can have several, parallel, near critical paths. An additional parallel path through the network with the total durations shorter than the critical path is called a sub-critical or non-critical path.

These results allow managers to prioritize activities for the effective management of project completion, and to shorten the planned critical path of a project by pruning critical path activities, by "**fast tracking**" (i.e., performing more activities in parallel), and/or by "**crashing the critical path**" (i.e., shortening the durations of critical path activities by adding resources).

Rapid application development (RAD): Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaption of the linear sequential model in which rapid development is achieved by using component-based construction.

If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days). Used primarily for information systems

applications, the RAD approaches encompass the following phases:

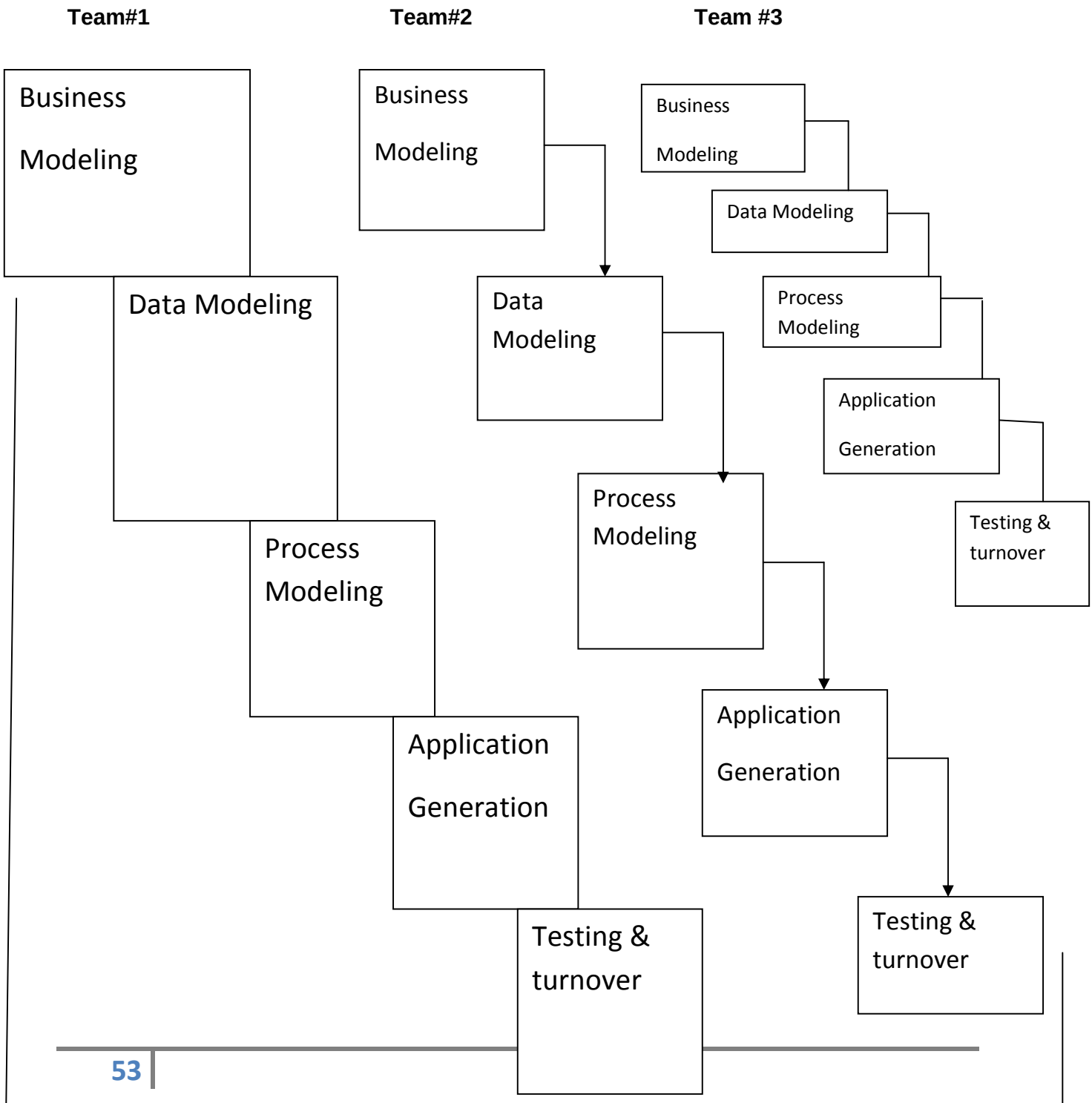
Business modeling. The information flow among business functions is modeled in a way that answers the following questions: what information drives the business process? What information is generated? Who generates it? Where does the information go? Who process it?

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

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

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

Testing and Turnover. Since the RAD process emphasizes reuse, many of the program components have been tested. This.


Team#1
Team#2
Team #3
Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing & turnover
Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing & turnover
Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing & turnover

60-90days

Reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised

The RAD process model is illustrated in fig. obviously, the time constraints imposed on a RAD project demand "scalable scope". If a business application can be modularized in a way that enables each major function to be completed in less than three months, it is a candidate for RAD. Each major function can be addressed by a separate RAD team and then integrated to form a whole.

The RAD approach has following drawbacks

- For large but scalable projects, RAD requires sufficient human resources 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. If commitment is lacking form either constituency, RAD projects will fail.
- Not all types of applications are appropriate for RAD. If a system cannot be popularly modularized, building the components necessary for RAD

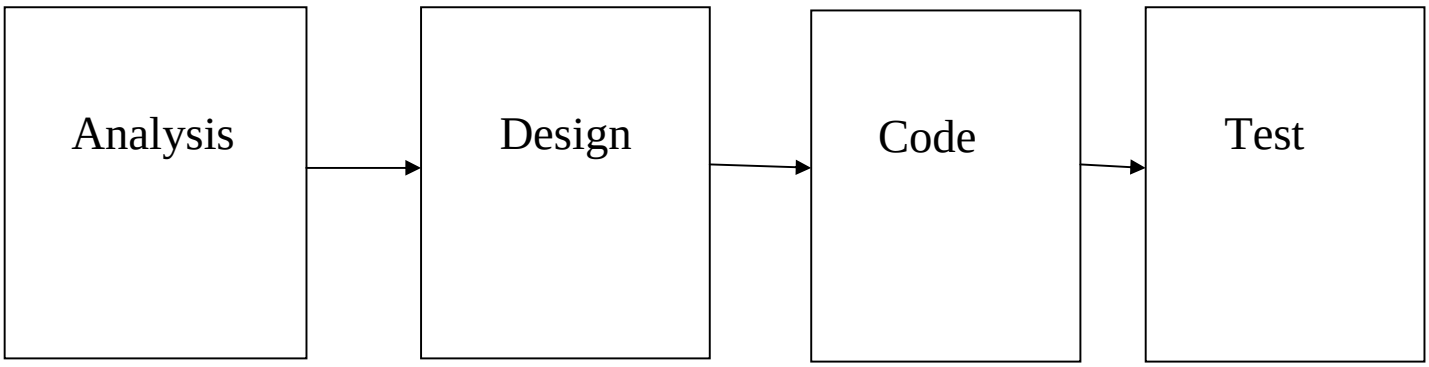
will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.RAD is not appropriate when technical risks are high. This occurs when a new application makes heavy use of new technology.

Program me Evaluation and Review Technique (PERT): In PERT analysis, time duration of each activity is no longer a single time estimate, but is a random variable characterized by some probability distribution- usually a β-distribution. To estimate the parameters of the β- distribution (the mean and variance), the PERT system is based on *three time estimates* of the performance time of an activity. They are

1) *The Optimistic Time Estimate:* The shortest possible time required for the completion of an activity, if all goes extremely well. No provisions are made for delays or setbacks while estimating this time.
2) *The pessimistic Time Estimate:* The maximum possible time the activity will take if everything goes bad. However, major catastrophes such as earthquakes, floods, storms and labor troubles are not taken into account while estimating this time.
3) *The Most Likely Time Estimate:* The time an activity will take if executed under normal conditions. It is the modal value.

For determining the single time estimates used in CPM, some historical data may be available, but the best way of predicting the three time estimates is by intelligent guessing. The experienced person who may be an engineer, foreman or worker having sufficient technical competence is asked to guess the various time estimates. For estimation the activity should be taken randomly, so that the guess of the assessor is not prejudiced by the predecessor and the successor activities.

Waterfall model: Waterfall model suggests a systematic sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. Fig illustrates the waterfall model for software engineering.



Analysis. The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program to be built, the software engineer must understand the information domain for the software as well as required

function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

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

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

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

Advantages and Disadvantages

Simplicity.
• Convey major software engineering activities to the uninitiated. (*Next few slides illustrate*).
• Clear high level view of the process.

• Provide crude milestones.
Inability to cope with:
Iteration.
Prototyping.
Levels of detail or abstraction.
More latterly
Modern methods, rapid development, OO etc...

COCOMO Model: A top-down model can depend on many different factors, instead of depending only on one variable, giving rise to multivariable models. One approach for building multivariable models is to start with an initial estimate determined by using the static single-variable model equations, which depend on size, and then adjusting the estimates based on other variables. This approach implies that size is the primary factor for cost; other factors have a lesser effect. Here we will discuss one such model called the constructive structure cost model (COCOMO) developed by Boehm. The basic steps in this model are:

1. Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KLOC).
2. Determine a set of 15 multiplying factors from different attributes of the project.
3. Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors.

The initial estimate is determined by an equation of the form used in the static single-variable models, using KLOC as the measure of size. To determine the initial effort E_i in person-

months the equation used is of the type $E_i=a*(KLOC)^{b.}$ The value of the constants a and b depend on the project type.

In COCOMO, projects are categorized into three types- organic, semidetached, and embedded. These categories roughly characterize the complexity of the project with organic projects being those that are relatively straight forward and developed by a small team, and embedded are those that are ambitious and novel, with stringent and reliability. The constraints a and b for different systems are:

System	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

The value of the constants for a cost model depends on the process and has to be determined from the past data. COCOMO has instead provided "global" constant values. These values should be considered as values to start with until data for some projects is available. With project data, the value of the constraints can be determined through regression analysis.

As an example, suppose a system for office automation has to be designed. From the requirements, it is clear that there will be four major modules in the system: data entry, data update, query, and report generator. It is also clear from the requirements that this project will fall in the organic category. The sizes for the different modules and the overall system are estimated to be:

Data entry	0.6 KLOC
Data update	0.6 KLOC

Query	0.8 KLOC
Reports	1.0 KLOC
TOTAL	3.0 KLOC

From the requirements, the ratings of the different cost driver attributes are assessed. These ratings, along with their multiplying factors, are:

Complexity	High	1.15
Storage	High	1.06
Experience	Low	1.13
Programmer capability	Low	1.17

All other factors had a nominal rating. From these, the effort adjustment factor (EAF) is

EAF= 1.15*1.06*1.13*1.17=1.61.

The initial effort estimate for the project is obtained from the relevant equations. We have

$E_{i=}$ $3.2*3^{1.05}$=10.14 PM

Using the EAF, the adjusted effort estimate is

E= 1.61*10.14= 16.3 PM.

Using the preceding table, we obtain the percentage of the total effort consumed in different phases. The office automation system's size estimate is 3 KLOC, so we will have to use interpolation to get the appropriate percentage. The percentages for the different phases are: design-16%, detailed design-25.83%, code and unit-test-41.66%, and integration and testing-16.5%. With these, the effort estimates for the different phases are:

System Design	.16*16.3= 2.6PM
Detailed Design	.258*16.3=4.2 PM
Code and Unit Test	.4166*16.3=6.8PM
Integration	.165*16.3=2.7 PM

Life cycle phases: Life cycle phases will be explained in six steps:

I. **Recognition of need**. The basis for a candidate system is recognition of a need for improving an information system or a procedure. This need leads to a preliminary survey or an initial investigation to determine whether an alternative system can solve the problem.

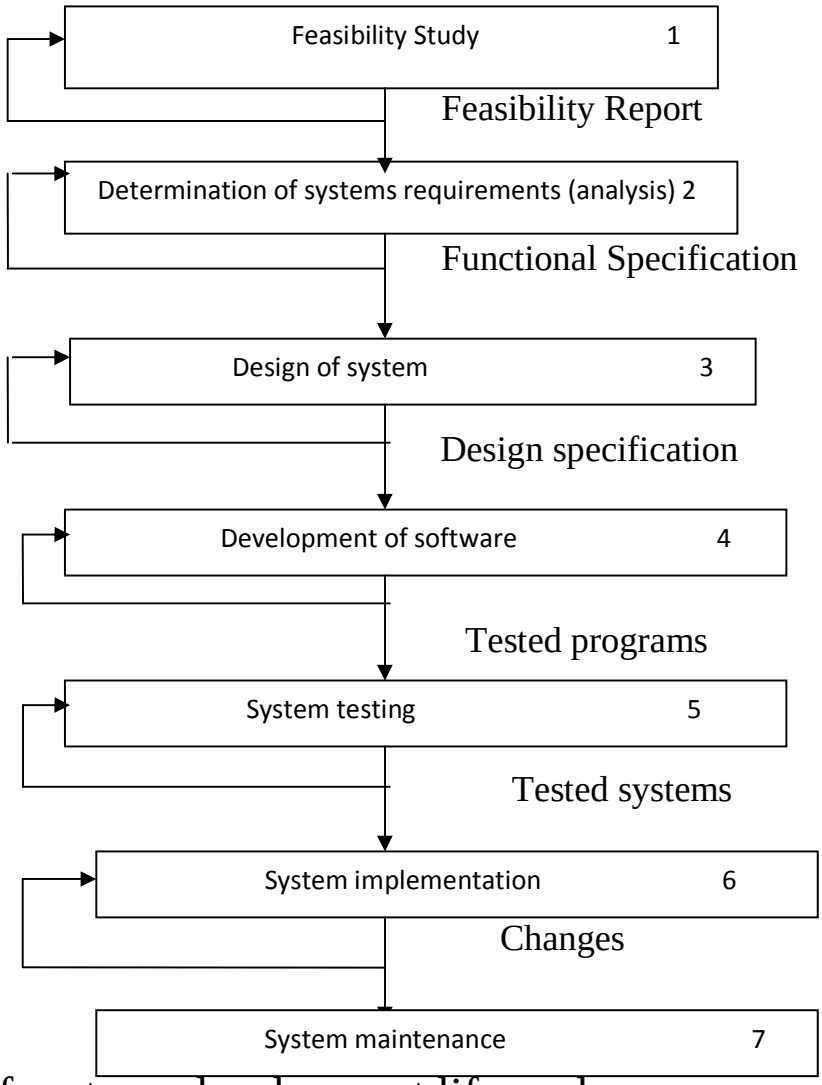


Fig. phases of systems development life cycle

II. **Feasibility study:** It is depending on the results of the initial investigation, the survey is expanded to more detailed feasibility study. A feasibility study is a test of a system proposal according to its workability, impact on the organization, ability to meet user need and effective use of resources. It revolves around investigation and evaluation of the problem, identification and description of candidate systems, specification of performance and the cost of each system, and final selection of the best system. The objective of a feasibility study is not to solve the problem but to acquire a sense of its scope. The result of the feasibility study is a formal proposal.

III. **Analysis:** Analysis is a detailed study of the various operations performed by a system and their relationships with in and outside of the system. One aspect of analysis is defining the boundaries of the system and determining whether or not a candidate system should consider other related systems. During analysis data are collected on the available files, decision prints and transactions handled by the present system.

IV. **Design:** The most creative and challenging phase of the system life cycle is “System Design”. The term ‘design’ describes a final system and the process by which it is developed. It refers to the technical specifications that will be applied in implementing the candidate system. It also includes the construction to program testing. .The first step is to determine how the output is to be produced and in what format. The second step is input data and master files have to be designed to meet the requirements of the proposed output.

V. **Implementation:** The implementation phase is less creative than system design. It is a primarily concerned with user training, site preparation, and file conversion. When the candidate system is linked to terminals or remote sites, the telecommunication network and tests of the network along with the system are also included under the implementation.

VI. **Post-Implementation and Maintenance:** After the installation phase is completed and the user staff is adjusted to the changes created by the candidate system, evaluation and maintenance begin.

Scheduling: Scheduling of a software project does not differ greatly from scheduling multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied to software with little modification.

Program evaluation and review technique (PERT) and critical path method (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities:

- Estimation of effort
- A decomposition of product function
- The selection of the appropriate process model
- The selection of project type and task set

Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project work breakdown structure (WBS), are defined for the product as a whole or for individual functions.

Both PERT and CPM provide quantitative tools that allow the software planner to (1) determine the critical path-the chain of tasks that determines the duration of the project; (2) establish most likely time estimates for individual tasks by applying statistical models; and (3) calculate boundary times define a time "window" for a particular task.

Boundary time calculations can be very useful in software project scheduling. Slippage in the design of one function, for example, can retard further development of other functions. Boundary time calculations lead to a determination of critical path and provide the manager with a quantitative method for evaluating progress as tasks are completed.

Tracking the schedule. The project schedule provides a road map for a software project manager. If it has been properly developed, the project schedule defines the tasks and milestones that must be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways:

- Conducting periodic project status meetings in which each team member reports progress and problems
- Evaluating the results of all reviews conducted throughout the software engineering process
- Determine whether formal project milestones have been accomplished by the scheduled date
- Comparing actual start date to planned start date for each project task listed in the project table.
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.

Control is employed by a software project manager to administrator project resources, cope with problems, and direct

project staff. If things are going well. Control is light. but when problems occur, the project manager must exercise control to reconcile them as quickly as possible. After a problem has been diagnosed, additional resources may be focused on the problem area: staff may be redeployed, or the project schedule can be redefined.

Requirements: Requirements analysis is a software engineering task that bridges the gap between system-level software allocation and software design (fig)

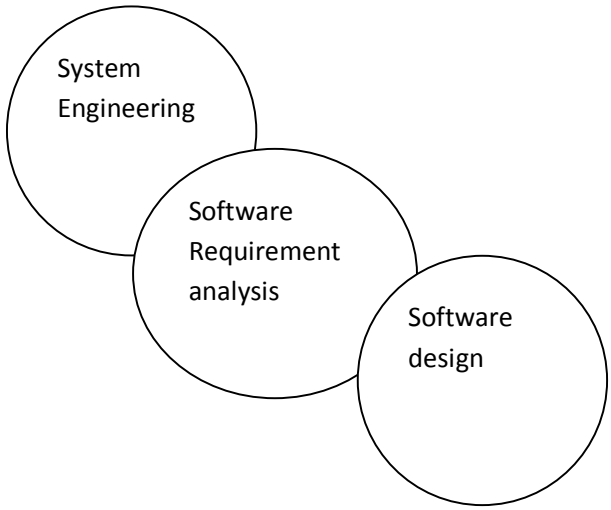


Fig. Analysis and a bridge between system engineering and software design

Requirements analysis enables the system engineer to specify software function and performance, indicate software's interface with other system elements, and establish constraints that software must meet.

Requirements analysis provides the software designer with models that can be translated in to data, architectural, interface, and procedural design. Finally, the requirements

specification provides the developer and the customer with the means to assess quality once software is built.

Software requirements analysis may be divided into five areas of effort: (1) problem recognition; (2) evaluation and synthesis; (3) modeling; (4) specification; (5) review.

Initially, the analyst studies the system specification and the software project plan. It is important to understand software in a system context and to review the software scope that was used to generate planning estimates.

Problem evaluation and solution synthesis is the next major area of effort for analysis. The analyst must define all externally observable data objects; evaluate the flow and content of information; define and elaborate all software functions; understand software behavior in the context of events that affect the system; establish system interface characteristics; and uncover additional design constraints. Each of these tasks serves to describe the problem so that overall approach or solution may be synthesized.

Throughout evaluation and solution synthesis, the analyst's primary focus is on "what", not "how". What data does the system produce and consume, what functions must the system perform, what interfaces are defined, and what constraints apply.

During the evaluation and solution synthesis activity, the analyst creates models of the system in an effort to better understand data and control flow, functional processing and behavioral operation, and information content. The model serves as a foundation for software design and as the basis for the creation of a specification for the software.

Analysis: The set of operational principles for all analysis methods are as follows:

1. The information domain of a problem must be represented and understood.
2. The functions that the software is to perform must be defined.
3. The behavior of the software must be represented.
4. The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered fashion.
5. The analysis process should move from essential information toward implementation detail.

By applying these principles, the analyst approaches a problem systematically. The information domain is examined so that function may be understood more completely. Models are used so that the characteristics of function and behavior can be communicated in a compact fashion. Partitioning is applied to reduce complexity. Essential and implementation views of the software are necessary to accommodate the logical constraints imposed by processing requirements and the physical constraints imposed by other system elements.

Davis [DAV95a] suggests a set of guiding principles for "requirements engineering":

- *Understand the problem before you begin to create the analysis model.* There is a tendency to rush to a solution, even before the problem is understood. This often leads to elegant software that solves the wrong problem!
- *Develop prototypes that enable a user to understand how human-machine interaction will occur.* Since the perception of the quality of software is often based on the perception of the "friendliness" of the interface, prototyping is highly recommended.

- *Record the origin of and the reason for every requirement.* This is the first step in establishing traceability back to the customer.
- *Use multiple views of requirements.* Building data, functional, and behavioral models provides the software engineer with three different views.
- *Prioritize requirements.* Tight deadlines may preclude the implementation of every software requirement. If an incremental process model is applied, those requirements to be delivered in the first increment must be identified.
- *Work to eliminate ambiguity.* Because most requirements are described in a natural language, the opportunity for ambiguity abounds. The use of formal technical reviews is one way to uncover and eliminate ambiguity.

A software engineer who takes these principles to heart is more likely to develop a software specification that will provide an excellent foundation for design.

Partitioning: Problems are often too large and complex to be understood as a whole. For this reason, we tend to partition such problems into parts that can be easily understood and establish interfaces between the parts so that overall function can be accomplished. The fourth operational analysis principle suggests that the information, functional, and behavioral domains of software can be partitioned.

In essence, partitioning decomposes a problem into its constituent parts. Conceptually, we establish a hierarchical representation of information or function and ten partition the uppermost element by (1) exposing increasing detail by moving vertically in the hierarchy. (2) Decomposing the problem by moving horizontally in the hierarchy.

Requirements for safe home software may be analyzed by partitioning the information, functional, and behavioral domains of the product. To illustrate the functional domain of the problem will be partitioned. Fig illustrates a horizontal decomposition of safe home software. The problem is partitioned by representing constituent safe home software functions, moving horizontally in the functional hierarchy. Three major functions are noted on the first level of the hierarchy.

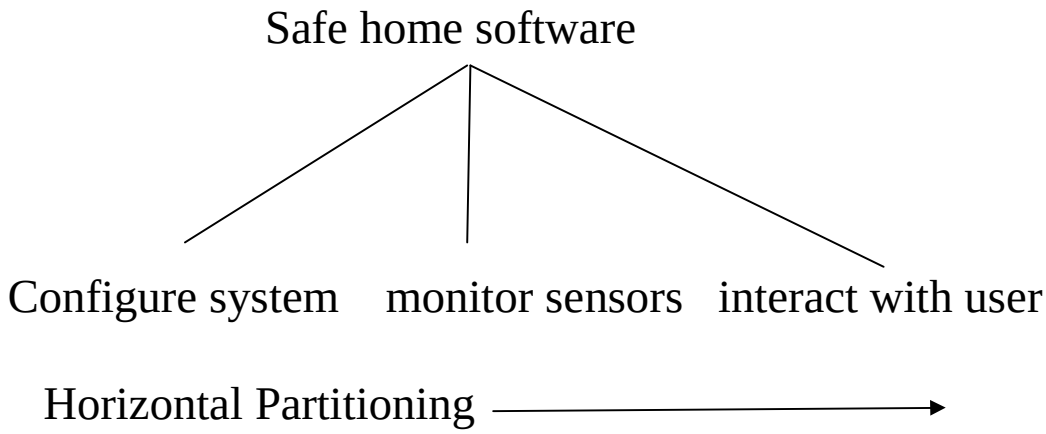


Fig1. safe home- horizontal partitioning function

The sub functions associated with a major safe home function may be examined by exposing detail vertically in the hierarchy, as illustrated in fig 2. Moving downward along a single path below the function monitor sensors, partitioning occurs vertically to show increasing levels of functional detail.

The partitioning approach that we have applied to safe home functions can also be applied to the information domain and behavioral domain as well. In fact, partitioning of information flow and system behavior will provide additional insight into system requirements. As the problem is partitioned, interfaces between functions are derived. Data and control items that move across an interface should be restricted to inputs required to

perform the stated function and outputs that are required by other functions or system elements.

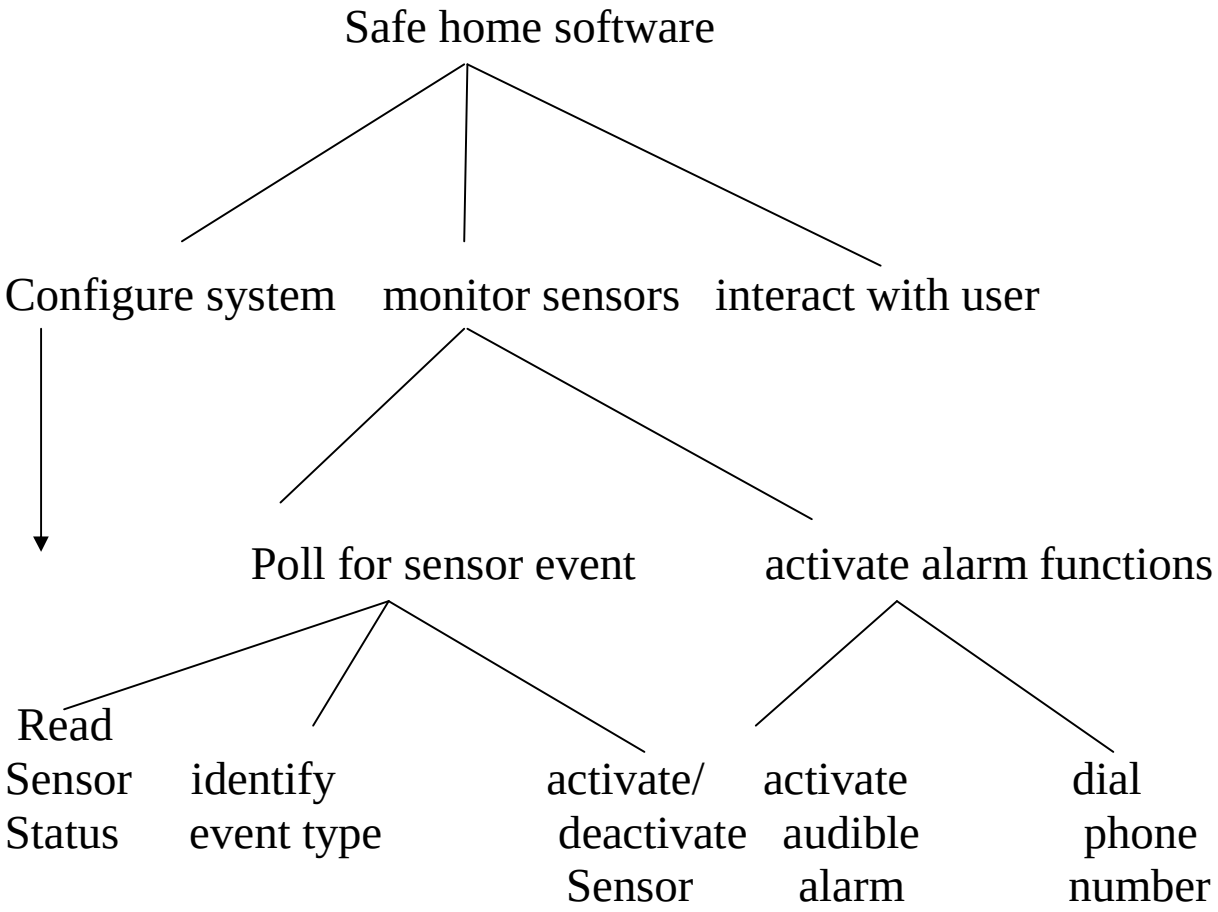


Fig2. safe home- vertical partitioning of function.

Estimation methodologies: Estimation is a form of problem solving, and in most cases, the problem to be solved is too complex to be considered in one piece.

Estimation can be divided into following categories:

- LOC based estimation
- Process-based estimation
- Empirical based estimation

- Cost estimation and Effort estimation

LOC based estimation. Lines of code (LOC) and function points (FP) were basic measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable that is used to size each element of the software (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

Process based estimation .The process is decomposed into a relatively small set of activities or tasks and the effort required to accomplish each task is estimated.

Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. Once problem functions and process activities are melded, the planner estimates the effort that will be required to accomplish each software process activity in each software function.

Empirical based estimation. An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. Values for LOC or FP are estimates are plugged into the estimation model.

The empirical data that support most estimation models is derived from limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments.

Cost and effort estimation. Cost and effort estimation will never be an exact science. Too many variables-human, technical, environmental, political-can affect the ultimate cost of software and effort applied to develop it.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple “decomposition techniques” to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

Unit- 4
Risk and Change Management

Project control: Project control is that element of a project that keeps it on-track, on-time and within budget. Project control begins early in the project with planning and ends late in the project with post-implementation review, having a thorough involvement of each step in the process. Each project should be assessed for the appropriate level of control needed: too much control is too time consuming, too little control is very risky. If project control is not implemented correctly, the cost to the business should be clarified in terms of errors, fixes, and additional audit fees.

Control systems are needed for cost, risk, quality, communication, time, change, procurement, and human resources. In addition, auditors should consider how important the projects are to the financial statements, how reliant the stakeholders are on controls, and how many controls exist. Auditors should review the development process and procedures for how they are implemented. The process of development and the quality of the final product may also be assessed if needed or requested. A business may want the auditing firm to be involved throughout the process to catch problems earlier on so that they can be fixed more easily. An auditor can serve as a controls consultant as part of the development team or as an independent auditor as part of an audit.

Businesses sometimes use formal systems development processes. These help assure that systems are developed successfully. A formal process is more effective in creating

strong controls, and auditors should review this process to confirm that it is well designed and is followed in practice. A good formal systems development plan outlines:

- A strategy to align development with the organization's broader objectives
- Standards for new systems
- Project management policies for timing and budgeting
- Procedures describing the process
- Evaluation of quality of change

Earned Value Analysis (EVA):

A technique for performing quantitative analysis of progress does exist. It is called Earned Value Analysis (EVA).

Humphrey discusses earned value in the following manner:

The earned value system provides a common value scale for every task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

Stated even more simply, earned value is a measure of progress. It enables us to assess the "percent of completeness" of a project using quantitative analysis rather then rely on a gut feeling. In fact, Fleming and Kop leman argue that earned value analysis "provides accurate and reliable readings of performance from as early as 15 percent into the project".

To determine the earned value, the following steps are performed:

1. The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. During the estimation activity, the work (in person-hours or person-days) of each software engineering task is planned. Hence, $BCWS_i$ is the effort planned for work task i. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
2. The BCWS values for all work tasks are summed to derive the budget at completion, BAC. Hence,

   $BAC=\sum (BCWS_k)$ for all tasks k.

3. Next, the value for budgeted cost of work performed (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by point in time on the project schedule.

According to Will kens “the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed”. Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

Scheduled performance index, SPI= BCWP/ BCWS

Scheduled variance, SV= BCWP- BCWS

SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

Percent Scheduled for completion= BCWS/ BAC

Provides an indication of the percentage of work that should have been completed by time, t.

Percent complete= BCWP/ BAC

Provides a quantitative indication of the percent of completeness of the project at a given point in time, t.

It is also possible to compute the actual cost of work performed, ACWP. The value for ACWP is the sum of the effort actually expanded on work tasks that have been completed by a

point in time on the project schedule. It is then possible to compute

Cost performance index, CPI= BCWP/ ACWP

Cost variance, CV= BCWP- ACWP

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings or shortfall at a particular stage of a project.

Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables the software project manager to take corrective action before a project crisis develops.

Risk identification: Risk identification is a systematic attempt to specify threats to the project plan. By identifying known and predictable risks, the project manger takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks for each of the categories which are generic risks and product-specific risks. *Generic* risks are a potential threat to every software project. *Product-specific risks* can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the project at hand.

One method for identifying risks is to create a *risk item checklist.* The checklist can be used for risk identification and

focuses on some subset of known and predictable risks in the following generic subcategories:

1. *Product size*- risks associated with the overall size of the software to be built or modified.
2. *Business impact*- - risks associated with constraints imposed by management or the market place.
3. *Customer characteristics*- - risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
4. *Process definition*-- risks associated with the degree to which the software process has been defined and is followed by the development organization.
5. *Development environment*-- risks associated with the availability and quality of the tools to be used to build the product.
6. *Technology to be built*- risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
7. *Staff size and experience*- - risks associated with the overall technical and project experience of the software engineers who will do the work.

Assessing overall project Risk

The following questions have derived form risk data obtained by surveying experienced software project managers in different

part of the world. The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end-users enthusiastically committed to the project and the system/product to be built?
3. Are requirements fully understood by the software engineering team and their customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end-users have realistic expectations?
6. Is project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?

Software Configuration Management (SCM): Software configuration management is a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

A primary goal of software engineering is to improve the ease with which changes can be accommodated and reduce the amount of effort expanded when changes must be made.

The output of the software process is information that may be divided into three broad categories: (1) computer programs (both source level and executable forms); (2) documents that describes the computer programs (targeted at both technical practioners and users), and (3) data (contained within the program or external to it). The items that comprise all information produced as part of the software process are collectively called a *software configuration.*

Baselines: A Baseline is a software configuration management concept that helps us to control change without seriously impeding justifiable change. The IEEE defines a baseline as:

A specification or product that been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

One way to describe a baseline is through analogy:

A baseline is analogous to the kitchen doors in the restaurant. Before a software configuration item becomes a baseline, change may be made quickly and informally. However, once a baseline is established, we figuratively pass through a swinging one-way door. Changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.

In the context of software engineering, a baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCI that is obtained through a formal technique review.

For example, the elements of a design specification have been documented and reviewed, corrected and then approved; the design specification becomes a baseline

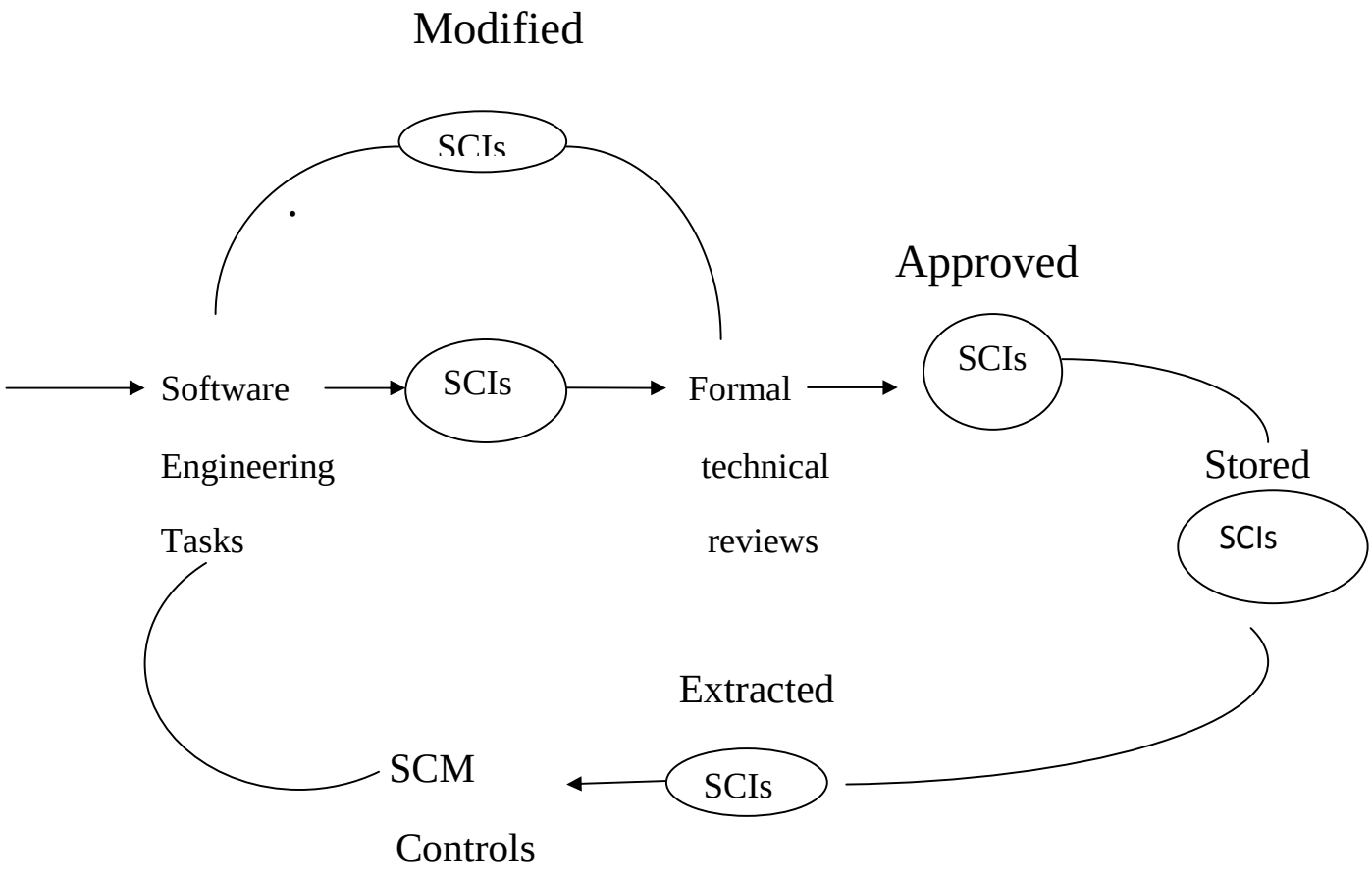


Fig Baseline SCI and the project database

SCM process: Software configuration management is an important element of software quality assurance. Its primary

responsibility is the control of change. However, SCM is also responsible for the identification of individual SCI and various versions of the software, the auditing of the software configuration to ensure that it has been properly developed, and the reporting of all changes applied to the configuration.

Any discussion of SCM introduces a set of complex questions:

- How does an organization identify and manage the many existing versions of a program in a manner that will enable change to be accommodated efficiently?
- How an organization control changes does before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?
- How can w ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made

SCM tasks: There are five types of **SCM tasks**:

1. Identification.
2. Version control
3. Change control.
4. Configuration auditing.
5. Reporting

CMM levels: The grading scheme determines compliance with a capability maturity model (CMM) that defines key

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

LEVEL	CHARACTERISTIC
5. Optimizing	Improvement fed back in to process
4. Managed	Measured Process (Quantitative)
3. Defined	Process defined and institutionalized (Qualitative)
2. Repeatable	Process dependent on individuals
1. Initial	Ad hoc or chaotic process

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

Level2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the

organization's process for developing and supporting software. This level includes all characteristics defined for level 2

Level4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level5: Optimizing. Continuous process improvement is enabled by quantitative feedback form the process and form testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

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

The SEI has associated key process areas (KPA) with each of the maturity levels. The KPA describe those software engineering functions that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

- *Goals*- the overall objectives that the KPA must achieve.
- *Commitments*- requirements that must be met to achieve the goals or provide proof of intent to comply with the goals.

- *Abilities*-those things that must be in place to enable the organization to meet the commitments.
- *Activities*- the specific tasks required to achieve the KPA function.
- *Methods for monitoring implementation*- the manner in which the activities are monitored as they are put into place.
- *Methods for verifying implementation*- the manner in which the proper practice for the KPA can be verified.

Risk Management: A *risk* is anything that could cause a project to fail to meet one or more of its goals. Risk management is the area that tries to ensure that the impact of risks on cost, quality and schedule is minimal.

Risk management can be considered as dealing with the possibility and actual occurrence of those events that are not "regular" or commonly expected, that is, they are probabilistic.

The idea of risk management is to minimize the possibility of risks materializing, if possible, or to minimize the effects if risks actually materialize. For example, when constructing a building, there is a risk that the building may later collapse due to an earthquake. That is, the possibility of an earthquake is a risk. If the building is a large residential complex, then the potential cost in case the earthquake risk materializes can be enormous. This risk can be reduced by shifting to a zone that is not an earthquake prone. Alternatively, if this is not acceptable, then

the effects of the risk materializes are minimized by suitably constructing the building. At the same time, if a small dumping ground is to be constructed, no such approach might be followed, as the financial and other impact of an actual earthquake on such a building is so low that it does not warrant special measures.

It should be clear that risk management has to deal with identifying the undesirable events that can occur, the probability of their occurring, and the loss if an undesirable event does occur. Once this is known, strategies can be formulated for either reducing the probability of the risk materializing or reducing the effect of risk materializing. So the risk management revolves around risk assessment and risk control. For each of these major activities, some sub activities must be performed.

Ideally in risk management, a risk prioritization process is followed in which those risks that pose the threat of great loss and have great probability of occurrence are dealt with first. Refer to table
below

IMPACT	ACTIONS		
SIGNIFICANT	Considerable Management Required	Must Manage and Monitor Risks	Extensive Management essential
MODERATE	Risk are bearable to certain extent	Management effort worthwhile	Management effort required
MINOR	Accept Risks	Accept but monitor Risks	Manage and Monitor Risks
	LOW	MEDIUM	HIGH
	LIKELIHOOD		

:

The above chart can be used to strategize in various situations. The two factors that govern the action required are the probability of occurrence and the impact of the risk. For example a condition where the impact is minor and the probability of occurrence is low, it is better to accept the risk without any interventions. A condition where the likelihood is high and the impact is significant, extensive management is required. This is how a certain priority can be established in dealing with the risk.

Risk Source: The source can be either internal or external to the system. External sources are beyond control whereas internal sources can be controlled to a certain extent. For example, the amount of rainfall, weather over an airport etc!

Problem: A problem at the surface level could be the threat of accident and casualty at the plant, a fire incident etc.

When any or both of the above two are known beforehand, certain steps can be taken to deal with the same.

After the risk/s has been identified then it/they must be assessed on the potential of criticality. Here we arrive upon risk prioritization. In generic terms 'likelihood of occurrence × impact' is equal to risk.

This is followed by development of a risk management plan and implementation of the same. It comprises of the effective security controls and control mechanisms for mitigation of risk.

Change control board: In software development, a **Change Control Board** (CCB) or **Software Change Control Board** (SCCB) is a committee that makes decisions regarding whether or not proposed changes to a software project should be implemented. The change control board is constituted of project stakeholders or their representatives. The authority of the change control board may vary from project to project, but decisions reached by the change control board are often accepted as final and binding.

Project control:

Unit-5

Project testing and Project success

White-box testing: White-box testing sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Using White-box testing methods, the software engineer can derive test cases that

1) Guarantee that all independent paths within a module have been exercised at least once,
2) Exercise all logical decisions on their true and false sides,
3) Execute all loops at their boundaries and within their operational bounds,
4) Exercise internal data structures to ensure their validity.

A reasonable question might be posed at this juncture: "why spend time and energy worrying about (and testing) logical minutiae when we might better expend effort ensuring that program requirements have been met?" Stated another way, why don't we spend all of our energy on black-box tests? The answers lie in the nature of software defects:

- *Logic errors and incorrect assumptions are inversely proportional to the probability that a program path will be*

executed. Errors tend to creep into our work when we design and implement function, conditions, or control that is out of the mainstream. Everyday processing tends to be well understood, while "special case" processing tends to fall into the cracks.

- *We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis.* The logical flow of a program is sometimes counterintuitive, meaning that our unconscious assumptions about flow of control and data may lead us to make design errors that are uncovered only once path testing commences.
- *Typographical errors are random.* When a program is translated into programming language source code, it is likely that some typing errors will occur. Many will be uncovered by syntax and type checking mechanisms, but others may go undetected until testing begins. It is as likely that a typo will exist on an obscure logical path as on a mainstream path.

Each of these reasons provides an argument for conducting white-box tests. White box testing is far more likely to uncover them.

Black-Box Testing: Black-box testing also called behavioral testing, focuses on the functional requirements of the software. That is, Black-box testing enables the software engineer to

derive sets of input conditions that will fully exercise all functional requirements of a program.

Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external data base access, (4) behavior or performance errors, and (5) initialization and termination errors.

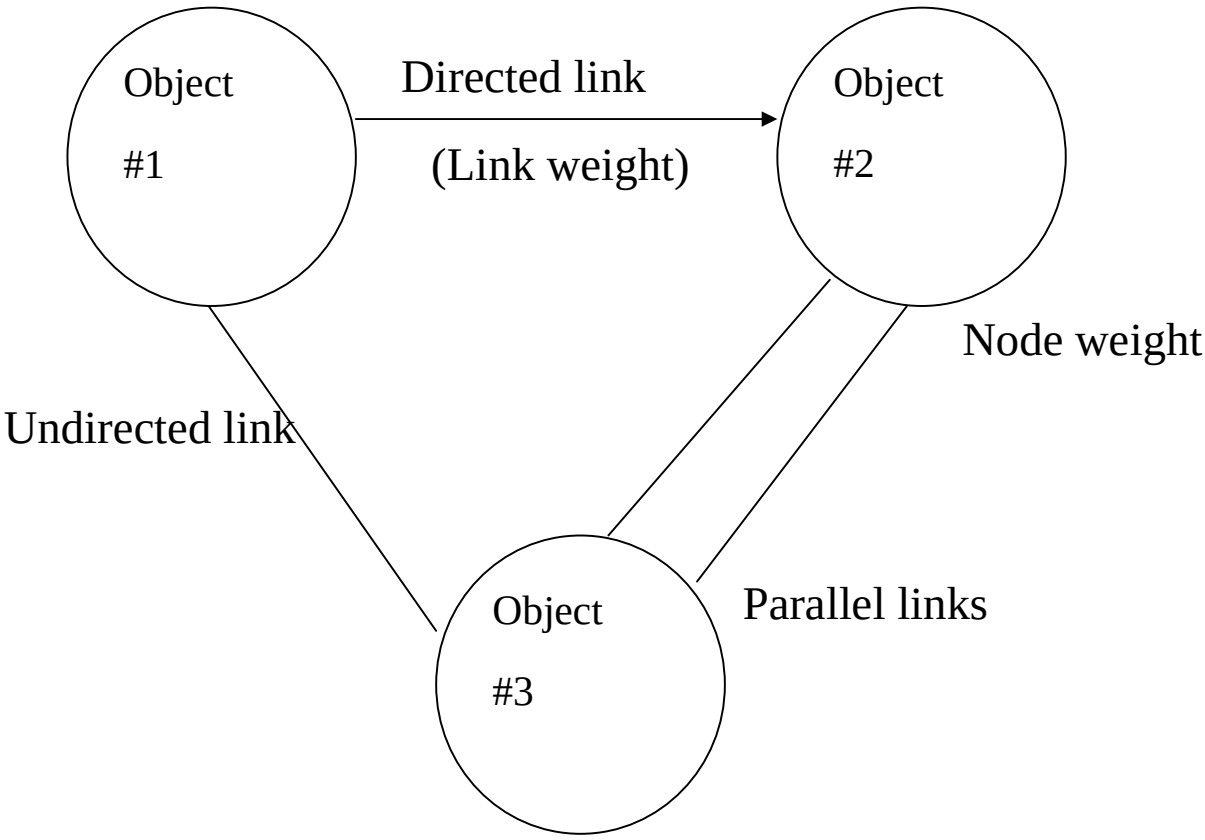
Black-box testing purposely disregards control structure, attention is focused on the information domain. Tests are designed to answer the following questions:

- How is functional validity tested?
- How is system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

Graph Based Testing Methods. The first step in Black-box testing is to understand the objects that are modeled in software and the relationship that connect these objects. Once this has been accomplished, the next step is to define a series of tests that verify “all objects have the expected relationship to one another”. Stated in another way, software testing begins by creating a graph of important objects and their relationships and

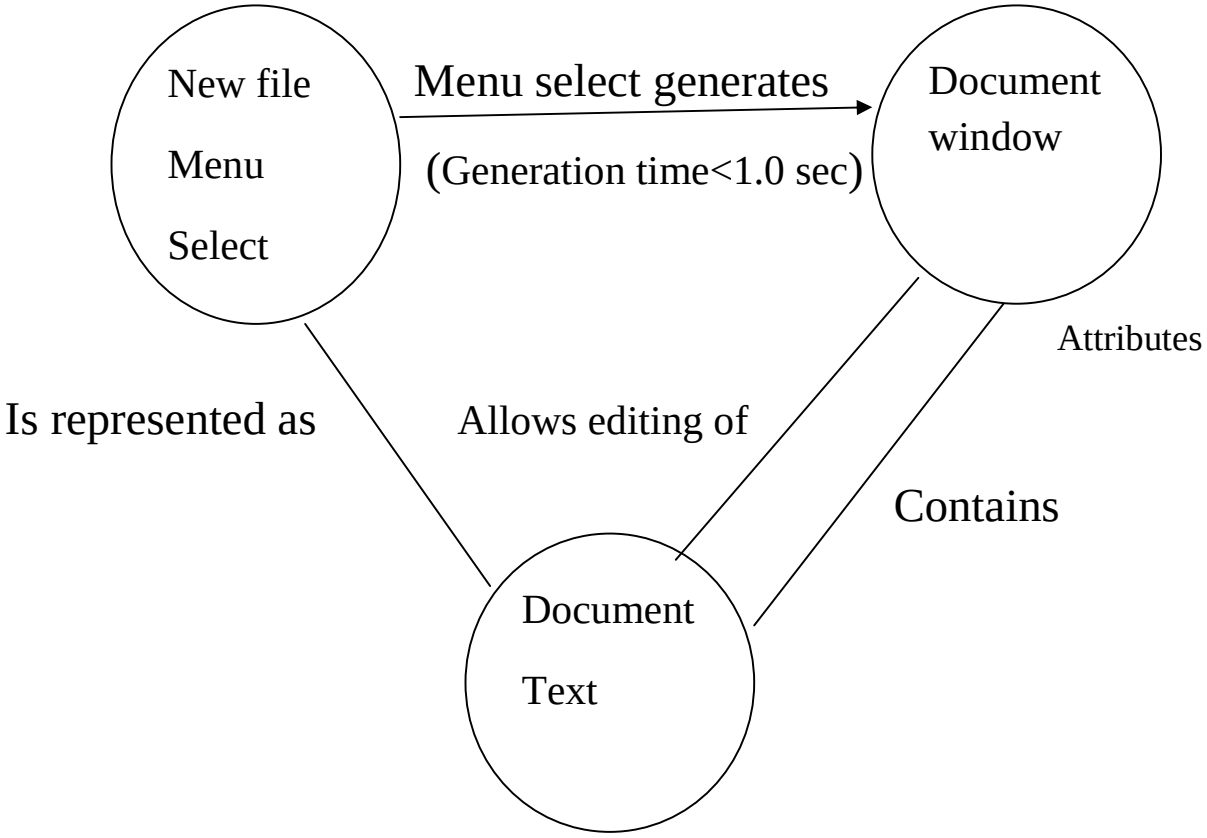
then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

To accomplish these steps, the software engineer begins by creating a graph—a collection of nodes that represent objects; links that represent the relationships between objects; node weights that describe the properties of a node and link weights that describe some characteristic of a link.



(A) Graph notation

The symbolic representation of a graph is shown in fig A. Nodes are represented as circles connected by links that take a number of different forms. A directed link, also called a symmetric link, implies that the relationships are established between graph nodes.



(B) Simple notation

Equivalence Partitioning. Equivalence partitioning is a black-box testing method that divides the input domain of a program

into classes of errors that might otherwise require many cases to be executed before the general error is observed.

Equivalence partitioning strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed. As an example, consider data maintained as part of an automated banking application. The user can access the bank using a personal computer, provide a six-digit password, and follow with a series of typed commands that trigger various banking functions. During the log-on sequence, the software supplied for the banking application accepts data in the form

Area code-blank or three-digit number

Prefix-three-digit number not beginning with 0 or 1

Suffix-four digit number

Password-six digit alphanumeric string

Commands-check, deposit, bill pay, and the like

Applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed. Test cases are selected so that the largest number of attributes of an equivalence class is exercised at once.

Unit-Testing: Unit testing focuses verification effort on the smallest unit of software design-the software component or

module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module.

Unit-Test Considerations. The tests that occur as part of unit tests are illustrated schematically in fig1. The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. And finally, all error handling paths are tested

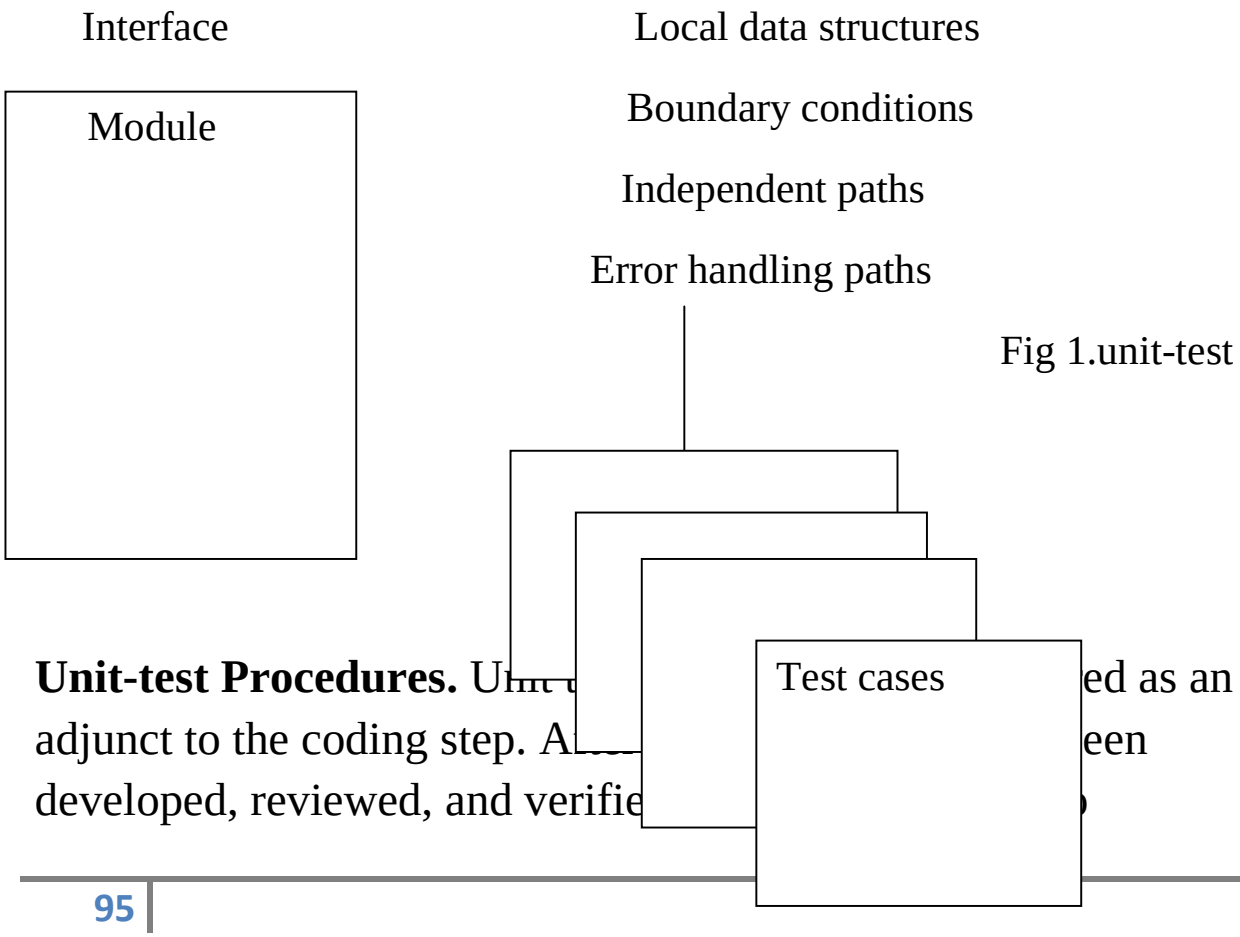


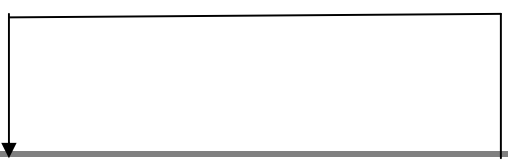
Fig 1.unit-test

Unit-test Procedures. Unit t ed as an adjunct to the coding step. A… een developed, reviewed, and verifie

component level design, unit test case design begins. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories. Each test case should be coupled with a set of expected results.

Because a component is not a stand-alone program, driver and/or stub software must be developed for each unit test. The unit test environment is illustrated in figure. In most applications a driver is nothing more than a main program that accepts test case data, passes such data to the component and prints relevant results.

Drivers and stubs represent overhead. That is, both are software that must be written but there is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with “simple” overhead software. In such cases, complete testing can be postponed until the integration test step.



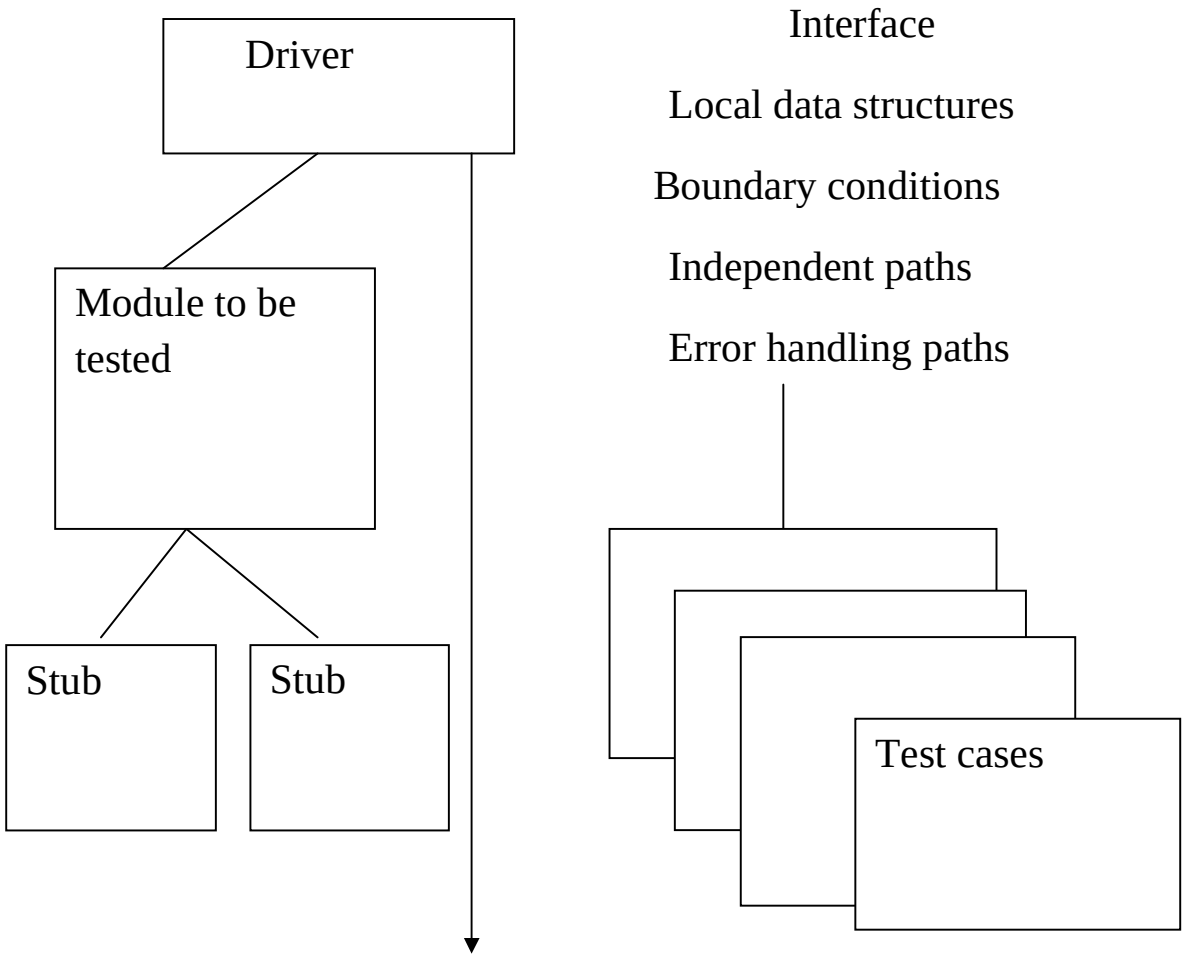


Fig2. Unit-test environment

Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.

Integration Testing: Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with

interfacing. The objective is to take unit tested components and build a program structures that has been dictated by design.

Integration testing is of 6 types which are as follows:

1. Top-down Integration
2. Bottom-up Integration
3. Regression testing
4. Smoke testing
5. Comments on Integration testing
6. Integration test documents

Top-down Integration. Top-down Integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Modules subordinate to the main control module are incorporated into the structure in either a depth-first or Breadth-first manner

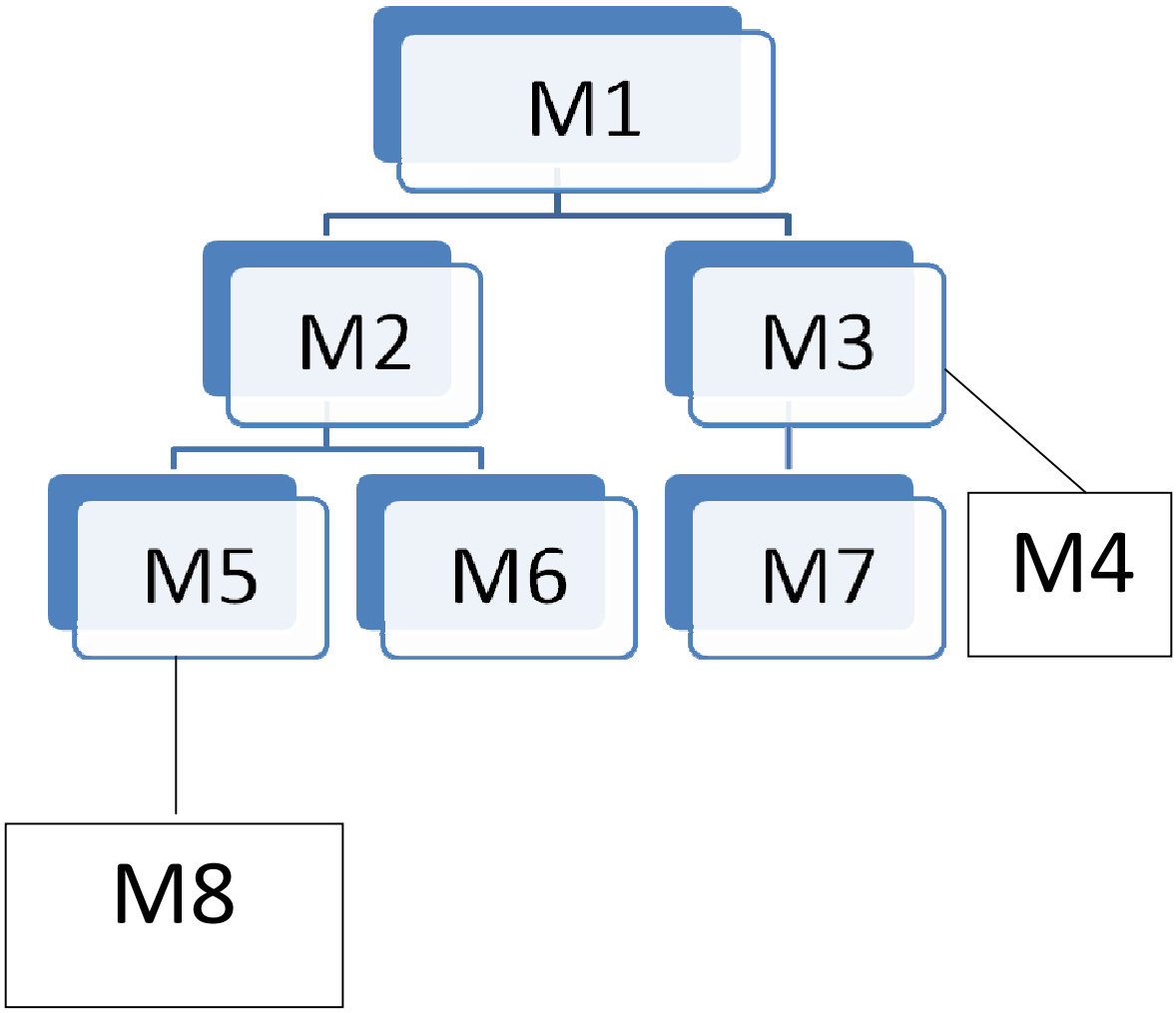


Fig Top down Integration

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

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

The top-down integration strategy verifies major control or decision points early in the test process. In a well-factored program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated.

For example, consider a classic transaction structure in which a complex series of interactive inputs is requested, acquired, and validated via an incoming path. The incoming path may be integrated in a top-down manner. All input processing may be demonstrated before other elements of the structure have been integrated. Early demonstration of functional capability is a confidence builder for both the developer and the customer.

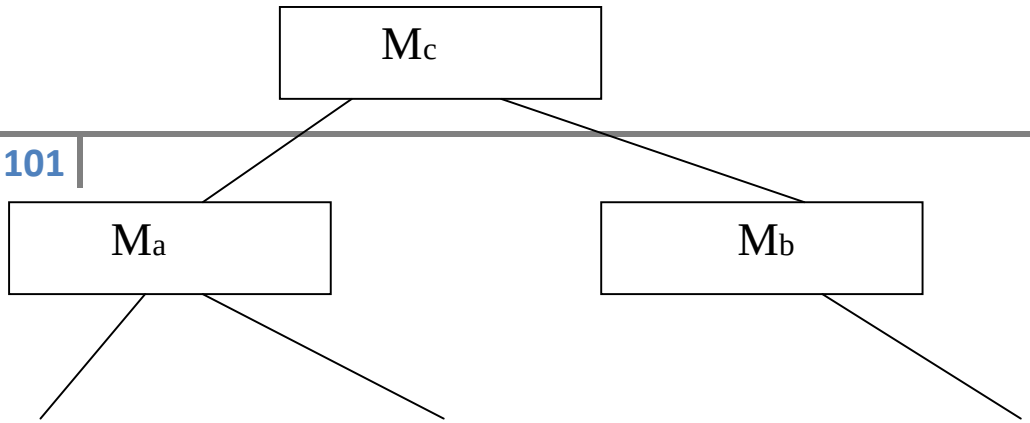
Bottom-up Integration. Bottom-up Integration testing, as it name implies, begins construction and testing with atomic modules. Because components are integrated from the bottom

up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

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

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

Integration follows the pattern illustrated in fig. Components are combined to from clusters 1, 2, and 3. Each of the clusters is tested using a driver. Components in clusters 1 and 2 are subordinate to M_a. Drivers D_1 and D_2 are removed and the clusters are interfaced directly to M_a. Similarly driver D_3 for cluster 3 is removed prior to integration with module M_b .Both M_a. And M_b will ultimately be integrated with component M_c, and so forth.



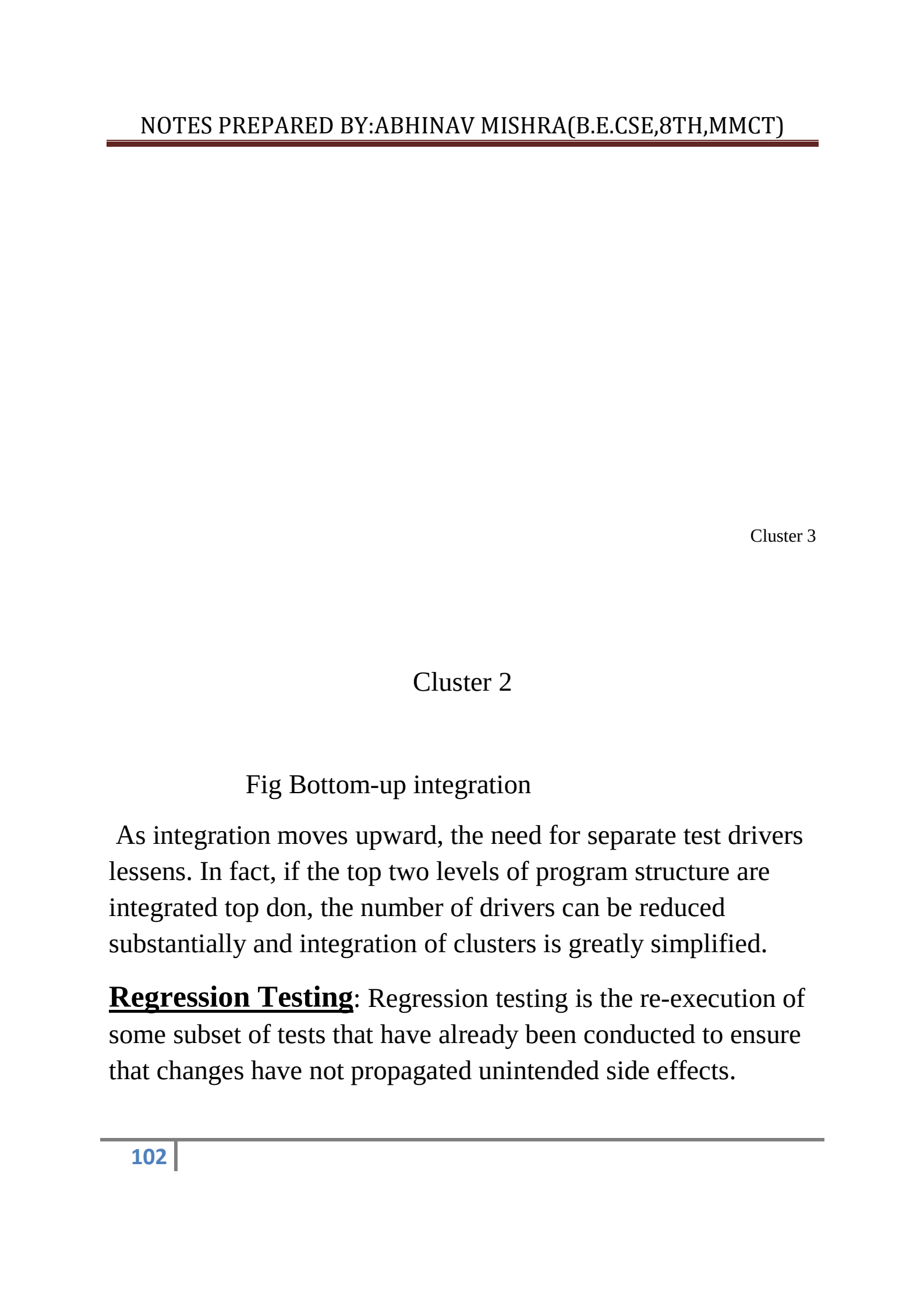


Fig Bottom-up integration

As integration moves upward, the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top don, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

Regression Testing: Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

In a broader context successful tests result in the discovery of errors, and errors must be corrected. Whenever software is corrected, some aspect of the software configuration is changed. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors.

Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools. Capture/playback tools enable the software engineer to capture test cases and results for subsequent playback and comparison.

The Regression test contains three different classes of test cases:

1. A representative sample of tests that will exercise all software functions.
2. Additional tests that focus on software functions that are likely to be affected by the change.
3. Tests that focus on the software components that have been changed.

As integration testing proceeds, the number of Regression tests can grow quite large. Therefore, the Regression test suite should be designed to include only those tests that address one or more classes of errors in each of the major program functions. It is impractical and inefficient to re-execute every test for every program function once a change has occurred.

System Testing: System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test have been properly integrated and perform allocated functions. In this we discuss the types of system tests that are worthwhile for software –based systems.

1. Recovery testing.

2. Security testing.

3. Stress testing.

4. Performance testing.

1. *Recovery testing*: Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that Recovery is properly performed. If Recovery is automatic, reinitialization, check pointing mechanisms, data recovery and restart are evaluated for correctness. If Recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

2. *Security testing*: Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. "The system's security must, of course, be tested for invulnerability from frontal attack-but must also be tested for invulnerability from flank or rear attack".

Stress testing: Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that

generate ten interrupts per second when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond. (3) Test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed. (5) Test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.

A variation of stress testing is a technique called stress testing. In some situations, a very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation. Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

Testing: Testing presents an interesting anomaly for the software engineer. During earlier software engineering activities, the engineer attempts to build software from an abstract concept to a tangible product. The Engineer creates a series of test cases that are intended to “demolish” the software that has been built. In fact testing is the one step in the software process that could be viewed as destructive rather than their own constructive.

Testing Objectives: Testing objectives are as follows:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as-yet-undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error.

Testing Principles: Testing principles should be explained in the following points:

- **All tests should be traceable to customer requirements.** The objective of software testing is to uncover errors. It follows that the most severe defects are those that cause the program to fail to meet its requirements.
- **Test should be planned long before testing begins.** Test planning can begin as soon as the requirements model is complete. Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.
- **The Pareto principle applies to software testing.** Stated simply, the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem of course, is to isolate these suspect components and to thoroughly test them.
- **Testing should begin "in the small" and progress toward testing "in the large".** The first tests planned and

executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

- **Exhausting Testing is not possible**. The number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing.

Validation: Validation involves checking that the program meets the exceptions of users and customers. Validation is the set of activities that ensures the software has been built according to the customer's requirements.

Verification- Verification refers to the set of activities that ensure the software correctly implements specific function. Verification involves checking that a program conforms to its specification.

Quality Assurance: Quality assurance is an essential activity for any business that produce product to be used by others. The first formal quality assurance and control function was introduced at Bell labs in 1916 and spread rapidly throughout the manufacturing world. During the $1940_{s,}$ more formal approaches to quality control were suggested. These relied on measurement and continuous process improvement as key elements of quality management.

Today, every company has mechanisms to ensure quality in its products. In fact explicit statements of a company's concern for quality have become a marketing ploy during the past few decades.

The history of quality assurance in software development parallels the history of quality in hardware manufacturing. During the early days of computing quality was the sole responsibility of the programmer. Standards for quality assurances for software were introduced in military contact software development during the 1970_s and have spread rapidly into software development in the commercial world.

SQA activities: software Quality assurance is composed of a variety of tasks associated with two different constituencies- the software engineers who do technical work and an SQA group that has responsibility for Quality assurance planning, oversight, record keeping, analysis and reporting.

The charter of the SQA group is to assist the software team in achieving a high quality end product. These activities are performed by an independent SQA group that:

1. **Prepares an SQA plan for a Project**: The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies:

- Evaluations to be performed
- Audits and reviews to be performed
- Standards that are applicable to the project.
- Procedures for error reporting and tracking
- Documents to be produced by the SQA group
- Amount of feedback provided to the software project team

2. **Participates in the development of the project's software process description.** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards and other parts of the software project plan.
3. **Reviews software engineering activities to verify compliance with the defined software process.** The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as part of the software process.** The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the result of its work to the project manager.
5. **Ensures that deviations in software work and work products are documented and handle according to a**

documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

6. **Records any non compliance and reports to senior management.** Non compliance items are tracked until they are resolved.

In addition to these activities, the SQA group coordinates the control and management of change and helps to collect and analyze software metrics.

Decision making: **Decision making** can be regarded as the mental processes (cognitive process) resulting in the selection of a course of action among several alternatives. Every decision making process produces a final choice. The output can be an action or an opinion of choice.

Overview

Human performance in decision-making terms has been the subject of active research from several perspectives. From a psychological perspective, it is necessary to examine individual decisions in the context of a set of needs, preferences an individual has and values they seek. From a cognitive perspective, the decision making process must be regarded as a continuous process integrated in the interaction with the environment. From a normative perspective, the analysis of individual decisions is concerned with the logic of decision making and rationality and the invariant choice it leads to.[2]

Yet, at another level, it might be regarded as a problem solving activity which is terminated when a satisfactory solution is found. Therefore, decision making is a reasoning or emotional process which can be rational or irrational, can be based on explicit assumptions or tacit assumptions.

Logical decision making is an important part of all science-based professions, where specialists apply their knowledge in a given area to making informed decisions. For example, medical decision making often involves making a diagnosis and selecting an appropriate treatment. Some research using naturalistic methods shows, however, that in situations with higher time pressure, higher stakes, or increased ambiguities, experts use intuitive decision making rather than structured approaches, following a recognition primed decision approach to fit a set of indicators into the expert's experience and immediately arrive at a satisfactory course of action without weighing alternatives. Recent robust decision efforts have formally integrated uncertainty into the decision making process. However, Decision Analysis, recognized and included uncertainties with a structured and rationally justifiable method of decision making since its conception in 1964.

Characteristics:

Objectives must first be established
• Objectives must be classified and placed in order of importance
• Alternative actions must be developed

• The alternative must be evaluated against all the objectives
• The alternative that is able to achieve all the objectives is the tentative decision
• The tentative decision is evaluated for more possible consequences
• The decisive actions are taken, and additional actions are taken to prevent any adverse consequences from becoming problems and starting both systems (problem analysis and decision making) all over again