

UNIT I

PROJECT

A general definition of project is: “It is an temporary endeavor with set of well-defined activities that leads achievement of a specific goal(s)”. A Project has following characteristics:

- Project has specific goal(s)
- It has a definite start date and end date
- It is not group of routine tasks or daily activities rather involves planned activities
- Unlike routine activities, project comes to end when its goal(s) is achieved
- Every project requires enough resources in terms of time, skilled workforce, budget, material and other support

What is management?



This involves the following activities:

- ▶ **Planning** – deciding what is to be done
- ▶ **Organizing** – making arrangements
- ▶ **Staffing** – selecting the right people for the job
- ▶ **Directing** – giving instructions
- ▶ **Monitoring** – checking on progress
- ▶ **Controlling** – taking action to remedy hold-ups
- ▶ **Innovating** – coming up with solutions when problems emerge
- ▶ **Representing** – liaising with clients, users, developers and other stakeholders

What is Software Project?

A Software Project can be considered as a subset of general Project. It involves the process managing software life-cycle right from software requirement gathering, designing, to testing and maintenance, carried out according to a given project management methodologies, in a stipulated time frame to achieve intended software product/service delivery.

What is Software Project Management

“Software Project Management is the art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled.”

Why Is Software Project Management Required?

Unlike machines or buildings, software does not have a physical form or it is not a tangible product. Today organizations are using software to drive business processes. One can imagine the complexity involved in mapping business process to a software. Also business process for one organization can not be same for other, it means requirement of a software for one organization will be different from other. Given the rapid changes in the technology platform as well as globalized but integrated economies induce element of risks in the software already developed or under development. Hence to reduce the risk factor and ensure project delivery will meet stakeholder’s expectations, there is a need to follow structured, process based approach; which is nothing but software project management.

ACTIVITIES COVERED BY SOFTWARE PROJECT MANAGEMENT

1. Feasibility Study: Feasibility study is need to determine that project is worth(useful) starting. The development, operational cost, benefits are estimated.

2. Planning: Once feasibility study is done, then project planning can be started. For Larger project it is not possible to do all the planning in the beginning. Planning is done for different stages.

3. Project Execution: Project execution include design and implementation.

4. Requirement Analysis: which investigates what the potential(work done) users and their managers and employers require as features and qualities of the new system.

5. Architecture Design: This maps the requirements to the components of the system that is to be built.(system architecture, software requirements, software components).

6. Detailed Design:Each software component is made up of a number of software units that can be separately coded and tested. Design of these units is carried out sepetately.

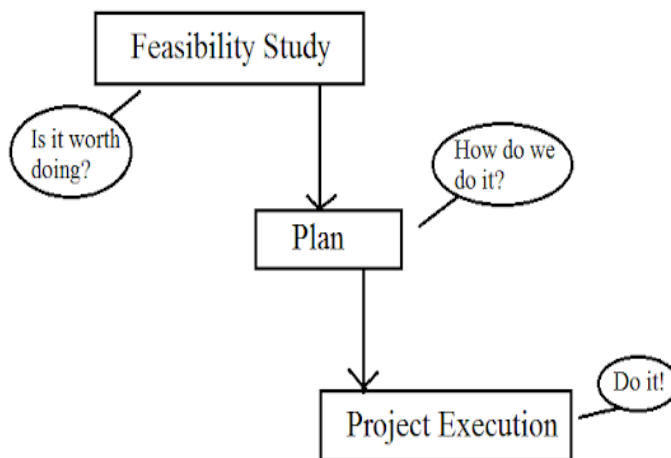
7. Code and Test:This could refer to writing code for each software unit in a procedural language such as java,c,python etc,Application builder such as Microsoft access.

8. Integration: Integration could be at the level of software where different software components are combined. Such as the hardware platforms and networks and the user procedures are brought together.

9.Qualification Testing: The system, Including the software components, has to be tested carefully.(to fulfilled requirements)

10.Installation:This is the process of making the new system operational.(such as payroll details(salary))

11.Acceptance support: This is the resolving(solve) of problems with the newly installed system, including the correction of any error that might have crepts into the system and any extensions and improvements that are required.



CHALLENGES IN SOFTWARE PROJECTS

Extremely high competition:The competition is extremely high both at the local and international level and it affects software business in terms of pricing, customer reach and retention(store),etc..

Project managers have to work closely with business owners and other stakeholders to identify the right market segment.

Old legacy systems:Software companies often spend significant resources on maintaining and upgrading the old legacy systems.

Having invested a lot of financial and human resources, stakeholders become resistant and don't want to change the existing systems, even when it no longer meets their needs.

High-level software expertise: When it comes to software selection and implementation, the best variant for business owners is finding project managers with the relevant software expertise.

Third-party integration:Modern companies are no longer interested in standalone solutions and look for third party integration.In general,it looks like implementing multiple systems in one project.

Multiple level users: Most companies look for systems that allow different types of users- from basic users to strictly IT users.

Project managers who are responsible for the system implementation,must be familiar with all types of users.

Quality testing :Successful system implementation requires numerous` testing iterations to ensure that the final outcomes align with the desired results.

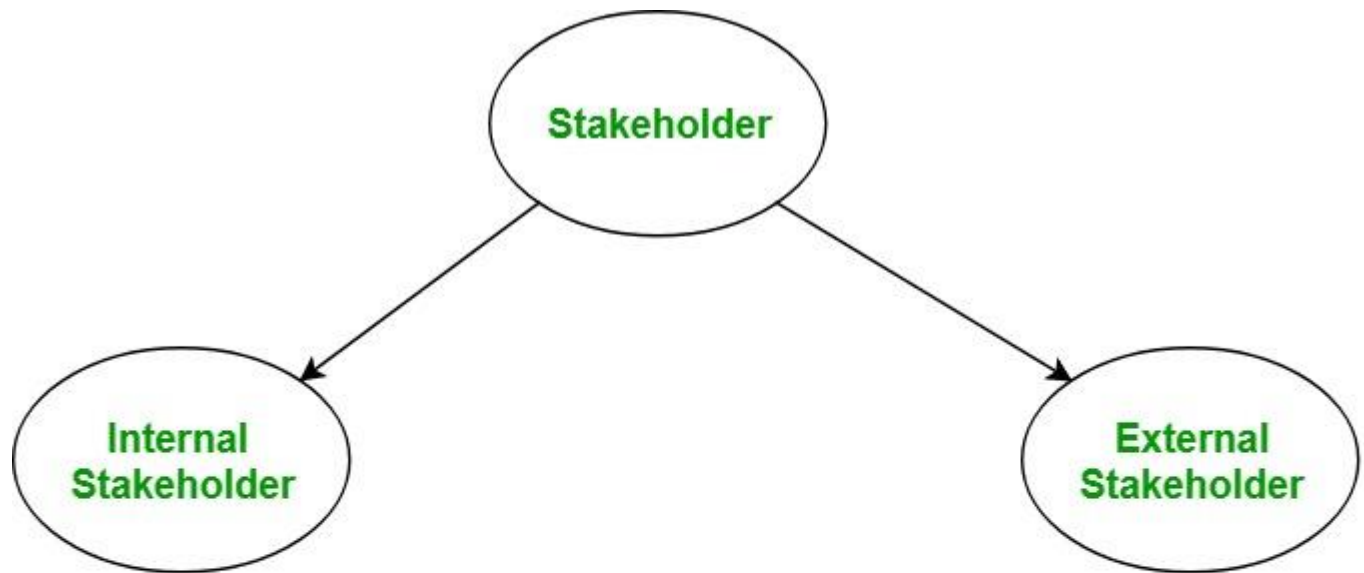
Project manager need to make sure all bugs are discovered and all issues are fixed before the system goes live.

STAKEHOLDERS

The term **Software Project Stakeholder** refers to, “a person, group or company that is directly or indirectly involved in the project and who may affect or get affected by the outcome of the project”.

What is Stakeholder Identification?

It is the process of identifying a person, group or a company which can affect or get affected by a decision, activity or the outcome of the software project. It is important in order to identify the exact requirements of the project and what various stakeholders are expecting from the project outcome.



* **Project Manager**

* **Project Team**

* **Company**

* **Funder**

* **Customer**

* **Government**

* **Supplier**

Type of Stakeholders:

1. Internal Stakeholder:

An internal stakeholder is a person, group or a company that is directly involved in the project.

For example,

Project Manager:

Responsible for managing the whole project. Project Manager is generally never involved in producing the end product but he/she controls, monitors and manages the activities involved in the production.

Project Team:

Performs the actual work of the project under the Project Manager including development, testing, etc.

Company:

Organisation who has taken up the project and whose employees are directly involved in the development of the project.

Funders:

Provides funds and resources for the successful completion of the project.

2. External Stakeholder:

An external stakeholder is the one who is linked indirectly to the project but has significant contribution in the successful completion of the project.

For example,

Customer:

Specifies the requirements of the project and helps in the elicitation process of the requirement gathering phase. Customer is the one for whom the project is being developed.

Supplier:

Supplies essential services and equipment for the project.

Government:

Makes policies which helps in better working of the organisation.

GOALS AND OBJECTIVES

Goals and objectives are statements that describe what the project will accomplish, or the business value the project will achieve.

Goals are high level statements that provide overall context for what the project is trying to achieve, and should align to business goals.

Objectives are lower level statements that describe the specific, tangible products and deliverables that the project will deliver.

The definition of goals and objectives is more of an art than a science, and it can be difficult to define them and align them correctly.

Goals

Goals are high-level statements that provide the overall context for what the project is trying to accomplish. Let's look at an example and some of the characteristics of a goal statement. One of the goals of a project might be to "increase the overall satisfaction levels for clients calling to the company helpdesk with support needs".

Because the goal is at a high-level, it may take more than one project to achieve. In the above example, for instance, there may be a technology component to increasing client satisfaction. There may also be new procedures, new training classes, reorganization of the helpdesk department and modification of the company rewards system. It may take many projects over a long period of time to achieve the goal.

The goal should reference the business benefit in terms of cost, speed and / or quality. In this example, the focus is on quality of service. Even if the project is not directly in support of the business, there should be an indirect tie. For instance, an IT infrastructure project to install new web servers may ultimately allow faster client response, better price performance, or other business benefit. If there is no business value to the project, the project should not be started.

Generally, non-measurable: If you can measure the achievement of your goal, it is probably at too low a level and is probably more of an objective.

If your goal is not achievable through any combination of projects, it is probably written at too high a level. In the above example, you could envision one or more projects that could end up achieving a higher level of client satisfaction. A goal statement that says you are trying to achieve a perfect client experience is not possible with any combination of projects. It may instead be a vision statement, which is a higher level statement showing direction and aspiration, but which may never actually be achieved.

It is important to understand business and project goal statements, even though goals are not a part of the TenStep Project Definition. Goals are most important from a business perspective. The project manager needs to understand the business goals that the project is trying to contribute to. However, you do not need to define specific project goals. On the other hand, objectives definitely are important.

Objectives

Objectives are concrete statements describing what the project is trying to achieve. The objective should be written at a lower level, so that it can be evaluated at the conclusion of a project to see whether it was achieved or not. Goal statements are designed to be vague. Objectives should not be vague. A well-worded objective will be Specific, Measurable, Attainable/Achievable, Realistic and Time-bound (SMART).

An example of an objective statement might be to “upgrade the helpdesk telephone system by December 31 to achieve average client wait times of no more than two minutes”.

Note that the objective is much more concrete and specific than the goal statement.

The objective is measurable in terms of the average client wait times the new phone system is trying to achieve.

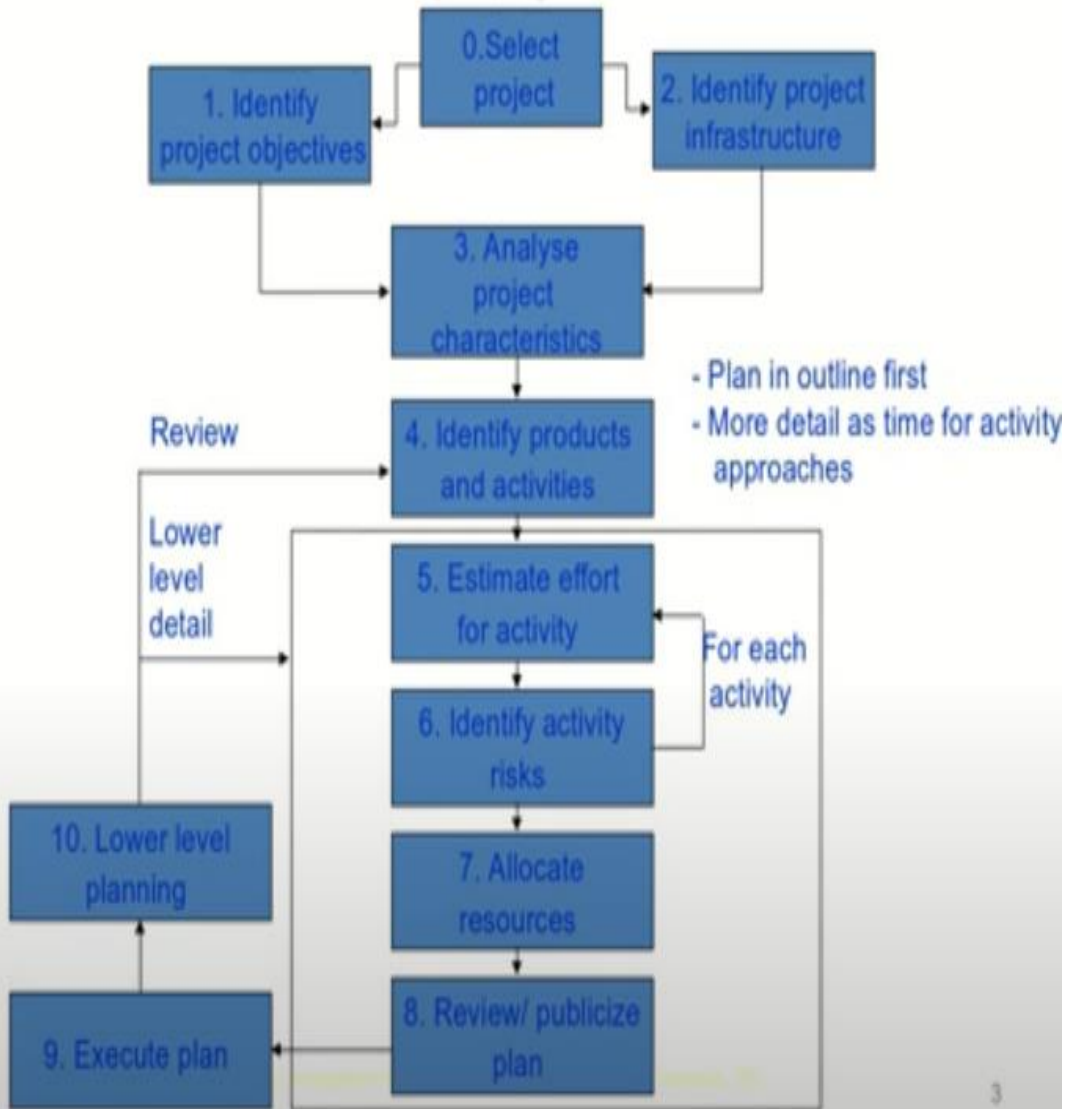
We must assume that the objective is achievable and realistic.

The objective is time-bound, and should be completed by December 31.

Objectives should refer to the deliverables of the project. In this case, it refers to the upgrade of the telephone system. If you cannot determine what deliverables are being created to achieve the objective, then the objective may be written at too high a level. On the other hand, if an objective describes the characteristics of the deliverables, they are written at too low a level. If they describe the features and functions, they are requirements, not objectives.

PROJECT PLANNING

main planning activities



Step	Activities within step
0	Select project
1	Identify project scope and objectives <ul style="list-style-type: none"> 1.1 Identify objectives and measures of effectiveness in meeting them 1.2 Establish a project authority 1.3 Identify stakeholders 1.4 Modify objectives in the light of stakeholder analysis 1.5 Establish methods of communication with all parties
2	Identify project infrastructure <ul style="list-style-type: none"> 2.1 Establish relationship between project and strategic planning 2.2 Identify installation standards and procedures 2.3 Identify project team organization
3	Analyse project characteristics <ul style="list-style-type: none"> 3.1 Distinguish the project as either objective- or product-driven 3.2 Analyse other project characteristics 3.3 Identify high-level project risks 3.4 Take into account user requirements concerning implementation 3.5 Select general life-cycle approach 3.6 Review overall resource estimates
4	Identify project products and activities <ul style="list-style-type: none"> 4.1 Identify and describe project products (including quality criteria) 4.2 Document generic product flows 4.3 Recognize product instances 4.4 Produce ideal activity network 4.5 Modify ideal to take into account need for stages and checkpoints
5	Estimate effort for each activity <ul style="list-style-type: none"> 5.1 Carry out bottom-up estimates 5.2 Revise plan to create controllable activities
6	Identify activity risks <ul style="list-style-type: none"> 6.1 Identify and quantify activity-based risks 6.2 Plan risk reduction and contingency measures where appropriate 6.3 Adjust plans and estimates to take account of risks
7	Allocate resources <ul style="list-style-type: none"> 7.1 Identify and allocate resources 7.2 Revise plans and estimates to take account of resource constraints
8	Review/publicize plan

Scope of Project:

Project scope defines the concept and range of the proposed solution, and limitations identify certain capabilities that the product will not include. Clarifying the scope and limitations helps to establish realistic stakeholder's expectations. Propose requirements that are out of scope must be rejected. Keep a record of these requirements and why they were rejected, as they have a way of reappearing.

PROJECT SCOPE: Project scope describes what work should be performed to meet all those requirements.

Narrative: It is used as a written confirmation of what your project is going to produce and how what's the key is useful projects scope statement.

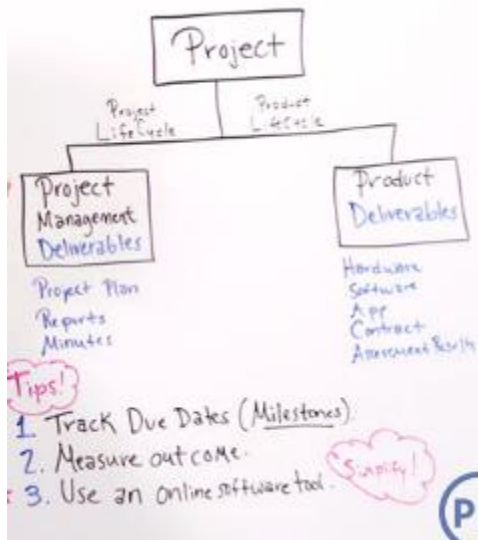
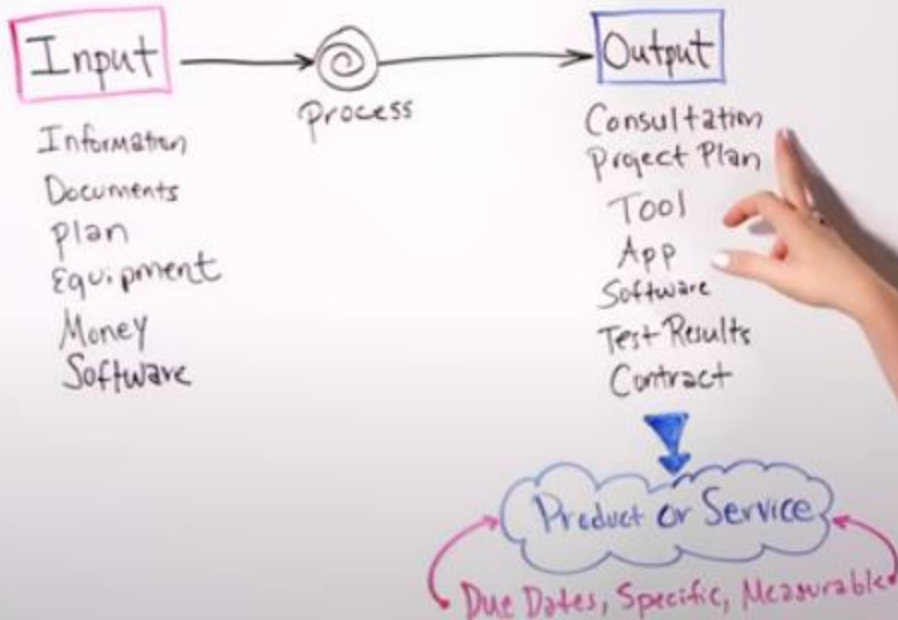
REQUIREMENTS: Requirements is condition or capability that is required to be present in a product, service or result to satisfy a contract or other formally imposed specifications.

Deliverables: Today we are talking about what our project deliverables. We here that term all the time on projects so whether you are the project manager asking your team members what the status of their deliverables are or the team member with your project manager asking you for the status or may be even you asking you other team members what status of their deliverables .

Delivarbles: something produced or provided as result of a process.

Process: so if we look at the process we get inputs and even the input into the process could be deliverables it could be taking some form of information, some type of documents from project, a plan, maybe even some equipment or money, even software that's not all the different types but these are examples of a input you could take into the process and outputs you may get something like a consultation, you may get a project plan, some type of tool or appor software test results or may be even a contract. if you look these again these are just examples but in a project you are producing deliverables that are either a product or service and with those you want to be sure that you get a due date for each deliverables make sure that it is specific or measurable.

Deliverable = Something Produced or Provided as a result of a process.



2.6 Identify project products and activities

The following activities are:

➤ *Identify and describe project products (or deliverables)*

The products will form the hierarchy. The main product will have sets of component products which in turn may have sub-component products and so on. This relationship can be documented in a Product Breakdown Structure (PBS). It is shown as in the figure.

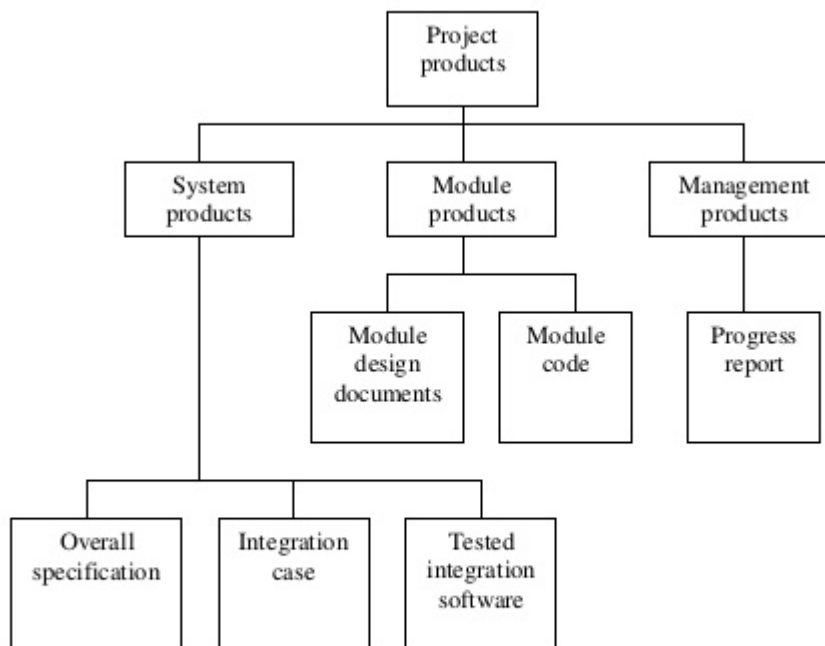


Fig. A framework of a product breakdown structure for a system development task

➤ *Document generic product flows*

Some products will need one or more other products to exist first before they can be created. For example, a program design must be created before the program can be written and the program specification must exist before the design can be concerned. These relationships can be shown by the Product Flow Diagram (PFD). It is shown as in figure.

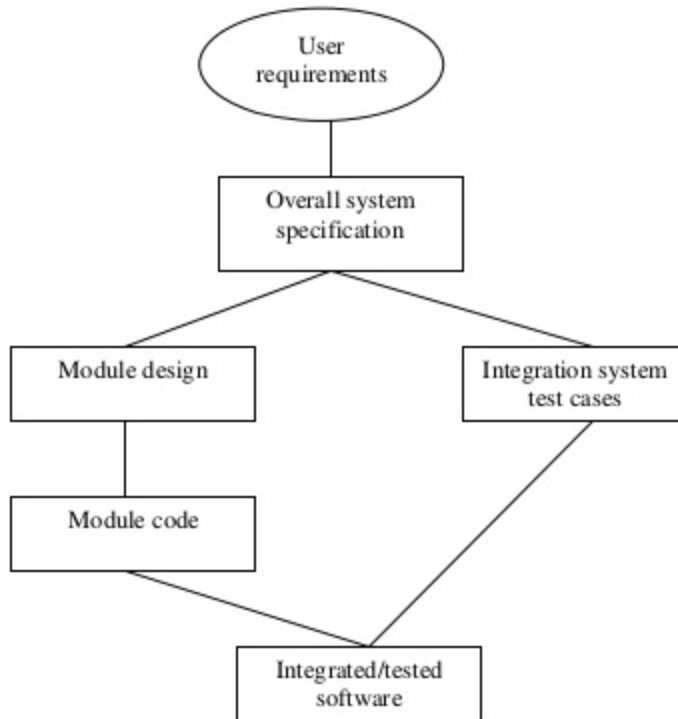


Fig. A framework of a PDF for a software development task

➤ *Recognize product instances*

There may be delayed to later in the product when more information is known.

➤ *Produce ideal activity network*

It is explained by an example of activities network.

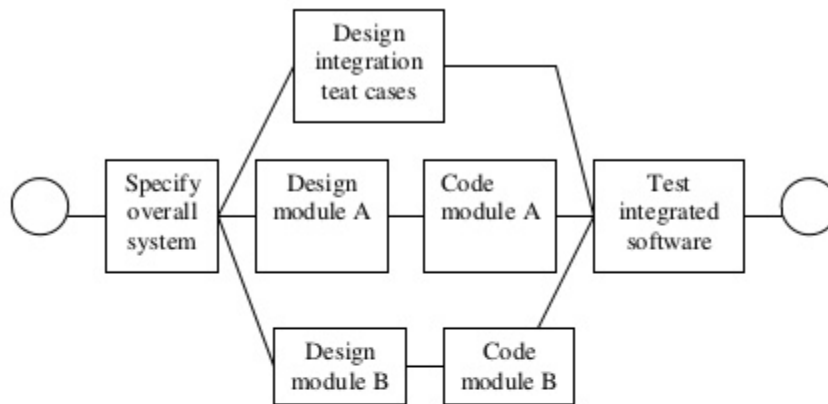


Fig. An example of an activity network

- *Modify the ideal to take into account need for stages and checkpoints*

There might be a need to modify this by dividing the project into stages and introduces checkpoint activities. These are activities which draw together the products activities to check that they are compatible.

There could be some key attributes, or milestones, which represent the completion of important stages of the project of which they would want to take particular note.

2.7 Estimate effort for each activity

The following activities are:

- *Carry out bottom-up estimates*

At this point, estimates of the staff effort required, the probable elapsed time and the non-staff resources needed for each activity will need to be produced.

Effort is the amount of work that needs to be done.

Elapsed time is the time between the start and end of a task.

- *Revise plan to create controllable activities*

- Try to make activities about the length of the reporting period used for monitoring and controlling the project.

UNIT II

Software Development Life Cycle Models and Methodologies

Introduction

Software development life cycle (**SDLC**) is a series of phases that provide a common understanding of the software building process. How the software will be realized and developed from the business understanding and requirements elicitation phase to convert these business ideas and requirements into functions and features until its usage and operation to achieve the business needs. The good software engineer should have enough knowledge on how to choose the SDLC model based on the project context and the business requirements.

Therefore, it may be required to choose the right SDLC model according to the specific concerns and requirements of the project to ensure its success. I wrote another article on how to choose the right SDLC, you can follow this [link](#) for more information. Moreover, to learn more about [Software Testing life cycles](#) and [SDLC phases](#) you follow the links highlighted here. In this article, we will explore the different types of SDLC models and the advantages and disadvantages of each one and when to use them.

You can think of SDLC models as tools that you can use to better deliver your software project. Therefore, knowing and understanding each model and when to use it, the advantages and disadvantages of each one are important to know which one is suitable for the project context.

Types of Software developing life cycles (SDLC)

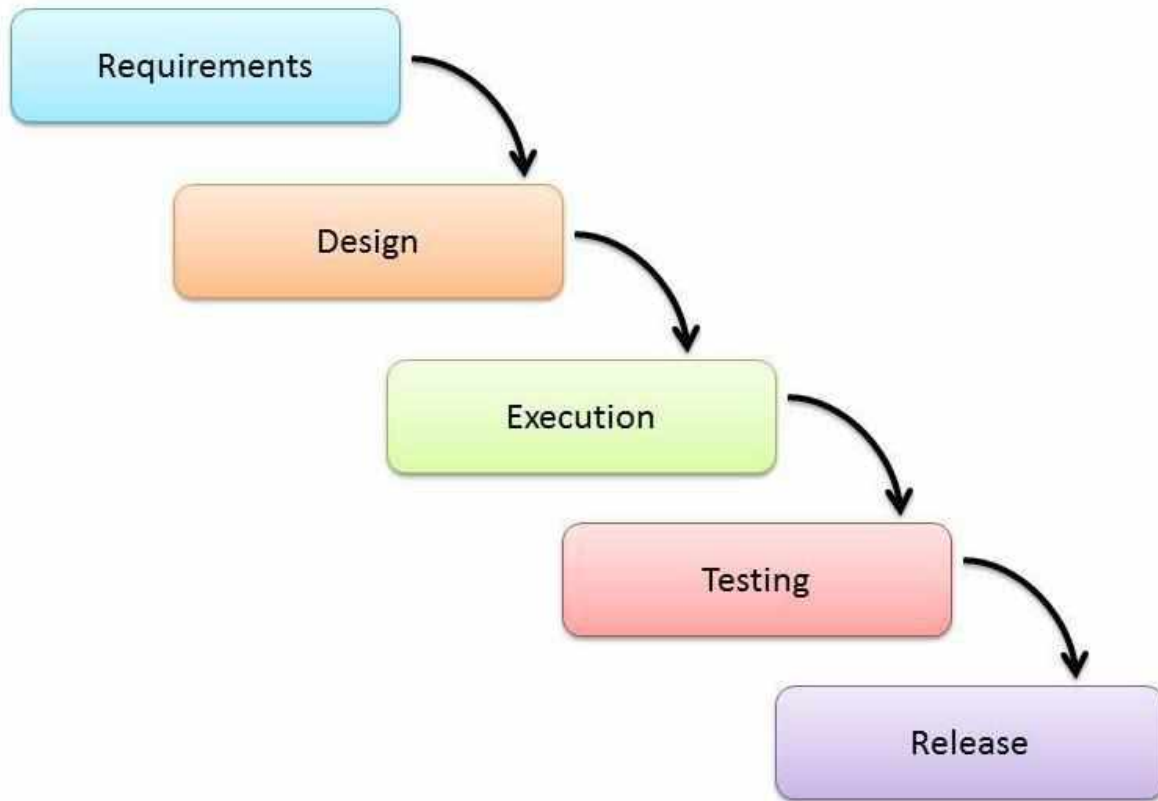
- [Waterfall Model](#)
- [V-Shaped Model](#)
- [Evolutionary Prototyping Model](#)
- [Spiral Method \(SDM\)](#)
- [Iterative and Incremental Method](#)
- [Agile development](#)

Waterfall Model

Description

The [Waterfall Model](#) is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle

changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.



The usage

Projects which not focus on changing the requirements, for example, projects initiated from a request for proposals (RFPs), the customer has a very clear documented requirements

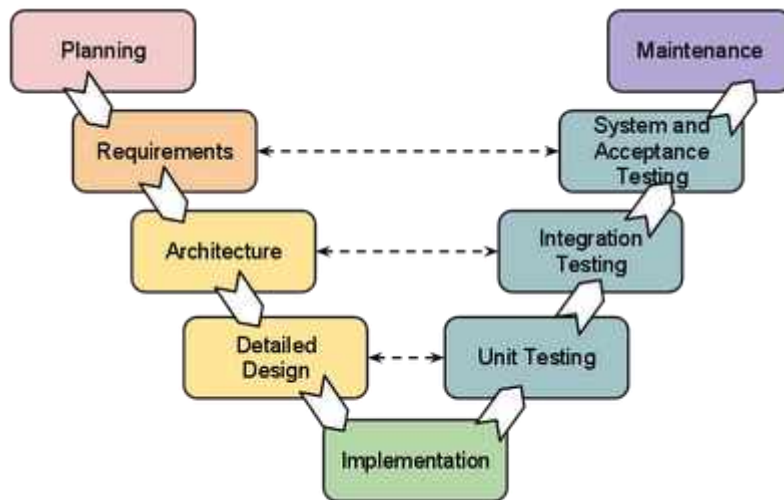
Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none">• Easy to explain to the users.• Structures approach.• Stages and activities are well defined.• Helps to plan and schedule the project.• Verification at each stage ensures early detection of errors/misunderstanding.• Each phase has specific deliverables.	<ul style="list-style-type: none">• Assumes that the requirements of a system can be frozen.• Very difficult to go back to any stage after it finished.• A little flexibility and adjusting scope is difficult and expensive.• Costly and required more time, in addition to the detailed plan.

V-Shaped Model

Description

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.



The usage

- Software requirements clearly defined and known
- Software development technologies and tools are well-known

Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none">• Simple and easy to use• Each phase has specific deliverables.• Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.• Works well for where requirements are easily understood.• Verification and validation of the product in the early stages of product development.	<ul style="list-style-type: none">• Very inflexible, like the waterfall model.• Adjusting scope is difficult and expensive.• The software is developed during the implementation phase, so no early prototypes of the software are produced.• The model doesn't provide a clear path for problems found during testing

	<p>phases.</p> <ul style="list-style-type: none"> • Costly and required more time, in addition to a detailed plan
--	--

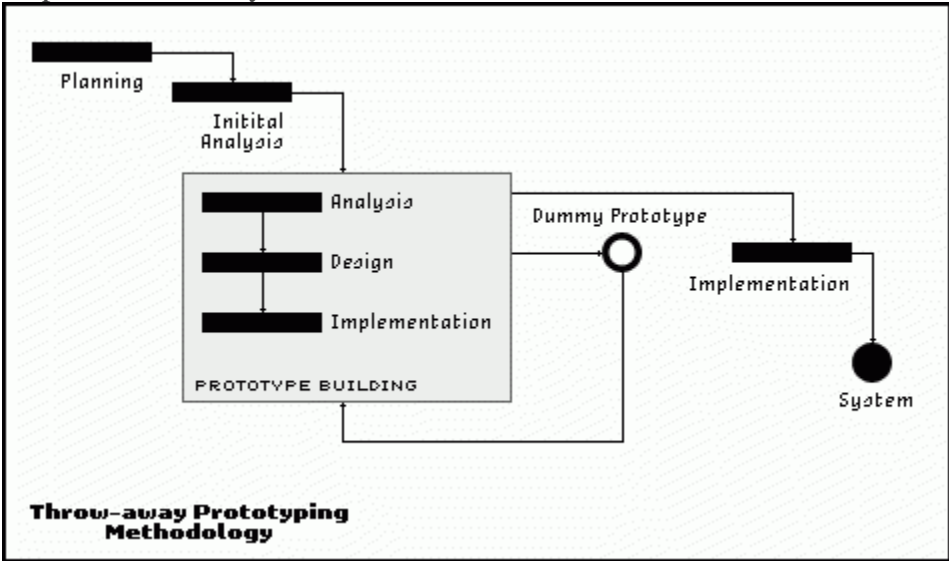
Prototyping Model

Description

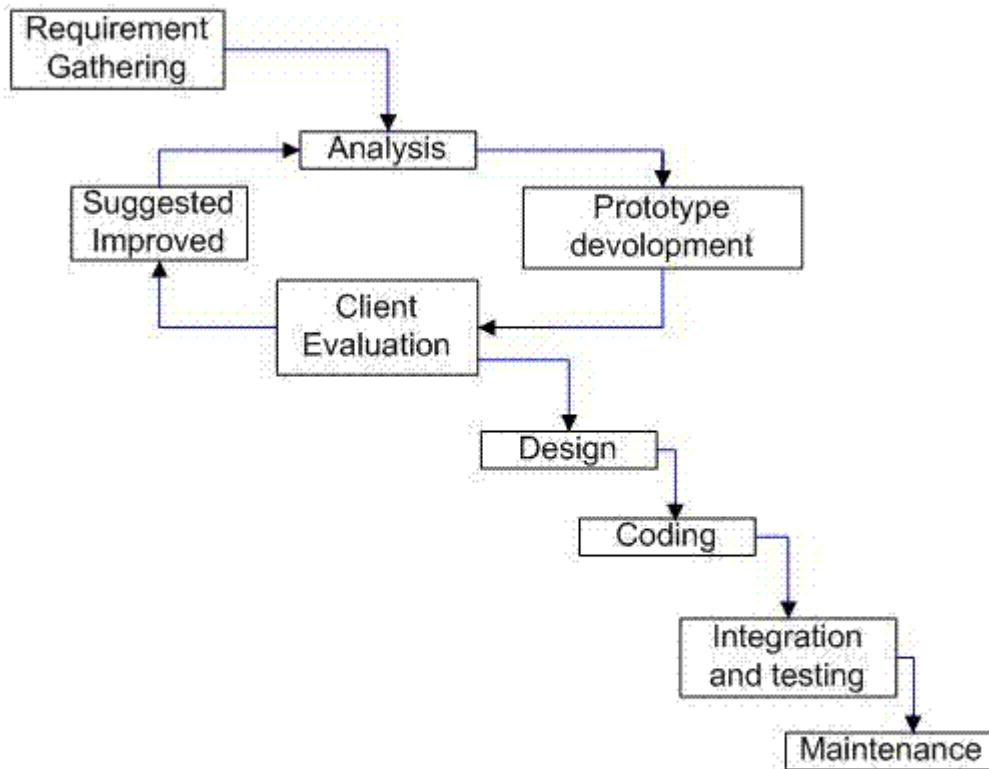
It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development and it is used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations that may occur in the waterfall approach and is hard to be implemented due to the inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

It has some types, such as:

- **Throwaway prototyping:** Prototypes that are eventually discarded rather than becoming a part of the finally delivered software



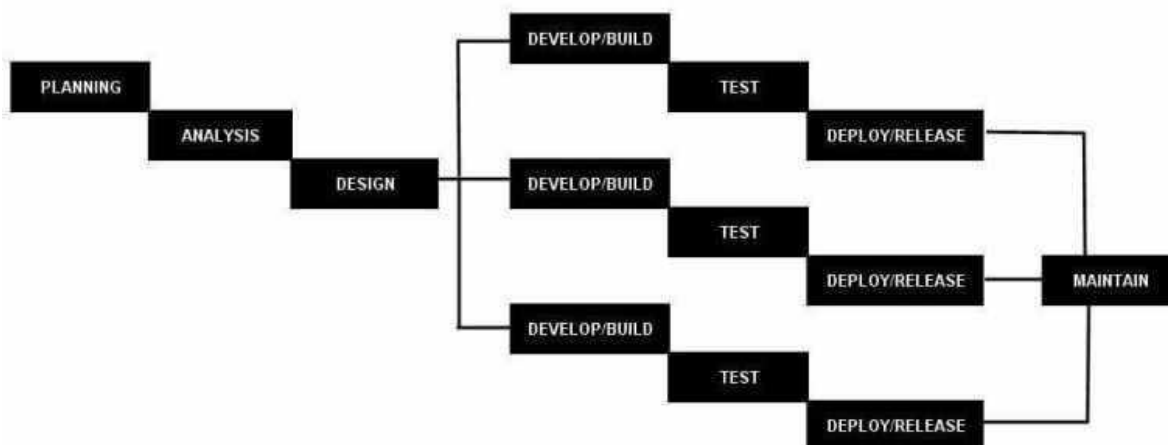
- **Evolutionary prototyping:** prototypes that evolve into the final system through an iterative incorporation of user feedback.



Evolutionary Prototyping Model

- Incremental prototyping: The final product is built as separate prototypes. In the end, the separate prototypes are merged in an overall design.

STAGED / INCREMENTAL MODEL OF SDLC



- Extreme prototyping: used in web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented

The usage

- This process can be used with any software developing life cycle model. While this shall be chosen when you are developing a system has user interactions. So, if the system does not have user interactions, such as a system does some calculations shall not have prototypes.

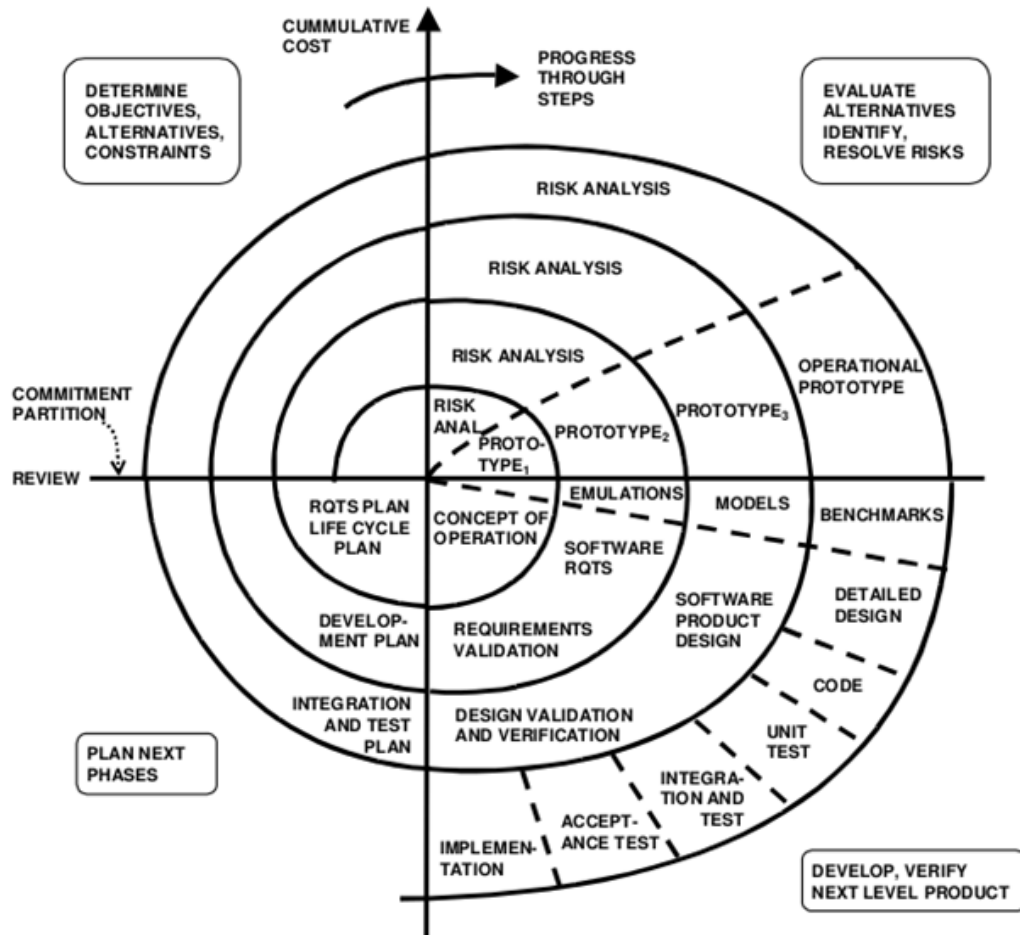
Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none"> • Reduced time and costs, but this can be a disadvantage if the developer loses time in developing the prototypes. • Improved and increased user involvement. 	<ul style="list-style-type: none"> • Insufficient analysis. User confusion of prototype and finished system. • Developer misunderstanding of user objectives. • Excessive development time of the prototype. • It is costly to implement the prototypes

Spiral Model (SDM)

Description

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.



The usage

It is used in the large applications and systems which built-in small phases or segments.

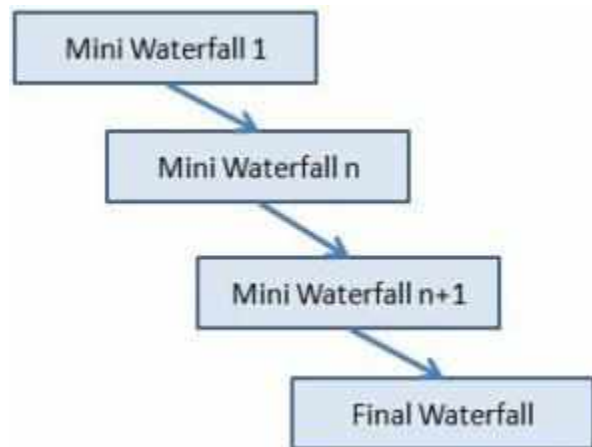
Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none"> • Estimates (i.e. budget, schedule, etc.) become more realistic as work progressed because important issues are discovered earlier. • Early involvement of developers. • Manages risks and develops the system into phases. 	<ul style="list-style-type: none"> • High cost and time to reach the final product. • Needs special skills to evaluate the risks and assumptions. • Highly customized limiting re-usability

Iterative and Incremental Model

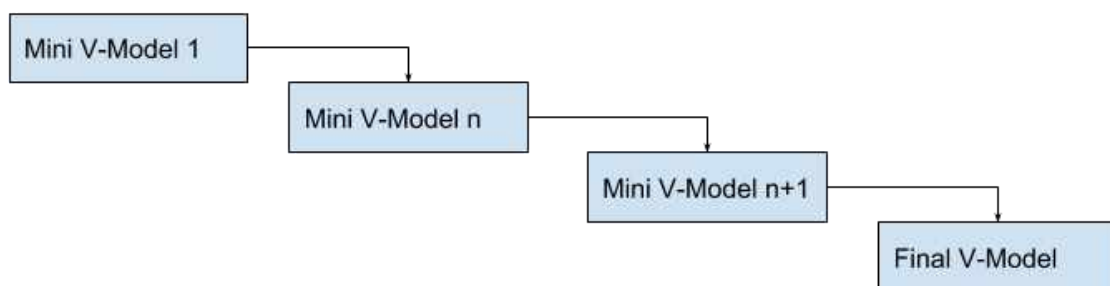
Description

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during the development of earlier parts or versions of the system. It can consist of mini waterfalls or mini V-Shaped model



The usage

It is used in shrink-wrap application and large system which built-in small phases or segments. Also, can be used in a system has separated components, for example, ERP system. Which we can start with the budget module as a first iteration and then we can start with the inventory module and so forth.



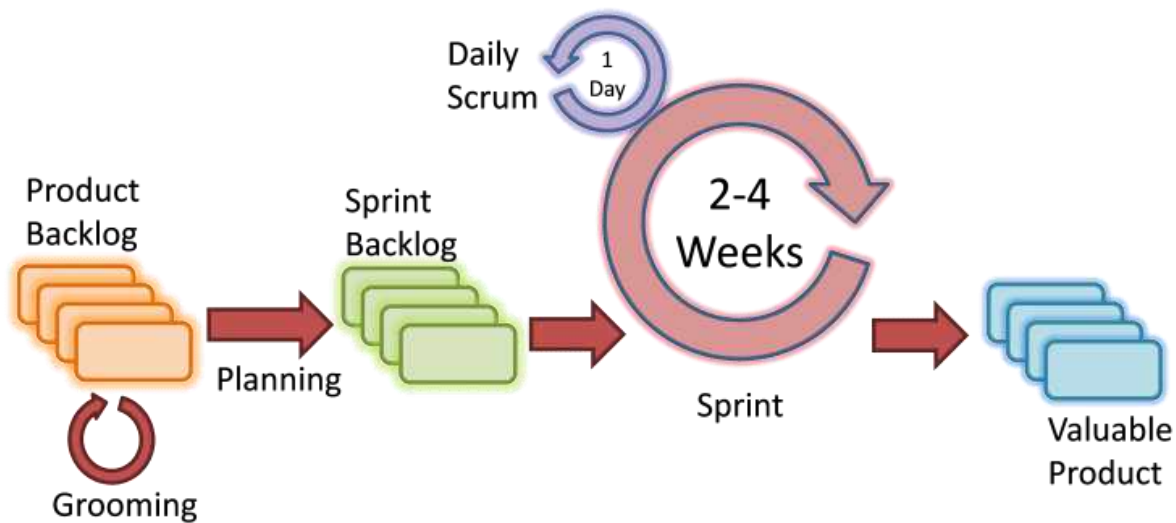
Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none"> • Produces business value early in the development lifecycle. • Better use of scarce resources through proper increment definition. • Can accommodate some change requests between increments. • More focused on customer value than the linear approaches. • We can detect project issues and changes earlier. 	<ul style="list-style-type: none"> • Requires heavy documentation. • Follows a defined set of processes. • Defines increments based on function and feature dependencies. • Requires more customer involvement than the linear approaches. • Partitioning the functions and features might be problematic. • Integration between the iterations can be an issue if it is not considered during the development and project planning.

Agile Model

Description

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.



Scrum Agile Model

The usage

It can be used with any type of the project, but it needs more engagement from the customer and to be interactive. Also, we can use it when the customer needs to have some functional requirement ready in less than three weeks and the requirements are not clear enough. This will enable more valuable and workable piece for software early which also increase the customer satisfaction.

Advantages and Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none"> • Decrease the time required to avail some system features. • Face to face communication and continuous inputs from customer representative leaves no space for guesswork. • The end result is the high-quality software in the least possible time duration and satisfied customer. 	<ul style="list-style-type: none"> • Scalability. • The ability and collaboration of the customer to express user needs. • Documentation is done at later stages. • Reduce the usability of components. • Needs special skills for the team.

Life cycle phases

Characteristic of a successful software development process is the well-defined separation between "research and development" activities and "production" activities. Most unsuccessful projects exhibit one of the following characteristics:

- An overemphasis on research and development
- An overemphasis on production.

Successful modern projects-and even successful projects developed under the conventional process-tend to have a very well-defined project milestone when there is a noticeable transition from a research attitude to a production attitude. Earlier phases focus on achieving functionality. Later phases revolve around achieving a product that can be shipped to a customer, with explicit attention to robustness, performance, and finish.

A modern software development process must be defined to support the following:

- Evolution of the plans, requirements, and architecture, together with well defined synchronization points
- Risk management and objective measures of progress and quality
- Evolution of system capabilities through demonstrations of increasing functionality

ENGINEERING AND PRODUCTION STAGES

To achieve economies of scale and higher returns on investment, we must move toward a software manufacturing process driven by technological improvements in process automation

and component-based development. Two stages of the life cycle are:

1. The **engineering stage**, driven by less predictable but smaller teams doing design and synthesis activities
2. The **production stage**, driven by more predictable but larger teams doing construction, test, and deployment activities

TABLE 5-1. *The two stages of the life cycle: engineering and production*

LIFE-CYCLE ASPECT	ENGINEERING STAGE EMPHASIS	PRODUCTION STAGE EMPHASIS
Risk reduction	Schedule, technical feasibility	Cost
Products	Architecture baseline	Product release baselines
Activities	Analysis, design, planning	Implementation, testing
Assessment	Demonstration, inspection, analysis	Testing
Economics	Resolving diseconomies of scale	Exploiting economies of scale
Management	Planning	Operations

The transition between engineering and production is a crucial event for the various stakeholders. The production plan has been agreed upon, and there is a good enough understanding of the problem and the solution that all stakeholders can make a firm commitment to go ahead with production.

Engineering stage is decomposed into two distinct phases, inception and elaboration, and the production stage into construction and transition. These four phases of the life-cycle process are loosely mapped to the conceptual framework of the spiral model as shown in Figure 5-1

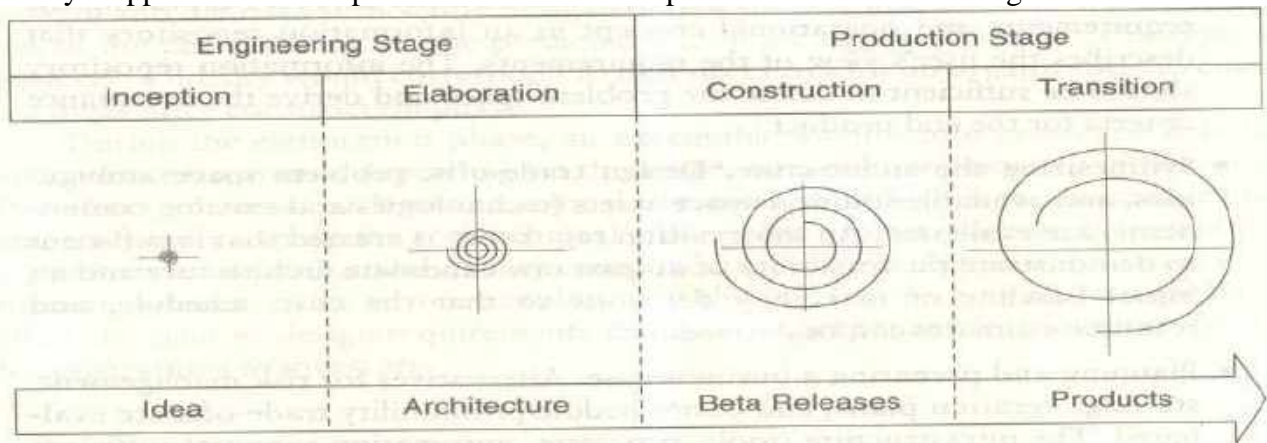


FIGURE 5-1. *The phases of the life-cycle process*

INCEPTION PHASE

The overriding goal of the inception phase is to achieve concurrence among stakeholders on the life-cycle objectives for the project.

PRIMARY OBJECTIVES

- Establishing the project's software scope and boundary conditions, including an operational concept, acceptance criteria, and a clear understanding of what is and is not intended to be in the product
- Discriminating the critical use cases of the system and the primary scenarios of operation that will drive the major design trade-offs
- Demonstrating at least one candidate architecture against some of the primary scenanos
- Estimating the cost and schedule for the entire project (including detailed estimates for the elaboration phase)
- Estimating potential risks (sources of unpredictability)

ESSENTIAL ACTMTIES

- Formulating the scope of the project. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.
- Synthesizing the architecture. An information repository is created that is sufficient to demonstrate the feasibility of at least one candidate architecture and an, initial baseline of make/buy decisions so that the cost, schedule, and resource estimates can be derived.
- Planning and preparing a business case. Alternatives for risk management, staffing, iteration plans, and cost/schedule/profitability trade-offs are evaluated.

PRIMARY EVALUATION CRITERIA

- Do all stakeholders concur on the scope definition and cost and schedule estimates?
- Are requirements understood, as evidenced by the fidelity of the critical use cases?
- Are the cost and schedule estimates, priorities, risks, and development processes credible?
- Do the depth and breadth of an architecture prototype demonstrate the preceding criteria? (The primary value of prototyping candidate architecture is to provide a vehicle for understanding the scope and assessing the credibility of the development group in solving the particular technical problem.)
- Are actual resource expenditures versus planned expenditures acceptable

ELABORATION PHASE

At the end of this phase, the "engineering" is considered complete. The elaboration phase activities must ensure that the architecture, requirements, and plans are stable enough, and the risks sufficiently mitigated, that the cost and schedule for the completion of the development can be predicted within an acceptable range. During the elaboration phase, an executable architecture prototype is built in one or more iterations, depending on the scope, size, & risk.

PRIMARY OBJECTIVES

- Baselineing the architecture as rapidly as practical (establishing a configuration-managed snapshot in which all changes are rationalized, tracked, and maintained)

- Baseline the vision
- Baseline a high-fidelity plan for the construction phase
- Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time

ESSENTIAL ACTIVITIES

- Elaborating the vision.
- Elaborating the process and infrastructure.
- Elaborating the architecture and selecting components.

PRIMARY EVALUATION CRITERIA

- Is the vision stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the construction phase plan of sufficient fidelity, and is it backed up with a credible basis of estimate?
- Do all stakeholders agree that the current vision can be met if the current plan is executed to develop the complete system in the context of the current architecture?
- Are actual resource expenditures versus planned expenditures acceptable?

CONSTRUCTION PHASE

During the construction phase, all remaining components and application features are integrated into the application, and all features are thoroughly tested. Newly developed software is integrated where required. The construction phase represents a production process, in which emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.

PRIMARY OBJECTIVES

- Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework
- Achieving adequate quality as rapidly as practical
- Achieving useful versions (alpha, beta, and other test releases) as rapidly as practical

ESSENTIAL ACTIVITIES

- Resource management, control, and process optimization
- Complete component development and testing against evaluation criteria
- Assessment of product releases against acceptance criteria of the vision

PRIMARY EVALUATION CRITERIA

- Is this product baseline mature enough to be deployed in the user community? (Existing defects are not obstacles to achieving the purpose of the next release.)

- Is this product baseline stable enough to be deployed in the user community? (Pending changes are not obstacles to achieving the purpose of the next release.)
- Are the stakeholders ready for transition to the user community?
- Are actual resource expenditures versus planned expenditures acceptable?

TRANSITION PHASE

The transition phase is entered when a baseline is mature enough to be deployed in the end-user domain. This typically requires that a usable subset of the system has been achieved with acceptable quality levels and user documentation so that transition to the user will provide positive results. This phase could include any of the following activities:

1. Beta testing to validate the new system against user expectations
2. Beta testing and parallel operation relative to a legacy system it is replacing
3. Conversion of operational databases
4. Training of users and maintainers

The transition phase concludes when the deployment baseline has achieved the complete vision.

PRIMARY OBJECTIVES

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

ESSENTIAL ACTIVITIES

- Synchronization and integration of concurrent construction increments into consistent deployment baselines
- Deployment-specific engineering (cutover, commercial packaging and production, sales rollout kit development, field personnel training)
- Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set

EVALUATION CRITERIA

- Is the user satisfied?
- Are actual resource expenditures versus planned expenditures acceptable?

6. Artifacts of the process

THE ARTIFACT SETS

To make the development of a complete software system manageable, distinct collections of information are organized into artifact sets. *Artifact* represents cohesive information that typically is developed and reviewed as a single entity.

Life-cycle software artifacts are organized into five distinct sets that are roughly partitioned by the underlying language of the set: management (ad hoc textual formats), requirements (organized text and models of the problem space), design (models of the solution space), implementation (human-readable programming language and associated source files), and deployment (machine-process able languages and associated files). The artifact sets are shown in Figure 6-1.

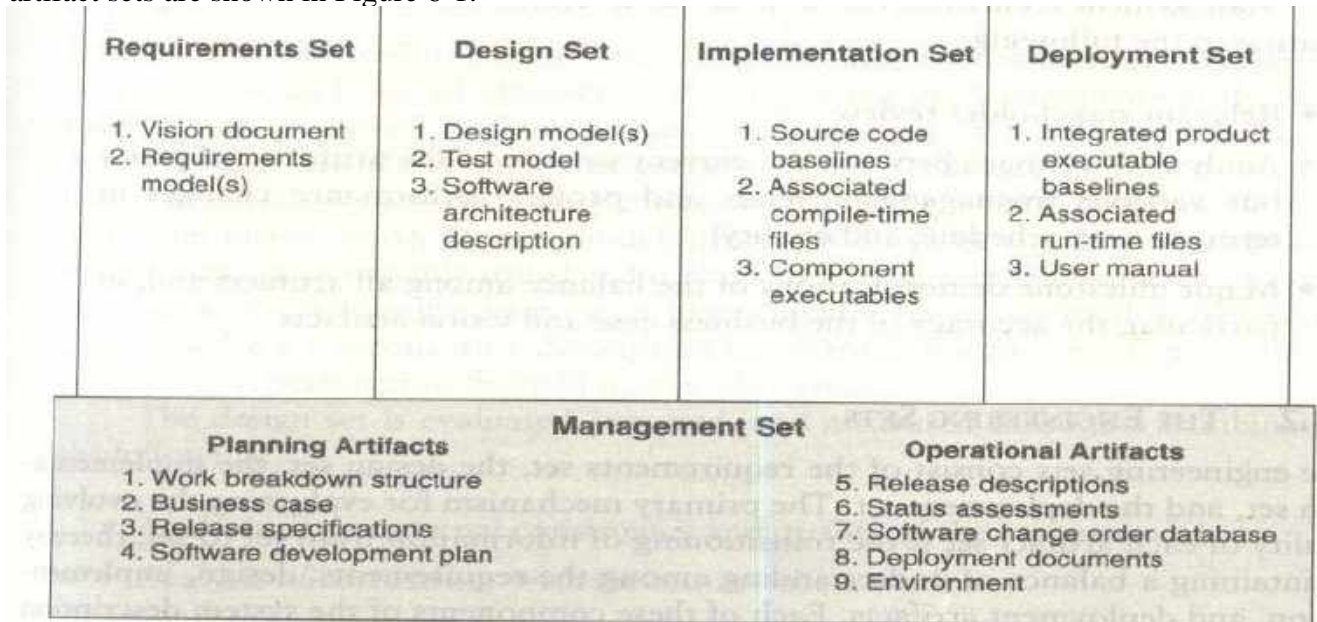


FIGURE 6-1. Overview of the artifact sets

THE MANAGEMENT SET

The management set captures the artifacts associated with process planning and execution. These artifacts use ad hoc notations, including text, graphics, or whatever representation is required to capture the "contracts" among project personnel (project management, architects, developers, testers, marketers, administrators), among stakeholders (funding authority, user, software project manager, organization manager, regulatory agency), and between project personnel and stakeholders. Specific artifacts included in this set are the work breakdown structure (activity breakdown and financial tracking mechanism), the business case (cost, schedule, profit expectations), the release specifications (scope, plan, objectives for release baselines), the software development plan (project process instance), the release descriptions (results of release baselines), the status assessments (periodic snapshots of project progress), the software change orders (descriptions of discrete baseline changes), the deployment documents (cutover plan, training course, sales rollout kit), and the environment (hardware and software tools, process automation, & documentation).

Management set artifacts are evaluated, assessed, and measured through a combination of the following:

- Relevant stakeholder review
- Analysis of changes between the current version of the artifact and previous versions
- Major milestone demonstrations of the balance among all artifacts and, in particular, the accuracy of the business case and vision artifacts

THE ENGINEERING SETS

The engineering sets consist of the requirements set, the design set, the implementation set, and the deployment set.

Requirements Set

Requirements artifacts are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the release specifications of the management set
- Analysis of consistency between the vision and the requirements models
- Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets
- Analysis of changes between the current version of requirements artifacts and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Design Set

UML notation is used to engineer the design models for the solution. The design set contains varying levels of abstraction that represent the components of the solution space (their identities, attributes, static relationships, dynamic interactions). The design set is evaluated, assessed, and measured through a combination of the following:

- Analysis of the internal consistency and quality of the design model
- Analysis of consistency with the requirements models
- Translation into implementation and deployment sets and notations (for example, traceability, source code generation, compilation, linking) to evaluate the consistency and completeness and the semantic balance between information in the sets
- Analysis of changes between the current version of the design model and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Implementation set

The implementation set includes source code (programming language notations) that represents the tangible implementations of components (their form, interface, and dependency relationships)

Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the design models
- Translation into deployment set notations (for example, compilation and linking) to evaluate the consistency and completeness among artifact sets
- Assessment of component source or executable files against relevant evaluation criteria through inspection, analysis, demonstration, or testing
- Execution of stand-alone component test cases that automatically compare expected results with actual results
- Analysis of changes between the current version of the implementation set and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Deployment Set

The deployment set includes user deliverables and machine language notations, executable software, and the build scripts, installation scripts, and executable target specific data necessary to use the product in its target environment.

Deployment sets are evaluated, assessed, and measured through a combination of the following:

- Testing against the usage scenarios and quality attributes defined in the requirements set to evaluate the consistency and completeness and the semantic balance between information in the two sets
- Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system (platform type, number, network topology)
- Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management
- Analysis of changes between the current version of the deployment set and previous versions (defect elimination trends, performance changes)
- Subjective review of other dimensions of quality

Each artifact set is the predominant development focus of one phase of the life cycle; the other sets take on check and balance roles. As illustrated in Figure 6-2, each phase has a predominant focus: Requirements are the focus of the inception phase; design, the elaboration phase; implementation, the construction phase; and deployment, the transition phase. The management artifacts also evolve, but at a fairly constant level across the life cycle.

Most of today's software development tools map closely to one of the five artifact sets.

1. Management: scheduling, workflow, defect tracking, change management, documentation, spreadsheet, resource management, and presentation tools
2. Requirements: requirements management tools
3. Design: visual modeling tools
4. Implementation: compiler/debugger tools, code analysis tools, test coverage analysis tools, and test management tools
5. Deployment: test coverage and test automation tools, network management tools, commercial components (operating systems, GUIs, RDBMS, networks, middleware),

and installation tools.

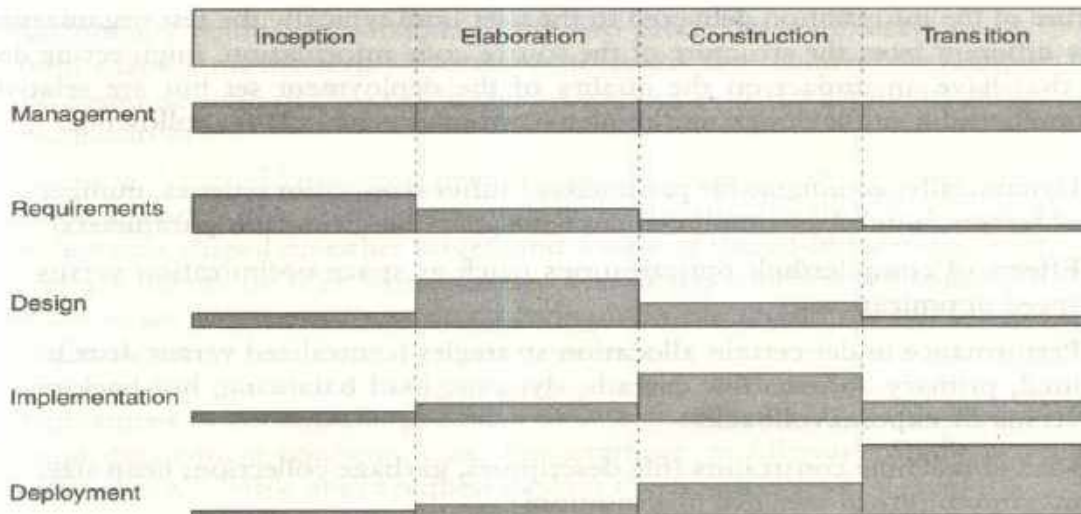


FIGURE 6-2. Life-cycle focus on artifact sets

Implementation Set versus Deployment Set

The separation of the implementation set (source code) from the deployment set (executable code) is important because there are very different concerns with each set. The structure of the information delivered to the user (and typically the test organization) is very different from the structure of the source code information. Engineering decisions that have an impact on the quality of the deployment set but are relatively incomprehensible in the design and implementation sets include the following:

- Dynamically reconfigurable parameters (buffer sizes, color palettes, number of servers, number of simultaneous clients, data files, run-time parameters)
- Effects of compiler/link optimizations (such as space optimization versus speed optimization)
- Performance under certain allocation strategies (centralized versus distributed, primary and shadow threads, dynamic load balancing, hot backup versus checkpoint/rollback)
- Virtual machine constraints (file descriptors, garbage collection, heap size, maximum record size, disk file rotations)
- Process-level concurrency issues (deadlock and race conditions)
- Platform-specific differences in performance or behavior

ARTIFACT EVOLUTION OVER THE LIFE CYCLE

Each state of development represents a certain amount of precision in the final system description. Early in the life cycle, precision is low and the representation is generally high. Eventually, the precision of representation is high and everything is specified in full detail. Each phase of development focuses on a particular artifact set. At the end of each phase, the overall system state will have progressed on all sets, as illustrated in Figure 6-3.

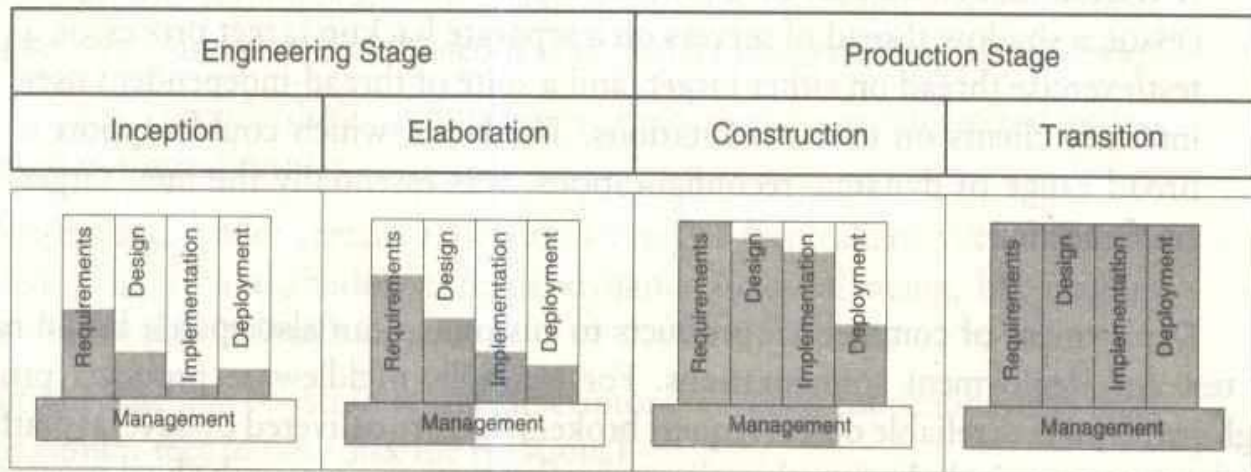


FIGURE 6-3. *Life-cycle evolution of the artifact sets*

The **inception** phase focuses mainly on critical requirements usually with a secondary focus on an initial deployment view. During the **elaboration phase**, there is much greater depth in requirements, much more breadth in the design set, and further work on implementation and deployment issues. The main focus of the **construction** phase is design and implementation. The main focus of the **transition** phase is on achieving consistency and completeness of the deployment set in the context of the other sets.

TEST ARTIFACTS

- The test artifacts must be developed concurrently with the product from inception through deployment. Thus, testing is a full-life-cycle activity, not a late life-cycle activity.
- The test artifacts are communicated, engineered, and developed within the same artifact sets as the developed product.
- The test artifacts are implemented in programmable and repeatable formats (as software programs).
- The test artifacts are documented in the same way that the product is documented.
- Developers of the test artifacts use the same tools, techniques, and training as the software engineers developing the product.

Test artifact subsets are highly project-specific, the following example clarifies the relationship between test artifacts and the other artifact sets. Consider a project to perform seismic data processing for the purpose of oil exploration. This system has three fundamental subsystems: (1) a sensor subsystem that captures raw seismic data in real time and delivers these data to (2) a technical operations subsystem that converts raw data into an organized database and manages queries to this database from (3) a display subsystem that allows workstation operators to examine seismic data in human-readable form. Such a system would result in the following test artifacts:

- Management set. The release specifications and release descriptions capture the objectives, evaluation criteria, and results of an intermediate milestone. These

artifacts are the test plans and test results negotiated among internal project teams. The software change orders capture test results (defects, testability changes, requirements ambiguities, enhancements) and the closure criteria associated with making a discrete change to a baseline.

- Requirements set. The system-level use cases capture the operational concept for the system and the acceptance test case descriptions, including the expected behavior of the system and its quality attributes. The entire requirement set is a test artifact because it is the basis of all assessment activities across the life cycle.
- Design set. A test model for nondeliverable components needed to test the product baselines is captured in the design set. These components include such design set artifacts as a seismic event simulation for creating realistic sensor data; a "virtual operator" that can support unattended, after-hours test cases; specific instrumentation suites for early demonstration of resource usage; transaction rates or response times; and use case test drivers and component stand-alone test drivers.
- Implementation set. Self-documenting source code representations for test components and test drivers provide the equivalent of test procedures and test scripts. These source files may also include human-readable data files representing certain statically defined data sets that are explicit test source files. Output files from test drivers provide the equivalent of test reports.
- Deployment set. Executable versions of test components, test drivers, and data files are provided.

MANAGEMENT ARTIFACTS

The management set includes several artifacts that capture intermediate results and ancillary information necessary to document the product/process legacy, maintain the product, improve the product, and improve the process.

Business Case

The business case artifact provides all the information necessary to determine whether the project is worth investing in. It details the expected revenue, expected cost, technical and management plans, and backup data necessary to demonstrate the risks and realism of the plans. The main purpose is to transform the vision into economic terms so that an organization can make an accurate ROI assessment. The financial forecasts are evolutionary, updated with more accurate forecasts as the life cycle progresses. Figure 6-4 provides a default outline for a business case.

Software Development Plan

The software development plan (SDP) elaborates the process framework into a fully detailed plan. Two indications of a useful SDP are periodic updating (it is not stagnant shelfware) and understanding and acceptance by managers and practitioners alike. Figure 6-5 provides a default outline for a software development plan.

- I. Context (domain, market, scope)**
- II. Technical approach**
 - A. Feature set achievement plan
 - B. Quality achievement plan
 - C. Engineering trade-offs and technical risks
- III. Management approach**
 - A. Schedule and schedule risk assessment
 - B. Objective measures of success
- IV. Evolutionary appendixes**
 - A. Financial forecast
 - 1. Cost estimate
 - 2. Revenue estimate
 - 3. Bases of estimates

FIGURE 6-4. *Typical business case outline*

- I. Context (scope, objectives)**
- II. Software development process**
 - A. Project primitives
 - 1. Life-cycle phases
 - 2. Artifacts
 - 3. Workflows
 - 4. Checkpoints
 - B. Major milestone scope and content
 - C. Process improvement procedures
- III. Software engineering environment**
 - A. Process automation (hardware and software resource configuration)
 - B. Resource allocation procedures (sharing across organizations, security access)
- IV. Software change management**
 - A. Configuration control board plan and procedures
 - B. Software change order definitions and procedures
 - C. Configuration baseline definitions and procedures
- V. Software assessment**
 - A. Metrics collection and reporting procedures
 - B. Risk management procedures (risk identification, tracking, and resolution)
 - C. Status assessment plan
 - D. Acceptance test plan
- VI. Standards and procedures**
 - A. Standards and procedures for technical artifacts
- VII. Evolutionary appendixes**
 - A. Minor milestone scope and content
 - B. Human resources (organization, staffing plan, training plan)

FIGURE 6-5. *Typical software development plan outline*

Work Breakdown Structure

Work breakdown structure (WBS) is the vehicle for budgeting and collecting costs. To monitor and control a project's financial performance, the software project manager must have insight into project costs and how they are expended. The structure of cost accountability is a

serious project planning constraint.

Software Change Order Database

Managing change is one of the fundamental primitives of an iterative development process. With greater change freedom, a project can iterate more productively. This flexibility increases the content, quality, and number of iterations that a project can achieve within a given schedule. Change freedom has been achieved in practice through automation, and today's iterative development environments carry the burden of change management. Organizational processes that depend on manual change management techniques have encountered major inefficiencies.

Release Specifications

The scope, plan, and objective evaluation criteria for each baseline release are derived from the vision statement as well as many other sources (make/buy analyses, risk management concerns, architectural considerations, shots in the dark, implementation constraints, quality thresholds). These artifacts are intended to evolve along with the process, achieving greater fidelity as the life cycle progresses and requirements understanding matures. Figure 6-6 provides a default outline for a release specification

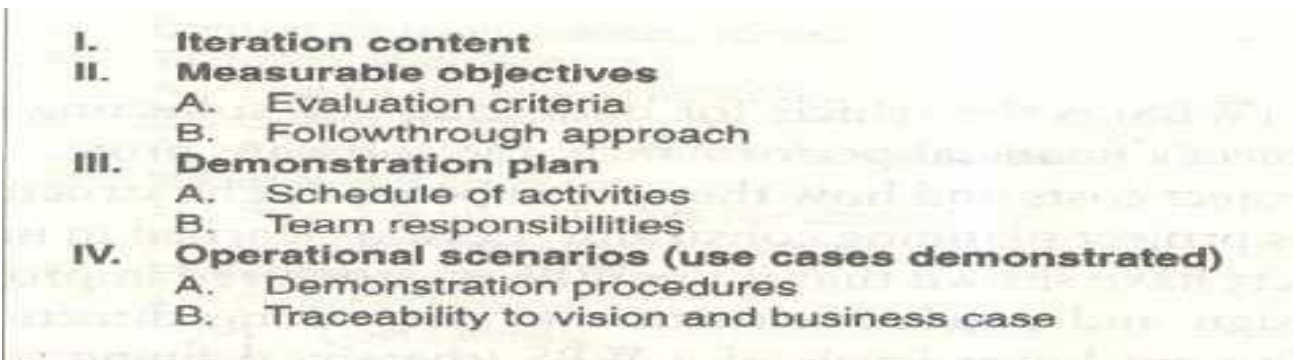


FIGURE 6-6. *Typical release specification outline*

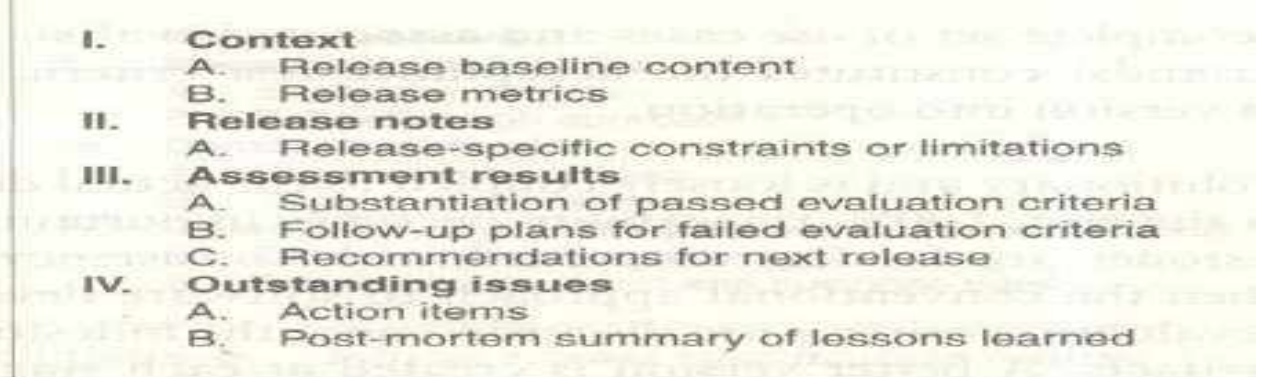
Release Descriptions

Release description documents describe the results of each release, including performance against each of the evaluation criteria in the corresponding release specification. Release baselines should be accompanied by a release description document that describes the evaluation criteria for that configuration baseline and provides substantiation (through demonstration, testing, inspection, or analysis) that each criterion has been addressed in an acceptable manner. Figure 6-7 provides a default outline for a release description.

Status Assessments

Status assessments provide periodic snapshots of project health and status, including the software project manager's risk assessment, quality indicators, and management indicators.

Typical status assessments should include a review of resources, personnel staffing, financial data (cost and revenue), top 10 risks, technical progress (metrics snapshots), major milestone plans and results, total project or product scope & action items



I.	Context
	A. Release baseline content
	B. Release metrics
II.	Release notes
	A. Release-specific constraints or limitations
III.	Assessment results
	A. Substantiation of passed evaluation criteria
	B. Follow-up plans for failed evaluation criteria
	C. Recommendations for next release
IV.	Outstanding issues
	A. Action items
	B. Post-mortem summary of lessons learned

FIGURE 6-7. *Typical release description outline*

Environment

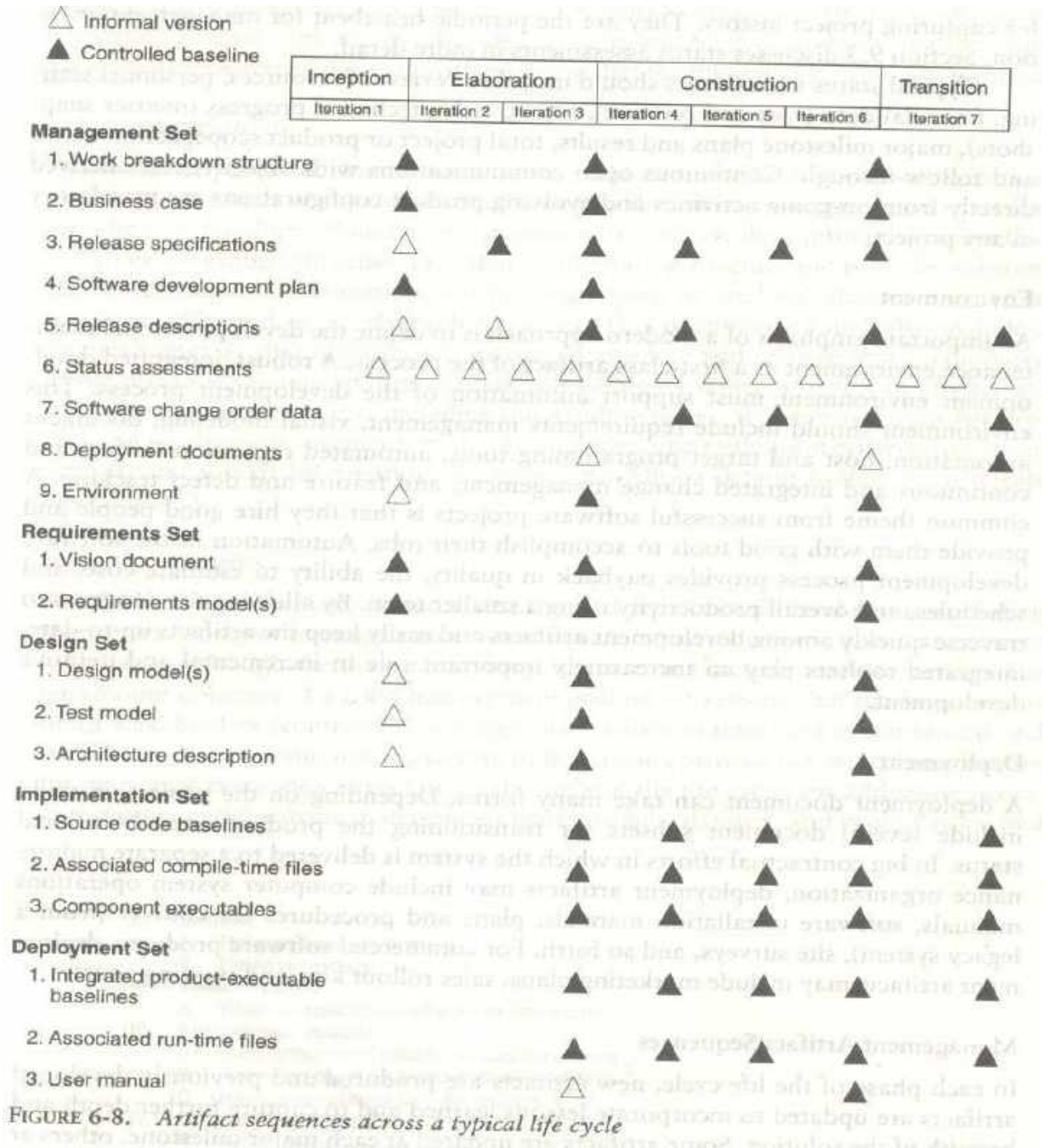
An important emphasis of a modern approach is to define the development and maintenance environment as a first-class artifact of the process. A robust, integrated development environment must support automation of the development process. This environment should include requirements management, visual modeling, document automation, host and target programming tools, automated regression testing, and continuous and integrated change management, and feature and defect tracking.

Deployment

A deployment document can take many forms. Depending on the project, it could include several document subsets for transitioning the product into operational status. In big contractual efforts in which the system is delivered to a separate maintenance organization, deployment artifacts may include computer system operations manuals, software installation manuals, plans and procedures for cutover (from a legacy system), site surveys, and so forth. For commercial software products, deployment artifacts may include marketing plans, sales rollout kits, and training courses.

Management Artifact Sequences

In each phase of the life cycle, new artifacts are produced and previously developed artifacts are updated to incorporate lessons learned and to capture further depth and breadth of the solution. Figure 6-8 identifies a typical sequence of artifacts across the life-cycle phases.



ENGINEERING ARTIFACTS

Most of the engineering artifacts are captured in rigorous engineering notations such as UML, programming languages, or executable machine codes. Three engineering artifacts are explicitly intended for more general review, and they deserve further elaboration.

Vision Document

The vision document provides a complete vision for the software system under development and supports the contract between the funding authority and the development organization. A project vision is meant to be changeable as understanding evolves of the requirements, architecture, plans, and technology. A good vision document should change slowly. Figure 6-9 provides a default outline for a vision document.

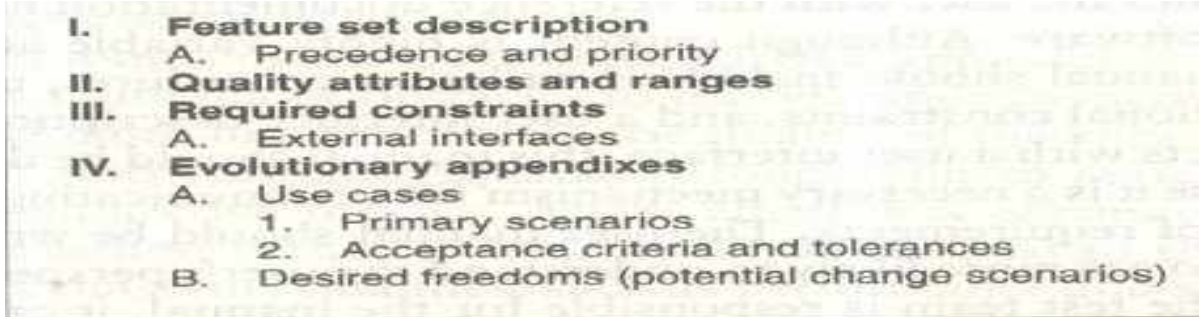
- 
- I. Feature set description**
 - A. Precedence and priority
 - II. Quality attributes and ranges**
 - III. Required constraints**
 - A. External interfaces
 - IV. Evolutionary appendixes**
 - A. Use cases
 - 1. Primary scenarios
 - 2. Acceptance criteria and tolerances
 - B. Desired freedoms (potential change scenarios)

FIGURE 6-9. *Typical vision document outline*

Architecture Description

The architecture description provides an organized view of the software architecture under development. It is extracted largely from the design model and includes views of the design, implementation, and deployment sets sufficient to understand how the operational concept of the requirements set will be achieved. The breadth of the architecture description will vary from project to project depending on many factors. Figure 6-10 provides a default outline for an architecture description.

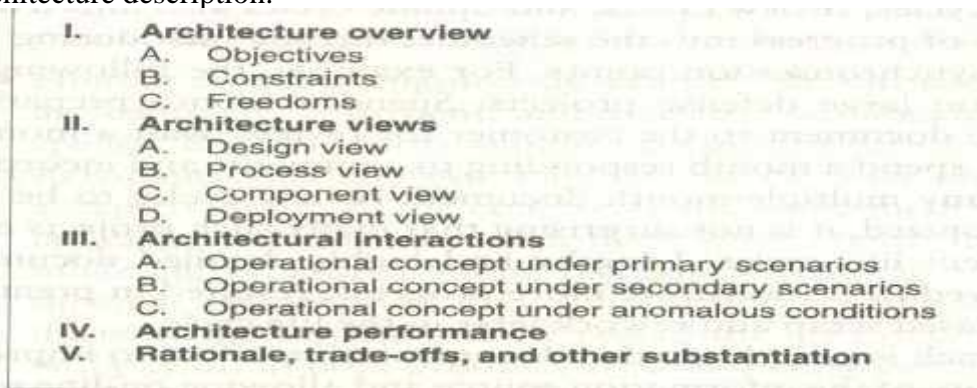
- 
- I. Architecture overview**
 - A. Objectives
 - B. Constraints
 - C. Freedoms
 - II. Architecture views**
 - A. Design view
 - B. Process view
 - C. Component view
 - D. Deployment view
 - III. Architectural interactions**
 - A. Operational concept under primary scenarios
 - B. Operational concept under secondary scenarios
 - C. Operational concept under anomalous conditions
 - IV. Architecture performance**
 - V. Rationale, trade-offs, and other substantiation**

FIGURE 6-10. *Typical architecture description outline*

Software User Manual

The software user manual provides the user with the reference documentation necessary to support the delivered software. Although content is highly variable across application domains, the user manual should include installation procedures, usage procedures and guidance, operational constraints, and a user interface description, at a minimum. For software products with a user interface, this manual should be developed early in the life cycle because it is a necessary mechanism for communicating and stabilizing an important subset of

requirements. The user manual should be written by members of the test team, who are more likely to understand the user's perspective than the development team.

PRAGMATIC ARTIFACTS

•People want to review information but don't understand the language of the artifact.

Many interested reviewers of a particular artifact will resist having to learn the engineering language in which the artifact is written. It is not uncommon to find people (such as veteran software managers, veteran quality assurance specialists, or an auditing authority from a regulatory agency) who react as follows: "I'm not going to learn UML, but I want to review the design of this software, so give me a separate description such as some flowcharts and text that I can understand."

•People want to review the information but don't have access to the tools. It is not very common for the

development organization to be fully tooled; it is extremely rare that the/other stakeholders have any capability to review the engineering artifacts on-line. Consequently, organizations are forced to exchange paper documents. Standardized formats (such as UML, spreadsheets, Visual Basic, C++, and Ada 95), visualization tools, and the Web are rapidly making it economically feasible for all stakeholders to exchange information electronically.

•Human-readable engineering artifacts should use rigorous notations that are complete, consistent, and used in a self-documenting manner. Properly spelled English words should be used for all identifiers and descriptions. Acronyms and abbreviations should be used only where they are well accepted jargon in the context of the component's usage. Readability should be emphasized and the use of proper English words should be required in all engineering artifacts. This practice enables understandable representations, browse able formats (paperless review), more-rigorous notations, and reduced error rates.

•Useful documentation is self-defining: It is documentation that gets used.

•Paper is tangible; electronic artifacts are too easy to change. On-line and Web-based artifacts can be changed easily and are viewed with more skepticism because of their inherent volatility.

UNIT III

EFFORT ESTIMATION & ACTIVITY PLANNING

EFFORT ESTIMATION:

Software Effort Estimation

- Successful project is that the system is delivered on time and within budget and with the required quality.

Software effort estimation

Difficulties in Software estimation

- Subjective Nature of estimating
- Political Implications
- Changing Technology
- Lack of homogeneity of project experience

Project Data

Note:
SLOC - Source Number of Lines
WM - Work in Month

Clip slide

Project	Design		Coding		Testing		Total	
	wm	(%)	wm	(%)	wm	(%)	wm	SLOC
a	3.9	(23)	5.3	(32)	7.4	(44)	16.7	6050
b	2.7	(12)	13.4	(59)	6.5	(26)	22.6	8363
c	3.5	(11)	26.8	(83)	1.9	(6)	32.2	13334
d	0.8	(21)	2.4	(62)	0.7	(18)	3.9	5942
e	1.8	(10)	7.7	(44)	7.8	(45)	17.3	3315
f	19.0	(28)	29.7	(44)	19.0	(28)	67.7	38988
g	2.1	(21)	7.4	(74)	0.5	(5)	10.1	38614
h	1.3	(7)	12.7	(66)	5.3	(27)	19.3	12762
i	8.5	(14)	22.7	(38)	28.2	(47)	59.5	26500

Clip slide

Where are estimates done?

- Estimates are carried out at various stages of software project.
- **Strategic Planning**
 - Decide priority to each project.
- **Feasibility Study**
 - Benefits of potential system
- **System Specification**
 - Detailed requirement analysis at design stage.
- **Evaluation of Suppliers Proposals**
 - Tender Management
- **Project Planning**
 - Detailed estimates of smaller work components during implementation.

Software effort estimation techniques

- Algorithm or parametric model
- Analog model
- Expert judgment
- Parkinson
- Price to win
- Top-down
- Bottom-up

Bottom-up Estimating

Clip slide

- **Work Breakdown Structure**
- **Assumptions about characteristics of final system**
- **Number and Size of software modules.**
- **Appropriate at detailed stages of project planning.**
- **When a project is completely novel or no historical data available.**

Expert Judgment

Clip slide

- **Asking for estimate of task effort from someone who is knowledgeable about either application or development environment.**
- **Experts use the combination of informal analogy approach where similar projects from past are identified and bottom up estimating.**

Top-down Approach and Parametric Models

Clip slide

- **Effort = (system size) * (productivity rate)**
- **System size in the form of KLOC**
- **Productivity rate 40 days per KLOC**
- **Software module to be constructed is 2 KLOC**
- **Effort = 2 * 80 = 160 days**

Estimating by Analogy

Clip slide

- **Called “Case Based Analogy”**
- **Estimator identifies completed projects source cases with similar characteristics to new project (target case)**
- **Effort of the source case used as base estimate for target.**
- **TOOL – ANGEL software tool**
- **Measuring Euclidean Distance between the cases**

Functional Point (FP) Analysis

FPA provides standardized method to functionally size the software work product. This work product is the output of software new development and improvement projects for subsequent releases. It is the software which is relocated to the production application at project implementation. It measures functionality from the users point of view i.e. on the basis of what the user requests and receives in return.

Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements. It measures the logical view of an application not the physically implemented view or the internal technical view.

The Function Point Analysis technique is used to analyse the functionality delivered by software and *Unadjusted Function Point (UFP)* is the unit of measurement.

Objectives of FPA:

1. The objective of FPA is to measure functionality that the user requests and receives.
2. The objective of FPA is to measure software development and maintenance independently of technology used for implementation.
3. It should be simple enough to minimize the overhead of the measurement process.
4. It should be a consistent measure among various projects and organizations.

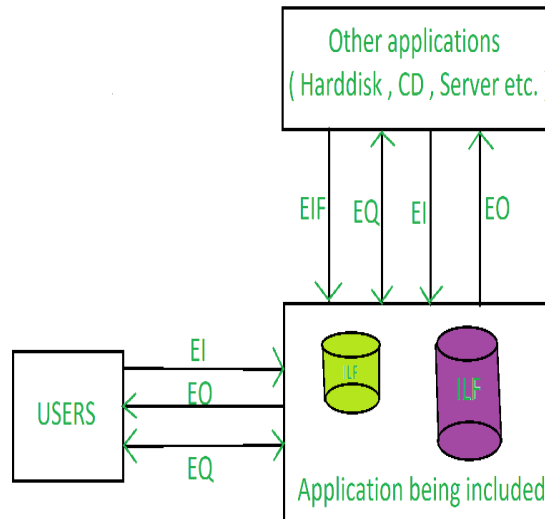
Types of FPA:

1. Transactional Functional Type –

- (i) **External Input (EI):** EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
- (ii) **External Output (EO):** EO is an elementary process that generates data or control information sent outside the application's boundary.
- (iii) **External Inquiries (EQ):** EQ is an elementary process made up of an input-output combination that results in data retrieval.

2. Data Functional Type –

- (iv) **Internal Logical File (ILF):** A user identifiable group of logically related data or control information maintained within the boundary of the application.
- (v) **External Interface File (EIF):** A group of user recognizable logically related data allusion to the software but maintained within the boundary of another software.



Benefits of FPA:

- FPA is a tool to determine the size of a purchased application package by counting all the functions included in the package.
- It is a tool to help users discover the benefit of an application package to their organization by counting functions that specifically match their requirements.
- It is a tool to measure the units of a software product to support quality and productivity analysis.
- It is a vehicle to estimate cost and resources required for software development and maintenance.
- It is a normalization factor for software comparison.

IFPUG

- PURCHASE_ORDER
- PURCHASE_ORDER_ITEM

Table 5.3 *IFPUG file type complexity*

Number of record types	Number of data types		
	<20	20 to 50	>50
1	low	low	average
2 to 5	low	average	high
> 5	average	high	high

Function Points Mark II

- Sponsored by CCTA (Central Computer and Telecommunications Agency)
- Mark II – Improvement and replacement in Albrecht method
- In Albrecht method
 - Information Processing Size is measured in UFPs (Unadjusted Functional Points)
 - Then TCA (Technical Complexity Adjustment) is applied



Figure 5.2 Model of a transaction.

For each transaction the UFPs are calculated:

$$\begin{aligned}
 &W_i \times (\text{number of input data element types}) + \\
 &W_e \times (\text{number of entity types referenced}) + \\
 &W_o \times (\text{number of output data element types})
 \end{aligned}$$

For each transaction UFPs are calculated

- $UFPs = W_i * (\text{number of input data element types}) + W_e * (\text{number of entity types referenced}) + W_o * (\text{number of output data element types})$
- W_i W_e W_o are **weightings** derived by asking the developers the proportions of effort spent.
- FP counters use industry averages which are:
 - $W_i = 0.58$
 - $W_e = 1.66$
 - $W_o = 0.26$

COSMIC Full Function Points

- Cosmic deals with decomposing the system architecture into hierarchy of software layers.
- Inputs and outputs are aggregated into data groups
- Each data group brings together data items that relate to the same object of interest.
- Data Groups can be moved in 4 ways:
 - Entries(E)
 - Exits(X)
 - Reads (R)
 - Writes(W)

□ COCOMO (Constructive Cost Model)-Boehm

□ Formula :

- $(\text{effort}) = c(\text{size})^k$
 - Effort measured in pm(number of person-month)
 - Size in kdsi (Thousands of delivered source code instructions)
 - C,K constants

□ C and K are from

System Type	C	K
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

- Organic Mode:
 - Small teams develop software in a highly familiar environment (Small & Flexible)
- Embedded Mode:
 - Operate within very tight constraints and changes to the system very costly
- Semi-Detached Mode:
 - Combined elements of both

COCOMO II

COCOMO II

- COCOMO 81 makes a variety of assumptions about the software development process in order to produce its estimates. The latter will only be somewhat accurate when the project uses the waterfall process model and every line of code is produced from scratch. It also fails to take into account that nowadays higher-level programming languages are employed, supported by various automated tools. We will not elaborate on this version, since it has been obsolete by COCOMO 2.

The Application Composition model involves prototyping efforts to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity. The costs of this type of effort are best estimated by the Applications Composition model.

The Early Design model involves exploration of alternative software/system architectures and concepts of operation. At this stage, not enough is generally known to support fine-grain cost estimation. The corresponding COCOMO II capability involves the use of function points and a course-grained set of 7 cost drivers (e.g. two cost drivers for Personnel Capability and Personnel Experience in place of the 6 COCOMO II Post-Architecture model cost drivers covering various aspects of personnel capability, continuity, and experience).

The Post-Architecture model involves the actual development and maintenance of a software product. This stage proceeds most cost-effectively if a software life-cycle architecture has been developed; validated with respect to the system's mission, concept of operation, and risk; and established as the framework for the product. The corresponding COCOMO II model has about the same granularity as the previous COCOMO and Ada COCOMO models. It uses source instructions and / or function points for sizing, with modifiers for reuse and software breakage; a set of 17 multiplicative cost drivers; and a set of 5 factors determining the project's scaling exponent. These factors replace the development modes (Organic, Semidetached, or Embedded) in the original COCOMO model, and refine the four exponent-scaling factors in Ada COCOMO.

The post-architecture level: Once the system architecture has been designed a reasonably accurate estimate of the software size can be made. The estimate at this level uses a more extensive set of multipliers reflecting personnel capabilities, product and project characteristics.

$$P_m = A * size^{sf} * (em_1) * (em_2) * \dots * (em_n)$$

The activity-based approach

The **activity based approach** consists of creating a list of all activities that the project is thought to involve. How?

- Brainstorming session involving the whole project team
- The analysis of similar past projects.
- One useful way is to divide you projects into stages and think what activities might be required at each stage.

One way of creating the activity or task list is to create

WBS (Work Breakdown Structure).

The activity-based approach (cont'd)

In WBS we:

- Identify the main (high level) tasks (activities) required to complete a project.
- Then break each of these down into a set of lower-level tasks.

The activity-based approach (cont'd)

When preparing the WBS:

- Too great depth should be avoided. **Why?**
 - Will result in a large number of tasks that will need to be managed.
- Too shallow structure should be avoided. **Why?**
 - This will provide insufficient detail for project control.

The activity-based approach (cont'd)

Advantages (WBS)

- More likely to obtain a task catalogue that is:
 - Complete and *كاملة*
 - Composed of non-overlapping tasks *غير متداخلة*
- WBS represents a structure that can be refined as the project proceeds.
- Early in the project, It can start shallow *في البداية مبسط*
- Developed as information becomes available e.g. during project analysis and specification phases. *لاحقًا بتفاصيل أكثر*

Once the project activities have been identified (whether using the WBS or not) they will need to be sequenced.

The product-based approach

It consists of producing a **product breakdown structure PBS**, and a **product flow diagram PFD**.

- Product Breakdown Structure (PBS)**
 - To show how a system can be broken down into different products for development.
 - Advantage: It is less likely to omit a product.

The product-based approach (cont'd)

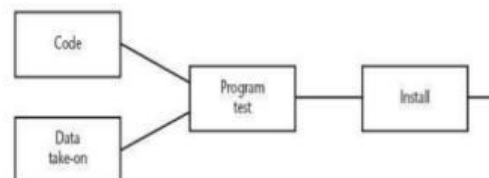
- Product Flow Diagram (PFD)**
 - Once a product breakdown structure has been created, work can then begin on creating a product flow diagram which identifies the order of precedence of products and will typically include multiple and complex parallel paths.

Formulating A Network Model

- Constructing Precedence Network Rules**
- A project network should have only one start node
 - More than one activity starting at once? Invent a 'start' activity with zero duration
- A project network should have only one end node
 - If necessary, invent an 'end' activity
- A node has duration
- Links normally have no duration
- Precedents are the immediate preceding activities
 - All have to be completed before an activity can be started
- Time moves from left to right
- A network may not contain loops
- A network should not contain dangles
 - If necessary, connect to the final node

Fragment of Precedence Network

- Installation cannot start until program testing is completed
- Program test cannot start until both code and data take-on have been completed

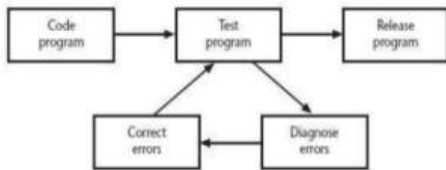


- ◆ In the precedence networks:
 - ◆ The nodes represent activities.
 - ◆ The lines between nodes represent dependencies.



Network Contains Loop

- A loop is an error in that it represents a situation that cannot occur in practice
- Program testing cannot start until errors have been corrected?



Formulating A Network Model A Dangle

- A dangling activity such as "write user manual" should not exist as it is likely to lead to errors in subsequent analysis

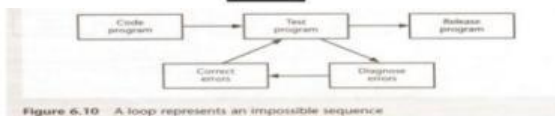
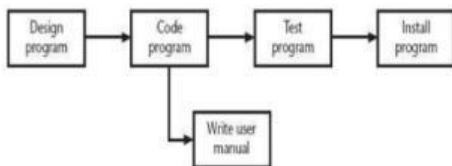


Figure 6.10 A loop represents an impossible sequence

- **A network may not contain loops** Figure 6.10 demonstrates a loop in a network. A loop is an error in that it represents a situation that cannot occur in practice. While loops, in the sense of iteration, may occur in practice, they cannot be directly represented in a project network.
- **A network should not contain dangles.** A dangling activity such as 'Write user manual' in Figure 6.11 should not exist as it is likely to lead to errors in subsequent analysis.
- Redraw the network with a final completion activity — which, at least in this case, is probably a more accurate

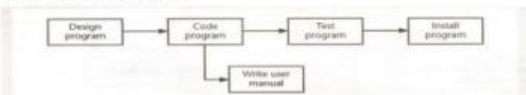


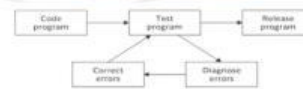
Figure 6.11 A dangle

Precedence Networks (cont'd)

Rules and Conventions

- A project network should have only one start node.
- A project network should have only one end node.
- A node has duration.
- Links have no duration.
- Precedents are referred to the immediate preceding activities.
- Time moves from left to right.
- A network may not contain loops.
- A network should not contain dangles.

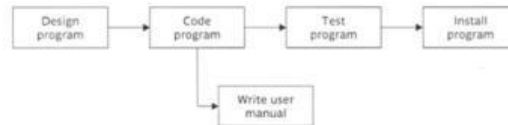
Precedence Networks (cont'd)



Loops can't be directly represented in a project network.

- ◆ If you know the number of times we expect to repeat a set of activities, then
- ◆ we can draw them in a sequence repeating them for the appropriate number of times.

Precedence Networks (cont'd)

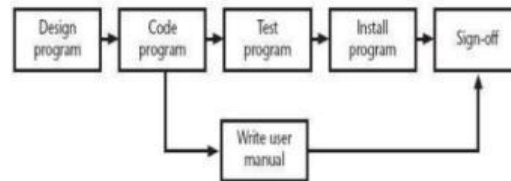


A dangle: Here what is meant by the diagram is: that the project will be finished once we "the program has been installed" and "the user manual is written".

"Write user manual" is a dangling activity.

Formulating A Network Model Resolving The Dangle

- The figure implies that the project is complete once the software has been installed and the user manual written
- We should redraw the network with a final completion activity



Representing Lagged Activities

- **Representing lagged activities**
- We might come across situations where we wished to undertake two activities in parallel so long as there is a lag between the two. We might wish to document amendments to a program as it was being tested - particularly if evaluating a prototype.
- Where activities can occur in parallel with a time lag between them we represent the lag with a duration on the linking arrow as shown in Figure 6.13. This indicates that documenting amendments can start one day after the start of proto-type testing and will be completed two days after prototype testing is completed.

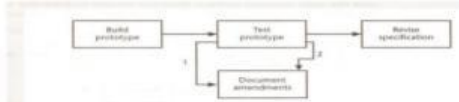


Figure 6.13 Indicating lags

- **Lag activities:** are two activities that will be undertaken in parallel but there is a lag between them.
- Here two activities will be undertaken at the same time with some lag between them.
- “Document amendments” will start one day after “Test prototype” starts and finish two days after “Test prototype” ends.

Adding The Time Dimension

- The critical path approach
 - Planning the project in such way that it is completed as quickly as possible
 - Identifying delayed activities
- The method requires the estimation of duration of each activity
 - *Forward pass:* calculate the earliest dates at which activities may commence and the project completed
 - *Backward pass:* calculate the latest start dates for activities and the *critical path*

Adding the Time Dimension

- After we create the logical network model showing the activities and the interrelationships between those activities. We should think of when each activity will be undertaken.
- The critical path approach is concerned with:
 - Planning the project in a way that it will be completed as quickly as possible.
 - Identifying the activities where a delay in their execution is likely to affect
 - The overall end date of the project or
 - Later activities start dates.

Adding the Time Dimension (cont'd)

- For each activity we will estimate its duration.
- The network is then analyzed by carrying out the forward pass and a backward pass.
- **The forward pass:**
 - Calculates the earliest dates at which activities may be started, finished
 - Project completion time.
- **The backward pass:**
 - Calculates the latest dates at which activities may be started, finished, the float and
 - The critical path.

6.9 Adding The Time Dimension

- The critical path approach
 - Planning the project in such way that it is completed as quickly as possible
 - Identifying delayed activities
- The method requires the estimation of duration of each activity
 - *Forward pass:* calculate the earliest dates at which activities may commence and the project completed
 - *Backward pass:* calculate the latest start dates for activities and the *critical path*

Activity	Duration (weeks)	Precedents
A Hardware selection	6	
B System configuration	4	
C Install hardware	3	A
D Data migration	4	B
E Draft office procedures	3	B
F Recruit staff	10	
G User training	3	E, F
H Install and test system	2	C, D

6.10 The Forward Pass

The Calculation of Earliest Start Date [1/4]

- Activities A, B and F may start immediately
 - The earliest date for their start is zero
- Activity A will take 6 weeks
 - The earliest it can finish is week 6
- Activity F will take 10 weeks
 - The earliest it can finish is week 10

6.10 The Forward Pass

The Calculation of Earliest Start Date [2/4]

- Activity C can start as soon as A has finished
 - Its earliest start date is week 6
 - It will take 3 weeks, so the earliest it can finish is week 9
- Activities D and E can start as soon as B is complete
 - The earliest they can each start is week 4
 - Activity D will take 4 weeks, so the earliest it can finish is week 8
 - Activity E will take 3 weeks, so the earliest it can finish is week 7

6.10 The Forward Pass

The Calculation of Earliest Start Date [3/4]

- Activity G cannot start until both E and F have been completed
 - It cannot start until week 10 - the later of weeks 7 (activity E) and 10 (for activity F)
 - It takes 3 weeks and finishes in week 13
- Similarly, activity H cannot start until week 9 - the later of the two earliest finished dates for the preceding activities C and D

6.10 The Forward Pass

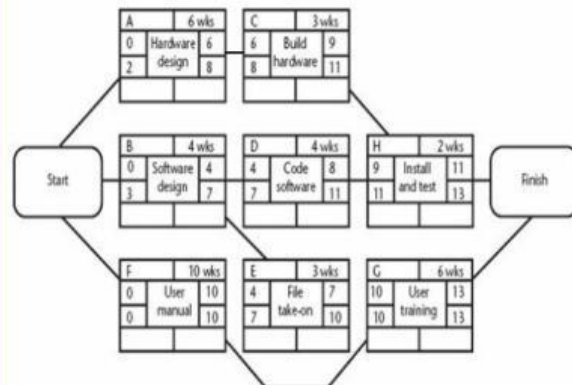
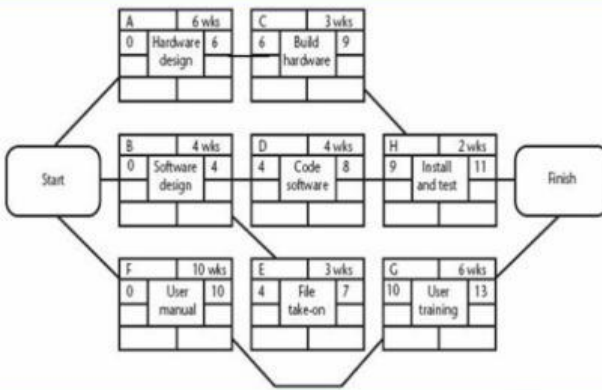
The Calculation of Earliest Start Date [4/4]

- The project will be complete when both activities H and G have been completed
 - The earliest project completion date will be the later of weeks 11 and 13 - that is, week 13

6.11 The Backward Pass

The Latest Activity Dates Calculation [1/3]

- The latest completion date for activities G and H is assumed to be week 13
- Activity H must therefore start at week 11 at the latest (13-2) and the latest start date for activity G is week 10 (13-3)
- The latest completion date for activities C and D is the latest date at which activity H must start - that is week 11
 - The latest start date of week 8 (11-3), and week 7 (10-3) respectively



6.11 The Backward Pass The Latest Activity Dates Calculation [2/3]

- Activities E and F must be completed by week 10
 - The earliest start dates are weeks 7 (10-3) and 0 (10-10) respectively
- Activity B must be completed by week 7 (the latest start date for both activities D and E)
 - The latest start is week 3 (7-4)

6.11 The Backward Pass The Latest Activity Dates Calculation [3/3]

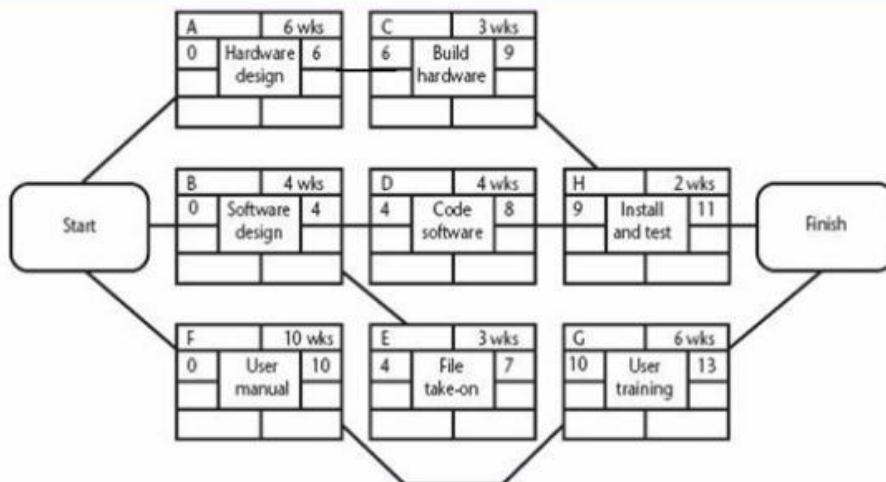
- Activity A must be completed by week 8 (the latest start date for activity C)
 - Its latest start is week 2 (8-6)
- The latest start date for the project start is the earliest of the latest start dates for activities A, B and F
 - This week is week zero
 - It tells us that if the project does not start on time it won't finish on time

6.12 Identifying The Critical Path The Critical Path [1/3]

- **Critical path:** One path through the network that defines the duration of the project
- Any delay to any activity of this critical path will delay the completion of the project

6.12 Identifying The Critical Path The Critical Path [2/3]

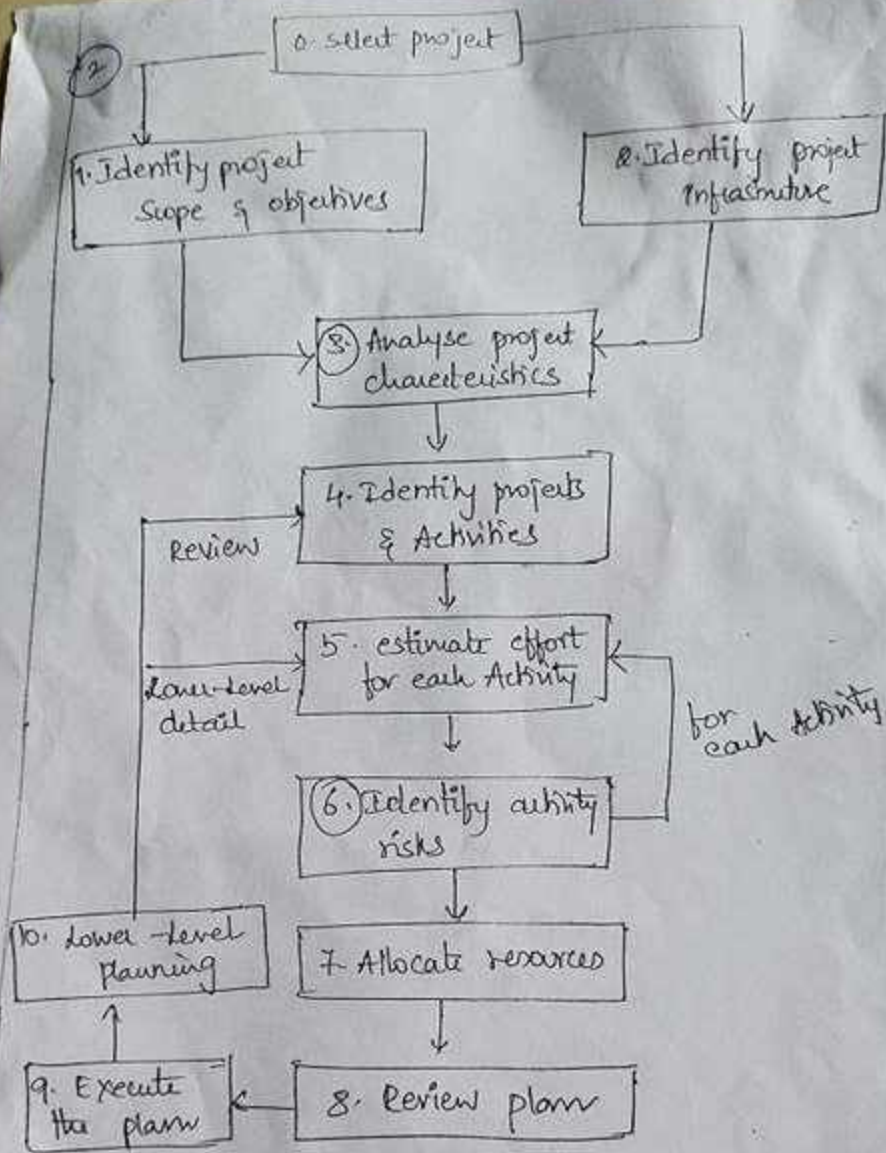
- Activity's *float*: the difference between an activity's earliest start date and its latest start date (or, equally, the difference between its earliest and latest finish dates)
 - A measure of how much the start date or completion of an activity may be delayed without affecting the end date of the project
- Activity *span*: the difference between the earliest start date and the latest finish date
 - Measure of maximum time allowable for the activity



Introduction

Risk:-

- Risk is an uncertain event that occurs and affects the project.
- The key elements of risks are budget, staff & technology
- Budget relates to the future uncertainty. If the cost was under-estimated, it leads to the loss in the project
- If the new technology implemented is difficult to use, it leads to loss.
- untrained staff or inexperienced staff results in low productivity which leads to risks.
- The causes of risks are:-
 - * staff inexperience
 - * Inaccurate estimation
 - * uncertain requirements taken.
 - * poor specifications
- The effects results in project scope & time increment delay in getting changes, testing taking long period budget increases etc.
- The project is planned initially basing on assumpt and the risk management tries to plan & control the situations where those assumptions become incorrect.



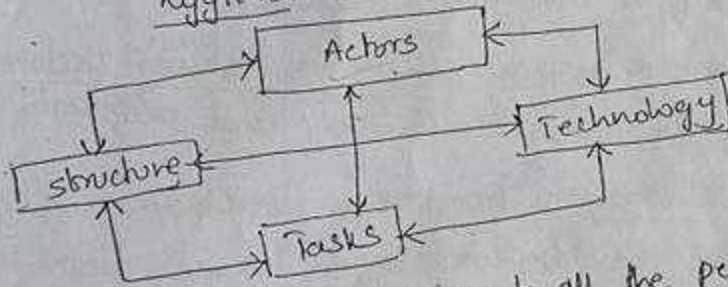
Risk planning is carried out in step 3 & step 6.

① Categories of Risk

③

- Risks are specified based on deadlines like time, budget and quality.
- Risks that occur after the successful implementation of application is business failure.
- for ex: an e-commerce site is established to sale a product, the site may be correctly implemented, but customers fail to use the site because of demand of the price, this is called business risk.
- Kalle Lyytinen & his colleagues proposed diagramatic representation of risks.

Lyytinen-Mathiasen-Ropponen Risk Framework.



- The box labelled 'Actors' refers to all the people involved in development of application.
- The risk in this area is inexperienced/untrained staff or high staff turnover leads to the project being lost.
- The next box labelled 'Technology' is used to implement the application.
- The risk in this area is to know the appropriateness of technology & to find possible faults if they are novel (new).

- (4) - The next box is 'Structure' which describes the management structure and system.
- It includes effective planning & control.
 - The risk in this area is sometimes implementation may need user participation in some tasks, but the responsibility of managing the users contribution leads to confusion & appropriate planning is needed.
- The last box is 'Tasks' which relates to the work planned.
- The risk in this area is the complexity of work leads to delays because of the additional time required to integrate large number of components.
 - Here all the boxes are interlinked.
 - Risks arises from relationships between factors for example between technology and people.
 - If technology is novel then developers (actors) might not be experienced in its use and delay results (task), so all are interrelated here.

④ Risk Identification - Boehm's Top 10 Risks

The 2 main approaches to the identification of risks are use of checklists and brainstorming.

software project risks and strategies for risk reduction

<u>Risk</u>	<u>Risk Reduction techniques</u>
1. Personnel shortfalls	Staffing with top talent, team building, early scheduling of staff, training and development.
2. Unrealistic time & cost estimates	Estimation techniques, design to cost, analysis of past projects, incremental development.
3. Developing wrong SW functions	Improved SW evaluation, <u>checking the specification</u> methods, prototyping.
4. Developing wrong user interface	prototyping, task analysis, user involvement.
5. Gold plating	Requirements <u>scrubbing</u> , prototyping, cost-benefit analysis, design to cost.
6. Late changes to requirements	strict <u>change</u> procedures, incremental development.
7. Shortfall in extremely supplied components	Benchmarking, inspection, quality assurance procedure, formal specifications, contractual agreements.

- | | | |
|----|--|---|
| 8 | shortfall in extremely performed tasks | prototyping, competitive design, contract incentives, quality assurance procedures |
| 9 | Real-time performance Shortfalls | Simulation, benchmarking, prototyping, technical analysis. |
| 10 | Development technically too difficult | Technical analysis, cost-benefit analysis, staff training & development, prototyping. |

Here we identify the problems of the project & steps are taken to resolve them.

This checklist can be used to the new projects.

③ Risk Assessment

- The common problem with risk assessment is the list of risks is potentially endless.
- The estimation of risk for each risk is given by the formula

$$\text{risk exposure} = (\text{potential damage}) \times (\text{probability of occurrence})$$

- potential damage is assessed as money value.
- for ex:- It is estimated if fire occurs, the new computer configuration could be established for \$500,000. It is estimated that there is a chance of

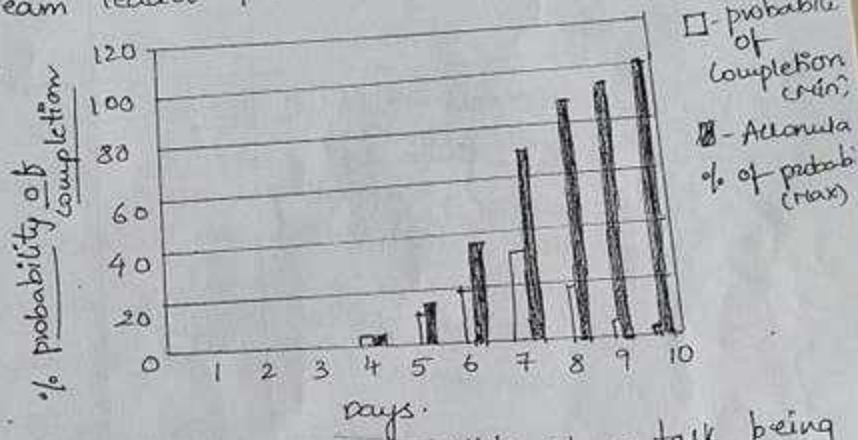
1 in 1000 for the chance of fire accident. so, the probability would be 0.001.

The risk exposure in this case would be:

$$\$500,000 \times 0.001 = \$500.$$

→ probability chart

The team leader produces a probability chart as shown below:



The above fig shows the probability of a task being completed in four days is 5%, five days is 15%, seventh day is 70% and so on.

- At eight day the task being completed will be 85%, showing the probability of failure 15%.

- Here the loss is measured in days rather than money

→ Amanda's Risk Exposure Assessment

- The Amanda's Group accounts project has been done.

- The table is taken showing the importance of risks as Hazard (threat, danger), likelihood (probability), impact (effect)

3 and Risk of the project

Table 1: Amanda's Risk Exposure Assessment

SNO	Hazard	likelihood	Impact	Risk
R1	changes to requirements specifications during coding	8	8	64
R2	Specification takes longer than expected	3	7	21
R3	staff sickness affecting critical path activities	5	7	35
R4	staff sickness affecting non-critical activities	10	3	30
R5	Module coding takes longer than expected	4	5	20
R6	Module testing demonstrates errors	4	8	32

Table 2: Qualitative descriptors of risk probability and associated range values.

SNO	probability level	Range
1	High	Greater than 50% chance of happening
2	Significant	30% - 50% chance of happening
3	Moderate	10% - 30% " "
4	Low	Less than 10% " "

\$ 500,000. It is estimated

→ qualitative descriptors of impact on cost and associated range values (4)

table 3

Impact level	Range
High	More than 30% above budget expected
Significant	20% to 30% above budget expected
Moderate	10% to 20% above budget expected
Low	within 10% of budget expected

The above table (3) shows the associated qualitative description with range of values

- The potential damage is categorized in terms of its impact on project cost & its range.
- in terms of risk and its range.

→ Impact Matrix

- The impact matrix has the grids, the cells in the in the top right of matrix are zoned by tolerance line

				R1
High				
Significant		R4, R3		
Moderate		R2		R5
Low				
	Low	Moderate	Significant	High
	<u>probability</u>			

In the table 1, R5- refers to module coding take

- 10 longer than expected. This impacts on duration of the project and cost as more staff time is needed.
- The probability of this risk would be moderate, if it occurs its impact would be significant as shown in the impact matrix table above.
 - In the same manner the risks R1, R2, R3, R4 are shown in the matrix.

(4) Risk Planning

The major tasks in Risk planning are:

- ↳ Risk Acceptance
- ↳ Risk Avoidance
- ↳ Risk Reduction & Mitigation
- ↳ Risk Transfer

Risk Acceptance:

- This is do nothing option.
- We prioritize the risks and ignore low priority risk & concentrate more likely & damaging risks.
- Calculate the cost of action and reduce the probability of risk happening.

Risk Avoidance:

- Some activities may be prone to accident, it is best to avoid them altogether. If you are worried abt sharks then don't go into water.
- In slw, give all problems with solutions from shard

----- it is assumed

of

(5)
- Managers decide to retain methods of existing or buy an off-the-shelf solution.

Risk Reduction:

- Now go ahead with action despite the risks, take precautions to reduce the probability of risk.

- For ex: If two staff are scheduled to work for the development of project and if this is identified as a risk, steps are taken to reduce the risk, here developers might be promised to be paid bonus on successful completion of the project.

- Richard Fairley propose four COTS - Commercial off-the-shelf software acquisition risks shown in the table below

Integration

upgrading

No source code

Supplier failures

difficulties in integrating data formats of different applications.

when supplier upgrades the package, the package may no longer meet the user requirements.

If you want to enhance the system, you might not be able to do so as you do not have access to source code.

The supplier of the application might go out of business or brought by a rival one.

12

- keeping these fairly four risks in mind, Risk reduction attempts are made ~~the res~~ to reduce the occurrence of risk.

- Risk ^{mitigation} ~~negotiation~~ is the action taken to ensure the impact of data corruption.

- Taking regular back ups of data storage would reduce the impact of data corruption. To reduce the risk it is the technique implemented.

Risk transfer:

- Here risk is transferred to another person or another organization.

- with s/w projects, an example of this would be the software development task is outsourced to an outside company for a fixed fee.

- here you might expect to quote a higher figure to cover the risk that ~~the~~ project takes longer than the 'expected average' time.

- on the other hand, a well established organization might have productivity advantage as it has well experienced developers.

- The need to compare with other s/w development specialists & this competition tends to drive the prices down.

⑤ Risk Management

13

The 3 concepts / cases of Risk Management are:

↳ Contingency.

↳ Deciding on risk actions.

↳ Creating & maintaining risk register.

→ Contingency: contingency should plan
what if work

- Contingency is a plan or action to be carried out if particular risk materializes.

- For ex:- If a team member involved in urgent work was ill, then the project manager should draft in another member of staff to cover that work.

- This is called contingency plan. This is a planned action to be carried out if particular risk materializes.

→ Deciding on risk actions:

- For each risk, specific actions have to be planned.

- Risk exposure value can be calculated as:

$(\text{Value of damage}) \times (\text{probability of occurrence})$

- Cost-effectiveness of risk reduction action is assessed by calculating risk reduction leverage (RRL)

$$\text{risk reduction leverage} = \frac{RE_{\text{before}} - RE_{\text{after}}}{\text{cost of risk reduction}}$$

- RE_{before} is the risk exposure before risk reduction actions have been taken.

- RE_{after} is the risk exposure after taking risk reduction actions

17

- For Ex: It might take \$200,000 to replace ^{new} configuration used to develop sw application.
- There is 1% chance of fire. As the risk exposure would be 1% of \$200,000 it is \$2,000.
 - Installing fire alarms at a cost of \$500 would reduce the chance of fire to 0.5%.
 - The new risk exposure would be \$1,000 i.e. the reduction of \$1,000 on the previous exposure.
 - The RRL would be $(2000 - 1000) / 500$ that is 2.0
 - In the above case, the premium you pay to cover against fire for insurance is reduced from \$2,000 to \$1,000 if you install fire alarms.
 - As fire alarms would cost you only \$500 & save \$1,000 & the cost would clearly be worthwhile.

→ creating and maintaining Risk Registers

- When project planners want to examine the most threatening risks to the project, they need to record their findings in the risk registers.
- The typical content of such register is shown in the fig below.
- After work starts on the project, more risks will emerge and will be added to the register.
- Many risks just threaten two or three activities when project staff have completed these risks then it is be 'closed' and no longer relevant.

15

(3)

RISK RECORD

Risk id	Risk title				
owner	Date raised	Status			
Risk description					
Impact description					
Recommended Risk Mitigation: (Action)					
probability / Impact values					
	probability	Impact			
		cost	duration	quality	
Pre-mitigation					
Post-mitigation					
Incident / Action history (previous project)					
Date	Incident / Action	Actor	outcome / comment		

Risk Register Page:

- (6) PERT technique (Program Evaluation and Review) technique.
- PERT was developed to estimate task durations
 - It was developed in expensive & high-risk environment
 - PERT requires 3 estimates:
 Most likely time, optimistic time, pessimistic time.

10

Most likely time: time we would expect the task would take under normal circumstances. It is denoted by m .

Optimistic time: the shortest time we would expect the task would take, it is denoted by a .

Pessimistic time: the max or the worst possible time that we would expect the task would take, it is denoted by b .

PERT combines these 3 estimates to form a single expected duration t_e , using the formula

$$t_e = \frac{a + 4m + b}{6}$$

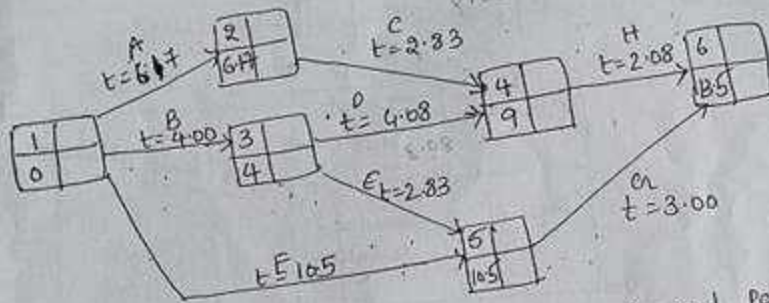
The table below shows the following estimates a, b, m and calculate the expected duration t_e for each activity.

Activity	optimistic (a)	Activity duration (weeks) Most likely	pessimistic (b)
A	5	6	8
B	3	4	5
C	2	3	3
D	3.5	4	5
E	1	3	4
F	8	10	15
G	2	3	4
H	2	2	2.5

PERT Network

17

- The PERT network illustrated that we expect the project to complete in 13.5 weeks
- In fig below we used an activity-on-arrow network to make the representation easier.
- This is also called as activity-on-node diagram.



The PERT network after the forward pass.

- The expected duration t_e is calculated for the following

activities:

for activity A: $t_e = \frac{a+4m+b}{6} = \frac{5+4(6)+8}{6} = 6.17$

Activity B: $t_e = \frac{a+(4 \times m)+b}{6} = \frac{17}{6} = 2.83$

Activity B: $t_e = \frac{3+(4 \times 4)+5}{6} = \frac{24}{6} = 4$

Activity D: $t_e = \frac{3.5+(4 \times 4)+5}{6} = \frac{24.5}{6} = 4.08$

Activity E: $t_e = \frac{1+(4 \times 3)+4}{6} = \frac{17}{6} = 2.83$

19

$$\text{Activity F: } t_e = \frac{8 + (4 \times 10) + 15}{6} = \frac{63}{6} = 10.50$$

$$\text{Activity G: } t_e = \frac{2 + (4 \times 3) + 4}{6} = \frac{18}{6} = 3$$

$$\text{Activity H: } t_e = \frac{2 + (4 \times 2) + 2.5}{6} = \frac{12.5}{6} = 2.08$$

- The pert network represented the numbers and the completion dates in the fig.

- PERT generally consists of four values,

Number	Target date
Expected date	Standard deviation

Pert basic structure

- The advantage of this approach is, it places an emphasis on the uncertainty of the real world.

- Rather than saying "The completion date for the project is", PERT leads us to say "we expect to complete the project by".

→ standard deviation: A quantitative measure of the degree of uncertainty is obtained by calculating the standard deviation 's' using the formula:

$$s = \frac{b-a}{6}$$

- The activity standard deviation is proportional to the difference between pessimistic time (b) & optimistic time (a).

Calculating standard deviations for Activities

for Activity A: $A \Rightarrow S_A = \frac{b-a}{6} = \frac{8-5}{6} = 0.50$

Activity B: $B \Rightarrow S_B = \frac{5-3}{6} = \frac{2}{6} = 0.33$

This calculation is performed on all the activities.

The table below shows the expected time & standard deviations

Activity	optimistic time (a)	Most likely time (M)	Pessimistic time (b)	Expected time (te)	standard deviations (s)
A	5	6	8	6.17	0.50 ✓
B	3	4	5	4	0.33
C	2	3	3	2.33	0.17 ✓
D	3.5	4	5	4.08	0.25
E	1	3	4	2.33	0.50
F	3	10	15	10.05	1.17
G	2	3	4	3	0.33
H	2	2	2.5	2.08	0.08

- There is a small difference to add two standard deviations, we must add their squares and find the square root of their sum.

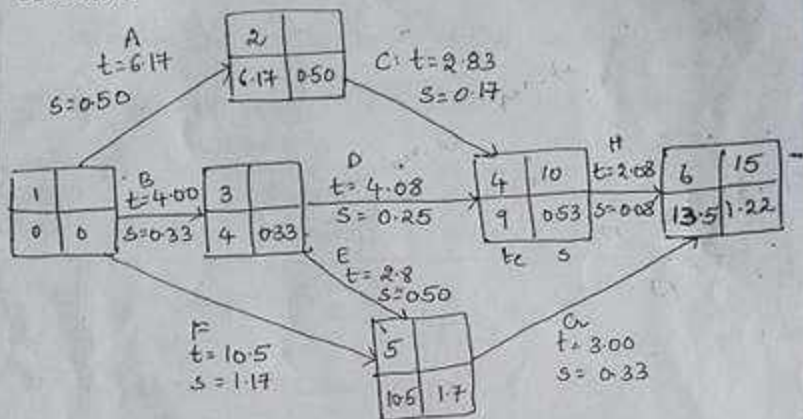
- calculating standard deviations for two activities:

- for Activities A & C: $\sqrt{(0.50)^2 + (0.17)^2} = \sqrt{0.25 + 0.0289}$
 $= \sqrt{0.278} = 0.53$

For Activity B and D:-

$$z = \frac{\sqrt{(0.33)^2 + (0.25)^2}}{\sqrt{1.08 + 0.0625}} = \frac{\sqrt{0.1705}}{\sqrt{1.1425}} = 0.41$$

- The fig below shows PERT network including standard deviations



- We must now calculate the target date i.e. we must complete the project within the target date.

- The target date is obtained by adding expected date and standard deviation.

- In the abv ex, the project takes maximum of 15 weeks i.e. the target date. $T = 13.5(t_e) + 1.22(s) = 14.72$

→ Missing target (Z):-

- The pert technique uses 3 steps for calculating the probability of missing the target date.

- calculate standard deviations, expected date, target date for each activity.

- (10)
- Calculate z for activity with the above mentioned.
 - To calculate z value the formula is $z = \frac{T - t_e}{s}$

$$z = \frac{T - t_e}{s}$$

- For Activity 4 $z = \frac{10 - 9}{0.53} = \frac{1}{0.53} = 1.8867$

(7) Monte-carlo Approach / Monte Carlo Simulation

- Monte Carlo is a general analysis technique used to solve any problem which is complex and involves uncertain parameters.
- Monte Carlo Simulation involves repeated random sampling to compute the result.
- Since it is based on repeated computations of random numbers it becomes more easier.
- It is used to analyse the risk of not meeting the project deadline.
- The Activity duration can be specified in variety of forms depending upon the information available.
- MCS calculates the results repeatedly.
- Each time it uses different set of random values for each function.
- So, MCS involves thousands of calculations depending upon the number of parameters of the project.
- After the results are available, they are analysed &

represented graphically as histograms shown in the below.

-Steps involved in carrying out MCS are:-

Step 1:- set activities be 'n' and completion time in terms of duration is $(x_1=1, x_2=2, x_3=3, \dots, x_n=n)$

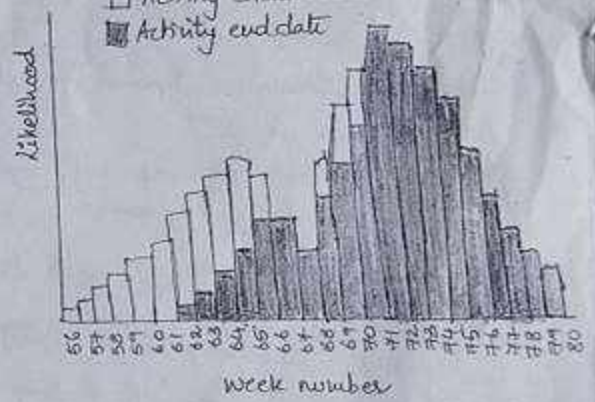
Step 2:- Generate set of random inputs as $x_{11}, x_{12}, x_{13}, \dots, x_{1n}$ and

Step 3:- Evaluate the result and store (u result 'd')

Step 4:- Repeat the steps 2 and 3 number of times.

Step 5:- Analyse the result & display in histogram.

- Activity start date
- Activity end date



RISK profile for an activity generated using Monte Carlo simulation.

..., expected date, target date for each activity.

UNIT V

PROJECT MONITORING & CONTROL, RESOURCE ALLOCATION

Monitoring and control processes continually track, review, adjust and report on the project's performance. It's important to find out how a project's performing and whether it's on time, as well as implement approved changes. This ensures the project remains on track, on budget and on time.

What is project control?

According to [*the PMBOK® Guide*](#) (the Project Management Body of Knowledge), project control is a “project management function that involves comparing actual performance with planned performance and taking appropriate corrective action (or directing others to take this action) that will yield the desired outcome in the project when significant differences exist.”

Essentially, project controls are a series of tools that help keep a project on schedule. Combined with people skills and project experience, they deliver information that enables accurate decision making. The project control process mainly focuses on:

- Measuring planned performance vs actual performance.
- Ongoing assessment of the project's performance to identify any preventive or corrective actions needed.
- Keeping accurate, timely information based on the project's output and associated documentation.
- Providing information that supports status updates, forecasting and measuring progress.
- Delivering forecasts that update current costs and project schedule.
- Monitoring the implementation of any approved changes or schedule amendments.

Importance of project monitoring and control

Monitoring and control keeps projects on track. The right controls can play a major part in completing projects on time. The data gathered also lets project managers make informed decisions. They can take advantage of opportunities, make changes and avoid crisis management issues.

Put simply, monitoring and control ensures the seamless execution of tasks. This improves productivity and efficiency.

Monitoring and control method

When setting up a project's monitoring and control process, first establish the project baselines. This includes the scope, schedule and budget. Use this information to benchmark the project's progress throughout the lifecycle.

Use a Work Breakdown Structure (WBS) to break a project down into small units of work, or sub-tasks. This makes the work easier to manage and evaluate. This enables easier detection of issues, keeps the project under control and allows for easier progress verification. It also helps prevent team members from feeling overwhelmed.

With a WBS in place, follow this sequence throughout the project's lifecycle:

Monitoring and control techniques

There are a range of monitoring and control techniques that can be used by project managers, including:

A Requirements Traceability Matrix (RTM). This maps, or traces, the project's requirements to the deliverables. The matrix correlates the relationship between two baseline documents. This makes the project's tasks more visible. It also prevents new tasks or requirements being added to the project without approval.

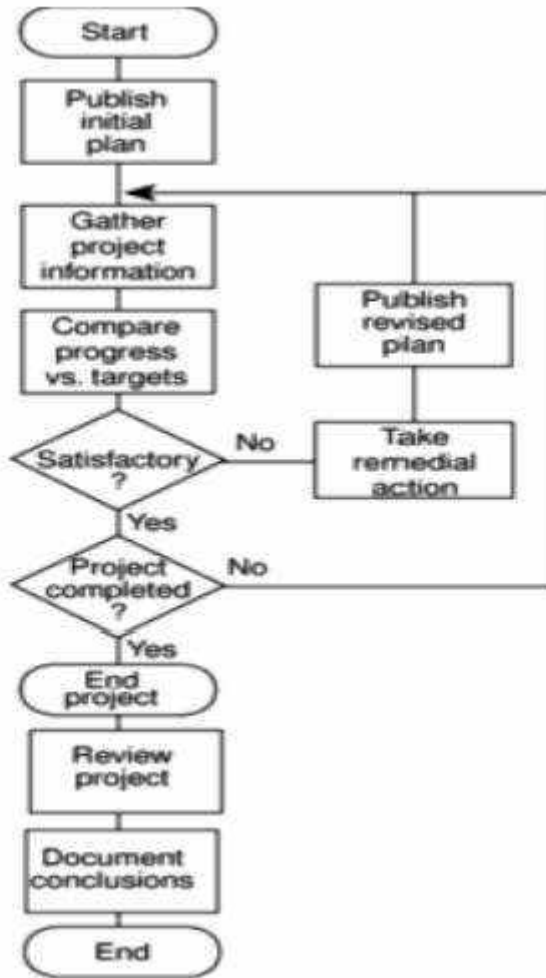
This makes the project's tasks more visible. It also prevents new tasks or requirements being added to the project without approval.

A control chart monitors the project's quality. There are two basic forms of control chart – a univariate control chart displays one project characteristic, while a multivariate chart displays more than one.

Review and status meetings further analyse problems, finding out why something happened. They can also highlight any issues that might happen later.

1. Creating framework

Project control cycle



Responsibility

Project steering committee

▪

Project board

▪

Reporting formal or informal

Project reporting structures.

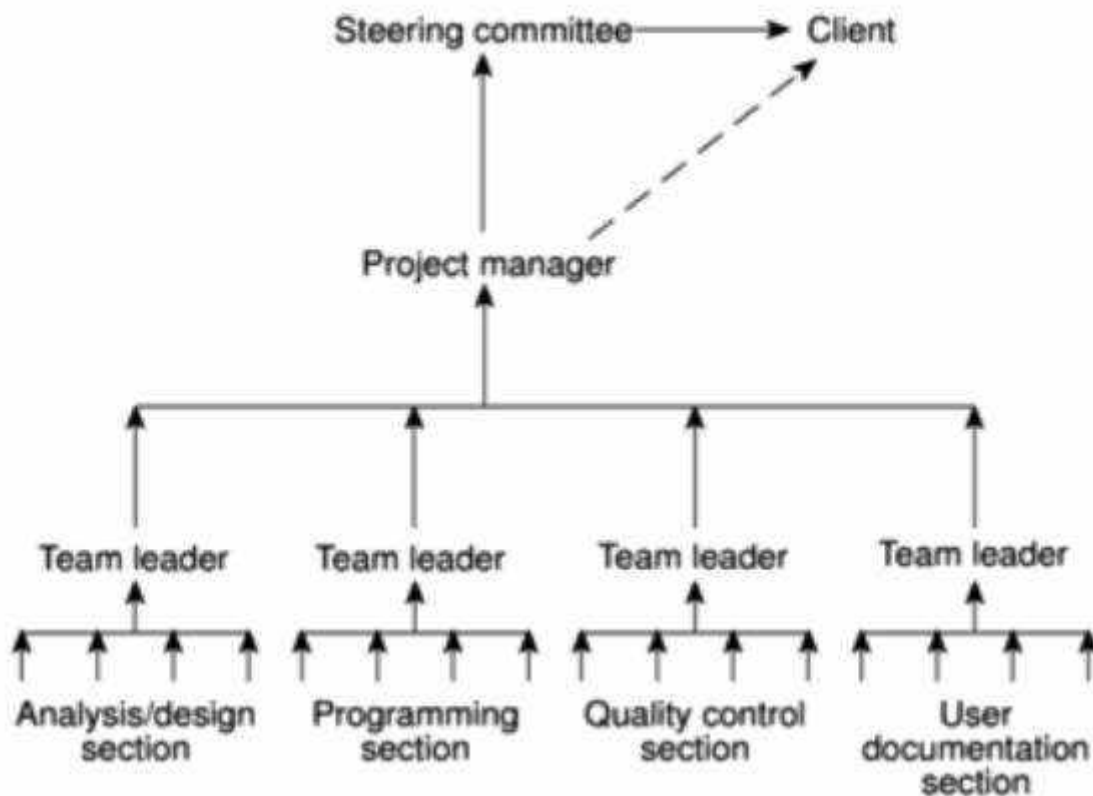
Reporting may be oral or written, formal or informal, or regular or ad hoc and some examples of each type are given in Table 9.1. While any effective team leader or project manager will be in touch with team members and available to discuss problems, any such informal reporting of project progress

must be complemented by formal reporting procedures - and it is those we are concerned with in this chapter.

Assessing progress

Progress assessment will normally be made on the basis of information collected and collated at regular intervals or when specific events occur. Wherever possible, this information will be objective and tangible - whether or not a particular report has been delivered, for example. However, such end-of-activity deliverables might Short, Monday morning team progress meetings are a common way of motivating staff to meet short term targets.

not occur sufficiently frequently throughout the life of the project. Here progress assessment will have to rely on the judgement of the team members who are carrying out the project activities.



Setting checkpoints

Regular

-

Tied to specific events

Setting checkpoints

It is essential to set a series of checkpoints in the initial activity plan. Checkpoints may be:

- tied to specific events such as the production of a report or other deliverable.

Taking snapshots

Review points or control points

-

Assess progress daily

The frequency with which a manager needs to receive information about progress will depend upon the size and degree of risk of the project or that part of the project under their control. Team leaders, for example, need to assess progress daily (particularly when employing inexperienced staff) whereas project managers may find weekly or monthly reporting appropriate. In general, the higher the level, the less frequent and less detailed the reporting needs to be.

There are, however, strong arguments in favour of formal weekly collection of information from staff carrying out activities. Collecting data at the end of each week ensures that information is provided while memories are still relatively fresh and provides a mechanism for individuals to review and reflect upon their progress during the past few days.

Major, or project-level, progress reviews will generally take place at particular points during the life of a project - commonly known as review points or control points. PRINCE 2, for example, designates a series of checkpoints where the status of work in a project or for a team is reviewed. At the end of each project Stage, PRINCE 2 provides for an End Stage Assessment where an assessment of the project and consideration of its future are undertaken.

COLLECTING DATA

- Partial completion reporting
- Risk reporting

PARTIAL COMPLETION REPORTING

Time Sheet						
Staff <u>John Smith</u>			Week ending <u>26/3/99</u>			
Rechargeable hours						
Project	Activity code	Description	Hours this week	% Complete	Scheduled completion	Estimated completion
P21	A243	Code mod A3	12	30	24/4/99	24/4/99
P34	B771	Document take-on	20	90	1/4/99	29/3/99
Total recharged hours:			32			
Non-rechargeable hours						
Code	Description	Hours	Comment & authorization			
z99	day in lieu	8	Authorized by RB			
Total non-rechargeable hours:			8			

RISK REPORTING

Red/Amber/Green (RAG) reporting

- Identify key tasks
- Break down into sub-tasks
- Assess subtasks as:
 - Green** – ‘on target’
 - Amber** – ‘not on target but recoverable’
 - Red** – ‘not on target and recoverable only with difficulty’
- Status of ‘critical’ tasks is particularly important

- “Traffic light technique’ for risk reporting followed by IBM
- Step 1: Identify the first level (higher level) key elements to assess the work
- Step 2: Break down the first level key elements to second level elements
- Step 3: Judge each of the second level element’s progress in 3 scales as below.

Green(G)	– As per target
Amber (A)	– Not as per target but can be brought back to control
Red (R)	– Not as per target and cannot be brought back to control without involving additional cost/resource/time
- Step 4: Based on the second level assessment, judge the first level on the same 3 point scale (Green/Amber/Red)
- Step 5: Review all the first level assessments to decide on the overall assessment of the project.

Risk Assessment Report

Name: _____

Dated: _____

Project Name: _____

Ref: _____

Project Risk Level : R

First Level Activity-Risk Level							
Week No.	10	11	12	13	14	15	16
First level activity risk assessment	G	G	A	A	R		
Second level activity-Risk level							
a)Screen handling	G	G	G	A	A		
b)DB update	G	G	A	G	A		
c)Feedback message	G	A	G	G	G		
d)Compilation	G	G	G	A	R		
e)Test Run	G	G	A	A	R		
f)Documentation	G	A	A	A	R		

VISUALIZING PROGRESS

- **The Gantt chart**
- **The Slip chart**
- **Ball charts**
- **The timeline**

THE GANTT CHART

WHAT IS GANTT CHART?

Gantt chart is a type of **bar chart**. In which a series of horizontal lines **shows the amount of work done** or production completed in certain period of time in relation to the amount planned for those period.

Creating a Gantt Chart:

There are two methods to creating a Gantt Chart (Maylor, 2005).

1. Using a **Forward Schedule**: starting with the list of activities and a given start date (6th Sept in previous example) follow them forwards in time until you hit given deadline.
2. Using a **Backward Schedule**: look at the deadline, from that date work in the logical list of activities.

Both of these methods allow you to ensure that all necessary activities can possibly be completed within the given project time frame.

Steps to Creating a Gantt Chart:

1. Determine Project start date and deadline.
2. Gather all information surrounding the list of activities within a project – the Work Breakdown Structure may be useful for this.
3. Determine how long each activity will take
4. Evaluate what activities are dependant on others
5. Create Graph shell including the timeline and list of activities.
6. Using either **Forward Scheduling** or **Backward Scheduling**, Begin to add bars ensuring to include dependencies and the full duration for each activity.

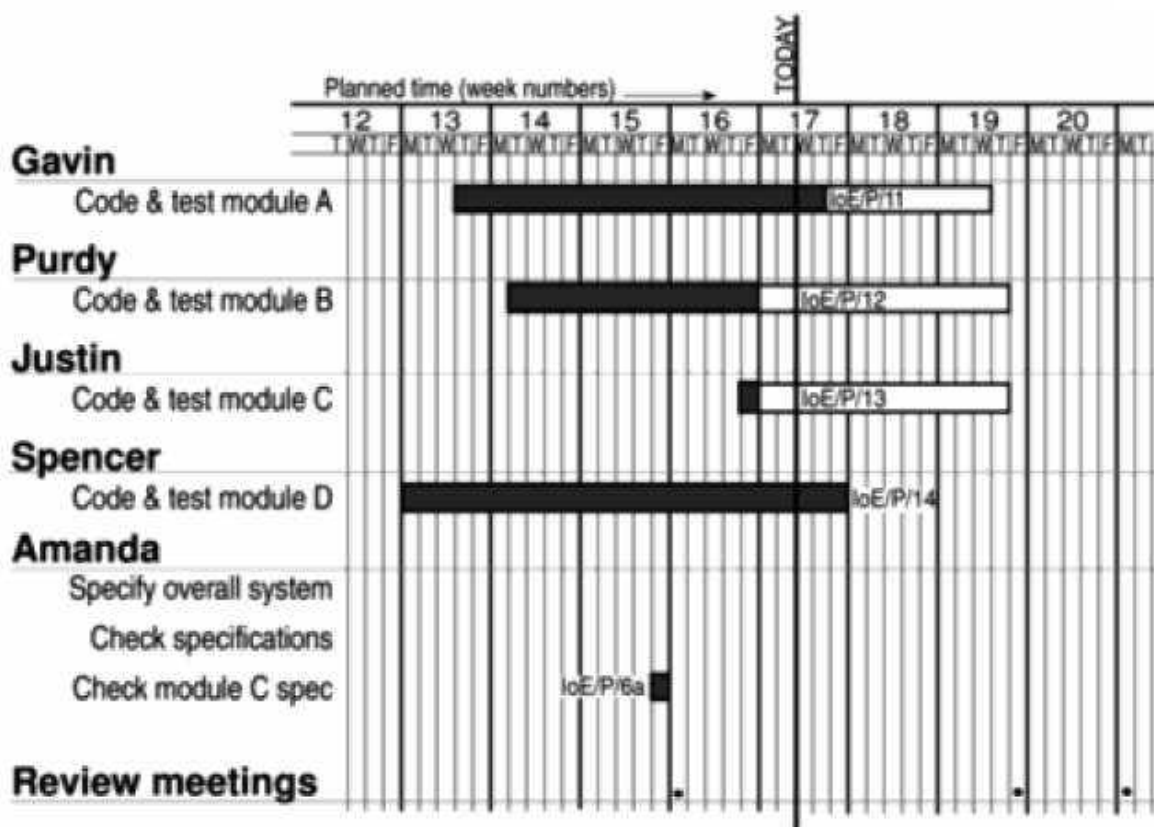


Figure 9.5 Part of Amanda's Gantt chart with the 'today cursor' in week 17.

THE SLIP CHARTTHE SLIP CHART

The Advantages:

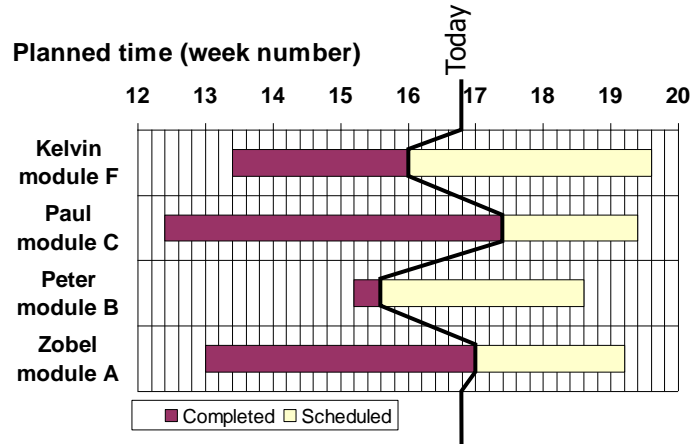
- A useful tool for displaying time-based information within a project.
- Very simple to create
- They provide a useful overview of project activities, a good starting point for project planning.
- The charts are widely used and understood.
- There exists several PC software packages that allow you to build Gantt Charts.

THE SLIP CHART

The Slip Chart

- Add a slip line on the Gantt chart
- The slip line indicates those activities that are either ahead or behind the schedule
- Too much bending indicates a need for rescheduling of the overall plan

The Slip Chart (cont'd)



BALL CHARTS

A somewhat more striking way of showing whether or not targets have been met is to use a ball chart as in Figure 9.7. In this version of the ball chart, the circles indicate start and completion points for activities. The circles initially contain the original scheduled dates. Whenever revisions are produced these are added as second dates in the appropriate circle until an activity is actually started.

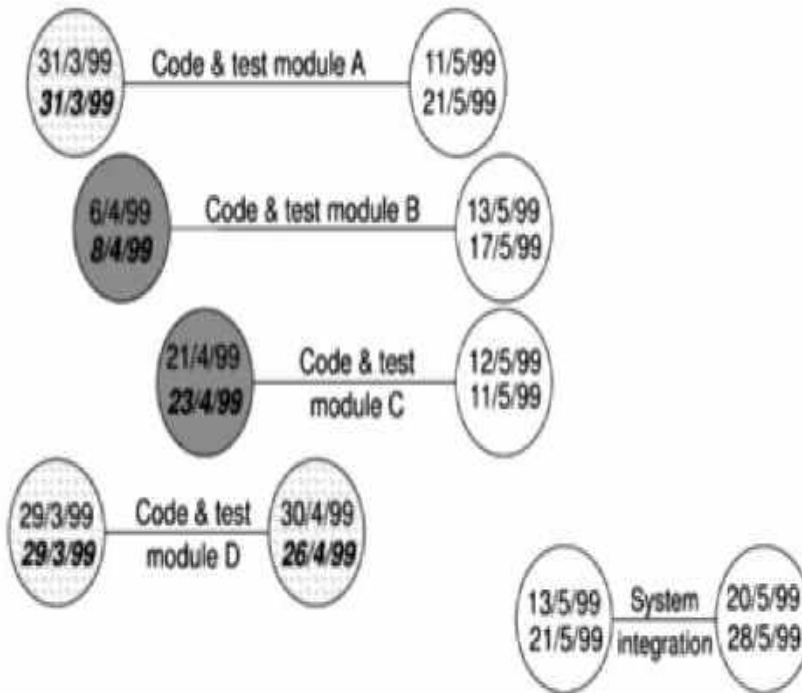
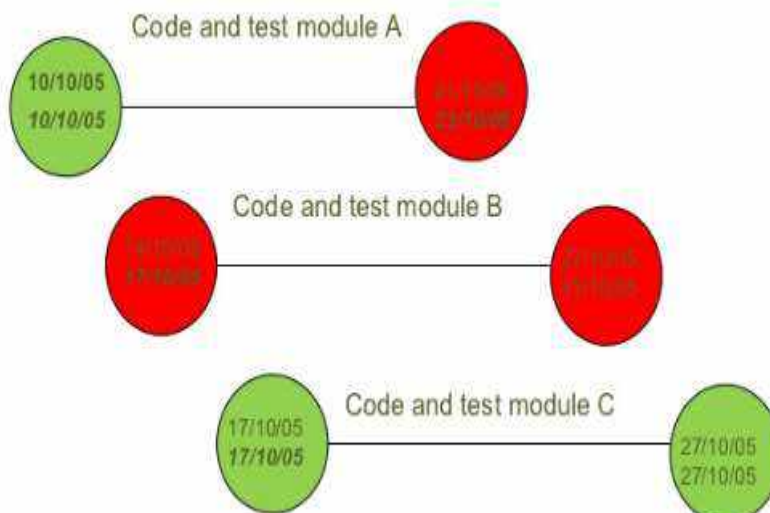


Figure 9.7 The ball wall chart provides an incentive for meeting targets.

Ball charts



Green: On time

Red: Missed the target

THE TIMELINE



The Timeline

- A plot of the elapsed time against the planned time of the activities indicating
 - the actual progress of the activities; and
 - the rescheduled activities by the end of each week
- show where and when the targets have changed through the life of a project



The Timeline (cont'd)

- Can show the slippage of the activities through the life of the project
 - The Gantt chart cannot
- Help to analyze and understand the trends and reason for changes
 - to avoid slippage in future projects

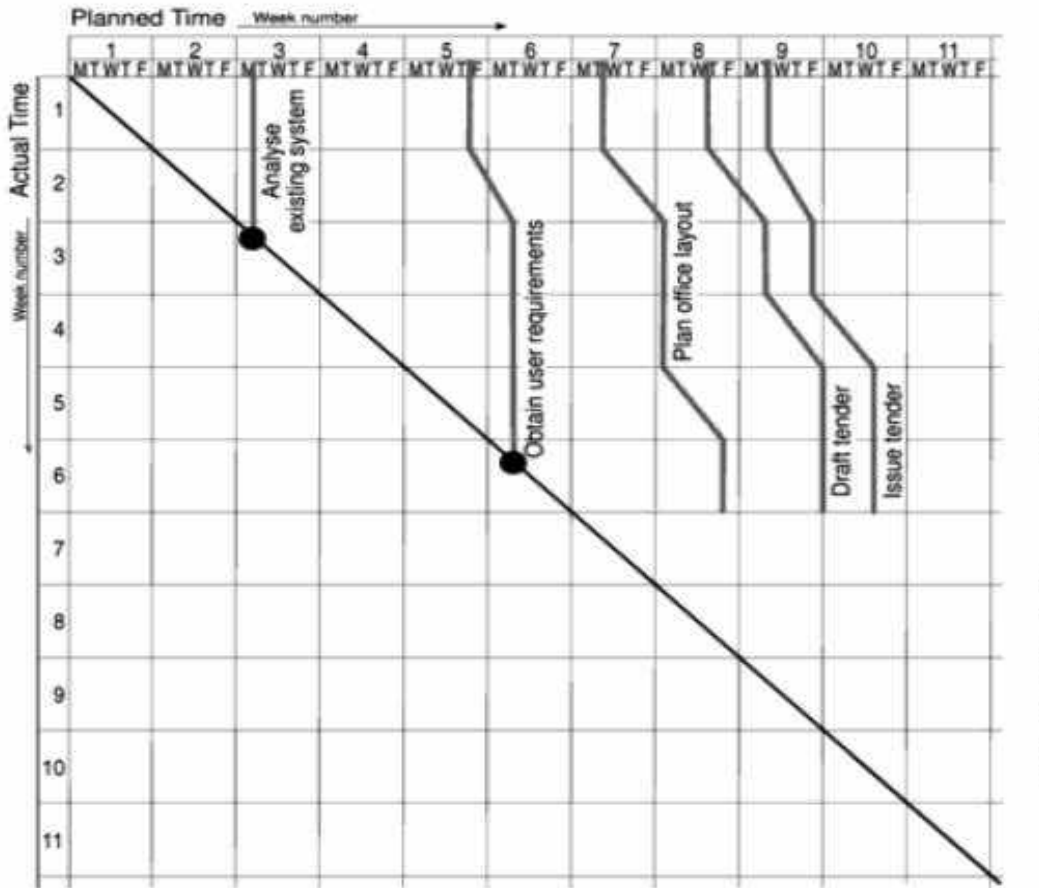
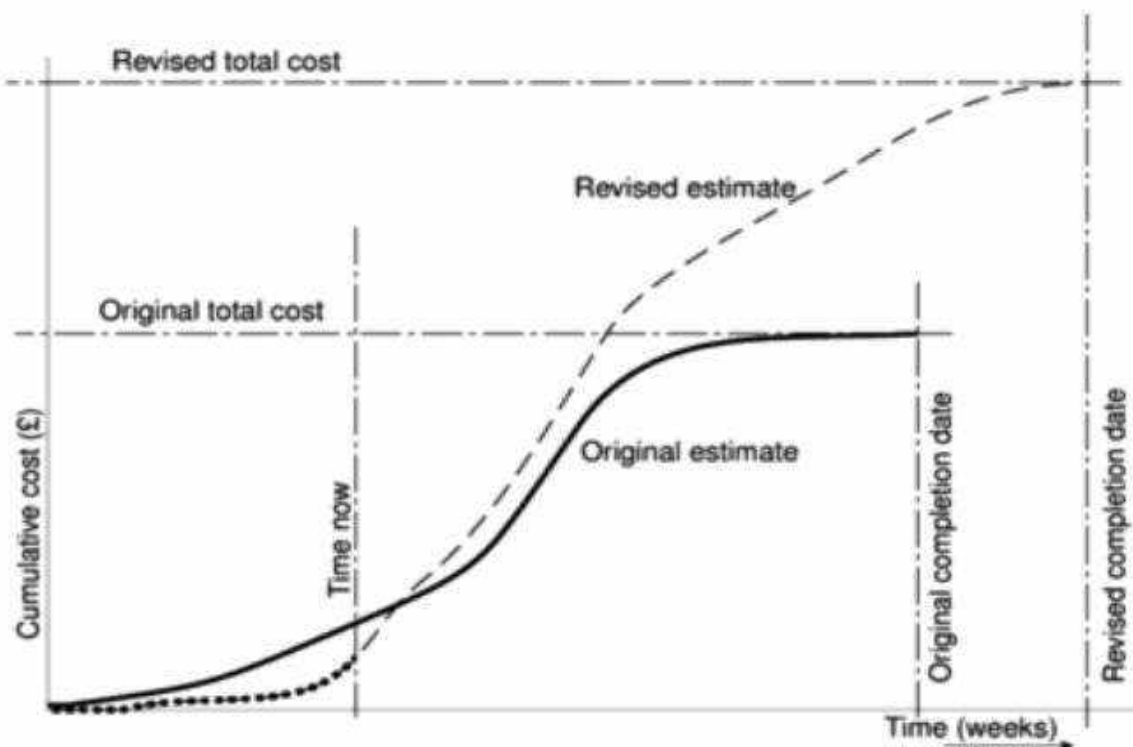


Figure 9.8 *Brigitte's timeline chart at the end of week six.*

COST MONITORING

Cost monitoring

- A project could be late because the staff originally committed, have not been deployed
- In this case the project will be *behind time* but *under budget*
- A project could be *on time* but only because additional resources have been added and so by *over budget*
- Need to monitor both *achievements* and *costs*



EARNED VALUE ANALYSIS

Objective:

- To measure the progress of an activity, deliverable and/or project by comparing the actual value to planned value, thereby indicating the probability of meeting the scope, time & cost budget of the activity, deliverable and/or project, and need for any corrective actions.
- To analyze the project performance, calculate the variance for schedule and cost and indicates where the project stands in comparison to the estimates calculated earlier for this point in time.

Many a times one could easily be on time, however may overspend, or may be on time & within budget however scope may be incomplete. In simple terms, EV analysis is better than comparing actual to planned results or by simply guessing the project status.

Earned value analysis

- *Planned value (PV) or Budgeted cost of work scheduled (BCWS)* – original estimate of the effort/cost to complete a task (compare with idea of a 'price')
- *Earned value (EV) or Budgeted cost of work performed (BCWP)* – total of PVs for the work completed at this time

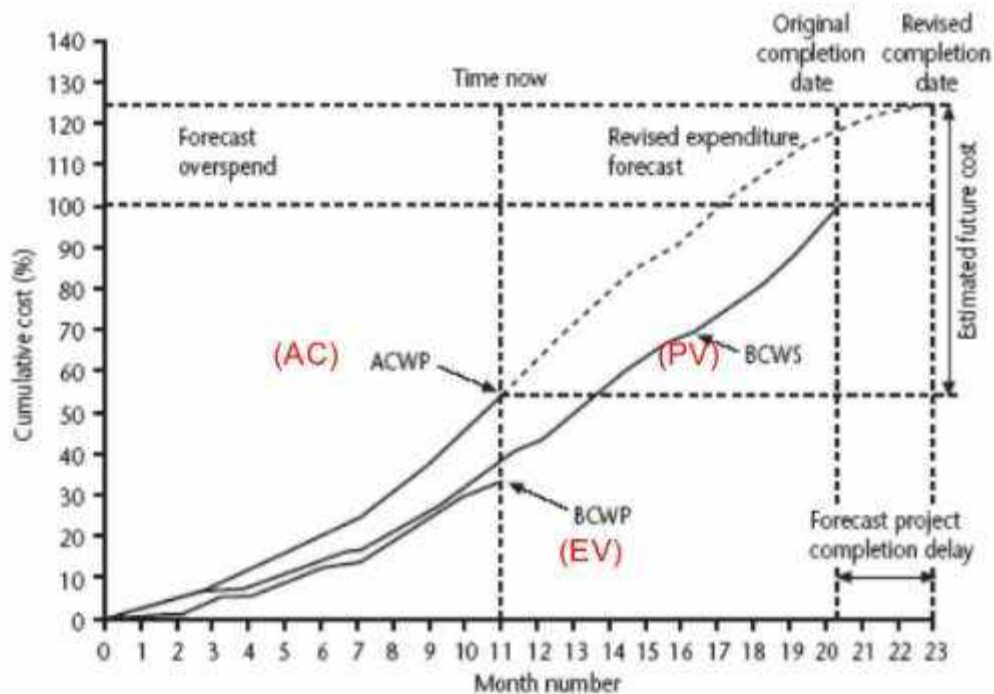
Earned value – an example

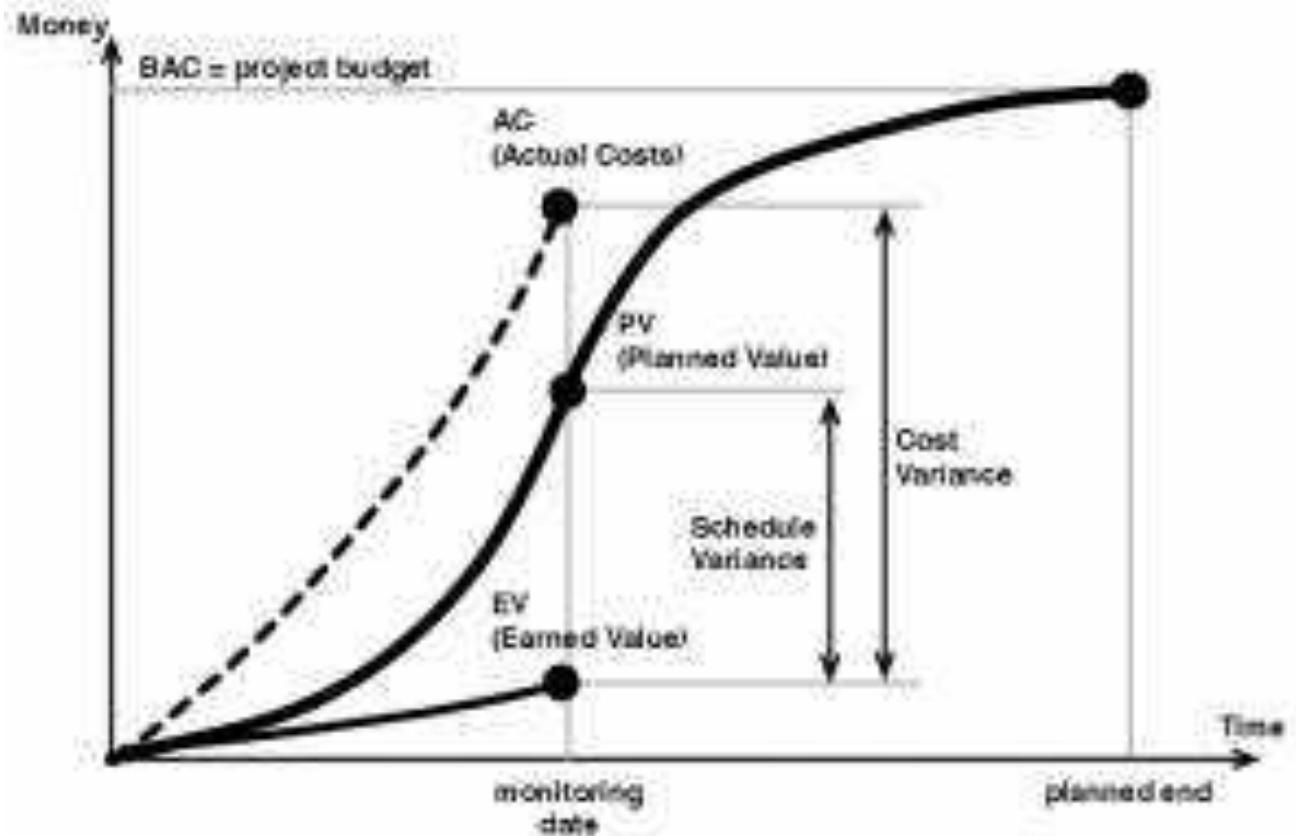
- Tasks
 - Specify module 5 days
 - Code module 8 days
 - Test module 6 days
- At the beginning of day 20, PV = 19 days
- If everything but testing completed, EV = 13 days
- Schedule variance = EV-PV i.e. 13-19 = -6
- Schedule performance indicator (SPI) = EV/PV
i.e 13/19 = 0.68

Earned value analysis – actual cost

- Actual cost (AC) is also known as Actual cost of work performed (ACWP)
- In previous example, if
 - ‘Specify module’ actually took 3 days (planned 5 days)
 - ‘Code module’ actually took 4 days (planned 8 days)
- Actual cost = 7 days
- Cost variance (CV) = EV-AC
i.e. 13-7 = 6 days
- Cost performance indicator (CPI) = EV/AC
i.e = 13/7 = 1.86
- Positive CV or CPI > 1.00 means project under budget or the work is completed better than planned

Earned value chart with revised forecasts





EARNED VALUE ANALYSIS

- The baseline budget
- Monitoring earned value
- Schedule variance(SV)
- Cost variance(CV)
- Performance ratios

Ref: <http://www.spmbook.com/downloads/slides/pdf/c03.08-09-executionmonitoringcontrol.key.pdf>
Definitions

Planned Value (PV) or Budgeted Cost of Work Scheduled (BCWS)	Originally planned cost of the work that should have been done by this time
Actual Cost (AC) or Actual Cost of Work Performed (ACWP)	Actual cost expenses on this project upto this time
Earned Value (EV) or Budgeted cost of Work Performed (BCWP)	Estimated cost of budgeted work completed
Budget at Completion (BAC)	Total budget for the project
Estimate At Completion (EAC)	Estimated final cost of the project
Estimate To Completion (ETC)	Estimated cost of the remaining work of the project

Description	EVA Formulas	Result
Cost Variance (CV)	$CV = EV - AC$	Positive value is good. Negative value unfavourable.
Schedule Variance (SV)	$SV = EV - PV$	Value below 1.0 = below par performance. Value above 1.0 = above par performance The further away the ratio is from 1.0 the more urgent need to investigate
Cost Performance	$CPI = EV / AC$, compares	>1 means project efficient, <1 means project inefficient

Indicator (CPI)	performed to actual cost	
Schedule Performance Indicator (SPI)	SPI = EV / PV, compares work performed to work planned	>1 means project ahead of schedule, <1 means project behind schedule
Estimate At Completion (EAC)	EAC = BAC/CPI	
Estimate To Complete (ETC)	ETC = (BAC – EV) / CPI	

Example

Assume a project that has exactly one task. The task was baselined at 100 hours, but 110 hours have been spent and the estimate to complete is 10 additional hours. The task was to have been completed already. Assume an hourly rate of \$100 per hour.

Description	Formulae	Result
PV	Hourly Rate * Total Hours Planned or Scheduled	100*100 = 10,000
AC	Hourly Rate * Total Hours Spent	100*110 = 11,000
% Complete	AC divided by estimated cost at completion which is 11,000 plus cost of 10 additional hours	11000/(11000+1000) = 91.667%

EV	Baselined Cost * % Complete Actual	9166.667 (baseline of 10,000 * 91.667% complete)
BAC	Baselined Effort in hours * Hourly Rate	10000 (100 hours * 100) <i>indicates initially budget signed off for the project</i>
EAC	AC + ETC	12000 (11000 + 1000) <i>notice this is over budget</i>
VAC	BAC – EAC	-2000 (10000 – 12000) <i>indicates additional funds required to complete work</i>
% Completed Planned	PV / BAC	100% (10000/10000)
% Completed Actual	AC / EAC	91.7% (11000/12000) <i>lesser than planned completion</i>
SV	Earned Value (EV) – Planned Value (PV)	-833.33 (9166.667 – 10000) <i>negative schedule variance or behind schedule</i>
SPI	SPI = EV / PV	0.9167 (9166.667 / 10000) <i>indicating poor schedule performance</i>
CV	Earned Value (EV) – Actual Cost (AC)	-1.833.33 (9166.67 – 11000) <i>indicating a cost overrun</i>

CPI

Earned Value (EV) /Actual Cost
(AC)

0.833 (9166.667 /
11000) *indicating over budget*

RESOURCE ALLOCATION

Effective Resource Management for Team Projects and Goals

Project Insight gives project managers power over the management of resource allocation for software development, marketing, product development teams and more. Assigning teammembers to business goals, projects and individual tasks is simple and easy with our PMI and PMBOK® Guide compliant solution. Mass assign team members' tasks grouped by skill set, department or resource type, or handle resource allocation management for a single person. It is equally simple to change a resource on a set of project tasks as well.

Our portfolio system allows resource allocation managers and project managers to use project level and/or cross project resource allocation to manage workloads in order to achieve their goals. The software application reports evenly divide the work (hours) among the workdays (duration) scheduled for the tasks to calculate the total work or effort assigned to a resource within a specified date range.

Efficient Resource Allocation and Workload Management

Resource information may be accessed from the 'Resources' tab within a project to review the availability of resources. Project Insight, web project management software provides real-time resource allocation data based on the allocation of their assignments to project tasks systemwide. Project managers can also view all resources across all projects in Project Insight. This information

is accessed in 'My Reports,' 'Cross Project Resource Allocation.' Data may be hidden or displayed

according to each person's preferences, supporting a wide variety of applications for these reports. Hundreds of permutations of resource allocation reports are available.

Other project management software applications claim to have extensive resource allocation capabilities in their marketing materials; however, they often fall short. Project Insight not only allows resource managers or project managers to see the total workload each resource has per day, week or other time period, it allows them to drill down on all of the projects and tasks that are causing the over allocation in one view. Tasks can easily be reassigned using Project Insight's

simple drag and drop functionality. It's perfect for the management of all kinds of goals, tasks and projects including IT projects, interactive or marketing projects, product development projects,

professional services and more. All tasks are efficiently managed with proper resource allocation and tracking, down to the last detail.

Notes If your goal is to make sure that you're on top of resource allocation, then Project Insight's powerful resource management applications will make it happen.

IDENTIFYING RESOURCE REQUIREMENTS

As you plan for application compatibility testing, keep in mind the future state of your computing

environment. Are you planning to upgrade some of your software to versions that fully use new Windows 2000 features? Are you planning to implement new standard desktop configurations

or use Terminal Services? Issues such as these determine the resources that are required and the applications that are to be tested as a suite.

If you plan to deploy new applications with Windows 2000 during the rollout, test these applications with the current applications.

You can facilitate testing by setting up a lab where testers can conduct their tests. In such a lab, you can have the necessary tools and equipment available at all times. Some organizations have a lab for testing applications that is separate from the Windows 2000 lab. If you do not have the budget for a separate lab, you might share a lab with another project or with training. If you share a lab, try to choose one that has compatible scheduling and equipment requirements.

SCHEDULING RESOURCES

In this unit we will learn about scheduling resources in projects. We will begin by discussing the nature of resource requirements (both people and machines) and the problems associated with managing resources in a project environment. Given the finite nature of resource availability, a project plan may have to be modified so that it is practical. This is the major thrust of resource planning and management. In this unit, we will examine, at some length, the four major stages of the resource scheduling process. These stages are **resource definition**, **resource allocation**, **resource aggregation**, and **resource leveling**. *Resource definition* involves identifying

the critical resources that need to be planned and managed for the successful completion of the project. In a multi-project environment as projects are competing for scarce resources, *resource allocation* addresses the problem of the optimum use and timing of the assignment of these resources to the various project activities. *Resource aggregation* involves determining the aggregate

resources that will be needed, period by period, to complete all project activities. Having identified

the necessary resource requirements, the last stage in the process is *resource leveling*. In this stage,

we attempt to ensure that the demand for resources does not exceed availability. Specifically, demand for resources is smoothed to ensure that the peaks and valleys are reduced. In this lesson, we will also learn about the “critical chain approach” to tackle resource dependencies that occur in projects due to reduced slack.

UNIT VI

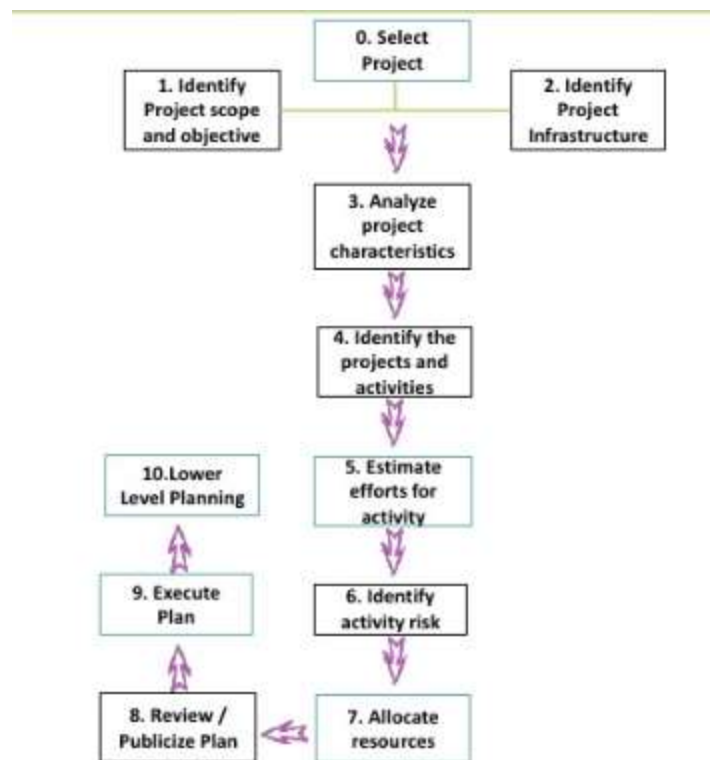
SOFTWARE QUALITY

INTRODUCTION

Quality is generally agreed to be a good thing. In a practice the quality of a system can be a vague, undefined, attribute. We therefore need to define precisely what qualities were require of a system. However, this is not enough – we need to objectively weather a system meet out quality requirements and this need measurement.

PLACE OF SOFTWARE QUALITY IN PROJECT MANAGEMENT

Quality will be of concern at all stages of project planning and execution. But will be of particular interest at the following points in the step wise framework



1. Identify Project Scope and objective: Some objective could relate to the qualities of the application to be delivered.
2. Identify project infrastructure: Identify the installation standard and procedures. Some of these almost certainly be about quality
3. Analyze project characteristics: To identify the other qualities based requirement.
4. Identify the products and activities of the project: It is at this point the entry, exist and process requirement are identified for each activity
5. Review and publicize Plan: At his stage the overall quality aspects of the project plan are reviewed

IMPORTANCE OF SOFTWARE QUALITY

We would expect quality to be concern of all procedures of goods and services.

- Increasing criticality of software: the end user of a software generally anxious about the quality of software especially about the reliability. They are concern about the safety because of their dependency on the software system such as aircraft control system are more safety critical systems

- The intangibility of software: This make it difficulty to know that a particular tasks in project has been completed satisfactory. The results of these tasks can be made tangible by demanding that the developer produce deliverables that can be examined for quality

- Accumulating errors during software development:

As computer system developed in made up of a number of steps where the output from one step is the input to the next, the error in the earlier deliverables will be added to those in the later steps leading to an accumulating determined effects. In general the later in a project that an error is found the more expensive it will be to fix. In addition because the number of errors in the system is unknown, the debugging phases of a project are particularly difficult to control

DEFINING THE SOFTWARE QUALITY

Quality is rather vague term and we need to define carefully what we mean by it

- A functional specification describing what the system is to do
- A quality specification concerned with how well the functions are to operate
- A resource specification concerned with how much is to be spent on the system

Attempt to identify specific product qualities that are appropriate to software, for instance, grouped software qualities into three sets. Product operation qualities, Product revision qualities and product transition qualities.

Product operation qualities

Correctness: The extent to which a program satisfies its specification and fulfills user objectives

Reliability: The extent to which a program can be expected to perform its intended function with required precision

Efficiency: The amounts of computer resource required by software

Integrity: The extent to which access to software or data by unauthorized persons can be controlled

Usability: The effort required to learn, operate, prepare input and interpret output

Product revision qualities

Maintainability: the effort required to locate and fix an error in an operational program

Testability: The effort required to test a program to ensure it performs its intended function

Flexibility: The effort required to modify an operational program,

Product Transition qualities

Portability: The efforts required to transfer a program from one hardware configuration and/or software system environment to another

Reusability: The extent to which a program can be used in other applications.

Interoperability: The efforts required to couple one system to another

Table 12.1 *Software quality criteria*

<i>Quality factor</i>	<i>Software quality criteria</i>
Correctness	traceability, consistency, completeness
Reliability	error tolerance, consistency, accuracy, simplicity
Efficiency	execution efficiency, storage efficiency
Integrity	access control, access audit
Usability	operability, training, communicativeness, input/output volume, input/output rate
Maintainability	consistency, simplicity, conciseness, modularity, self-descriptiveness
Testability	simplicity, modularity, instrumentation, self-descriptiveness
Flexibility	modularity, generality, expandability, self-descriptiveness
Portability	modularity, self-descriptiveness, machine independence, software system independence
Reusability	generality, modularity, software system independence, machine independence, self-descriptiveness
Interoperability	modularity, communications commonality, data commonality

During quality is not enough. If we are to judge whether a system meets our requirements we need to be able to measure its qualities. For each criterion, one or more measures have to be invented to measure the degree to which the quality is present. In general the user of software would be concerned with measuring what McCall called quality factors while the developers would be concerned with quality criteria. The following should be laid down for each quality.

Scale – the unit of measurement

Test – the practical test of the extent to which the attribute quality exists

Worst – the worst acceptable value

Plan – the value that is planned to achieve

Best – the best value that appears to be feasible

Now – the values that applies currently

ISO 9126

ISO 9126 standard was published in 1991 to tackle the question of the definition of software quality this 13 pages document was designed as foundation upon which further, more detailed standard could be built.

ISO 9126 identifies six software quality characteristics

- **Functionality:** which covers the functions that a software product provides to satisfy user needs
- **Reliability:** Which relates to the capability of the software to maintain its level of performance
- **Usability:** Efforts need to use a software
- **Efficiency:** physical resource used when a software is executed
- **Maintainability:** Effort needed to the make changes to the software
- **Portability:** Availability of the software to be transferred to a different environment.

Functionality: which covers the functions that a software product provides to satisfy user needs.

Functionality sub characteristics: suitability, Accuracy, Interoperability, Compliance and Security.

Compliance refers to the degree to which the software adheres to application-related standards or legal requirements. Typically these could be auditing requirement. Interoperability refers to the ability of software to interact with others.

Reliability: Which relates to the capability of the software to maintain its level of performance

Reliability sub characteristics: Maturity, Fault Tolerance and recoverability.

Maturity refers to frequency of failures due to fault in software more identification of fault more chances to remove them. Recoverability describe the control of access to a system

Usability: Efforts need to use a software.

Usability sub characteristics: Understand ability, Learnability, operability.

Understand-ability is a clear quality to grasp, although the definition attributes that bear on the user efforts for recognizing the logical concept and its applicability in our view actually makes it less clear. Learnability has been distinguished from operability. A software tool might be easy to learn but time-consuming to use say it uses a large number of nested menus. This is for a package that is used only intermittently but not where the system is used or several hours each day by the end user. In this case learnability has been incorporated at the expense of operability.

Efficiency: physical resource used when a software is executed

Efficiency sub characteristics: Time behaviour, Resource behaviour.

Maintainability: Effort needed to the make changes to the software

Maintainability sub characteristics: Analysability, Changeability, Stability and Testability.

Analysability is the quality that McCall called diagnose ability, the ease with which the cause of failure can be determined. Changeability is the quality that other have called flexibility: the latter name is perhaps a better one as changeability has a slightly different connotation in plain English It implies that the suppliers of the software are always changing it. Stability, on the other hand, does not means that the software never changes, It means that there is a low risk of a modification to the software having unexpected effects.

Portability: Availability of the software to be transferred to a different environment.

Portability sub characteristics: Adaptability, Install ability , Conformance and Replace ability.

Conformance is distinguished from compliance relates to those standard that have bearing on portability. The use of a standard programming language common to many software/hardware environment is an example of conformance. Replace ability refers to the factors that give upwards compatibility between old software components and the new ones. Downwards compatibility is specifically excluded from definition.

Quality Metrics selection: Measurements that correlate to the characteristics of each quality have to be identified. No specific guidance is given by ISO 9129 standard on the applicability of the various measurements that might be used.

Rating Level Definition: The metrics used must be mapped onto scales that indicate the degree to which the requirement have been satisfied for example in one application time behaviour in the sense of response time might be important for a key transaction actual response time might be mapped onto quality scale.

<i>response time (seconds)</i>	<i>quality score</i>
< 2	5
2-3	4
4-5	3
6-7	2
8-9	1
>9	0

Assessment criteria definitions: The way that the quality scores are combined or summarized or give an overall view of the product has to be defined. There software product as now to be evaluated by measuring its qualities, converting them to quality score or rating and summarising them the rating to obtain an overall judgment.

Table 12.2 *Quality rating scores*

<i>product quality</i>	<i>importance rating (a)</i>	<i>product A</i>		<i>product B</i>	
		<i>quality score (b)</i>	<i>weighted score (a × b)</i>	<i>quality score (c)</i>	<i>weighted score (a × c)</i>
usability	3	1	3	3	9
efficiency	4	2	8	2	8
maintainability	2	3	6	1	2
overall			17		19

Practical software quality measures : Below are some way of measuring particular qualities.

Reliability: might be measure in terms of

Availability: the percentage of a particular time interval that a system is usable.

Means time between failures, the total service time divided by the number of failures

Failure on demand: the probability that a system will not be available at the time required on the probability that a transaction will fail.

Support activity: the number of fault reports that are dealt with

Maintainability: This is closely related to flexibility the ease with which the software can be modified. The main difference is that before an amendment can be made, the fault has to be diagnosed. Maintainability can therefore be seen as flexibility plus a new quality, diagnose ability which might be defined as the average amount of time needed to diagnose a fault.

Extendibility: This is a component of the more general quality of flexibility. It can be defined as the productivity needed to incorporate a new feature into an existing system expressed as a percentage of the normal productivity when developing the software from scratch.

The original IOE maintenance billing system comprised 5000 SLOC and took 400 works-days to implement. An amendment to the core system caused by the introduction of group accounts has led to 100 SLOC being added which took 20 works days to implement thus

$$\begin{aligned} \text{productivity for the original system} &= 5000/400 \\ &= 12.5 \text{ SLOC/staff day} \end{aligned}$$

$$\begin{aligned} \text{productivity for the amendment} &= 100/20 \\ &= 5 \text{ SLOC/staff day} \end{aligned}$$

$$\begin{aligned} \text{extendibility} &= 5/12.5 \times 100 \\ &= 40\% \end{aligned}$$

Product Versus Process Quality Management

The measurements described above can be taken only after the system is operational. It might be too late to do anything to remedy problems. What would be more helpful to someone like Amanda the IOE would be measurements and other checks that can be taken during development and that can help control what the final system will be like. The system development process is made up of a number of activities that are linked together so that the output from one activity is the input to the next step. Thus, program testing will depend on there being a program to test that will be the product of the program coding stage. Errors can enter the process at any stage. They can be introduced either because of a defect in the way a process is carried out as when programmers make mistakes in the logic of their programs or because information has not been passed clearly and unambiguously between stages.

Errors that creep in at the early stages are more expensive to correct at late stages for the following reasons

The later the error is found the more rework at more stages of development.

The general tendency is for each successive stage of development to be more detailed and less able to absorb change.

Error should therefore be eradicated by careful examination of the products of each stage before they are passed on to the next. The following process requirement should be specified for each activity.

Entry requirement. Which have to be in place before an activity can start.

Implementation requirement: Which define how the process is to be conducted.

Exit Requirement. Which have to be fulfilled before an activity is deemed to have been completed.

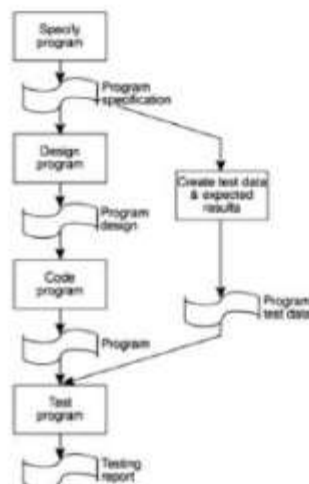


Figure 12.2 An example of the sequence of processes and deliverables.