Unit 5

Software Project Management

Introduction

Building computer software is a complex undertaking task, which particularly involves many people working over a relatively long time. That's why software projects needs to be managed. The software project management is the first layer of software engineering process. It starts before the technical work starts, continues as the software evolves from conceptual stage to implementation stage. It is a crucial activity because the success and failure of the software is directly depends on it.

Software project management is needed because professional software engineering is always subject to budget constraints, schedule constraints and quality oriented focus.

Definition

Project management involves the planning, monitoring and control of the people, process and events that occurs as software evolves from a preliminary concept to an operational implementation. Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development and support of computer software. The project management activity encompasses measurements and metrics estimation, risk analysis, schedules, tracking and control.

Effective software project management focuses on the **four P's**: people, product, process, and project. The order is not arbitrary. The manager who forgets that software engineering work is an intensely human endeavor will never have success in project management. A manager who fails to encourage comprehensive customer communication early in the evolution of a project risks building an elegant solution for the wrong problem. The manager who pays little attention to the process runs the risk of inserting competent technical methods and tools into a vacuum. The manager who embarks without a solid project plan jeopardizes the success of the product.

1. The People

a) The "people factor" is so important that the Software Engineering Institute has developed a ***people management capability maturity model*** (**PM-CMM**).

b) To enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability.

c) The PM-CMM is a companion to the software capability maturity model that guides organizations in the creation of a mature software process.

d) The PMCMM defines the following areas for software people.

   a. Recruiting
   b. Selection
   c. Performance Management
   d. Training
   e. Career Development
   f. Team Culture Development

2. The Product

Before a project can be planned,

a) Product objectives and scope should be established.

b) Alternative solutions should be considered.

c) Technical and management constraints should be identified.

d) The software developer and customer must meet to define product objectives and scope.

e) Objectives identify the overall goals for the product (from the customer's point of view) without considering how these goals will be achieved.

f) Scope identifies the primary data, functions and behaviors that characterize the product, and more important, attempts to *bound* these characteristics in a quantitative manner.

3. The Process

a) A software process provides the framework from which a comprehensive plan for software development can be established.

b) A small number of framework activities are applicable to all software projects, regardless of their size or complexity.

c) A number of different tasks set—tasks, milestones, work products and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.

d) Umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model.

4. The Project

a) We conduct planned and controlled software projects for one primary reason—it is the only known way to manage complexity.

b) The overall development cycle is called as Project.

c) In order to avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring and controlling the project.

The Software Team

The best team structure for any particular project depends on the nature of the project & product and also the individual characteristics of the team members. The basic team structures are as follows – a) Democratic teams, b) Chief programmer teams, c) Hierarchical team.

1. Democratic Teams or Democratic decentralized (DD)

It has following features.

1) The team leader position does not rotate among the team members because a team functions best when one individual is responsible for coordinating team activities and for making final decisions in situations where collective decisions can not work.
2) Here all the decisions are made by collective effort of the members.
3) All the activities carried out during project are collectively discussed and handled.

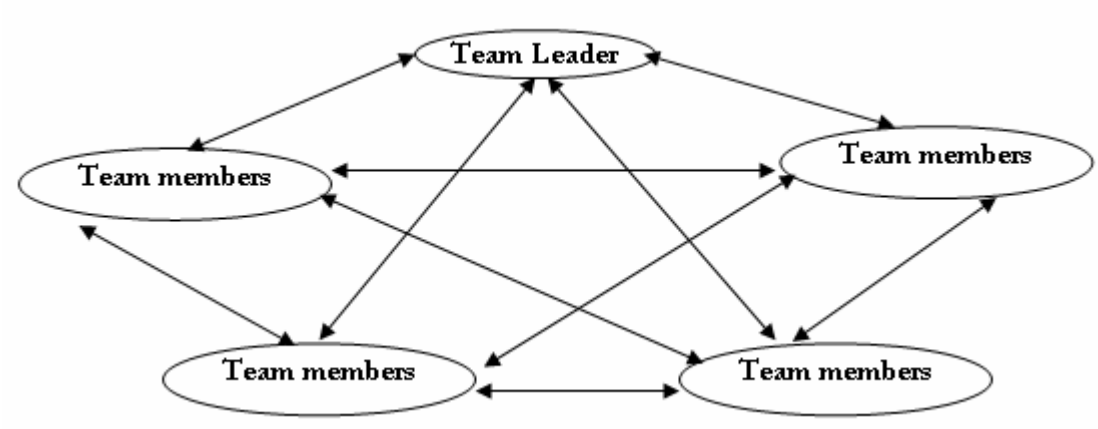


Fig Democratic Team Structure

Advantages: -

I) Opportunity for team members to contribute to decisions.

II) Opportunity for team members to learn from each other.

III) Increased job satisfaction due to equal importance and non threatening environment.

IV) These teams can stay together for several years and may work on several different projects.

Disadvantages:-

I) Communications overhead required for reaching to collective decisions.

II) A lot of coordination required between team members.

III) Less individual responsibility and authority results in less personal drive and initiative from team members.

2. Chief Programmer Teams or Controlled decentralized (CD)

It has following features.

1) They are highly structured
2) The chief programmer designs the product and makes all the decisions.
3) The chief programmer implements the critical parts of the project.
4) The chief programmer allocates the work for the individual programmer under him.
5) Usually the number of programmers ranges from 2 to 5 only.
6) The programmers do the coding, debug; document and unit test the system.
7) The chief programmer is assisted by a backup consultant programmer on various technical problems, provides connection with the customer provides interaction with quality assurance group and may participate in analysis, design and implementation phases.
8) The chief programmer is also assisted by an administrative program manager, who handles the administrative details which includes time cards for the employees, sick leave and vacation schedule.

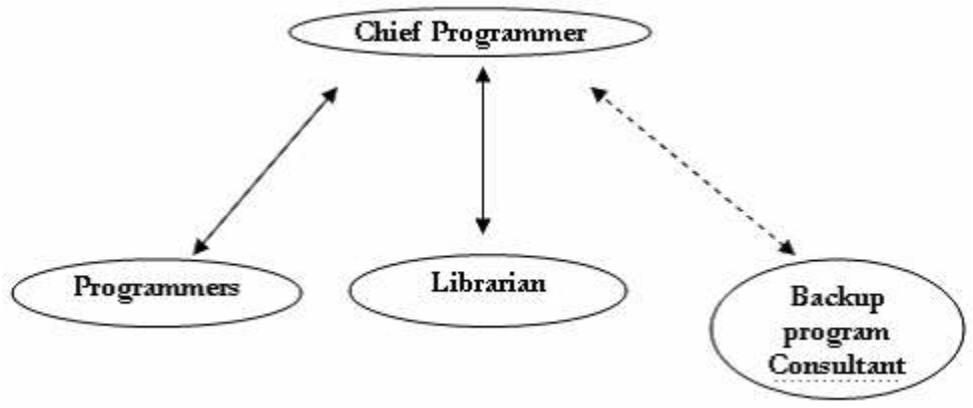


Fig Chief Programmer Team Structure

Advantages:-

I) Centralized decision making reduces the decision making time.

II) It reduces communication paths and related overheads.

Disadvantages:-

I) As all the decisions are taken by the chief programmer, hence it results in low moral among the programmers.

II) The effectiveness of this structure depends solely on the efficiency and knowledge of the chief programmer.

3. Hierarchical Team or Controlled Centralized (CC)

It has following features.

1) It s a mixed approach of Democratic and Chief programmer team structures.
2) Here the project leader has under his control, 2 to 5 senior programmers who individually have 5 to 7 junior programmers under their control.

3) The various jobs of the Project leader includes –
   a) Assigning tasks,
   b) Attending reviews and walkthroughs,
   c) Detecting problem areas,
   d) Balancing of the work load,
   e) Participation in various technical activities.
4) The major decisions are taken by the Project leader and who in-turn gives some decision making power to the senior programmers also.

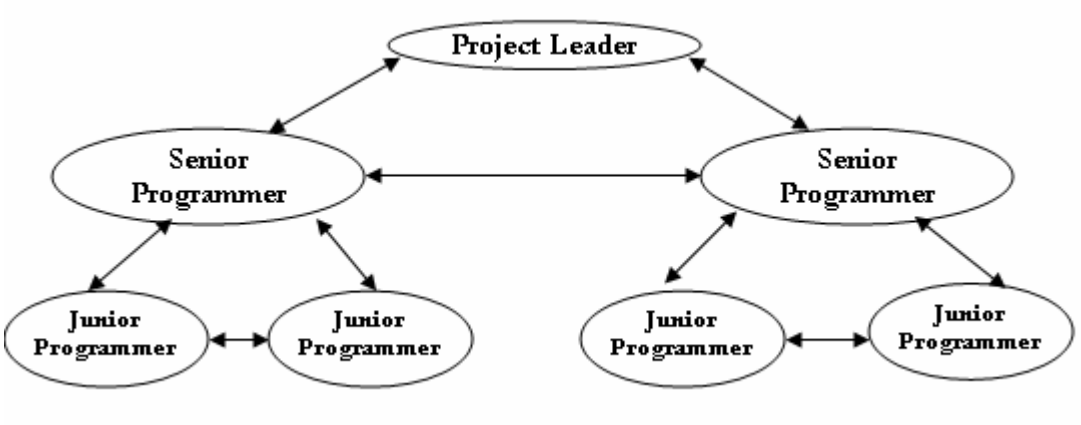


Fig Hierarchical Team Structure

Advantages:-

I) Here the number of communication paths are limited hence permitting effective communication
II) Here the time span required for deciding and implementing the decided processes takes less time
III) The job satisfaction is fairly good as the scope of promotions is good.

Disadvantages:-

I) The most technically efficient programmers tend to be promoted, so the best programmers are lost.
II) The best programmers may not be good managers hence promoted to a management post might result in reduction in productivity.

Software Project Planning – Following are the various project planning activities

1) Defining the problem –

a) Develop a definitive statement of the problem to be solved, including a description of the present situation, problem constraints and a statement of the goals to be achieved. The problem statement should be formulated in the customer's terminology.

b) Justify a computerized solution strategy for the problem.

c) Identify the various functions provided by the hardware subsystem, software subsystem and people subsystem. The constraints thereof also should be identified.

d) Determining system level goals and requirements for the development process and the work products.

e) Establish high-level acceptance criteria for the system.

2) Developing a solution strategy –

a) Create multiple solution strategies, with out considering the constraints.

b) Conduct a feasibility study for each strategy.

C) Recommend a solution strategy, indicating why other strategies were rejected.

d) Develop a list of the characteristics of the product priority wise.

3) Planning the development process –

a) Define a life-cycle model and an organizational structure for the project.

b) Planning of –

I) Configuration management activities,

II) Quality assurance activities and

III) Validation activities.

c) Determine phase-dependent factors –

I) Tools,

II) Techniques, and

III) Notations.

d) Establish preliminary cost estimates for the system development.

e) Establish a preliminary development schedule.

f) Establish preliminary staffing estimates.

g) Preliminary estimation of the required computing resources and maintenance of those systems.

h) Preparation of glossary of terms.

i) Identification of information sources for reference during the project development.

Detail Explanation of Software Project Planning activities

Problem Definition

There is a need to prepare a concise statement of the problem to be solved and the constraints that exist for its solution in customer's terminology. Problem definition requires understanding of the problem domain and the problem environment. Techniques for gaining this knowledge include –

a) Customer interviews,

b) Observation of the problem tasks, and

c) Actual performance of the tasks developed by the planner. Here the planner should not be biased in any way and should certainly be technically experienced.

After preparing the solutions the successive task includes the determination of the -

a) Appropriateness of the computerized solution,

b) Cost-effectiveness,

c) It should avoid displacing existing workers as it may not be acceptable in the society. etc.

Requirement analysis and Goal determination:-

Goals:

2) Goals are targets for achievement and serve to establish the frame work for a software development project.
3) Goals apply to both the development process and the work products.
4) Goals can be either qualitative or quantitative. They have the following categories-
    a) **Qualitative process goal :-** the development process should adhere to quality observed under quality assurance.
    b) **Quantitative process goal :-** the system should be delivered with in a fixed time.
    c) **Qualitative product goal :-** the system should make the user's job more easy & interesting.
    d) **Quantitative product goal :-** the system should reduce the cost of transaction by about 25 %.
5) Other common goals include a) transportability, b) early delivery, c) ease for nonprogrammers etc.

Requirement:

2) They include –
    a) Functional aspects,
    b) Performance aspects,
    c) Hardware aspects,
    d) Software aspects,
    e) User interface, etc.
3) They also specify development standards and quality assurance standards for both project process and product.
4) Special efforts should be made in the process of developing meaningful requirement statements and methods that will be used to verify those statements.

During goal determination and requirement analysis the quality attributes of the software has to be taken in to consideration. According to IEEE standards following are the desired quality attributes of a software product:-

1) Portability ,
2) Reliability,
3) Efficiency,
4) Accuracy,
5) Error,
6) Robustness,
7) Correctness.

Developing a Solution Strategy

1) A solution strategy is not a detailed solution plan. It's a general statement of solution concerning the nature of possible solutions.
2) A solution strategy should account for all external factors that are visible to the product users.
3) A strategy should be phrased to permit alternative approaches to product design.
4) Several strategies should be considered before one is adopted.
5) Solution strategies should be generated with out regard for feasibility because its not possible to be both creative & critical at the same time.
6) Often the best strategy is composite of ideas from several different approaches. And the best solution strategies may become apparent only after all the obvious solutions have been enumerated.
7) The feasibility of each proposed solution strategy can be established by examining solution constraints. Constraints prescribe the boundaries of the solution space.
8) A solution strategy is feasible if the project goals and requirements can be satisfied with in the constraints of available time , resources and technology using that strategy.
9) When recommending a solution strategy its extremely important to document the reasons for rejecting other strategies, this provides justification for the recommended strategy and may prevent ill-considered revisions at some later date.

A solution strategy should include a priority list of product features. There are several important reasons for stating product priorities. At some later time in the development cycle it may be necessary to postpone or eliminate some system capabilities due to inconsistencies in the requirements, technical bottlenecks or time and cost overruns.

Planning the Software Development Processes

A Software process consists of two parts

1) **Product Engineering Processes**, which consists of
   a) **Development process**: - This process consists of all kinds of activities which contribute towards the software development process which are performed by the programmers, designers, testing personnel, librarians etc. The major goal here is to improve quality of the software product.
   b) **Project Management process**: - This process includes all the activities related to the efficient management of various stages and resources in the development of the software. It includes scheduling, milestones, reviews, staffing etc. Here the major goal is optimal usage of available resource in order to reduce the overall cost.
   c) **Software Configuration Management process**: - It's the process of understanding and formulating the necessary system requirements and establishing the exact amount and type of resource needed to successfully complete the software product.

2) **Process Management Processes: -** This process is responsible for monitoring every process that are occurring during the software development procedures. It ensures that high standards are followed during every process. It includes understanding the current process, analyzing its properties, determining possible improvements and then implementing the improvements in the processes.

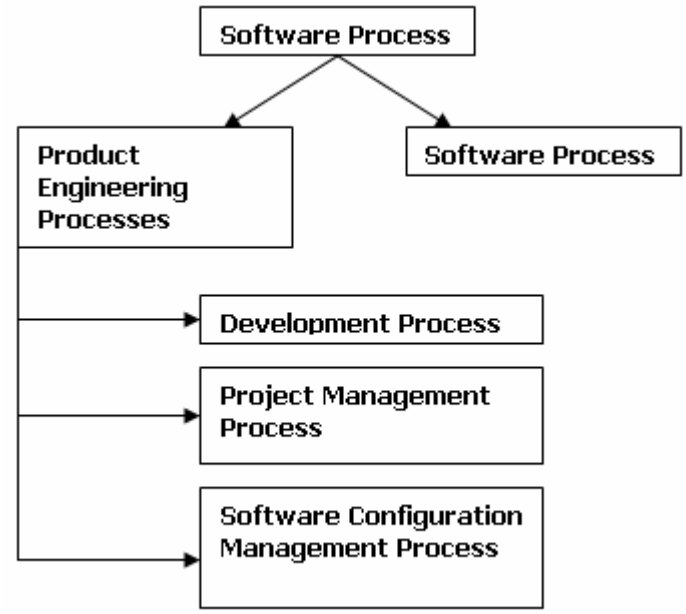


Fig Software Process

Characteristics of a Software Process

A software process should provide effective development, management and change management. Some of the desired characteristics are

1) **Predictability: -** It determines how accurately a process can give the desired outcomes if that particular process is followed and a prediction of the outcomes can be made before the product is actually completed. This is a fundamental property of any properly engineered process. The predictions could be in terms of size, cost, quality etc.
2) **Testability and Maintainability: -** A process is said to be proper if by following that process the final product be maintainable, also the cost on testing and maintenance should also reduce. Both testing and maintenance depend largely on the design and coding of software and these costs can be considerably reduced if the process that is being followed ensures that the coding is up to standards and flexible.

3) **Early defect removal and defect prevention: -** It's observed that errors occur throughout the development process. The cost of correcting errors of different phases is not the same and depends on when the error is detected and corrected. Detecting errors soon after they have been introduced is one of the major objectives of a software process.
4) **Process improvement: -** A good software process is such that it supports its own improvement. This is possible if the process itself provides some quantifiable data from time to time for its own evaluation and the data provided for evaluation should be able to show the strength and weakness in various stages of the process.

Software Metrics

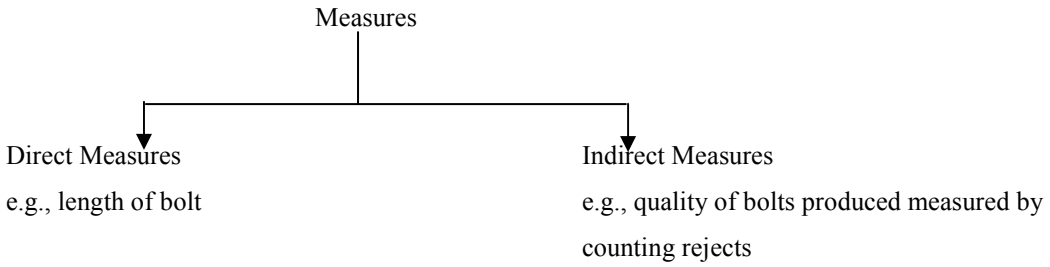
Software metrics refers to broad range of measures for computer software within the context of software project management. We are concerned with:

a) Productivity metrics output.
b) Quality metrics (customer requirements being satisfied or not)
c) Technical metrics (logically complexity of software, degree of modularity)

Why is software measured? Many reasons are there:

a) To indicate quality of product.
b) To access the productivity of people who produce the product.
c) To assess benefits (in terms of quality and productivity) derived from new software engineering methods and tools.
d) To form a baseline for estimation of cost, of resources and of schedules.
e) To help justify requests for new tools or additional training.

Types of Measures:



There are two types of measures:

1. **Direct Measures** – In software engineering process, following are the direct measures:
    a) Lines of Code (LOC)
    b) Execution Speed
    c) Memory Size
    d) Defects reported over some set period of time.
2. **Indirect Measures** – In software engineering process, following are the indirect measures:
    a) Functionality
    b) Quality
    c) Complexity
    d) Efficiency
    e) Maintainability

Note that one category of metrics is:

a) **Productivity Metrics** – That focuses on output of software engineering process.

b) **Quality Metrics** – That provides an indication of how closely software conforms to implicit and explicit customer requirements.

c) **Technical Metrics** – That focuses on character of software. E.g., its logical complexity, degree of modularity.

But there is another category of software metrics:

a) **Size oriented metrics** – Size oriented metrics are used to collect direct measures of software engineering output and quality.

b) **Function oriented metrics** – Function oriented metrics provide indirect measures and human oriented metrics collect information about the manner in which people develop computer software.

a) **Size oriented metrics** – It focuses on the size of the software that has been produced. If a software organization maintains simple records, a table of size oriented measures, such as shown below can be created. The table lists each software development project that has been completed over the past few years and corresponding measures for that project.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Fig Size oriented metrics

It shows that for project: alpha, 12,100 LOC were developed with 24 person months of effort, at a cost of $168,000. Note that the effort and cost recorded in the table represent all software engineering activities (i.e. analysis, design, code and test) and not just coding. Furthermore, it shows that 365 pages of documentation were developed, 134 errors were recorded before software was released and 29 defects were encountered after release to the customer within first year of operation and three people worked on this project alpha.

In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value. From the rudimentary data contained in the table, a set of simple size-oriented metrics can be developed for each project:

- Errors per KLOC (thousand lines of code).
- Defects4 per KLOC.
- $ per LOC.
- Page of documentation per KLOC.

In addition, other interesting metrics can be computed:

- Errors per person-month.
- LOC per person-month.

• $ per page of documentation.

Size-oriented metrics are not universally accepted as the best way to measure the process of software development . Most of the controversy swirls around the use of lines of code as a key measure. Proponents of the LOC measure claim that LOC is an "artifact" of all software development projects that can be easily counted, that many existing software estimation models use LOC or KLOC as a key input, and that a large body of literature and data predicated on LOC already exists. On the other hand, opponents argue that LOC measures are programming language dependent, that they penalize well-designed but shorter programs, that they cannot easily accommodate nonprocedural languages, and that their use in estimation requires a level of detail that may be difficult to achieve (i.e., the planner must estimate the LOC to be produced long before analysis and design have been completed).

b) **Function-Oriented Metrics** – Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the ***function point***. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Fig computing Function Point

Function points are computed by completing the table shown in Figure above. Five information domain characteristics are determined and counts are provided in the appropriate table location. Information domain values are defined in the following manner:

Number of user inputs – Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

Number of user outputs – Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

Number of user inquiries – An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files – Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.

Number of external interfaces – All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

Once these data have been collected, a complexity value is associated with each count. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective. To compute function points (FP), the following relationship is used:

FP = count total * [0.65 + 0.01 * Σ(*Fi*)]

Where count total is the sum of all FP entries obtained from Figure above.

The *Fi* (*i* = 1 to 14) are "complexity adjustment values" based on responses to the following questions:

1. Does the system require reliable backup and recovery?

2. Are data communications required?

3. Are there distributed processing functions?

4. Is performance critical?

5. Will the system run in an existing, heavily utilized operational environment?

6. Does the system require on-line data entry?

7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?

8. Are the master files updated on-line?

9. Are the inputs, outputs, files, or inquiries complex?

10. Is the internal processing complex?

11. Is the code designed to be reusable?

12. Are conversion and installation included in the design?

13. Is the system designed for multiple installations in different organizations?

14. Is the application designed to facilitate change and ease of use by the user?

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). The constant values in Equation and the weighting factors that are applied to information domain counts are determined empirically. Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes:

- Errors per FP.
- Defects per FP.
- $ per FP.
- Pages of documentation per FP.
- FP per person-month.

Advantages of FP

1. FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages.
2. It is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach.

Disadvantages of FP

1. The method requires some "sleight of hand" in that computation is based on subjective rather than objective data.
2. The counts of the information domain (and other dimensions) can be difficult to collect after the fact.
3. FP has no direct physical meaning—it's just a number.

Measures for software Quality

There are many measures of software quality, correctness; maintainability, integrity, and usability provide useful indicators for the project team.

1. Correctness.

1. Correctness is the degree to which the software performs its required function.

2. The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.
3. When considering the overall quality of a software product, defects are those problems reported by a user of the program after the program has been released for general use.
4. For quality assessment purposes, defects are counted over a standard period of time, typically one year.

2. Maintainability.

1. Maintainability is the ease with which a program can be
   - Corrected if an error is encountered,
   - Adapted if its environment changes, or
   - Enhanced if the customer desires a change in requirements
2. There is no way to measure maintainability directly; therefore, we must use indirect measures.
3. A simple time-oriented metric is ***mean-time-to change*** (MTTC), the time it takes to
   - Analyze the change request,
   - Design an appropriate modification,
   - Implement the change, test it, and
   - Distribute the change to all users.
   - On average, programs that are maintainable will have a lower MTTC (for equivalent types of changes) than programs that are not maintainable.

3. Integrity.

1. This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.
2. Attacks can be made on all three components of software: programs, data, and documents.
3. To measure integrity, two additional attributes must be defined:
   - ***Threat***
   - ***Security.***
4. ***Threat*** is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time.
5. ***Security*** is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled.
6. The integrity of a system can then be defined as

   Integrity = summation [(1 – threat) * (1 – security)]

   Where threat and security are summed over each type of attack

4. Usability.

If a program is not user-friendly, it is often doomed to failure, even if the functions that it performs are valuable.

Usability is an attempt to quantify user-friendliness and can be measured in terms of four characteristics:

(1) The physical and or intellectual skill required to learn the system,

(2) The time required to become moderately efficient in the use of the system,

(3) The net increase in productivity measured when the system is used by someone who is moderately efficient, and

(4) A subjective assessment of user's attitudes toward the system.

Metrics for Source Code

Halstead's Theory: Halstead software science assigns quantitative laws to the development of computer software, using a set of primitive measures that may be derived after code is generated or estimated once design is complete. These follow:

Say, n_1 = The number of distinct operators that appear in a program

n_2 = The number of distinct operands that appear in a program

N_1 = The total number of operator occurances.

N_2 = The total number of operand occurances.

Halstead uses these primitive measures to develop expressions for the overall program length, potential minimum volume for an algorithm, the actual volume (number of bits required to specify a program), the program level (a measure of software complexity), the language level (a constant for a given language) and other features such as development effort, development time and the projected number of faults in the software.

Halstead shows that length N can be estimated using the formula given below:

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

And program volume may be defined as follows:

$$V = N \log_2 (n_1 + n_2)$$

Where $N = N_1 + N_2$.

It should be noted that V will vary with programming language and represents the volume of information (in bits) required to specify a program.

Theoretically, a minimum volume must exist for a particular algorithm. Halstead defines a volume of the most compact form of a program to the volume of the actual program. In addition, L must always be less 1. In terms of premature measures, the volume ratio may be expressed as

$$L = 2/n_1 * n_1/n_2$$

Information Flow Metrics (IF – Metrics)

The IF – metrics is based on a premise that all systems (software) consist of components only that work together to influence the complexity of a system. System theory tells us that components that are highly coupled and that lack cohesion tend to be less reliable and less maintainable than those that are loosely coupled and are cohesive.

IF – metrics model the degree of cohesion and coupling for a particular system component. IF – metrics was applied to software systems by Henry and Kafura. They looked at UNIX OS and found a strong association between the IF – metrics and the level of maintainability applied to the components by developers.

IF – metrics are applied to the components of a system design. For e.g., for a component/ module – A, we can define 3 measures –

1. FAN – IN: A count of the number of other components that can call or pass control, to component – A.
2. FAN – OUT: It is the number of components that are called by component – A.
3. IF – metrics of component – A is

   IF (A) = [FAN – IN (A) * FAN – OUT (A)]

Q. 1) Computer the FP value for a project with the following information domain characteristics:s

No. of user inputs = 32

No. of user outputs = 60

No. of user inquiries = 24

No. of user files = 08

No. of External interfaces = 2

Assume that all complexity adjustment values are average. Assume 14 algorithms have been counted.

Sol: Given

No. of user inputs = 32

No. of user outputs = 60

No. of user inquiries = 24

No. of user files = 08

No. of External interfaces = 2

And complexity adjustment values are average, so,

Measurement Parameter	Count	Weighting Factor
No. of user inputs	32 * 4 =	128
No. of user outputs	60 * 5 =	300
No. of user inquiries	24 * 4 =	96
No. of user files	08 * 10 =	80
No. of External interfaces	2 * 7 =	14
Count total		618

So, FP = count total * [0.65 + 0.01 * Σ(*Fi*)]

Σ(*Fi*) = 14 * 3 = 42 (because in the question it is given that complexity adjustment values are average and all 14 algorithms are counted)

FP = 618 * [0.65 + 0.01 * 42]

FP = 661

Q. 2) Computer the FP value for a project with the following information domain characteristics:

Measurement Factor		Weighting Factor
No. of user inputs	40	4
No. of user outputs	50	5
No. of user inquiries	30	5
No. of user files	10	6
No. of External interfaces	5	10

Assume that all complexity adjustment values are average i.e. have a value of 3.

Q. 3) Compute FP for the following data set:

Inputs = 8

Outputs = 12

Inquiries = 4

Logical Files = 41

Interfaces = 1

Σ(*Fi*) = 41 (influence factor sum).

Software Cost Estimation

The major factors that influence software cost are

1. Programmer ability
2. Product complexity
3. Product size
4. Available time
5. Required reliability
6. Level of technology

Programmer Ability

It had been observed that on very large projects the differences in individual programmer ability will tend to average out and would not affect the project adversely. But on projects utilizing 5 or lesser programmers the factor of individual ability is significant and can affect the project positively or negatively. Hence individual programmer ability governs programmer productivity that affects the cost of the project.

Product Complexity

A software product can be classified on the basis of their complexity in to basic three types –

a) Application programs,

b) Utility programs,

c) Systems program.

Through extensive studies it had observed that Utility programs are 3 times as difficult to write as Application programs and that System programs are three times as difficult to write as Utility programs. So the levels of complexity can be stated as 1 - 3 - 9 for Application – Utility – Systems programs. The following equations were derived from the extensive research –

For calculating Programmer Months (PM): -

Application programs: - $PMap = 2.4 * (KDSI)^{1.05}$

Utility programs: - $PMup = 3.0 * (KDSI)^{1.12}$

Systems programs: - $PMsp = 3.6 * (KDSI)^{1.2}$

For calculating Development time for a program (TDEV): -

Application programs: - $TDEVap = 2.4 * (PM)^{1.05}$

Utility programs: - $TDEVup = 3.0 * (PM)^{1.12}$

Systems programs: - $TDEVsp = 3.6 * (PM)^{1.2}$

For calculating Average Staffing Level (SL): -

Application programs: - SLap= PMap / TDEVap

Utility programs: - SLup= PMup / TDEVup

Systems programs: - SLsp= PMsp / TDEVsp

Where, PM → Programmer Months and KDSI → Thousands of delivered source instructions.

A major consideration that needs to be taken in to account is extra coding required for housekeeping purpose. Housekeeping codes include that portion of coding that handles

a) Input/output,

b) Interactive user communication,

c) Human interface engineering and

d) Error checking & error handling.

So the estimation of cost on lines of code should include housekeeping codes.

Product Size

A large software product is obviously more expensive to develop than a small one. The equations derived for Programmer months (PM) and Development time (TDEV) show that the rate of increase in effort required grows with the number of source instructions at an exponential rate slightly greater than 1.

Available Time

Total project effort is sensitive to the calendar time available for the project completion. Software projects require more effort if the development time is compressed or expanded from the optimal time. After extensive research it had been observed that there is limit beyond which a software project cannot reduce its schedule even by buying more personnel and equipment. The limit occurs roughly at 75% of the nominal schedule.

Level of Technology

The level of technology provided by latest software products helps reducing overall cost. The type-checking and self-documentation aspects of high-level languages improve the reliability and modifiability of the programs created using these high-level languages. The familiarity of the programmer with the latest technology is also a contributing factor in effort reduction and cost reduction.

Software Cost Estimation Techniques

Cost estimates can be made either top-down or bottom-up.

Top-down estimation first focuses on system level costs, such as the computing resources and personnel required to develop the system, as well as the costs of

a) Configuration management,
b) Quality assurance,
c) System integration,
d) Training, and
e) Publications.

Personnel costs are estimated by examining the cost of similar past projects.

Bottom-up cost estimation first estimated the costs to develop each module or sub-system; those costs are combined to arrive at an overall estimate. Bottom-up estimation emphasizes the costs associated with developing individual system components, but may fail to account for system level costs, such as configuration management and quality control.

1) Expert Judgment Technique

The most widely used cost estimation technique is Expert judgment. Inherently it's a top-down estimation technique. Expert judgment relies on the experience background and business sense of one or more key people in the organization.

Groups of experts sometimes prepare a consensus estimate (collective estimate); this tends to minimize individual oversights and lack of familiarity with particular projects and neutralizes personal biases and the desire to win the contract through an overly optimistic estimate as seen in individual judgment style.

The major disadvantage of group estimation is the effect that interpersonal group dynamics may have on individuals in the group. Group members may not be candid enough due to political considerations. The presence of authority figures in the group or the dominance of an overly assertive group member.

2) Delphi cost estimation

This technique was developed to gain expert consensus without introducing the adverse side effects of group meetings. The Dephi can be adapted to software cost estimation in the following manners:-

A coordinator provides each estimator with the system definition document and a form for recording the cost estimation.

Estimators study the definition and complete their estimates anonymously. They may ask questions to the coordinator, but they do not discuss their estimates with one another.

The coordinator prepares and distributes a summary of the estimator's responses any unusual rationales noted by the estimators.

Estimators complete another estimate again anonymously , using the result from the previous estimate. Estimators whose estimates differ sharply from the group may be asked, anonymously, to provide justification for their estimates.

The process id iterated for as many rounds as required. No group discussion is allowed during the entire process.

3) The COCOMO Model

The Constructive Cost Model (COCOMO) is an empirical estimation model i.e., the model uses a theoretically derived formula to predict cost related factors. This model was created by "Barry Boehm". The COCOMO model consists of three models:-

1. **The Basic COCOMO model**: - It computes the effort & related cost applied on the software development process as a function of program size expressed in terms of estimated lines of code (LOC or KLOC).
2. **The Intermediate COCMO model**: - It computes the software development effort as a function of – a) Program size, and b) A set of cost drivers that includes subjective assessments of product, hardware, personnel, and project attributes.
3. **The Advanced COCOMO model**: - It incorporates all the characteristics of the intermediate version along with an assessment of cost driver's impact on each step of software engineering process.

The COCOMO models are defined for three classes of software projects, stated as follows:-

1. **Organic projects**: - These are relatively small and simple software projects which require small team structure having good application experience. The project requirements are not rigid.
2. **Semi-detached projects**: - These are medium size projects with a mix of rigid and less than rigid requirements level.
3. **Embedded projects**: - These are large projects that have a very rigid requirements level. Here the hardware, software and operational constraints are of prime importance.

In the following section we will discuss only the Basic and Intermediate COCMO models.

Basic COCOMO Model

The effort equation is as follows: -

$E = a * (KLOC)^{b}$

$D = c * (E)^{d}$

Where E → effort applied by per person per month, D → development time in consecutive months, KLOC → estimated thousands of lines of code delivered for the project. The coefficients a, b, and the coefficients c, d are given in the Table below

Software Project	a	b	c	D
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Intermediate COCOMO Model

The effort equation is as follows: -

$E = a * (KLOC)^{b} * EAF$

Where E → efforts applied by per person per month, KLOC → estimated thousands of lines of code delivered for the project, and the coefficients a and exponent b are given in the Table below. EAF is Effort Adjustment Factor whose typical ranges from 0.9 to 1.4 and the value of EAF can be determined by the tables published by "Barry Boehm" for four major categories – product attributes, hardware attributes, personal attributes and project attributes.

Software Project	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

Work Breakdown Structures (WBS)

The WBS method is a bottom-up estimation tool.

It's a hierarchical chart that accounts for the individual parts of system.

There are two main classifications - a) Product hierarchy, b) Process hierarchy.

Product hierarchy: - It identifies the product components and indicates the manner in which the components are interconnected.

Process hierarchy: - It identifies the work activities and the relationship between them.

The primary advantage of the WBS technique are it helps in identifying and accounting the various process & product factors and in identifying the exact costs included in the estimates

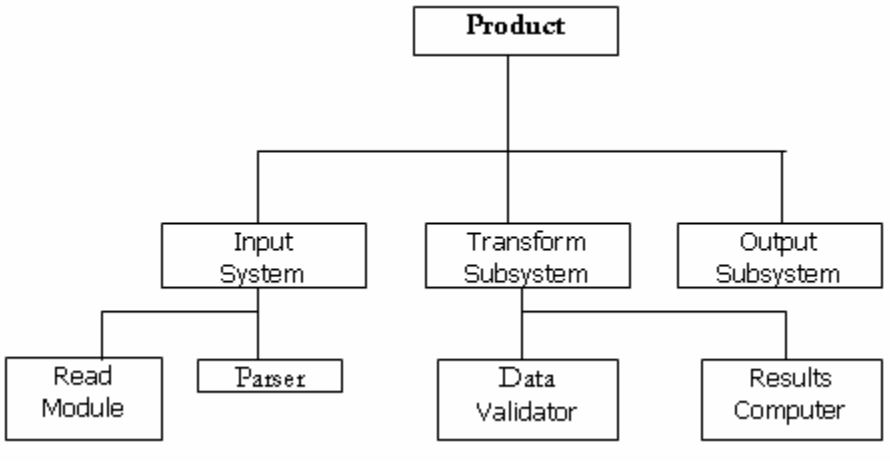


Fig Example of a Product Hierarchical Structure

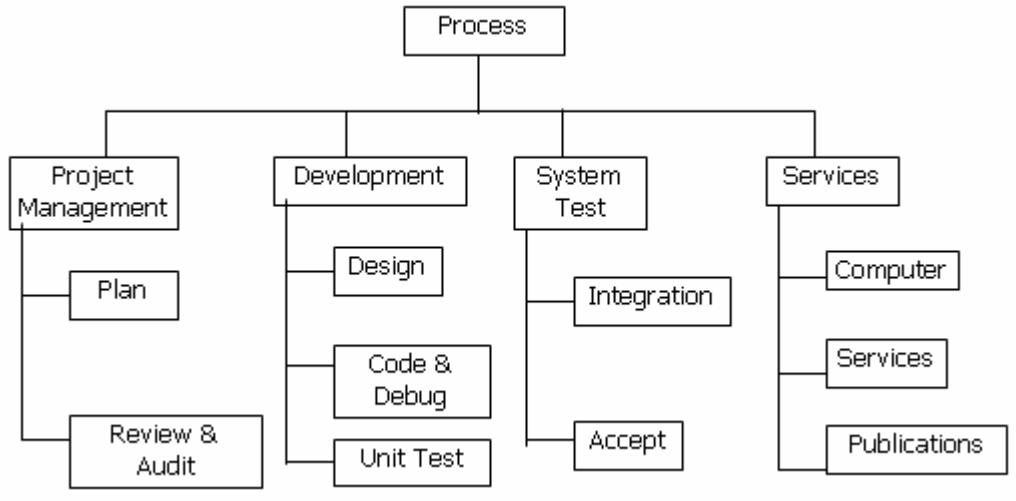


Fig Example of a Process Hierarchical Structure

Q.1) Consider a project to develop a full screen editor. The major components identified are screen edit, command language interpreter, file input and output, cursor movement, screen movement. The sizes for these are estimated to be 4K, 2K, 1K, 2K and 3K delivered source lines. Use COCOMO Model to determine:

a) Overall cost and schedule estimates.
b) Cost and schedules for different phases.

Sol:

a) Overall cost and schedule estimates using COCOMO Model:

For estimation we will go for Basic COCOMO model. The table for constants for Basic COCOMO Model is as follows:

Software Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

For our given project phases/ modules are there:

1. Screen Edit Size → 4K
2. Command Language Interpreter → 2K
3. File input and output → 1K
4. Cursor movement → 2K
5. Screen movement → 3K

Total size → 12K

Now, we will go for overall estimation of project:

If we are analyzing out project, then we find that full screen editor is a semidetached project. So, for estimation purpose, we will make, use of those constants.

$E = a * (KLOC)^b$

$D = c * (E)^d$

Where, E → Effort applied in person months.

D → Development time in chronological months.

$E = 3.0 * (12)^{1.12} = 48.50$ person months

$D = 2.5 * (48.50)^{0.35} = 9.72$ months.

Number of people N = E/D = 48.50 / 9.72 = 4.98 = 5 persons (approximately)

For completion of this project we will require 5 people.

(b) Now, we will go for estimation of each module:

All modules will be organic software project because they are simple and we will make use of concerned constants.

i. **Screen Edit** → 4K

$E = 2.4 * (4)^{1.05} = 10.28$ person months

$D = 2.5 * (10.20)^{0.38} = 6.06$ months

N = E/D = 1.6 = 2 persons (approximately)

ii. **Command Language Interpreter** → 2K

$E = 2.4 * (2)^{1.05} = 4.96$ person months

$D = 2.5 * (4.96)^{0.38} = 4.59$ months

N = E/D = 1.08 = 1 person (approximately)

iii. **File input and output** → 1K

$E = 2.4 * (1)^{1.05} = 2.4$ person months

$D = 2.5 * (2.4)^{0.38} = 3.48$ months

$N = E/D = 0.68$ persons

Not required dedicated individual, any person engaged with other module can do it simultaneously.

iv. **Cursor Movement** → 2K

$E = 2.4 * (2)^{1.05} = 4.96$ person months

$D = 2.5 * (4.96)^{0.38} = 4.59$ months

$N = E/D = 1.08 = 1$ person (approximately)

v. **Screen Movement** → 3K

$E = 2.4 * (3)^{1.05} = 7.6$ person months

$D = 2.5 * (7.6)^{0.38} = 5.4$ months

$N = E/D = 1.4 = 1$ person (approximately)

Q.2) Compute the estimation and duration of a project which is of type organic mode and estimated LOC is 10KLOC using COCOMO.

Sol: For estimation we will go for Basic COCOMO model. The table for constants for Basic COCOMO Model is as follows:

Software Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

For our given project estimated LOC = 10KLOC

Given mode is organic, so a = 2.4, b = 1.05, c = 2.5 and d = 0.38

Effort $E = a * (KLOC)^b$

$E = 2.4 * (10)^{1.05} = 26.93$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (26.93)^{0.38} = 8.74$ months

No of persons required $N = E/D = 26.93/8.74 = 3.08 = 3$ persons (approximately)

Q.3) Assume that the size of an organic type software product has been estimated to be 32,000 LOC and assume that the average salary of software engineer is Rs. 15,000 per month. Determine the effort required to develop the software product and the nominal development time.

Sol: For estimation we will go for Basic COCOMO model. The table for constants for Basic COCOMO Model is as follows:

Software Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

For our given project estimated LOC = 32KLOC

Given mode is organic, so a = 2.4, b = 1.05, c = 2.5 and d = 0.38

Effort $E = a * (KLOC)^b$

$E = 2.4 * (32)^{1.05} = 91$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (91)^{0.38} = 14$ months

Cost required to develop the product = 14 * 15,000 = Rs. 210,000

Q.4) Suppose that we are faced with developing a system such that we expect to have about 1,00,000 LOC. Compute the effort and development time for the organic and semidetached development mode.

Sol: For estimation we will go for Basic COCOMO model. The table for constants for Basic COCOMO Model is as follows:

Software Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

For our given project estimated LOC = 100 KLOC

a) **Organic Mode**: For organic mode, a = 2.4, b = 1.05, c = 2.5 and d = 0.38

Effort $E = a * (KLOC)^b$

$E = 2.4 * (100)^{1.05} = 302.14$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (302.14)^{0.38} = 21.89$ months

b) **Semidetached Mode**: For Semidetached mode, a = 3.0, b = 1.12, c = 2.5 and d = 0.35

Effort $E = a * (KLOC)^b$

$E = 3.0 * (100)^{1.12} = 521.34$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (521.34)^{0.35} = 22.33$ months

Q.5) The value of size of a program in KLOC and different cost drivers are given as size 300 KLOC, complexity 0.95, analyst capability 1.85, application of software engineering method 0.8, performance requirement 0.75. Calculate the effort for three types of projects i.e., organic, semidetached and embedded using COCOMO model.

Sol: For estimation we will go for intermediate COCOMO model. The table for constants for intermediate COCOMO Model is as follows:

Software Project	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

For our given project estimated LOC = 300 KLOC

Complexity → 0.95

Analyst Capability → 1.85

Application of software engineering method → 0.8

Performance Requirement → 0.75

So, EAF = 0.95 * 1.85 * 0.8 * 0.75 = 1.05

a) **Organic Mode** : For organic mode, a = 3.2, b = 1.05, c = 2.5 and d = 0.38

Effort $E = a * (KLOC)^b * EAF$

$E = 2.4 * (300)^{1.05} * 1.05 = 1340.7$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (1340.7)^{0.38} = 38.58$ months

No. of persons required N = E/D = 1340.7/38.58 = 34.75 = 35 persons (approximately)

b) **Semidetached Mode** : For Semidetached mode, a = 3.0, b = 1.12, c = 2.5 and d = 0.35

Effort $E = a * (KLOC)^b * EAF$

$E = 3.0 * (300)^{1.12} * 1.05 = 1873.6$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (1873.6)^{0.35} = 34.94$ months

No. of persons required N = E/D = 1873.6/34.94 = 53.62 = 54 persons (approximately)

c) **Embedded Mode** : For Embedded mode, a = 2.8, b = 1.20, c = 2.5 and d = 0.32

Effort $E = a * (KLOC)^b * EAF$

$E = 2.8 * (300)^{1.20} * 1.05 = 2759.9$ person months

Duration $D = c * (E)^d$

$D = 2.5 * (2759.9)^{0.32} = 31.55$ months

No. of persons required N = E/D = 2759.9/31.55 = 87.48 = 87 persons (approximately)

Scheduling

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time. During early stages of project planning, a ***macroscopic schedule*** is developed. This type of schedule identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a ***detailed schedule***. Here, specific software tasks (required to accomplish an activity) are identified and scheduled.

Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end-date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization. Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software. Unfortunately, the first situation is encountered far more frequently than the second.

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

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

- Estimates of effort
- A decomposition of the product function
- The selection of the appropriate process model and task set
- Decomposition of tasks

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

Both PERT and CPM provide quantitative tools that allow the software planner to

(1) Determine the *critical path*—the chain of tasks that determines the duration of the project;
(2) Establish "most likely" time estimates for individual tasks by applying statistical models; and
(3) Calculate "boundary times" that define a time "window" for a particular task.

Boundary time calculations can be very useful in software project scheduling. Slippage in the design of one function, for example, can retard further development of other functions. Riggs describes important boundary times that may be discerned from a PERT or CPM network:

(1) The earliest time that a task can begin when all preceding tasks are completed in the shortest possible time,
(2) The latest time for task initiation before the minimum project completion time is delayed,
(3) The earliest finish—the sum of the earliest start and the task duration,
(4) The latest finish— the latest start time added to task duration, and
(5) The total float—the amount of surplus time or leeway allowed in scheduling tasks so that the network critical path is maintained on schedule. Boundary time calculations lead to a determination of critical path and provide the manager with a quantitative method for evaluating progress as tasks are completed.

Both PERT and CPM have been implemented in a wide variety of automated tools that are available for the personal computer. Such tools are easy to use and make the scheduling methods described previously available to every software project manager.

Defining a Task Network

Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).

A ***task network,*** also called an ***activity network,*** is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering tasks. Figure below shows a schematic task network for a concept development project.

The concurrent nature of software engineering activities leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, the planner must determine intertask dependencies to ensure continuous progress toward completion. In addition, the project manager should be aware of those tasks that lie on the critical path. That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule.

It is important to note that the task network shown in Figure below is macroscopic. In a detailed task network (a precursor to a detailed schedule), each activity shown in Figure below would be expanded.

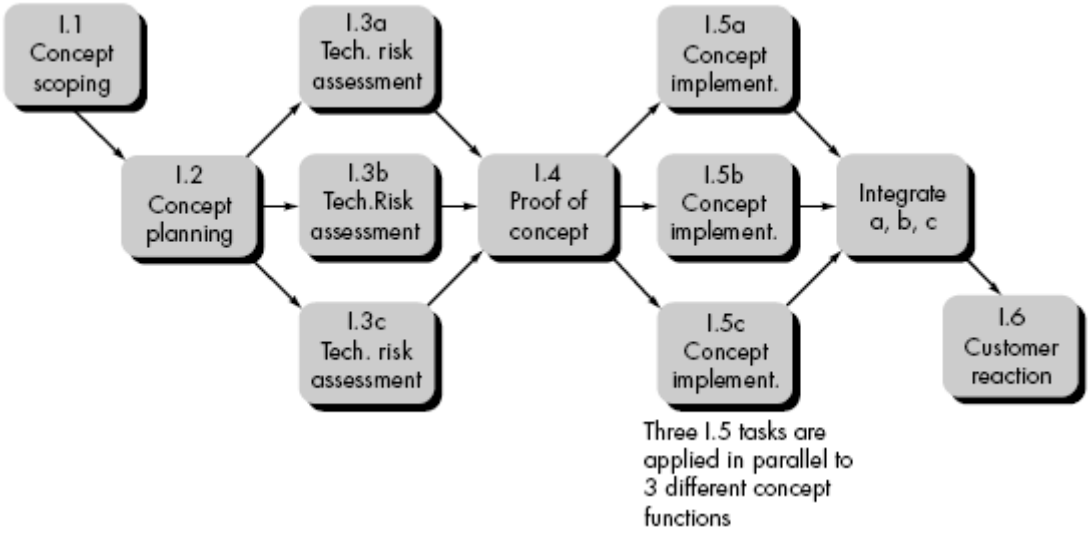


Fig A task set for concept development

Tracking the Schedule

Tracking can be accomplished in a number of different ways:

(1) Conducting periodic project status meetings in which each team member reports progress and problems.
(2) Evaluating the results of all reviews conducted throughout the software engineering process.
(3) Determining whether formal project milestones have been accomplished by the scheduled date.
(4) Comparing actual start-date to planned start-date for each project task listed in the resource table.
(5) Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
(6) Using earned value analysis to assess progress quantitatively.

Earned Value Analysis

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

1. The ***budgeted cost of work scheduled*** **(BCWS)** is determined for each work task represented in the schedule. During the estimation activity, the work (in person-hours or person-days) of each software engineering task is planned.

2. The BCWS values for all work tasks are summed to derive the budget at completion, BAC. Hence,

$$BAC = \Sigma (BCWS_k) \text{ for all tasks } k$$

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

Defect Removal Efficiency (DRE)

A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE). In essence, DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities. When considered for a project as a whole, DRE is defined in the following manner:

$$DRE = E/(E + D)$$

Where E is the number of errors found before delivery of the software to the end-user and D is the number of defects found after delivery.

The ideal value for DRE is 1. That is, no defects are found in the software. Realistically, D will be greater than 0, but the value of DRE can still approach 1. As E increases (for a given value of D), the overall value of DRE begins to approach 1. In fact, as E increases, it is likely that the final value of D will decrease (errors are filtered out before they become defects). If used as a metric that provides an indicator of

the filtering ability of quality control and assurance activities, DRE encourages a software project team to institute techniques for finding as many errors as possible before delivery.

DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task. For example, the requirements analysis task produces an analysis model that can be reviewed to find and correct errors. Those errors that are not found during the review of the analysis model are passed on to the design task (where they may or may not be found). When used in this context, we redefine DRE as

$DRE_i = E_i/(E_i + E_{i+1})$

Where E_i is the number of errors found during software engineering activity i and E_{i+1} is the number of errors found during software engineering activity $i+1$ that are traceable to errors that were not discovered in software engineering activity i. A quality objective for a software team (or an individual software engineer) is to achieve DRE_i that approaches 1. That is, errors should be filtered out before they are passed on to the next activity.

Error Tracking

Error tracking allows you to compare current work with past efforts and provides a quantitative indication of the quality of the work being conducted. Any errors that are not uncovered (but found in later tasks) are considered to be defects, D. Defect removal efficiency has been defined as

$DRE = E/(E + D)$

DRE is a process metric that provides a strong indication of the effectiveness of quality assurance activities.

Risk Management

Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problem—it might happen, it might not.

Steps of Risk Analysis and Management:-

1. Recognizing what can go wrong is the first step, called "risk identification."
2. Each risk is analyzed to determine the likelihood that it will occur and the damage that it will do if it does occur.
3. Risks are ranked, by probability and impact.
4. Finally, a plan is developed to manage those risks with high probability and high impact.

In short, the four steps are :

1. Risk identification
2. Risk Projection
3. Risk assessment
4. Risk management

Risk always involves two characteristics a set of risk information sheets is produced.

1. Uncertainty—the risk may or may not happen; that is, there are no 100% probable risks.

2. Loss—if the risk becomes a reality, unwanted consequences or losses will occur.

Types of risks that are we likely to encounter as the software is built

1. Project risks threaten the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project.

2. Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible.

3. Business risks threaten the viability of the software to be built. Business risks often jeopardize the project or the product. Candidates for the top five business risks are

- Market risk,
- Strategic risk,
- Management risk, and
- Budget risks.

4. Known risks are those that can be uncovered after careful evaluation of the project plan.

5. Predictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).

6. Unpredictable risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

Reactive risk strategies:-

1. Reactive risk strategies follows that the risks have to be tackled at the time of their occurrence.
2. No precautions are to be taken as per this strategy.
3. They are meant for risks with relatively smaller impact.

Proactive risk strategies:-

1. Proactive risk strategies follows that the risks have to be identified before start of the project.
2. They have to be analyzed by assessing their probability of occurrence, their impact after occurrence, and steps to be followed for its precaution.
3. They are meant for risks with relatively higher impact.

Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

One method for identifying risks is to create a **risk item checklist.** The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic categories:

- ***Product size***—risks associated with the overall size of the software to be built or modified.
- ***Business impact***—risks associated with constraints imposed by management or the marketplace.
- ***Customer characteristics***—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
- ***Process definition—***risks associated with the degree to which the software process has been defined and is followed by the development organization.
- ***Development environment***—risks associated with the availability and quality of the tools to be used to build the product.

- ***Technology to be built***—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- ***Staff size and experience***—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Projection

Risk projection also called ***risk estimation***, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities:

(1) Establish a scale that reflects the perceived likelihood of a risk,

(2) Delineate the consequences of the risk,

(3) Estimate the impact of the risk on the project and the product, and

(4) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Risk Assessment

At this point in the risk analysis process we have established a set of triplets of the form :

[ri,li,xi]

Where ri is risk, li is the likelihood (probability) of the risk, and xi is the impact of the risk.

During risk assessment, we further examine the accuracy of the estimates that were made during risk projection, attempt to rank the risks that have been uncovered, and begin thinking about ways to control and/or avert risks that are likely to occur.

Therefore, during risk assessment, we perform the following steps:

1. Define the risk referent levels for the project.
2. Attempt to develop a relationship between each (ri, li, xi) and each of the referent levels.
3. Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.
4. Try to predict how compound combinations of risks will affect a referent level.

Risk Mitigation, Monitoring, and Management

1. An effective strategy must consider three issues:
    a) Risk avoidance
    b) Risk monitoring
    c) Risk management and contingency planning
2. High staff turnover in any organization will have a critical impact on project cost and schedule.
3. To mitigate the risk, project management must develop a strategy for reducing turnover. Among the possible steps to be taken are
    - Meet with current staff to determine causes for turnover • Mitigate those causes that are under our control before the project starts.
    - Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
    - Organize project teams so that information about each development activity is widely dispersed.
    - Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.
    - Conduct peer reviews of all work
    - Assign a backup staff member for every critical technologist.

Computer Aided Software Engineering (CASE)

Computer Aided Software Engineering (CASE) is a set of automated tools for implementing software engineering activities and standards during software development process. Following is a brief overview of CASE architecture.

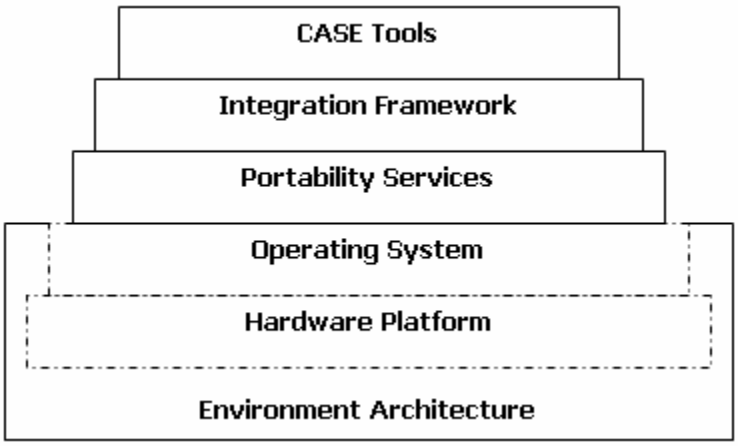


Fig CASE Building Blocks

1) The environment architecture is composed of the hardware platform and operating system support which includes networking and database management software, this forms the basic ground for CASE.
2) A set of portability services provides a bridge between CASE tools and their integration framework along with the environment architecture.
3) The integration framework is a collection of specialized programs that enable individual CASE tools to communicate with each other, to create a project database and to exhibit the same lookup for the software engineer as he would feel when doing the software engineering procedures manually.
4) Portability services provide CASE tools and their integration framework to become portable and migrate along different hardware platforms and operating systems without much change.

The CASE framework provides a very large range of tools for conducting all the possible software engineering activities. Here is a listing of some of the commonly used tools provided by a CASE framework :-

1) **Information Engineering Tools**:-These tools model business information as it moves between various organizations. They represent –
    a) Business data objects,
    b) Relationships between various business data objects and
    c) The flow of business data objects between organizations.
2) **Process Modeling and Management Tools**:- These tools provide –
    a) Key elements of a process for better understanding of the process,
    b) Links to the description of work tasks related to a process and
    c) Links to other related process management tools.
3) **Project Planning Tools**:- They provides tools related to –
    a) Software project effort and cost estimation,
    b) Project scheduling.
4) **Risk Analysis Tools**:- They provide information and methods to a project manager in the following areas –
    a) Building risk analysis tables, and
    b) Provide guidance for risk identification and analysis.

5) **Project Management Tools**:- These tools provides a project manager the following help –
   a) Creating, tracking and monitoring project plans and schedules, and
   b) Providing metrics to estimate & evaluate project and product quality factors.
6) **Documentation Tools**:- These tools help in the following manner –
   a) Document production, and
   b) Desk-top publishing of the documents.
7) **System software Tools**: -These tools include –
   a) E-mails,
   b) Electronic bulletin boards, and
   c) Network communication software.
8) **Software Configuration Management Tools** :- These tools help in the following areas of SCM –
   a) Identification,
   b) Version control,
   c) Change control,
   d) Auditing and
   e) Status accounting.
9) **PRO / SIM Tools**: - These are a set of Prototyping and Simulation tools for predicting the behavior of a real-time system before its production. They help the software engineers to build prototypes of the systems for analyzing the functionality, operation and response of the system.
10) **Test Management Tools**: - These tools help in the process of control and coordination for software testing procedures. It has a Test Driver that reads one or more test cases from a testing file, formats the test data to fit to the needs of the software under test, and then invokes the software to be tested.