UNIT-2 Notes

Software Project Management (Dr. A.P.J. Abdul Kalam Technical University)

UNIT -2

Project Lifecycle & Effort Estimation

Software processes are the activities for designing, implementing, and testing a software system. The software development process is complicated and involves a lot more than technical knowledge. A software process model is an abstract representation of the development process.

Software process model

A *software process model* is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the order of activities of the process and the sequence in which they are performed.
A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task
- The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

There are many kinds of process models for meeting different requirements. We refer to these as SDLC models (Software Development Life Cycle models). The most popular and important SDLC models are as follows:

- Waterfall model
- V model
- Incremental model
- RAD model
  - Agile model
  - Iterative model
  - Prototype model
  - Spiral model

Factors in choosing a software process:

Choosing the right software process model for your project can be difficult. If you know your requirements well, it will be easier to select a model that best matches your needs. You need to keep the following factors in mind when selecting your software process model:

1. Project requirements

Before you choose a model, take some time to go through the project requirements and clarify them alongside your organization's or team's expectations. Will the user need to specify requirements in detail after each iterative session? Will the requirements *change* during the development process?

2. Project size

Consider the size of the project you will be working on. Larger projects mean bigger teams, so you'll need more extensive and elaborate project management plans.

3. Project complexity

Complex projects may not have clear requirements. The requirements may change often, and the cost of delay is high. Ask yourself if the project requires constant monitoring or feedback from the client.

4. Cost of delay

Is the project highly time-bound with a huge cost of delay, or are the timelines flexible?

5. Customer involvement

Do you need to consult the customers during the process? Does the user need to participate in all phases?

6. Familiarity with technology

This involves the developers' knowledge and experience with the project domain, software tools, language, and methods needed for development.

7. Project resources

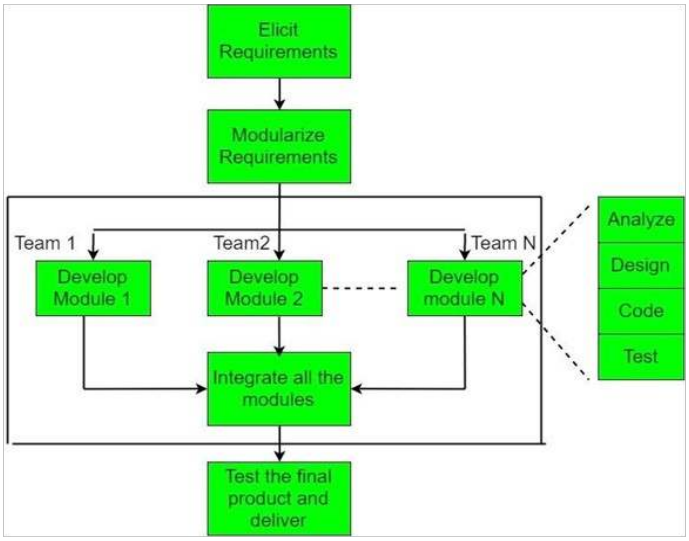
This involves the amount and availability of funds, staff, and other resources.

RAD-Rapid Application Development

The Rapid Application Development Model was first proposed by IBM in the 1980s. The critical feature of this model is the use of powerful development tools and techniques. A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product.
Development of each module involves the various basic steps as in the waterfall model i.e analyzing, designing, coding, and then testing, etc. as shown in the figure. Another striking feature of this model is a short time span i.e the time frame for delivery(time-box) is
generally 60-90 days.

Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.



RAD projects follow an iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.
The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

Following are the various phases of the RAD Model −

1.Business Modelling

The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

2.Data Modelling

The information gathered in the Business Modelling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

3. Process Modelling

The data object sets defined in the Data Modelling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

4. Application Generation

The actual system is built and coding is done by using automation tools to convert process and data

models into actual prototypes.

5. Testing and Turnover

The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

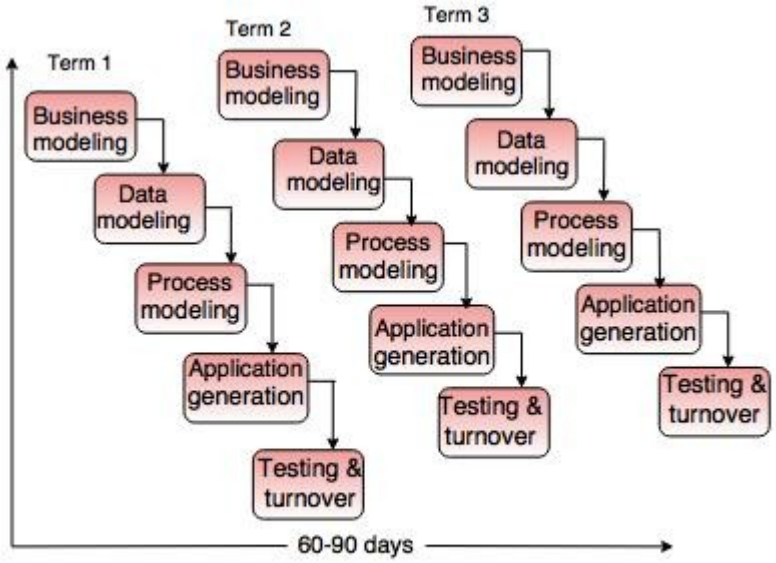


Fig. - RAD Model

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

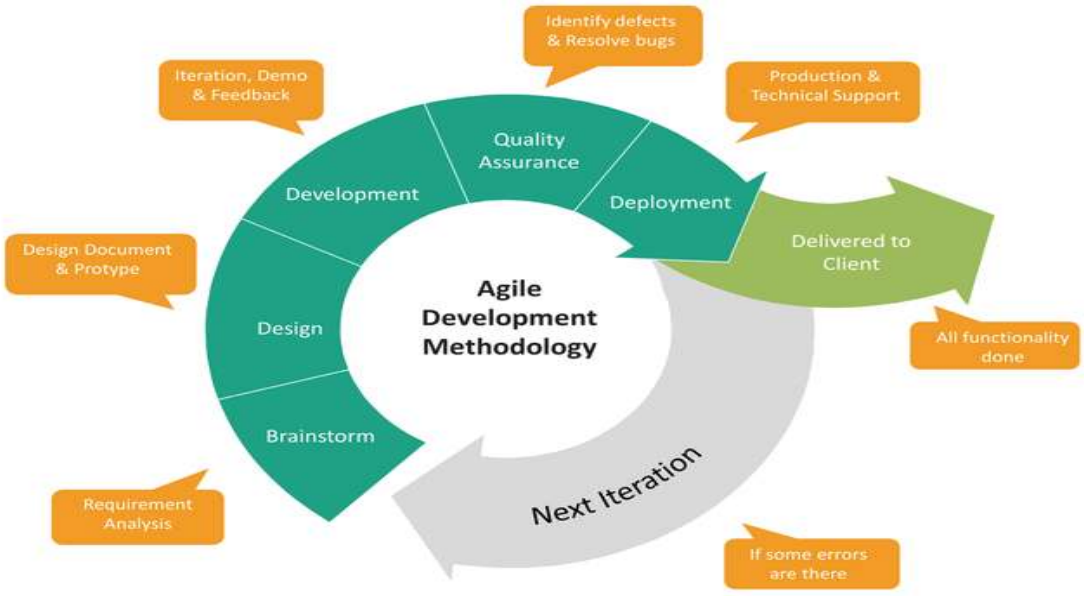
- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

Disadvantage of RAD Model

- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

Agile Model

The meaning of Agile is swift or versatile." **Agile process model**" refers to a software development approach based on iterative development.
Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.
Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and
testing before a working product is demonstrated to the client.



Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Advantages of Agile Methodology:

1. In Agile methodology the delivery of software is unremitting.
2. The customers are satisfied because after every Sprint working feature of the software is delivered to them.
3. Customers can have a look of the working feature which fulfilled their expectations.
4. If the customer has any feedback or any change in the feature then it can be accommodated in the current release of the product.
5. In Agile methodology the daily interactions are required between the business people and the developers.
6. In this methodology attention is paid to the good design of the product.
7. Changes in the requirements are accepted even in the later stages of the development.

8. An Agile/Scrum approach can improve organizational synergy by breaking down organizational barriers and developing a spirit of trust and partnership around organizational goals.

Disadvantages of the Agile Methodology:

1. In Agile methodology the documentation is less.
2. Sometimes in Agile methodology the requirement is not very clear hence it's difficult to predict the expected result.
3. In few of the projects at the starting of the software development life cycle it's difficult to estimate the actual effort required.
4. Because of the ever-evolving features, there is always a risk of the ever-lasting project.
5. For complex projects, the resource requirement and effort are difficult to estimate.

Agile Testing Methods:

- ◯ Scrum
- ◯ Crystal
- ◯ Dynamic Software Development Method(DSDM)
- ◯ Feature Driven Development(FDD)
- ◯ Lean Software Development
- ◯ eXtreme Programming(XP)

Dynamic Systems Development technique (DSDM)

The **Dynamic Systems Development technique (DSDM)** is an associate degree agile code development approach that provides a framework for building and maintaining systems. The DSDM philosophy is borrowed from a modified version of the sociologist principle—80 % of An application is often delivered in twenty percent of the time it'd desire deliver the entire (100 percent) application.

DSDM is An iterative code method within which every iteration follows the 80% rule that simply enough work is needed for every increment to facilitate movement to the following increment. The remaining detail is often completed later once a lot of business necessities are noted or changes are requested and accommodated.

DSDM life cycle defines 3 different iterative cycles, preceded by 2 further life cycle activities:

1. **Feasibility Study:**

   It establishes the essential business necessities and constraints related to the applying to be designed then assesses whether or not the application could be a viable candidate for the DSDM method.

2. **Business Study:**

   It establishes the use and knowledge necessities that may permit the applying to supply business value; additionally, it is the essential application design and identifies the maintainability necessities for the applying.

3. **Functional Model Iteration:**

   It produces a collection of progressive prototypes that demonstrate practicality for the client.(Note: All DSDM prototypes are supposed to evolve into the deliverable application.) The intent throughout this unvarying cycle is to collect further necessities by eliciting feedback from users as they exercise the paradigm.

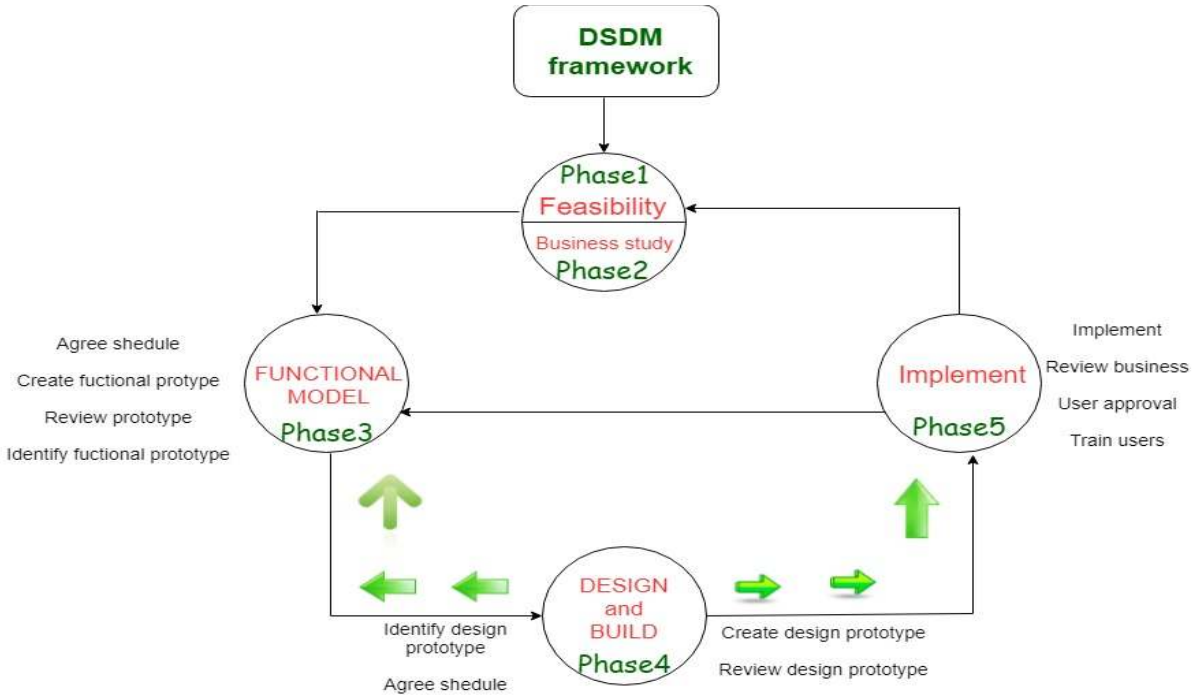
4. **Design and Build Iteration:**

   It revisits prototypes designed throughout useful model iteration to make sure that everyone has been designed during a manner that may alter it to supply operational business price for finish users. In some cases, useful model iteration and style and build iteration occur at the same time.

5. **Implementation:**

   It places the newest code increment (an "operationalized" prototype) into the operational surroundings. It ought to be noted that:

a. **(a)** the increment might not 100% complete or,
b. **(b)** changes are also requested because the increment is placed into place. In either case, DSDM development work continues by returning to the useful model iteration activity.



Dynamic Systems Development Method life cycle

Extreme Programming (XP)

Extreme programming (XP) is one of the most important software development frameworks of Agile models. It is used to improve software quality and responsiveness to customer requirements. The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels. **Good practices need to be practiced in extreme programming:** Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below:

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their works between them every hour.
- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.

- **Simplicity:** Simplicity makes it easier to develop good quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** It helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

Basic principles of Extreme programming: XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed. A *User Story* is a conventional description by the user of a feature of the required system. It does not mention finer details such as the different scenarios that can occur. Based on User stories, the project team proposes Metaphors. *Metaphors* are a common vision of how the system would work. The development team may decide to build a Spike for some features. A *Spike* is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype. Some of the basic activities that are followed during software development by using the XP model are given below:

- **Coding:** The concept of coding which is used in the XP model is slightly different from traditional coding. Here, the coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system, and choosing among several alternative solutions.
- **Testing:** XP model gives high importance to testing and considers it to be the primary factor to develop fault-free software.
- **Listening:** The developers need to carefully listen to the customers if they have to develop good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, the programmers should understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex and very difficult to understand the solution, thus making maintenance expensive. A good design results in elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.

Applications of Extreme Programming (XP): Some of the projects that are suitable to develop using the XP model are given below:

- **Small projects:** XP model is very useful in small projects consisting of small teams as face-to-face meeting is easier to achieve.

- **Projects involving new technology or Research projects:** This type of project face changing requirements rapidly and technical problems. So the XP model is used to complete this type of project.

Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation. XP emphasizes a close working relationship between the development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.

XP includes the following practices:

1. Continuous Integration: Code is integrated and tested frequently, with all changes reviewed by the development team.
2. Test-Driven Development: Tests are written before code is written, and the code is developed to pass those tests.
3. Pair Programming: Developers work together in pairs to write code and review each other's work.
4. Continuous Feedback: Feedback is obtained from customers and stakeholders through frequent demonstrations of working software.
5. Simplicity: XP prioritizes simplicity in design and implementation, with the goal of reducing complexity and improving maintainability.
6. Collective Ownership: All team members are responsible for the code, and anyone can make changes to any part of the codebase.
7. Coding Standards: Coding standards are established and followed to ensure consistency and maintainability of the code.
8. Sustainable Pace: The pace of work is maintained at a sustainable level, with regular breaks and opportunities for rest and rejuvenation.
9. XP is well-suited to projects with rapidly changing requirements, as it emphasizes flexibility and adaptability. It is also well-suited to projects with tight timelines, as it emphasizes rapid development and deployment.

DSDM is often combined with XP to supply a mixed approach that defines a solid method model (the DSDM life cycle) with the barmy and bolt practices (XP) that are needed to create code increments. Additionally, the ASD ideas of collaboration and self-organizing groups are often tailored to a combined method model.

Managing Interactive Processes:

Booch suggests that there are two levels of development:

- The macro process
- The micro process

Macro process

- Establish core requirements (conceptualization).
- Develop a model of the desired behavior (analysis).
- Create architecture (design).
- Evolve the implementation (evolution).
- Manage post delivery evolution (maintenance).

Micro process

- Identify the classes and objects at a given level of abstraction.
- Identify the semantics of these classes and objects.
- Identify the relationships among these classes and objects.
- Specify the interface and then the implementation of these classes and objects

In principle, the micro process represents the daily activity of the individual developer, or of a small team of developers.

Basics of Software Estimation

Estimation techniques are of utmost importance in software development life cycle, where the time required to complete a particular task is estimated before a project begins. Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable. This tutorial discusses various estimation techniques such as estimation using Function Points, Use-Case Points, Wideband Delphi technique, PERT, Analogy, etc

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

General Project Estimation Approach

The Project Estimation Approach that is widely used is **Decomposition Technique**. Decomposition techniques take a divide and conquer approach. Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.

Step 1 − Understand the scope of the software to be built.

Step 2 − Generate an estimate of the software size.

· Start with the statement of scope.

· Decompose the software into functions that can each be estimated individually.

· Calculate the size of each function.

· Derive effort and cost estimates by applying the size values to your baseline productivity metrics.

· Combine function estimates to produce an overall estimate for the entire project.

Step 3 − Generate an estimate of the effort and cost. You can arrive at the effort and cost estimates by breaking down a project into related software engineering activities.

· Identify the sequence of activities that need to be performed for the project to be completed.

· Divide activities into tasks that can be measured.

· Estimate the effort (in person hours/days) required to complete each task.

· Combine effort estimates of tasks of activity to produce an estimate for the activity.

· Obtain cost units (i.e., cost/unit effort) for each activity from the database.

· Compute the total effort and cost for each activity.

· Combine effort and cost estimates for each activity to produce an overall effort and cost estimate for the entire project.

Step 4 − Reconcile estimates: Compare the resulting values from Step 3 to those obtained from Step 2. If both sets of estimates agree, then your numbers are highly reliable. Otherwise, if widely divergent estimates occur conduct further investigation concerning whether −

· The scope of the project is not adequately understood or has been misinterpreted.

· The function and/or activity breakdown is not accurate.

· Historical data used for the estimation techniques is inappropriate for the application, or obsolete, or has been misapplied.

Step 5 − Determine the cause of divergence and then reconcile the estimates.

Estimation Accuracy

Accuracy is an indication of how close something is to reality. Whenever you generate an estimate, everyone wants to know how close the numbers are to reality. You will want every estimate to be as accurate as possible, given the data you have at the time you generate it. And of course you don't want to present an estimate in a way that inspires a false sense of confidence in the numbers.

Important factors that affect the accuracy of estimates are −

· The accuracy of all the estimate's input data.

· The accuracy of any estimate calculation.

· How closely the historical data or industry data used to calibrate the model matches the project you are estimating.

· The predictability of your organization's software development process.

· The stability of both the product requirements and the environment that supports the software engineering effort.

· Whether or not the actual project was carefully planned, monitored and controlled, and no major surprises occurred that caused unexpected delays.

Following are some guidelines for achieving reliable estimates −

- Base estimates on similar projects that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.
- Use one or more empirical estimation models for software cost and effort estimation.

To ensure accuracy, you are always advised to estimate using at least two techniques and compare the results.

Estimation Issues

There are many challenges in many aspects for project estimation. Below are some of the significant challenges:

▶ The uncertain gray area –The biggest issue is the uncertainty involved at the beginning of the project. Many times even the client is not clear about the whole complete requirement. If

there is no complete clear requirement then how it's possible to estimate it in term of effort and time?

▶ Not splitting bigger tasks- If somehow things are clear then many times the estimation is taken keeping in mind the bigger tasks instead of splitting it into smaller tasks for proper

estimation. Such estimation will definitely lead to the overhead tasks at a later stage.

▶ Idealistic & optimistic estimation-Most of the time, the estimation is done keeping in mind the ideal and optimistic conditions but things like version maintenance, unavailability of

some resource and change requests during the project etc. are not considered in project estimation.

▶ Estimation person- Estimation must be done by the developer or in assistance with the developer. Sometimes the estimation is not done by the developer which may lead to huge

mismatch in the estimation.

▶ Buffer & dependencies – It is always uncertain that how much buffer a PM should take. Usually 15-20% buffer is taken keeping in mind project elaboration as project progresses.

But this decision should also consider the things like skillsets, experience of the team and complexity of the project. Dependency of project's internal as well as external factors are not considered most of the time. It can be in terms of some functionality like payment integration or some license cost for some software etc.

Effort and Cost Estimation Techniques

▶ **top-down -** where an overall estimate is formulated for the whole project and is then broken down into the effort required for component tasks;

▶ **bottom-up** - where component tasks are identified and sized and these indiv idual estimates are aggregated.

▶**expert judgement** - where the advice of knowledgeable stall" is solicited;

▶**analogy -** where a similar, completed, project is identified and its actual effort is used as a basis for the new project;

▶ **three point estimate**-it involves three different estimates that are usually obtained from subject matter experts

Top Down estimate

▶ top-down estimating technique assigns an overall time for the project and divides the project into parts according to the work breakdown structure.

▶ For example, let's imagine a project that must be finalized in one year. By fitting the scope of the project on the timeline, you can estimate how much time is available for each activity that needs to be performed. The top-down method is best applied to projects similar to those you have completed previously. If details are sketchy or unpredictable, the top-down approach is likely to be inefficient and cause backlogs.

▶ The top-down approach is normally associated with parametric (or algorithmic) models. These may be explained using the analogy of estimating the cost of rebuilding a house. This would be of practical concern to a house-owner who needs sufficient insurance cover to allow for rebuilding the property if it were destroyed. Unless the house-owner happens to be in the building trade it is unlikely that he or she would be able to w ork out how many bricklayer-hours, how many carpenter-hours, electrician-hours and so on would be required. Insurance companies, however, produce convenient tables where the house- owner can find an estimate of rebuilding costs based on such parameters as ihe number of storeys and the floor space that a house has. 'ITiis is a simple parametric model.

▶ The effort needed to implement a project will be related mainly to variables associated with characteristics of the final system. The form of the parametric model will normally be one or more formulae in the form:

effort = (system size) x (productivity rate)

▶ For example, system size might be in the form 'thousands of lines of code' (KLOC) and the productivity rate 40 days per KLOC. The values to be used will often be matters of subjective judgement.
▶ A model to forecast software development effort therefore has two key components. The first is a method of assessing the size of the software development task to be undertaken. The second assesses the rate of work at which the task can be done. For example.
Amanda at IOE might estimate that the first software module to be constructed is 2 KLOC. She might then judge that if Kate undertook the development of the code, with her expertise she could work at a rate of 40 days per KLOC and complete the work in 2 x 40 days, that is. 80 days, while Ken. who is less experienced, would need 55 days per KLOC and take 2 x 55 that is, 110 days to complete the task.

Bottom up estimate

▶ The bottom-up method is the opposite of top-down. It approaches the project as a combination of small workpieces. By making a detailed estimate for each task and combining them together, you can build an overall project estimate.
▶ Creating a bottom-up estimate usually takes more time than the top-down method but has a higher accuracy rate. However, for the bottom-up method to be truly efficient, the project must be separated at the level of work packages.

Expert judgement

▶ The expert judgment technique requires consulting the expert who will perform the task to ask how long it will take to complete. This method relies on your trust in the expert's insights and experience

Analogous Estimating

▶ Analogous estimating is a technique for estimating based on similar projects completed in the past. If the whole project has no analogs, it can be applied by blending it with the bottom-up technique. In this case, you compare the tasks with their counterparts, then combine them to estimate the overall project.

Three-point Estimating

▶ Three-point estimating is very straightforward. It involves three different estimates that are usually obtained from subject matter experts:
▶ Optimistic estimate
▶ Pessimistic estimate
▶ Most likely estimate
▶ The optimistic estimate gives the amount of work and time that would be required if
everything went smoothly. A pessimistic estimate provides the worst-case scenario. The result will be most realistic when the two are averaged with the most likely estimate.

COSMIC full function points

• COSMIC FFP – Common Software Measurement International Consortium Full Function Point.

• COSMIC deals with decomposing the system architecture into a hierarchy of software layers.

• Unit is Cfsu(COSMIC functional size units).

A Data Movement moves one Data Group. A Data Group is a unique cohesive set of data (attributes) specifying an 'object of interest' (i.e. something that is 'of interest' to the user). Each Data Movement is counted as one CFP (COSMIC function point).

COSMIC recognizes 4 (types of) Data Movements

· Entry moves data from outside into the process

· Exit moves data from the process to the outside world

· Read moves data from persistent storage to the process

· Write moves data from the process to persistent storage.

Function Points

Function points were defined in 1979 in Measuring Application Development.

Productivity by *Allan Albrecht* at IBM. The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points. Each of these functional user requirements maps to an end-user business function, such as a data entry for an Input or a user query for an Inquiry. This distinction is important because it tends to make the functions measured in function points map easily into user-oriented requirements, but it also tends to hide internal functions (e.g. algorithms), which also require resources to implement.

There is currently no ISO recognized FSM Method that includes algorithmic complexity in the sizing result. Recently there have been different approaches proposed to deal with this perceived weakness, implemented in several commercial software products. The variations of the Albrecht- based IFPUG method designed to make up for this (and other weaknesses) include:

· Early and easy function points – Adjusts for problem and data complexity with two questions that yield a somewhat subjective complexity measurement; simplifies measurement by eliminating the need to count data elements.

· Engineering function points – Elements (variable names) and operators (e.g., arithmetic, equality/inequality, Boolean) are counted. This variation highlights computational function. The intent is similar to that of the operator/operand-based Halstead complexity measures.

· Bang measure – Defines a function metric based on twelve primitive (simple) counts that affect or show Bang, defined as "the measure of true function to be delivered as perceived by the user." Bang measure may be helpful in evaluating a software unit's value in terms of how much useful function it provides, although there is little evidence in the literature of such application. The use of Bang measure

could apply when re-engineering (either complete or piecewise) is being considered, as discussed in Maintenance of Operational Systems—An Overview.

· Feature points – Adds changes to improve applicability to systems with significant internal processing (e.g., operating systems, communications systems). This allows accounting for functions not readily perceivable by the user, but essential for proper operation.

· Weighted Micro Function Points – One of the newer models (2009) which adjusts function points using weights derived from program flow complexity, operand and operator vocabulary, object usage, and algorithm.

Benefits

The use of function points in favor of lines of code seek to address several additional issues:

• The risk of "inflation" of the created lines of code, and thus reducing the value of the measurement system, if developers are incentivized to be more productive. FP advocates refer to this as measuring the size of the solution instead of the size of the problem.

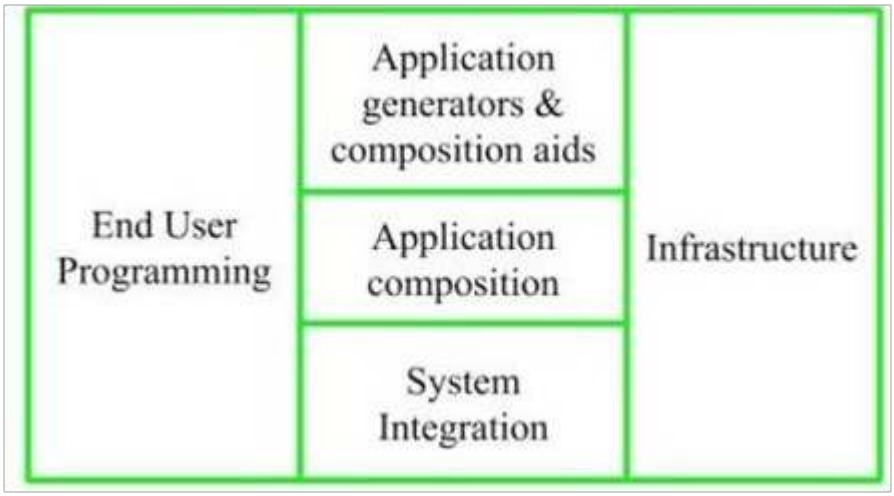
• Lines of Code (LOC) measures reward low level languages because more lines of code are needed to deliver a similar amount of functionality to a higher level language. C. Jones offers a method of correcting this in his work.

• LOC measures are not useful during early project phases where estimating the number of lines of code that will be delivered is challenging. However, Function Points can be derived from requirements and therefore are useful in methods such as estimation by proxy.

COCOMO II: A Parametric Productivity Model

COnstructive COst MOdel II (COCOMO II) is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. COCOMO II is the latest major extension to the original COCOMO (COCOMO 81) model published in 1981.

It consists of three sub-models:

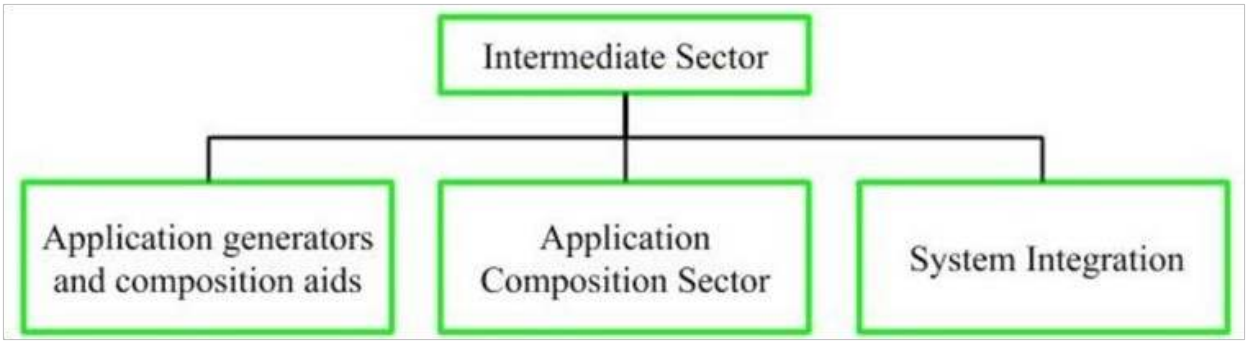


1. End User Programming:

Application generators are used in this sub-model. End user write the code by using these application generators.

Example – Spreadsheets, report generator, etc.

2. Intermediate Sector:



(a). Application Generators and Composition Aids –

This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus, Oracle, IBM, Borland, Novell.

(b). Application Composition Sector –

This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

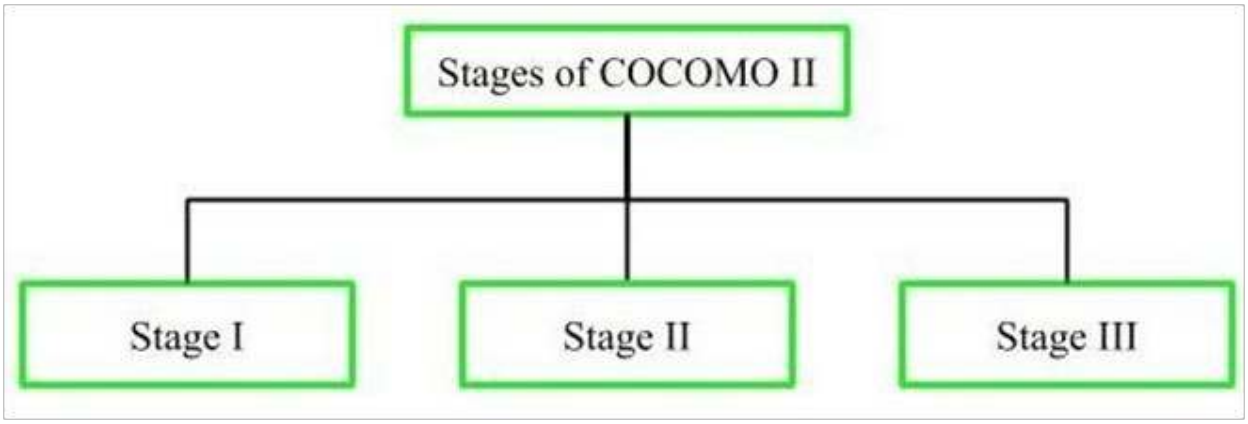
(c). System Integration –

This category deals with large scale and highly embedded systems.

3. Infrastructure Sector:

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

Stages of COCOMO II:



1. **Stage-I:**

   It supports estimation of prototyping. For this it uses *Application Composition Estimation Model*. This model is used for the prototyping stage of application generator and system integration.

2. **Stage-II:**

   It supports estimation in the early design stage of the project, when we less know about it. For this it uses *Early Design Estimation Model*. This model is used in early design stage of application generators, infrastructure, system integration.

3. **Stage-III:**

   It supports estimation in the post architecture stage of a project. For this it uses *Post Architecture Estimation Model*. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

COCOMO I	COCOMO II
COCOMO I is useful in the waterfall models of the software development cycle.	COCOMO II is useful in non-sequential, rapid development and reuse models of software.
It provides estimates of effort and schedule.	It provides estimates that represent one standard deviation around the most likely estimate.
This model is based upon the linear reuse formula.	This model is based upon the non linear reuse formula
This model is also based upon the assumption of reasonably stable requirements.	This model is also based upon reuse model which looks at effort needed to understand and estimate.
Effort equation's exponent is determined by 3 development modes.	Effort equation's exponent is determined by 5 scale factors.
Development begins with the requirements assigned to the software.	It follows a spiral type of development.
Number of submodels in COCOMO I is 3 and 15 cost drivers are assigned	In COCOMO II, Number of submodel are 4 and 17 cost drivers are assigned
Size of software stated in terms of Lines of code	Size of software stated in terms of Object points, function points and lines of code