Software Evolution: Program evolution dynamics, Software maintenance, Evolution processes, Legacy system evolution, Project Management, Management Activities, Project Planning, Project Scheduling, Risk Management (Chapter 5 and 21) - Eighth edition

- Once systems have been deployed, they i**nevitably have to change if they are to remain useful.** Once software is put into use, new requirements emerge and existing requirements change. Business changes often generate new requirements for existing software.
- The majority of changes are a consequence of **new requirements** that are generated in response to changing business and user needs.
- Consequently, you can think of software engineering as a spiral process with requirements, design, implementation and testing going on throughout the lifetime of the system. This is illustrated in Figure 2.1.

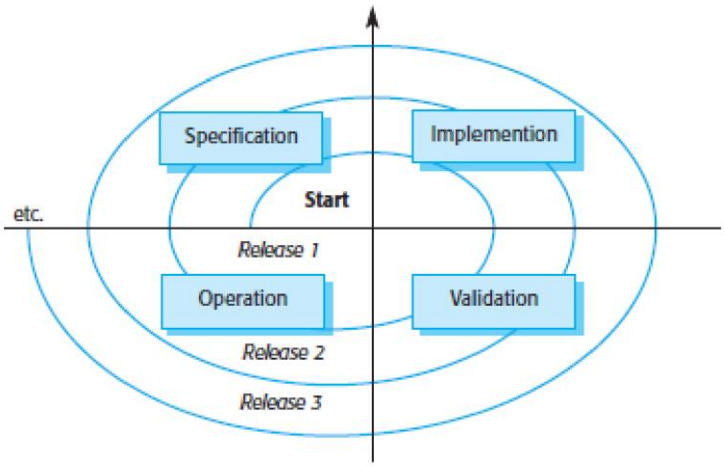


Figure 2.1 A spiral model of development and evolution

2.1 Evolution Processes

- Software evolution processes vary considerably depending on the **type of software being maintained**
- System change proposals are the driver for system evolution in all organizations. These change proposals may involve existing requirements that have not been implemented in

the released system, requests for new requirements and bug repairs from system stakeholders,

- As illustrated in Figure 2.2 , the processes of change identification and system evolution are cyclical and continue throughout the lifetime of a system.
- The cost and impact of these changes are assessed to see how much of the system is affected by the change and how much it might cost to implement the change.
- If the proposed changes are accepted, a new release of the system is planned. During release planning, all proposed changes (fault repair, adaptation and new functionality) are considered.

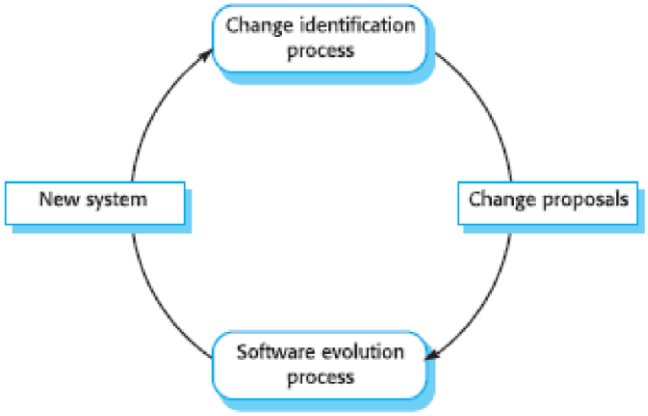


Figure 2.2 Change identification and evolution process

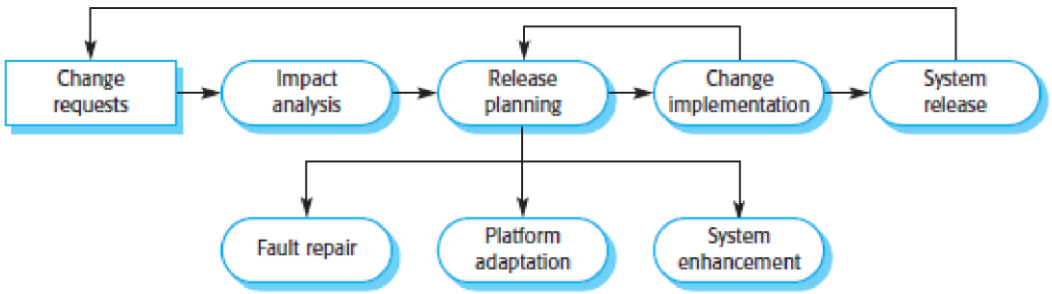


Figure 2.3 The system evolution process

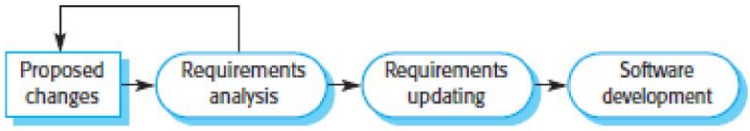


Figure 2.4 change implementation

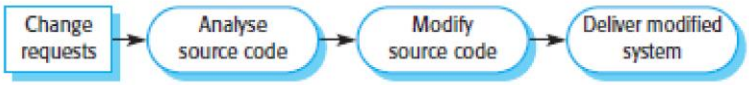


Figure 2.4 The emergency repair process

- A decision is then made on which changes to implement in the next version of the system. The changes are implemented and validated, and a new version of the system is released.
- The process then iterates with a new set of changes proposed for the next release. The change implementation stage of this process should modify the system specification, design and implementation to reflect the changes to the system (Figure 2.3).
- New requirements that reflect the system changes are proposed, analysed and validated. System components are redesigned and implemented and the system is re-tested As you change software, you develop succeeding releases of the system.

➢ **The urgent changes can arise for three reasons**:

1. If a **serious system fault** occurs that has to be repaired to allow normal operation to continue.
2. If changes to the system's operating environment have unexpected effects that disrupt normal operation
3. If there are unanticipated changes to the business running the system, such as the emergence of new competitors or the introduction of new legislation.

- Rather than modify the requirements and design, you make an emergency fix to the program to solve the immediate problem (Figure 2.4).

- When emergency code repairs are made, the change request should remain outstanding after the code faults have been fixed. It can then be re-implemented more carefully after further analysis.

2.2 Program Evolution Dynamics

Program evolution dynamics is the study of system change. The majority of work in this area has been carried out by Lehman and Belady. From these studies, they proposed a set of laws (Lehman's laws) concerning system change. They claim these laws (hypotheses, really) are invariant and widely applicable. Lehman and Belady examined the growth and evolution of a number of large software systems. The proposed laws, shown in Figure 2.5 , were derived from these measurements.

Law	Description
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Figure 2.5 Lehman's Laws

- **The first law** states that system maintenance is an inevitable process. As the system's environment changes, new requirements emerge and the system must be modified. When the modified system is re-introduced to the environment, this promotes more environmental changes, so the evolution process recycles.

- **The second law** states that, as a system is changed, its structure is degraded. The only way to avoid this happening is to invest in preventative maintenance where you spend time improving the software structure without adding to its functionality. Obviously, this means additional costs, over and above those of implementing required system changes.

- **The third law** is, perhaps, the most interesting and the most contentious of Lehman's laws. It suggests that large systems have a dynamic of their own that is established at an early stage in the development process. This determines the gross trends of the system maintenance process and limits the number of possible system changes.
- **Lehman's fourth law** suggests that most large programming projects work in what he terms a *saturated* state. That is, a change to resources or staffing has imperceptible effects on the long-term evolution of the system.
- **Lehman's fifth** law is concerned with the change increments in each system release. Adding new functionality to a system inevitably introduces new system faults. The more functionality added in each release, the more faults there will be. Therefore, a large increment in functionality in one system release means that this will have to be followed by a further release where the new system faults are repaired.
- **The sixth** and seventh laws are similar and essentially say that users of software will become increasingly unhappy with it unless it is maintained and new functionality is added to it.
- The final law reflects the most recent work on feedback processes, although it is not yet clear how this can be applied in practical software development.

2.3 Software Maintenance

Software maintenance is the general process of changing a system after it has been delivered. The term is usually applied to custom software where separate development groups are involved before and after delivery. There are **three different types** of software maintenance:

1. ***Maintenance to repair software faults*** Coding errors are usually relatively cheap to correct; design errors are more expensive as they may involve rewriting several program components. Requirements errors are the most expensive to repair because of the extensive system redesign that may be necessary.

2. ***Maintenance to adapt the software to a different operating environment*** This type of maintenance is required when some aspect of the system's environment such as the hardware, the platform operating system or other support software changes. The application system must be modified to adapt it to cope with these environmental changes.

3. ***Maintenance to add to or modify the system's functionality*** This type of maintenance is necessary when the system requirements change in response to organizational or business change. The scale of the changes required to the software is often much greater than for the other types of maintenance.

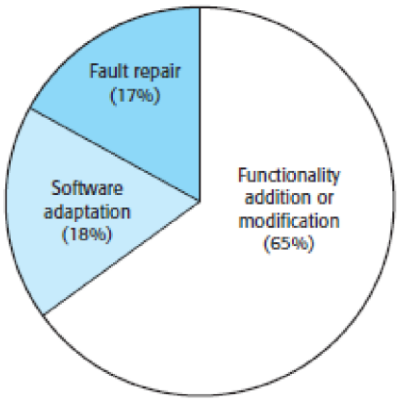


Figure 2.6 Maintenance effort distribution

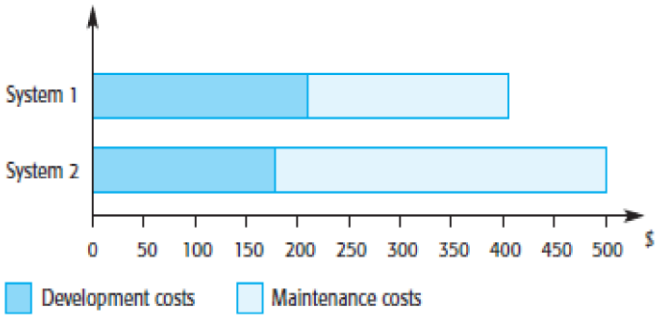


Figure 2.7 Development and maintenance costs

The key factors that distinguish development and maintenance, and which lead to higher maintenance costs, are:

1. ***Team stability*** After a system has been delivered, it is normal for the development team to be broken up and people work on new projects. The new team or the individuals responsible for system maintenance do not understand the system or the background to system design decisions.
2. ***Contractual responsibility*** The contract to maintain a system is usually separate from the system development contract. The maintenance contract may be given to a different company rather than the original system developer. This factor, along with the lack of team stability, means that there is no incentive for a development team to write the software so that it is easy to change.
3. ***Staff skills*** Maintenance staff are often relatively inexperienced and unfamiliar with the application domain. Maintenance has a poor image among software engineers. It is seen as a less skilled process than system development and is often allocated to the most junior staff
4. ***Program age and structure*** As programs age, their structure tends to be degraded by change, so they become harder to understand and modify. Some systems have been developed without modern software engineering techniques.

2.3.1 Maintenance prediction

Managers hate surprises, especially if these result in unexpectedly high costs. You should therefore try to predict what system changes are likely and what parts of the system are likely to be the most difficult to maintain. You should also try to estimate the overall maintenance costs for a system in a given time period. Figure 2.8 illustrates these predictions and associated questions.

These predictions are obviously closely related:

1. Whether a system change should be accepted depends, to some extent, on the maintainability of the system components affected by that change.
2. Implementing system changes tends to degrade the system structure and hence reduce its maintainability.
3. Maintenance costs depend on the number of changes, and the costs of change implementation depend on the maintainability of system components. Predicting the number of change requests

for a system requires an understanding of the relationship between the system and its external environment. To evaluate the relationships between a system and its environment, you should assess:

1. ***The number and complexity of system interfaces*** The larger the number of interfaces and the more complex they are, the more likely it is that demands for change will be made.

2. ***The number of inherently volatile system requirements*** The requirements that reflect organizational policies and procedures are likely to be more volatile than requirements that are based on stable domain characteristics.

3. ***The business processes in which the system is used*** As business processes evolve, they generate system change requests. The more business processes that use a system, the more the demands for system change.

Process metrics that can be used for assessing maintainability are

1. ***Number of requests for corrective maintenance*** An increase in the number of failure reports may indicate that more errors are being introduced into the program than are being repaired during the maintenance process. This may indicate a decline in maintainability.

2. ***Average time required for impact analysis*** This reflects the number of program components that are affected by the change request. If this time increases, it implies that more and more components are affected and maintainability is decreasing.

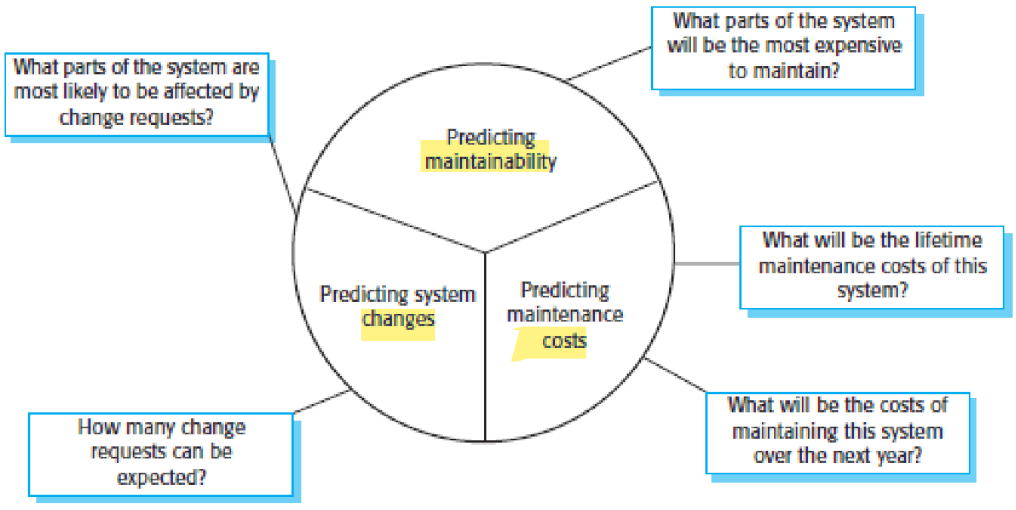


Figure 2.8 Maintenance prediction

3. *Average time taken to implement a change request:* This is not the same as the time for impact analysis although it may correlate with it. This is the amount of time that you need to actually modify the system and its documentation, after you have assessed which components are affected. An increase in the time needed to implement a change may indicate a decline in maintainability.

4. *Number of outstanding change requests* An increase in this number over time may imply a decline in maintainability.

2.3.2 System re-engineering

Re-engineering may involve re-documenting the system, organizing and restructuring the system, translating the system to a more modern programming language, and modifying and updating the structure and values of the system's data.

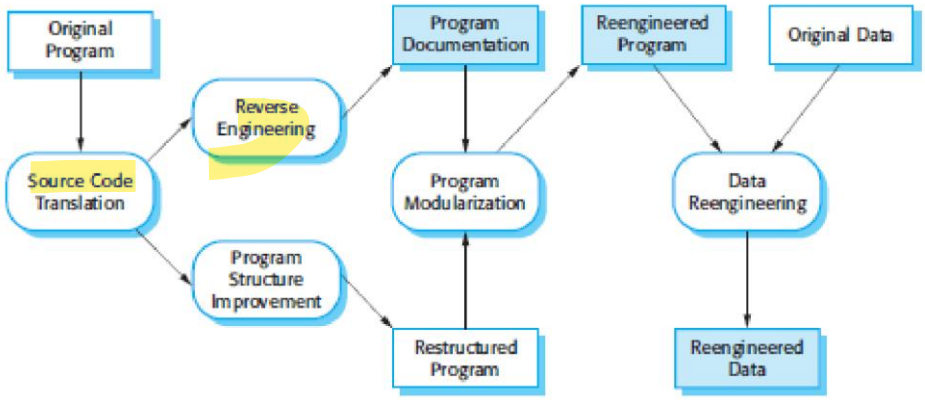


Fig 2.9: Re-engineering

The activities in this re-engineering process are:

1. *Source code translation* The program is converted from an old programming language to a more modern version of the same language or to a different language.

2. *Reverse engineering* The program is analysed and information extracted from it. This helps to document its organisation and functionality.

3. *Program structure improvement* The control structure of the program is analysed and modified to make it easier to read and understand.

4. *Program modularisation* Related parts of the program are grouped together and, where appropriate, redundancy is removed. In some cases, this stage may involve architectural

transformation where a centralised system intended for a single computer is modified to run on a distributed platform.

5. *Data re-engineering* The data processed by the program is changed to reflect program changes.

Re-engineering advantages are

1. *Reduced risk* There is a high risk in re-developing business-critical software. Errors may be made in the system specification, or there may be development problems. Delays in introducing the new software may mean that business is lost and extra costs are incurred.

2. *Reduced cost* The cost of re-engineering is significantly less than the cost of developing new software.

The costs of re-engineering obviously depend on the extent of the work that is carried out. There is a spectrum of possible approaches to re-engineering, as shown in Figure 2.10

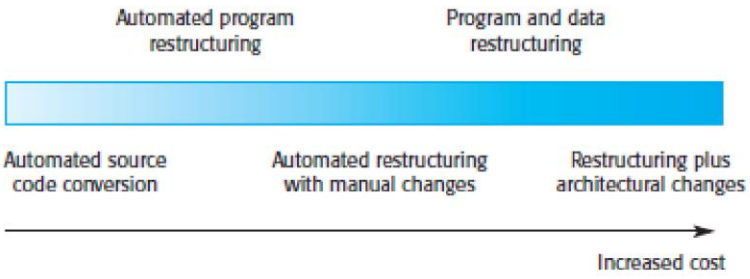


Figure 2.10 Reengineering Approaches

2.4 Legacy system evolution

Organizations that have a limited budget for maintaining and upgrading their legacy systems have to decide how to get the best return on their investment. This means that they have to make a realistic assessment of their legacy systems and then decide what is the most appropriate strategy for evolving these systems. There are **four strategic options:**

1. *Scrap the system completely* this option should be chosen when the system is not making an effective contribution to business processes. This occurs when business processes have changed since the system was installed and are no longer completely dependent on the system.

2. ***Leave the system unchanged and continue with regular maintenance*** This option should be chosen when the system is still required but is fairly stable and the system users make relatively few change requests.

3. ***Re-engineer the system to improve its maintainability*** this option should be chosen when the system quality has been degraded by regular change and where regular change to the system is still required.

4. ***Replace all or part of the system with a new system*** this option should be chosen when other factors such as new hardware mean that the old system cannot continue in operation or where off-the-shelf systems would allow the new system to be developed at a reasonable cost.

- **Assessment (judgment) of legacy system.**

1. Relative Business value
2. System quality

From a business perspective, you have to decide whether the business really needs the system. From a technical perspective, you have to assess the quality of the application software and the system's support software and hardware.

From Figure 2.12, you can see that there **are four clusters of systems**:

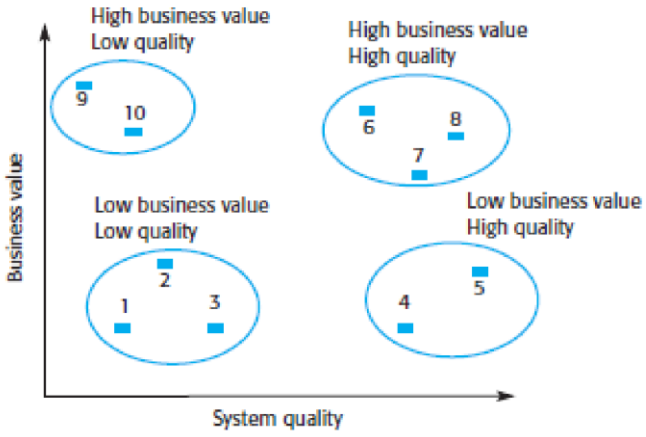


Figure 2.12 Legacy system assessments

1. ***Low quality, low business value:*** Keeping these systems in operation will be expensive and the rate of the return to the business will be fairly small. These systems should be scrapped.

2. ***Low quality, high business value:*** These systems are making an important business contribution so they cannot be scrapped. However, their low quality means that it is expensive to maintain them. These systems should be re-engineered to improve their quality or replaced, if a suitable off-the-shelf system is available.

3. ***High quality, low business value:*** These are systems that don't contribute much to the business but that may not be very expensive to maintain. It is not worth replacing these systems so normal system maintenance may be continued so long as no expensive changes are required and the system hardware is operational. If expensive changes become necessary, they should be scrapped.

4. ***High quality, high business value:*** These systems have to be kept in operation, but their high quality means that you don't have to invest in transformation or system replacement. Normal system maintenance should be continued.

There are **four basic issues for low business value**:

1. ***The use of the system:*** If systems are only used occasionally or by a small number of people, they may have a low business value. A legacy system may have been developed to meet a business need that has either changed or that can now
be met more effectively in other ways.

2. ***The business processes that are supported:*** When a system is introduced, business processes to exploit that system may be designed. However, changing these processes may be impossible because the legacy system can't be adapted. Therefore, a system may have a low business value because new processes can't be introduced.

3. ***The system dependability:*** System dependability is not only a technical problem but also a business problem. If a system is not dependable and the problems directly affect the business customers or mean that people in the business are diverted from other tasks to solve these problems, the system has a low business value.

4. ***The system outputs:*** The key issue here is the importance of the system outputs to the successful functioning of the business

Factors that you should consider during the environment assessment are shown in Figure 2.13.

Factor	Questions
Supplier stability	Is the supplier is still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers, for example, be used with current versions of the operating system? Is hardware emulation required?

Figure 2.13 Factors used in environment assessment

To assess the technical quality of an application system, you have to assess a range of factors (Figure 2.14 be collected are:

1. *The number of system change requests:* System changes tend to corrupt the system structure and make further changes more difficult. The higher this value, the lower the quality of the system

2. *The number of user interfaces:* This is an important factor in forms-based systems where each form can be considered as a separate user interface. The more interfaces, the more likely that there will be inconsistencies and redundancies in these interfaces.

3. *The volume of data used by the system:* The higher the volume of data (number of files, size of database, etc.), the more complex the system.

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and current?
Data	Is there an explicit data model for the system? To what extent is data duplicated across files? Is the data used by the system up-to-date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?

Figure 2.14 Factors used in application assessment

PROJECT PLANNING

Project planning is one of the most important jobs of a software project manager. The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Project planning takes place at three stages in a project life cycle:

1. At the proposal stage, when there is bidding for a contract to develop or provide a software system, a plan may be required at this stage to help contractors decide if they have the resources to complete the work and to work out the price that they should quote to a customer.
2. During the project start-up phase, there is a need to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
3. Periodically throughout the project, when the plan is modified in light of experience gained and information from monitoring the progress of the work

more information about the system being implemented and capabilities of the development team are learnt.

Project Plans

In a plan-driven development project, a project plan sets out the resources available for the project, the work breakdown, and a schedule for carrying out the work. The plan should identify risks to the project and the software under development, and the approach that is taken to risk management.

Plans normally include the following sections

1. Introduction: Describes the objectives of the project and sets out the constraints (e.g., budget, time, etc.) that affect the management of the project.

2. Project organization: This describes the way in which the development teamis organized, the people involved, and their roles in the team.

3. Risk analysis: This describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed.

4. Hardware and software resource requirements: These specifies the hardware and support software required to carry out the development. If hardware has tobe bought, estimates of the prices and the delivery schedule may be included.

5. Work breakdown: This sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity.

6. Project schedule: Shows dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities.

7. Monitoring and reporting mechanisms: This defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

The Planning Process

Project planning is an iterative process that starts when an initial project plan is created during the project start-up phase.

At the beginning of a planning process, it is necessary to assess the constraints affecting the project. These constraints are the required delivery date, staff available, overall budget available tools, and so on. It is also necessary to identify the project milestones and deliverables. Milestones are points in the schedule against which progress can be accessed for example, the

handover of the system for testing. Deliverables are work products that are delivered to the customer.

The process then enters a loop. You draw up an estimated schedule for the project and the activities defined in the schedule are initiated or given permission to continue. After some time (usually about two to three weeks), the progress must be reviewed and discrepancies must be noted from the planned schedule. Because initial estimates of project parameters are inevitably approximate, minor slippages are normal and thereby modifications will have to be made to the original plan.

```
Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled loop
    Draw up project schedule
    Initiate activities according to schedule
    Wait ( for a while )
    Review project progress
    Revise estimates of project parameters
    Update the project schedule
    Renegotiate project constraints and deliverables
    if ( problems arise ) then
        Initiate technical review and possible revision
    end if
end loop
```

The project planning process.

4.3 Project Scheduling

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.

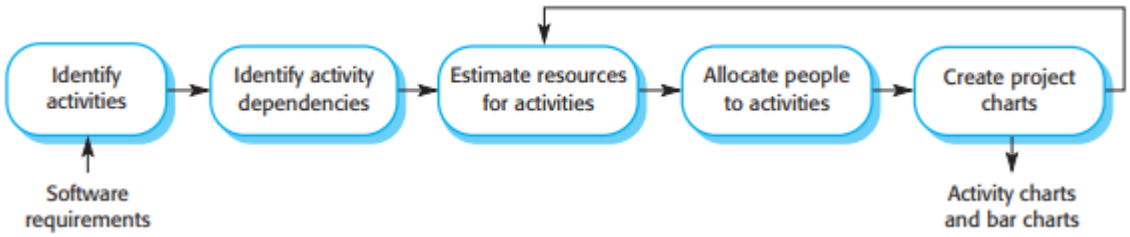
- Here there is an estimation of the calendar time needed to complete each task, the effort required, and who will work on the tasks that have been identified.

- It is essential to estimate the resources needed to complete each task, such as the diskspace required on a server, the time required on specialized hardware, such as a Simulator, and what the travel budget will be.

- Scheduling in plan-driven projects involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task

- Tasks should normally last at least a week and no longer than 2 months.

- Finer subdivision means that a disproportionate amount of time must be spent on re planning and updating the project plan.
- The maximum amount of time for any task should be around 8 to 10 weeks.
- If it takes longer than this, the task should be subdivided for project planning and scheduling.



The project scheduling process.

4.3.1 Schedule Representation

Project schedules may simply be represented in a table or spreadsheet showing the tasks, effort, expected duration, and task dependencies. There are two types of representation that are commonly used:

1. Bar charts, which are calendar-based, show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end. Bar charts are sometimes called Gantt charts.
2. Activity networks, which are network diagrams, show the dependencies between the different activities making up a project

Project Activities (tasks) are the basic planning element. Each activity has:

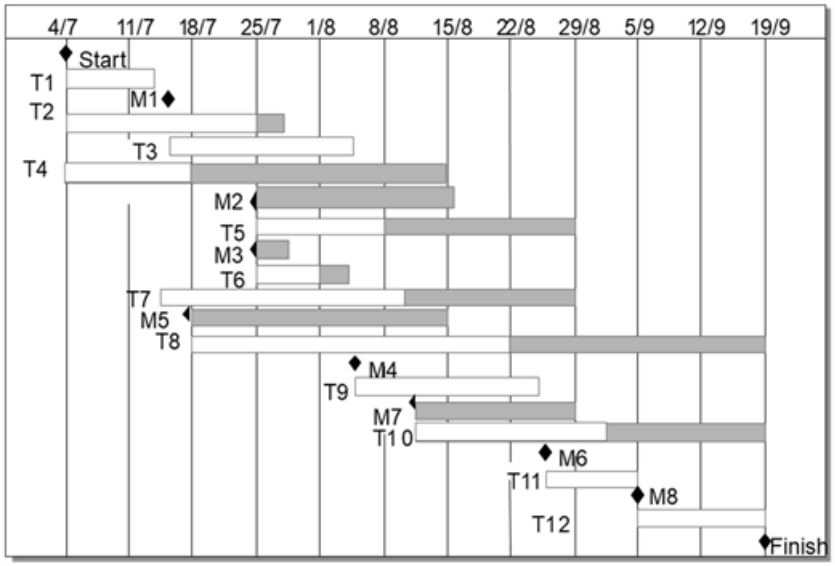
- A duration in calendar days or months,
- An effort estimate, which shows the number of person-days or person-months to complete the work,
- A deadline by which the activity should be complete,
- A defined end-point, which might be a document, the holding of a review meeting, the successful execution of all tests, etc.

Schedule representation is done using graphical notations**.** They Show project breakdown into tasks. Tasks should not be too small are too large (They should take about a week or two.). **Activity charts** show task dependencies and the critical path. **Bar charts** show schedule against calendar time.

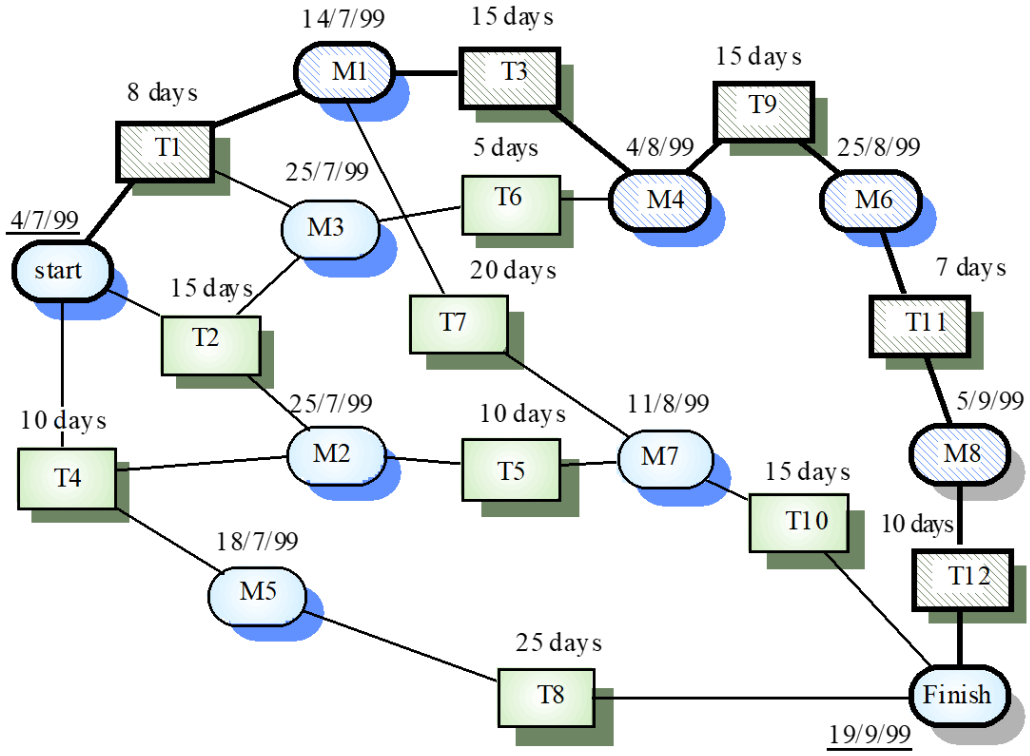
Task duration data for example:

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

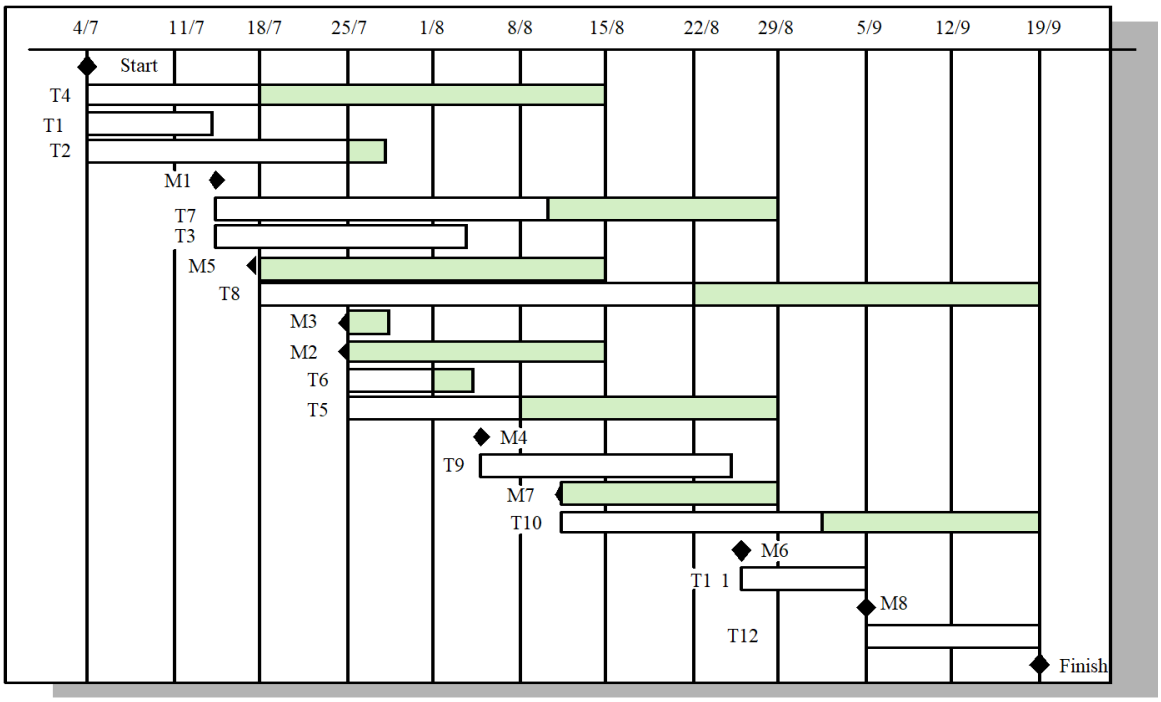
and the corresponding Bar chart is given below..



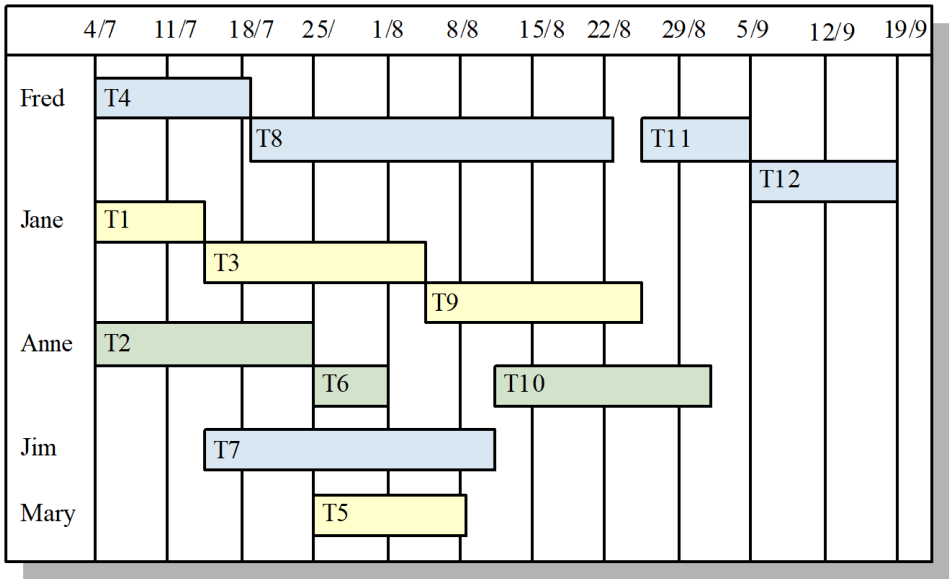
Activity Network



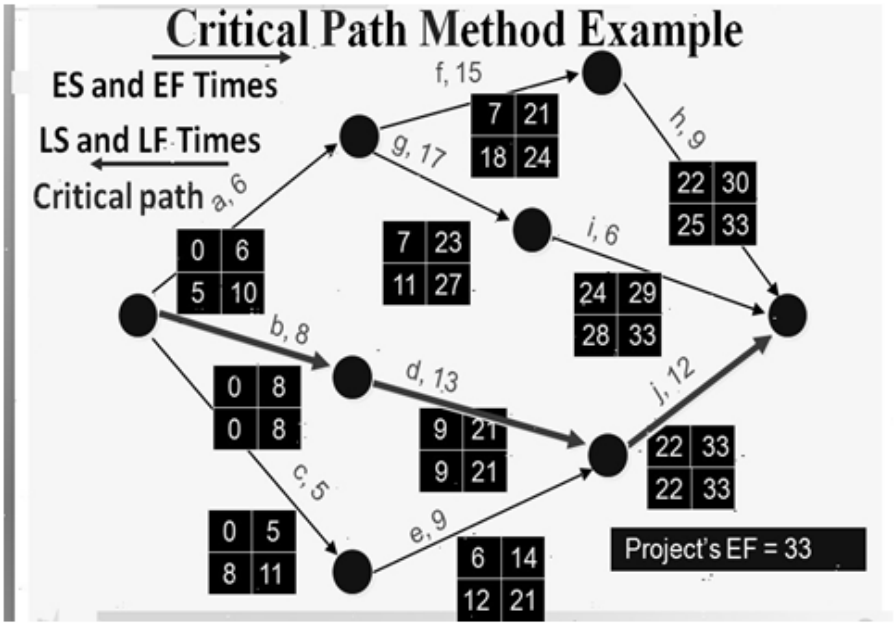
Gantt Chart



Staff Allocation Chart



Critical Path Calculation - Method



Note: For Risk Management ….refer to the PPTs