MODULE 4

SOFTWARE PROJECT MANAGEMENT

Software project management is an essential part of software engineering.

Important goals are:
1. to deliver the software to the customer at the agreed time;
2. to keep overall costs within budget
3. to deliver software that meets the customer's expectations;
4. to maintain a coherent and well-functioning development team.

The fundamental **project management activities** that are common to all organizations:
1. Project planning → Project managers are responsible for planning, estimating, and scheduling project development and assigning people to tasks.
2. Risk management → Project managers have to assess the risks that may affect a project, monitor these risks, and take action when problems arise.
3. People management → Project managers are responsible for managing a team of people. They have to choose people for their team and establish ways of working that lead to effective team performance.

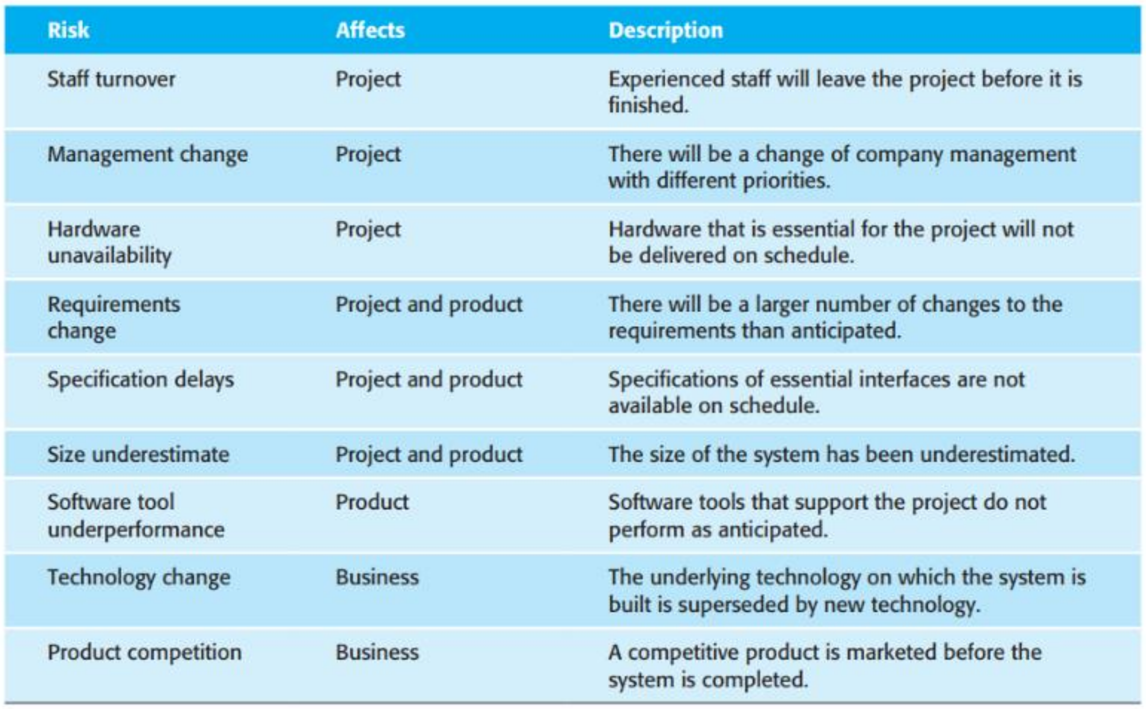
4. Reporting → Project managers are usually responsible for reporting on the progressof a project to customers and to the managers of the company developing the software.

5. Proposal writing → The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work. The proposal describes the objectives of the project and how it will be carried out. It usually includes cost and schedule estimates and justifies why the project contract should be awarded to a particular organization or team.

RISK MANAGEMENT

- Risk management is one of the most important jobs for a project manager.
- Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks.
- Risks can be categorized according to type of risk (technical, organizational, etc.)

Classification of risks according to what these risks affect:
1. Project risks → affect the project schedule or resources. An example of a project risk is the loss of an experienced system architect.
2. Product risks → affect the quality or performance of the software being developed. An example of a product risk is the failure of a purchased component to perform as expected.
3. Business risks → affect the organization developing or procuring the software. For example, a competitor introducing a new product is a business risk.



Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of company management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
Software tool underperformance	Product	Software tools that support the project do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Fig: Examples of common project, product, and business risks

Effective risk management makes it easier to cope with problems and to ensure that these do not lead to unacceptable budget or schedule slippage.

• For small projects, formal risk recording may not be required, but the project manager should be aware of them.

• The specific risks that may affect a project depend on the project and the organizational environment in which the software is being developed

• Software risk management is important because of the inherent uncertainties in software development.

RISK MANAGEMENT PROCESS

An outline of the process of risk management is presented inFigure. It involves several stages:

1.Risk identification→ You should identify possible project, product,and business risks.

2. Risk analysis → You should assess the likelihood and consequences of these risks.

3. Risk planning → You should make plans to address the risk, either by avoiding it or by minimizing its effects on the project.

4. Risk monitoring → You should regularly assess the risk and your plans for risk mitigation and revise these plans when you learn more about the risk.

•The risk management process is an iterative process that continues throughout a project.

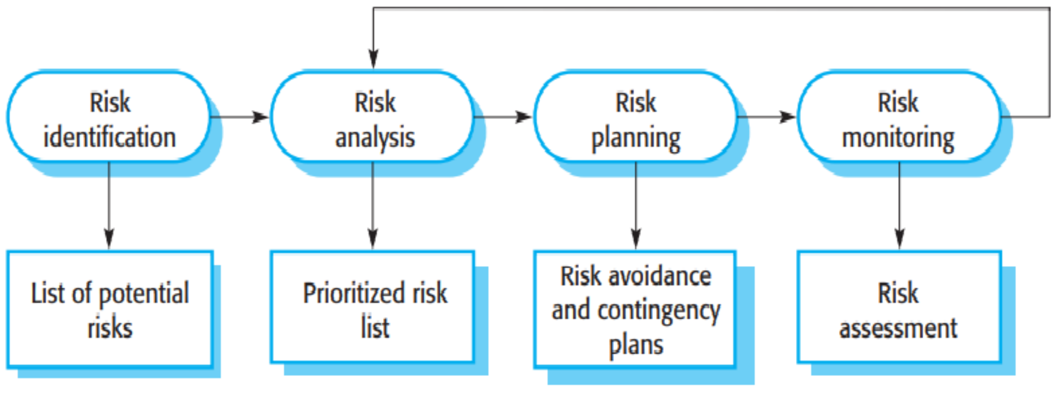


Fig: The Risk Management Process

<u>Risk Identification</u>:

• Risk identification is the first stage of the risk management process.

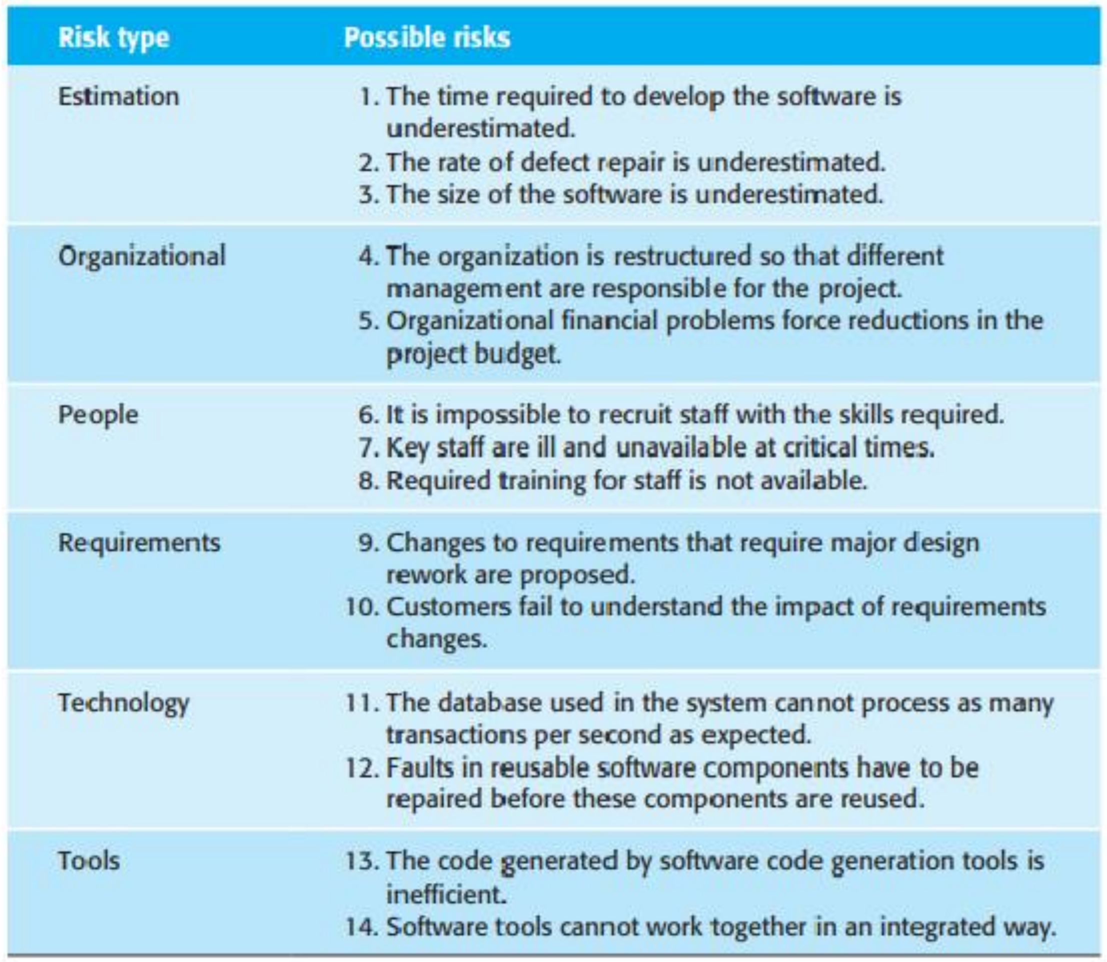
• It is concerned with identifying the risks that could pose a major threat to the software engineering process, the software being developed, or the development organization.

• Risk identification may be a team process in which a team gets together to brainstorm possible risks.

• As a starting point for risk identification, a checklist of different types of risk may be used

<u>6 types of risk may be included in a risk checklist:</u>

1. Estimation risks → arise from the management estimates of the resources required to build the system.
2. Organizational risks → arise from the organizational environment where the software is being developed.
3. People risks → are associated with the people in the development team.
4. Requirements risks → come from changes to the customer requirements and the process of managing the requirements change.
5. Technology risks → come from the software or hardware technologies that are used to develop the system.
6. Tools risks → come from the software tools and other support software used to develop the system.



Risk type	Possible risks
Estimation	1. The time required to develop the software is underestimated. 2. The rate of defect repair is underestimated. 3. The size of the software is underestimated.
Organizational	4. The organization is restructured so that different management are responsible for the project. 5. Organizational financial problems force reductions in the project budget.
People	6. It is impossible to recruit staff with the skills required. 7. Key staff are ill and unavailable at critical times. 8. Required training for staff is not available.
Requirements	9. Changes to requirements that require major design rework are proposed. 10. Customers fail to understand the impact of requirements changes.
Technology	11. The database used in the system cannot process as many transactions per second as expected. 12. Faults in reusable software components have to be repaired before these components are reused.
Tools	13. The code generated by software code generation tools is inefficient. 14. Software tools cannot work together in an integrated way.

<u>Risk Analysis:</u>

• During the risk analysis process, you have to consider each identified risk and make a judgment about the probability and seriousness of that risk.

• It is not possible to make precise, numeric assessment of the probability and seriousness of each risk.

• You should assign the risk to one of a number of bands:

1. The probability of the risk might be assessed as insignificant, low, moderate, high, or very high.
2. The effects of the risk might be assessed as catastrophic (threaten the survival of the project), serious (would cause major delays), tolerable (delays are within allowed contingency), or insignificant.

• You may then tabulate the results of this analysis process using a table ordered according to the seriousness of the risk.

Risk Planning:

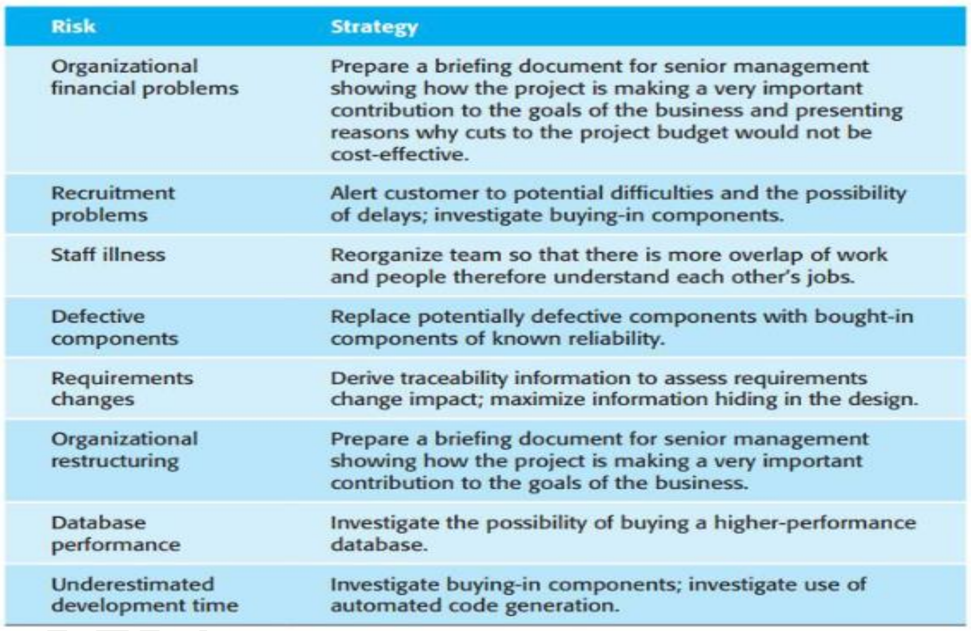
• The risk planning process develops strategies to manage the key risks that threaten the project.

• For each risk, you have to think of actions that you might take to minimize the disruption to the project if the problem identified in the risk occurs.

• You should also think about the information that you need to collect while monitoring the project so that emerging problems can be detected before they become serious.

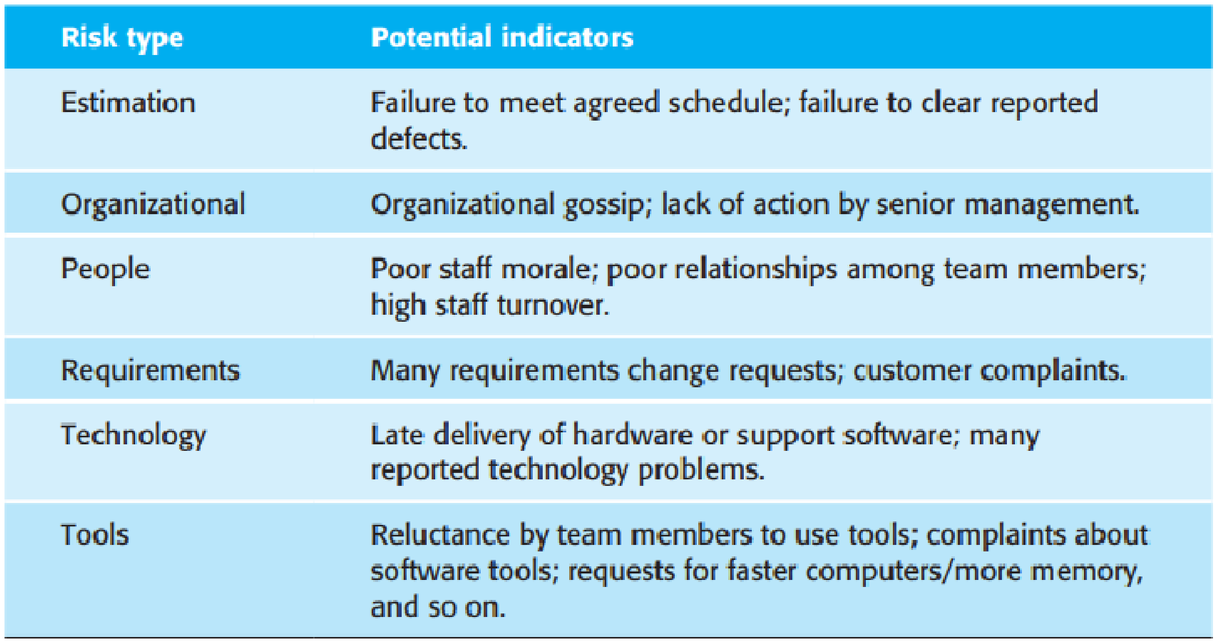
The possible **risk management strategies** fall into 3 categories:
1. **Avoidance strategies** → Following these strategies means that the probability that the risk will arise is reduced. An example of a risk avoidance strategy is the strategy for dealing with defective components.
2. **Minimization strategies** → Following these strategies means that the impact of the risk is reduced. An example of a risk minimization strategy is the strategy for staff illness.
3. **Contingency plans** → Following these strategies means that you are prepared for the worst and have a strategy in place to deal with it. An example of a contingency strategy is the strategy for organizational financial problems.

Risk Monitoring:
• Risk monitoring is the process of checking that your assumptions about the product, process, and business risks have not changed.
• You should regularly assess each of the identified risks to decide whether or not that risk is becoming more or less probable.



Risk	Strategy
Organizational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost-effective.
Recruitment problems	Alert customer to potential difficulties and the possibility of delays; investigate buying-in components.
Staff illness	Reorganize team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact; maximize information hiding in the design.
Organizational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying-in components; investigate use of automated code generation.

Risk Indicators



Risk type	Potential indicators
Estimation	Failure to meet agreed schedule; failure to clear reported defects.
Organizational	Organizational gossip; lack of action by senior management.
People	Poor staff morale; poor relationships among team members; high staff turnover.
Requirements	Many requirements change requests; customer complaints.
Technology	Late delivery of hardware or support software; many reported technology problems.
Tools	Reluctance by team members to use tools; complaints about software tools; requests for faster computers/more memory, and so on.

MANAGING PEOPLE

The people working in a software organization are its greatest assets.

•It is expensive to recruit and retain good people.

• Software managers have to ensure that the engineers working on a project are as productive as possible.

It is important that software project managers understand the technical issues that influence the work of software development.

• Software engineers often have strong technical skills but may lack the softer skills that enable them to motivate and lead a project development team.

• As a project manager, you should be aware of the potential problems of people management and should try to develop people management skills.

4 critical factors that influence the relationship between a manager and the people that he or she manages:

1. Consistency → All the people in a project team should be treated in a comparable way. No one expects all rewards to be identical, but people should not feel that their contribution to the organization is undervalued.

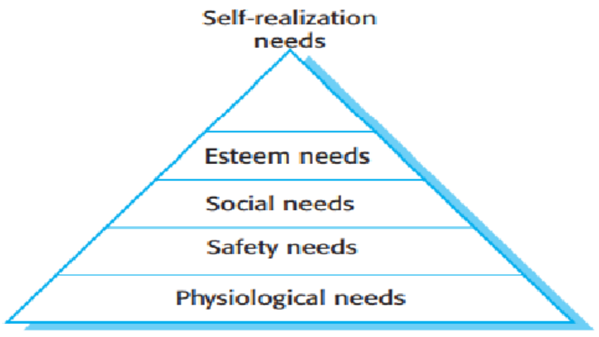
2. Respect → Different people have different skills, and managers should respect these differences.
3. Inclusion → People contribute effectively when they feel that others listen to them and take

account of their proposals. It is important to develop a working environment where all views, even those of the least experienced staff, are considered.

4. Honesty → As a manager, you should always be honest about what is going well and what is going badly in the team. You should also be honest about your level of technical knowledge and be willing to defer to staff with more knowledge when necessary.

Motivating People:

As a project manager, you need to motivate the people who work with you so that they will contribute to the best of their abilities.
• In practice, motivation means organizing work and its environment to encourage people to work as effectively as possible.
• To provide this encouragement, you should understand a little about what motivates people.
• People are motivated by satisfying their needs. These needs are arranged in a series of levels, as shown in Figure.



The lower levels of this hierarchy represent fundamental needs for food, sleep, and so on, and the need to feel secure in an environment.
• **Social need** is concerned with the need to feel part of a social grouping.
• **Esteem need** represents the need to feel respected by others, and self-realization need is concerned with personal development.
• **People need** to satisfy lower-level needs such as hunger before the more abstract, higher-level needs.
• People working in software development organizations are not usuallyhungry, thirsty, or physically threatened by their environment. Therefore,
making sure that peoples' social, esteem, and self-realization needs are satisfied is most important from a management point of view.

1. To satisfy social needs, you need to give people time to meet their co-workers and provide places for them to meet. This is relatively easy when all of the members of a development team work in the same place. Social networking systems and teleconferencing can be used for remote communications.

2. To satisfy esteem needs, you need to show people that they are valued by the organization. Public recognition of achievements is a simple and effective way of doing this.

3. Finally, to satisfy self-realization needs, you need to give people responsibility for their work, assign them demanding (but not impossible) tasks, and provide opportunities for training and development where people can enhance their skills. Training is an important motivating influence as people like to gain new knowledge and learn new skills.

Bass and Dunteman (Bass and Dunteman 1963) identified 3 classifications for professional workers:

1. **Task-oriented people** → who are motivated by the work they do. In software engineering, these are people who are motivated by the intellectual challenge of software development.

2. **Self-oriented people** → who are principally motivated by personal success and recognition. They are interested in software development as a means of achieving their own goals. They often have longer-term goals and they wish to be successful in their work to help realize these goals.

3. **Interaction-oriented people** → who are motivated by the presence and actions of co-workers.

People Capability Maturity Model (P-CMM) → is a framework forassessing how well organizations manage the development of their staff. It highlights best practice in people management and provides a basis for organizations to improve their people management processes. It is best suited to large rather than small, informal companies.

Teamwork

•As it is impossible for everyone in a large group to work together on a single problem, large teams are usually split into a number of smaller groups.

• Each group is responsible for developing part of the overall system.

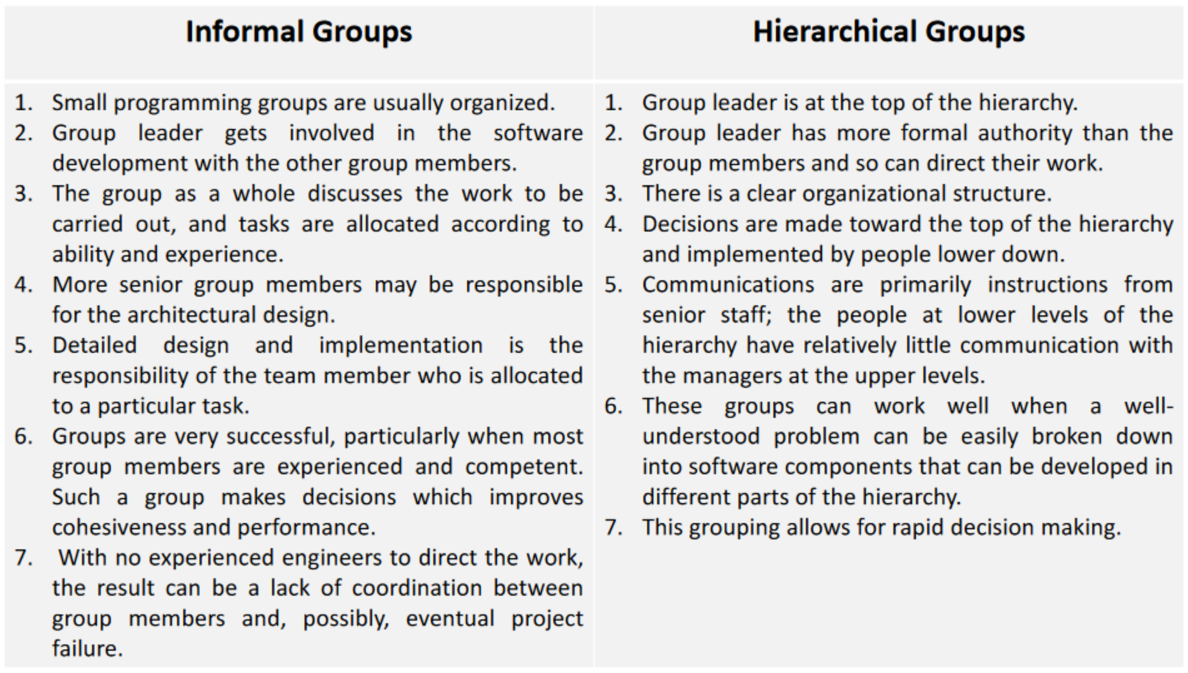
• The best size for a software engineering group is 4 to 6 members, and they should never have more than 12 members.

• When groups are small, communication problems are reduced.

In a **cohesive group**, members think of the group as more important than the individuals who are group members.
➢They are loyal to the group.
➢They identify with group goals and other group members.
➢They attempt to protect the group, as an entity, from outside interference. This makes the group robust and able to cope with problems and unexpected situations.

- **The benefits of creating a cohesive group are:**
  1. The group can establish its own quality standards.
  2. Individuals learn from and support each other.
  3. Knowledge is shared.
  4. Refactoring and continual improvement is encouraged
- Good project managers should always try to encourage group cohesiveness.

- A manager or team leader's job is to create a cohesive group and organize that group so that they work together effectively.
- This task involves selecting a group with the right balance of technical skills and personalities.
- Technical knowledge and ability should not be the only factor used to select group members.
- People who are motivated by the work are likely to be the strongest technically
- People who are self-oriented will probably be best at pushing the work forward to finish the job.
- People who are interaction-oriented help facilitate communications within the group.
- The project manager has to control the group so that individual goals do not take precedence over organizational and group objectives.
- This control is easier to achieve if all group members participate in each stage of the project.
- Individual initiative is most likely to develop when group members are given instructions without being aware of the part that their task plays in the overall project.
- If all the members of the group are involved in the design from the start, they are more likely to understand why design decisions have been made. They may then identify with these decisions rather than oppose them.



Informal Groups	Hierarchical Groups
1. Small programming groups are usually organized. 2. Group leader gets involved in the software development with the other group members. 3. The group as a whole discusses the work to be carried out, and tasks are allocated according to ability and experience. 4. More senior group members may be responsible for the architectural design. 5. Detailed design and implementation is the responsibility of the team member who is allocated to a particular task. 6. Groups are very successful, particularly when most group members are experienced and competent. Such a group makes decisions which improves cohesiveness and performance. 7. With no experienced engineers to direct the work, the result can be a lack of coordination between group members and, possibly, eventual project failure.	1. Group leader is at the top of the hierarchy. 2. Group leader has more formal authority than the group members and so can direct their work. 3. There is a clear organizational structure. 4. Decisions are made toward the top of the hierarchy and implemented by people lower down. 5. Communications are primarily instructions from senior staff; the people at lower levels of the hierarchy have relatively little communication with the managers at the upper levels. 6. These groups can work well when a well-understood problem can be easily broken down into software components that can be developed in different parts of the hierarchy. 7. This grouping allows for rapid decision making.

Lakshmi M B. SCET

Group Communications:

• It is absolutely essential that group members communicate effectively and efficiently with each other and with other project stakeholders.

• Good communication also helps strengthen group cohesiveness.

• Group members:

1. Exchange information on the status of their work, the design decisions that have been made, and changes to previous design decisions.
2. Resolve problems that arise with other stakeholders and inform these stakeholders of

changes to the system, the group, and delivery plans.
3. Come to understand the motivations, strengths, and weaknesses of other people in the group.

Project planning

Three main parameters should be used when computing the costs of a software development project:

■ effort costs (the costs of paying software engineers and managers);
■ hardware and software costs, including hardware maintenance and software support; and
■ travel and training costs.

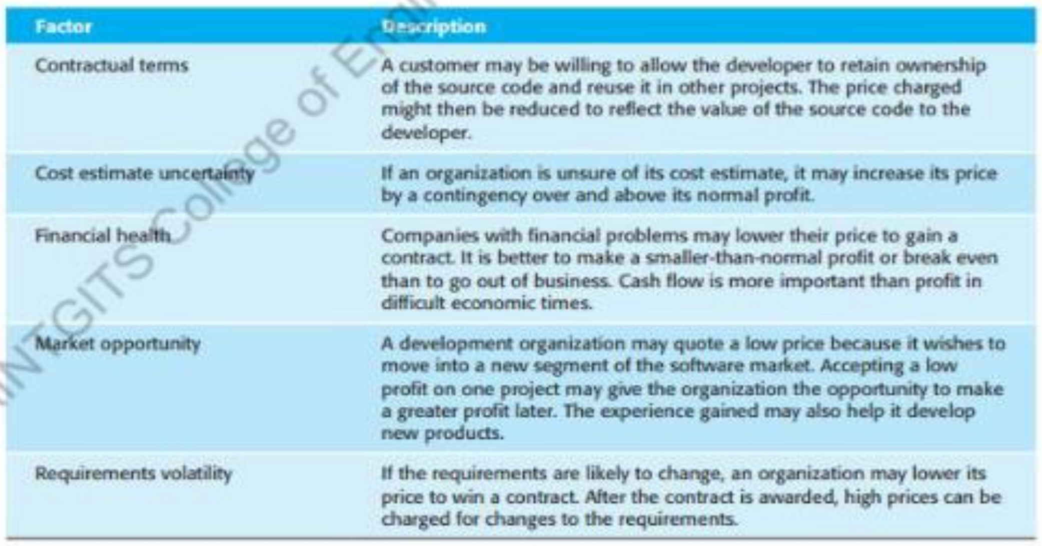
For most projects, the biggest cost is the effort cost.
You have to estimate the total effort (in person-months) that is likely to be required to complete the work of a project.

SOFTWARE PRICING

In principle, the price of a software system developed for a customer is simply the cost of development plus profit for the developer.

• In practice, however, the relationship between the project cost and the price quoted to the customer is not usually so simple.
• When calculating a price, you take broader organizational, economic, political, and business considerations into account

FACTORS AFFECTING SOFTWARE PRICING



Factor	Description
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged might then be reduced to reflect the value of the source code to the developer.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Financial health	Companies with financial problems may lower their price to gain a contract. It is better to make a smaller-than-normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

PLAN DRIVEN DEVELOPMENT

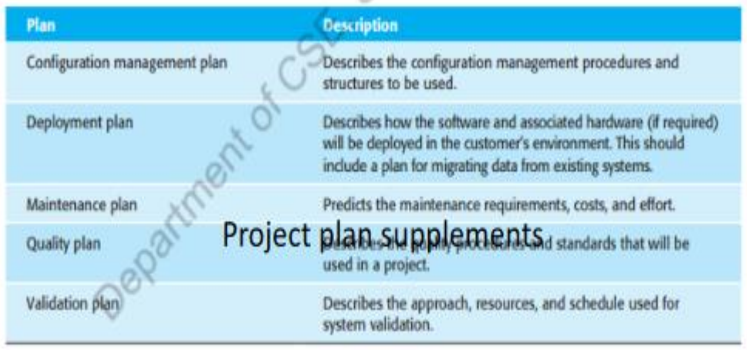
Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.

• A project plan is created that records the work to be done, who will do it, the development schedule, and the work products.

• Managers use the plan to support project decision making and as a way of measuring progress.
• Agile development involves a different planning process, where decisions are delayed.
• The problem with plan-driven development is that early decisions have to be revised because of changes to the environments in which the software is developed and used.• Delaying planning decisions avoids unnecessary rework.
• However, the arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be taken into account. Potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
• The best approach to project planning involves a sensible mixture of plan-based and agile development.

SECTIONS OF PROJECT PLAN

1. Introduction: Briefly describes the objectives of the project and sets out the constraints (e.g., budget, time) that affect the management of the project.
2. Project organization :Describes the way in which the development team is organized, the people involved, and their roles in the team.
3. Risk analysis: Describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed.
4. Hardware and software resource requirements: Specifies the hardware and support software required to carry out the development. If hardware has to be purchased, estimates of the prices and the delivery schedule may be included.
5. Work breakdown: Sets out the breakdown of the project into activities and identifies the inputs to and the outputs from each project activity.
6. Project schedule: Shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities. The ways in which the schedule may be presented are discussed in the next section of the chapter.
7. Monitoring and reporting mechanisms: Defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be use.



Plan	Description
Configuration management plan	Describes the configuration management procedures and structures to be used.
Deployment plan	Describes how the software and associated hardware (if required) will be deployed in the customer's environment. This should include a plan for migrating data from existing systems.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.

Project plan supplements

PLANNING PROCESS

Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.

•Figure is a UML activity diagram that shows a typical workflow for a project planning process.

•Plan changes are inevitable. As more information about the system and the project team becomes available during the project, you should regularly revise the plan to reflect requirements, schedule, and risk changes. Changing business goals also leads to changes in project plans.

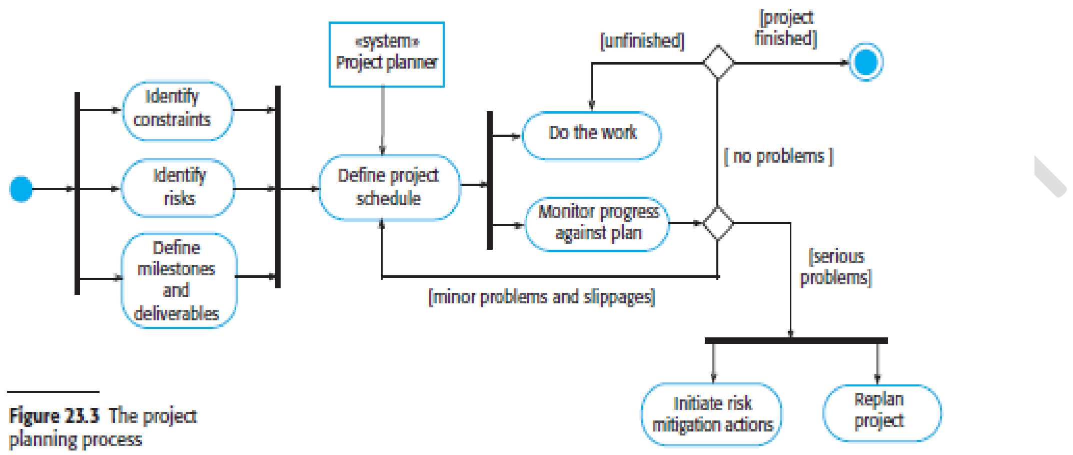


Figure 23.3 The project planning process

At the beginning of a planning process, you should assess the constraints affecting the project. These constraints are the required delivery date, staff available, overall budget, available tools, and so on. In conjunction with this assessment, you should also identify the project milestones and deliverables.
Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing. **Deliverables** are work products that are delivered to the customer, for example, a requirements document for the system.
The process then enters a loop that terminates when the project is complete. You draw up an estimated schedule for the project, and the activities defined in the schedule are initiated or are approved to continue. After some time (usually about two to three weeks), you should review progress and note discrepancies from the planned schedule.
Because initial estimates of project parameters are inevitably approximate, minor slippages are normal and you will have to make modifications to the original plan.

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.
•You estimate the calendar time needed to complete each task and the effort required, and you suggest who will work on the tasks that have been identified.
•You also have to estimate the hardware and software resources that are needed to complete each task.
•An initial project schedule is usually created during the project startup phase. This schedule is then refined and modified during development planning.
Scheduling in plan-driven projects (Figure) 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.

•The maximum amount of time for any task should be 6 to 8 weeks. If a task will take longer than this, it should be split intosubtasks for project planning and scheduling.

•Some of these tasks are carried out in parallel, with different people working on different components of the system. You have to coordinate these parallel tasks and organize the work so that the workforce is used optimally and you don't introduce unnecessary dependencies between the tasks.

•It is important to avoid a situation where the whole project is delayed because a critical task is unfinished.

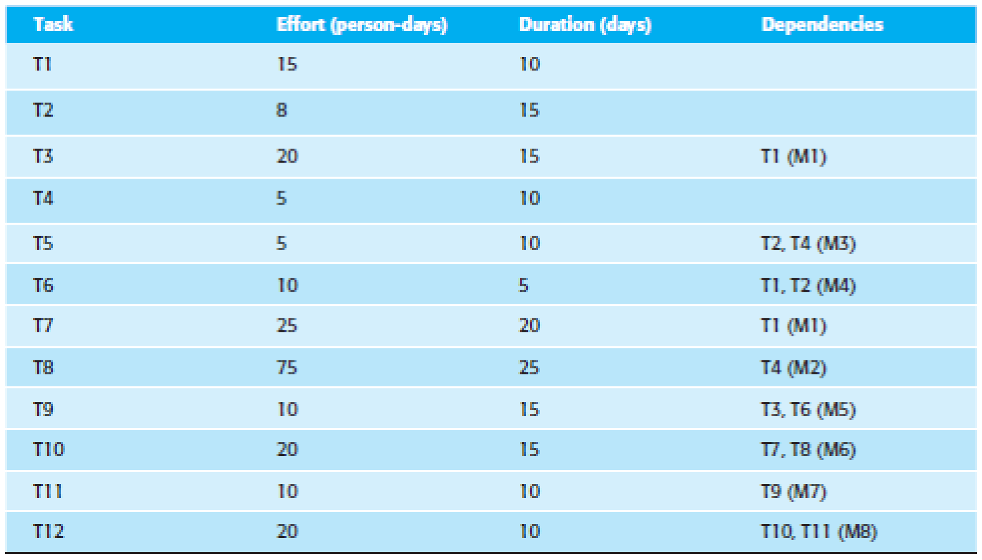
•When you are estimating schedules, you must take into account the possibility that things will go wrong. People working on a project may fall ill or leave, hardware may fail, and essential support software or hardware may be delivered late.

•If the project is new and technically advanced, parts of it may turn out to be more difficult and take longer than originally anticipated.

•A good rule of thumb is to estimate as if nothing will go wrong and then increase your estimate to cover anticipated problems.

•A further contingency factor to cover unanticipated problems may also be added to the estimate. This extra contingency factordepends on the type of project, the process parameters (deadline, standards, etc.), and the quality and experience of the software engineers workingon the project.

Project schedules may simply be documented in a table or spreadsheet showing the tasks, estimated effort, duration, and task dependencies (Figure 23.5). However, this style of presentation makes it difficult to see the relationships and dependencies between the different activities. For this reason, alternative graphical visualizations of project schedules have been developed that are often easier to read and understand.
Two types of visualization are commonly used:
1. Calendar-based bar charts show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end. Barcharts are also called Gantt charts, after their inventor, Henry Gantt.
2. Activity networks show the dependencies between the different activities making up a project. These networks are described in an associated web section.
Project activities 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; and
■ a defined endpoint, which might be a document, the holding of a review meeting, the successful execution of all tests, or the like.



Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Figure 23.5 Tasks, durations, and dependencies

2 Activity networks show the dependencies between the different activities mak

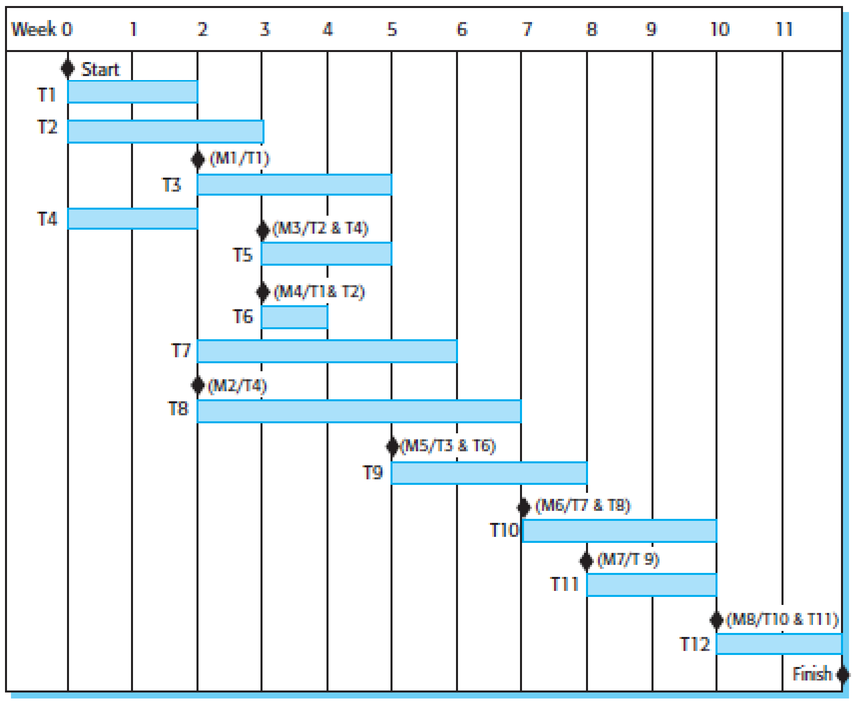


Figure 23.6 Activity bar chart

[illegible]

- Figure 23.6 takes the information in Figure 23.5 and presents the project schedule as a bar chart showing a project calendar and the start and finish dates of tasks.
- Reading from left to right, the bar chart clearly shows when tasks start and end. The milestones (M1, M2, etc.) are also shown on the bar chart. Notice that tasks that are

independent may be carried out in parallel. For example, tasks T1, T2, and T4 all start at the beginning of the project

- If a task is delayed, later tasks that are dependent on it may be affected. They cannot start until the delayed task is completed. Delays can cause serious problems with staff allocation, especially when people are working on several projects at the same time. If a task (T) is delayed, the people allocated to it may be assigned to other work (W).
- To complete this work may take longer than the delay, but, once assigned, they cannot simply be reassigned back to the original task. This may then lead to further delays in T as they complete W.
- Normally, you should use a project planning tool, such as **the Basecamp or Microsoft project**, to create, update, and analyze project schedule information.
- Project management tools usually expect you to input project information into atable, and they create a database of project information. Bar charts and activity chartscan then be generated automatically from this database.

AGILE PLANNING

Agile development methods such as Scrum (Rubin 2013) and Extreme Programming (Beck and Andres 2004) have a two-stage approach to planning, correspondingto the startup phase in plan-driven development and development planning:

1. ***Release planning***, which looks ahead for several months and decides on the features that should be included in a release of a system.
2. ***Iteration planning***, which has a shorter term outlook and focuses on planning the next increment of a system. This usually represents 2 to 4 weeks of work for the team.

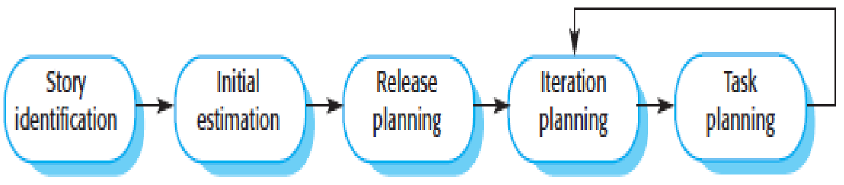


Figure 23.8 The

- **Release planning** involves selecting and refining the stories that will reflect the features to be implemented in a release of a system and the order in which the stories should be implemented.
    - The customer has to be involved in this process. A release date is then chosen, and the stories are examined to see if the effort estimate is consistent with that date.
    - If not, stories are added or removed from the list.
- **Iteration planning** is the first stage in developing a deliverable system increment.
    - Stories to be implemented during that iteration are chosen, with the number of stories reflecting the time to deliver an workable system (usually 2 or 3 weeks) and the team's velocity. When the delivery date is reached, the development iteration is complete,even if all of the stories have not been implemented.

At the start of each development iteration, there is a task planning stage where the developers break down stories into development tasks. A development task should take 4–16 hours. All of the tasks that must be completed to implement all of the stories in that iteration are listed. The individual developers then sign up for the specific planning tasks that they will implement.

•Each developer knows their individual velocity and so should not sign up for more tasks than they can implement in the time allotted

This approach to task allocation has two important benefits:

1. The whole team gets an overview of the tasks to be completed in an iteration. They therefore have an understanding of what other team members are doing and who to talk to if task dependencies are identified.
2. Individual developers choose the tasks to implement; they are not simply allocated tasks by a project manager. They therefore have a sense of ownership in these tasks, and this is likely to motivate them to complete the task.
Halfway through an iteration, progress is reviewed. At this stage, half of the story effort points should have been completed.

Advantages:

•This approach to planning has the advantage that a software increment is always delivered at the end of each project iteration.
•If the features to be included in the increment cannot be completed in the time allowed, the scope of the work is reduced.
•The delivery schedule is never extended..

Disadvantages

•A major difficulty in agile planning is that it relies on customer involvement and availability.
•This involvement can be difficult to arrange, as customer representatives sometimes have to prioritize other work and are notavailable for the planning game.
•Furthermore, some customers may be more familiar with traditional project plans and may find it difficult to engage in an agile planning process.

Configuration management

•The configuration management of a software system product involves four closely related activities (Figure 1):
1. Version control: This involves keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.
2. System building: This is the process of assembling program components, data, and libraries, then compiling and linking these to create an executable system.
3. Change management: This involves keeping track of requests for changes to delivered software from customers and developers, working out the costs and impact of making these changes, and deciding if and when the changes should be implemented.
4. Release management: This involves preparing software for external release and keeping track of the system versions that have been released for customer use.

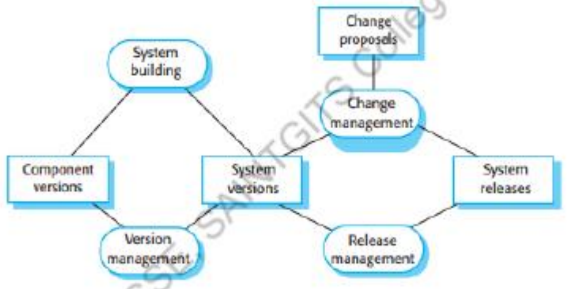


Figure 1: Configuration management activities

In large software projects, configuration management is sometimes part of software quality management.

•The **quality manager** is responsible for both quality management and configuration management.

•When a pre-release version of the software is ready, the development team hands it over to the **quality management team**.

•The QM team checks that the system quality is acceptable. If so, it then becomes a controlled system, which means that all changes to the system have to be agreed on and recorded before they are implemented.

•Many **specialized terms** are used in configuration management. Unfortunately, these are not standardized.

•Military software systems were the first systems in which software CM was used, so the terminology for these systems reflected the processes and terminology used in hardware configuration management.

•Commercial systems developers did not know about military procedures or terminology and so often invented their own terms.

•Agile methods have also devised new terminology in order to distinguish the agile approach from traditional CM methods

Version management

•A **codeline**is a sequence of versions of source code, with later versions in the sequence derived from earlier versions.

•Codelinesnormally apply to components of systems so that there are different versions of each component.

•A **baseline**is a definition of a specific system.

•The baseline specifies the component versions that are included in the system and identifies the libraries used, configuration files, and other system information.

Version control (VC) systems identify, store, and control access to the different versions of components.

•There are **two types of modern version control system**:

1.Centralized systems, where a single master repository maintains all versions of the software components that are being developed. Subversion is a widely used example of a centralized VC system.

2.**Distributed systems**, where multiple versions of the component repository exist at the same time. Git, is a widely used example of a distributed VC system.

Centralized and distributed VC systems provide comparable functionality but implement this functionality in different ways. **Key features** of these systems include:

1. Version and release identification: Managed versions of a component are assigned unique identifiers when they are submitted to the system. These identifiers allow different versions of the same component to be managed, without changing the component name. Versions may also be assigned attributes, with the set of attributes used to uniquely identify each version.

2. Change history recording: The VC system keeps records of the changes that have been made to create a new version of a component from an earlier version.

3. Independent development: Different developers may be working on the same component at the same time. The version control system keeps track of components that have been checked out for editing and ensures that changes made to a component by different developers do not interfere.

4. Project support: A version control system may support the development of several projects, which share components. It is usually possible to check in and check out all of the files associated with a project rather than having to work with one file or directory at a time.

5. Storage management: Rather than maintain separate copies of all versions of a component, the version control system may use efficient mechanisms to ensure that duplicate copies of identical files are not maintained. Where there are only small differences between files, the VC system may store these differences rather than maintain multiple copies of files. A specific version may be automatically re-created by applying the differences to a master version

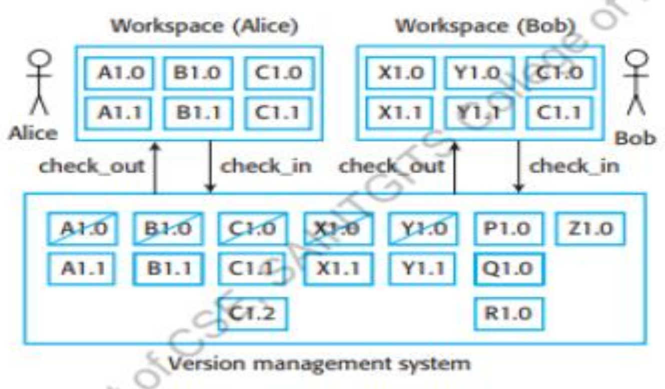


Fig 3: Check-in and check-out from a centralized version repository

In a **distributed VC system**, such as Git, a different approach is used.

•A "master" repository is created on a server that maintains the code produced by the development team.

•Instead of simply checking out the files that they need, a developer creates a clone of the project repository that is downloaded and installed on his or her computer.

•Developers work on the files required and maintain the new versions on their private repository on their own computer.

•When they have finished making changes, they "commit" these changes and update their private server repository.

•They may then "push" these changes to the project repository or tell the integration manager that changed versions are available.

•He or she may then "pull" these files to the project repository (see Figure 4). In this example, both Bob and Alice have cloned the project repository and have updated files.

•They have not yet pushed these back to the project repository.

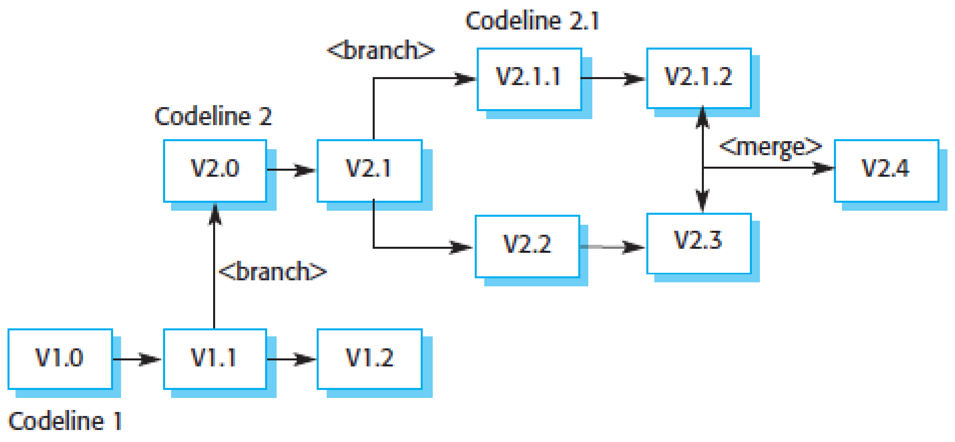
This model of development has a number of advantages:

1. It provides a backup mechanism for the repository. If the repository is corrupted, work can continue and the project repository can be restored from local copies.

2. It allows for offline working so that developers can commit changes if they do not have a network connection.

3. Project support is the default way of working. Developers can compile and test the entire system on their local machines and test the changes they have made.

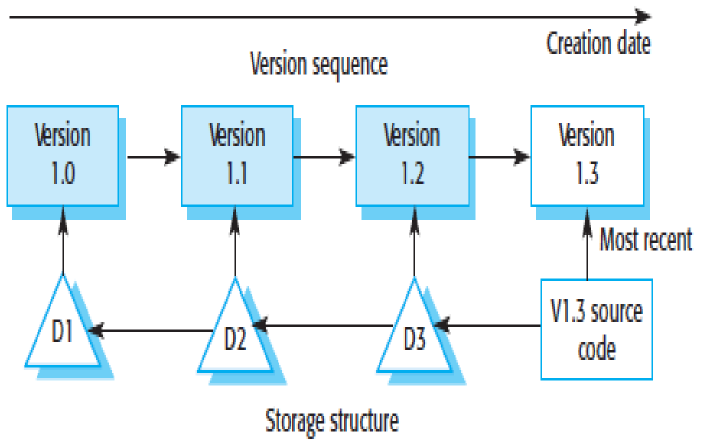
Branching and Merging



- A consequence of the independent development of the same component is that codelines may branch.
- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences, as shown in Figure 25.8. This is normal in system development, where different developers work independently on different versions of the source code and change it in different ways.

- **It is generally recommended when working on a system that a new branch should be created so that changes do not accidentally break a working system.**
- At some stage, it may be necessary to merge codeline branches to create a new version of a component that includes all changes that have been made. This is also shown in Figure 25.8, where component versions 2.1.2 and 2.3 are merged to create version 2.4.
- If the changes made involve completely different parts of the code, the component versions may be merged automatically by the version control system by combining the code changes.

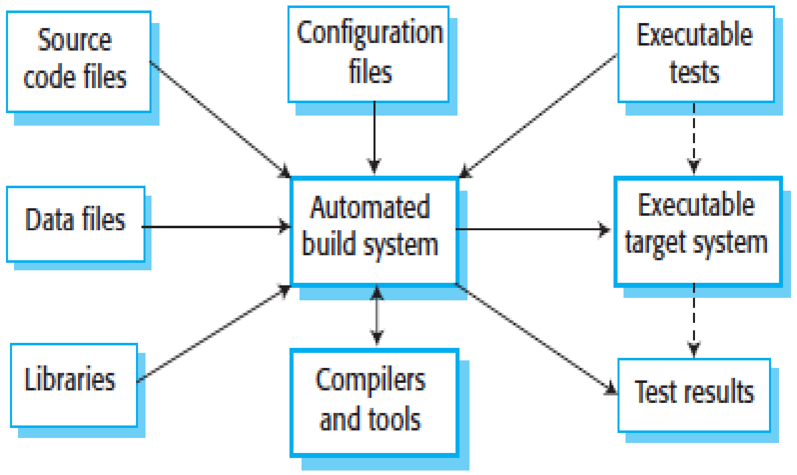
DELTA BASED VERSION MANAGEMENT



- When version control systems were first developed, **storage management** was one of their most important functions. Disk space was expensive, and it was important to minimize the disk space used by the different copies of components.
- Instead of keeping a complete copy of each version, the **system stores a list of differences (deltas) between one version and another.**
- By applying these to a master version (usually the most recent version), a target version can be re-created. This is illustratedin Figure 7.
- When a new version is created, the system simply stores a delta, a list of differences, between the new version and the olderversion used to create that new version.
- In Figure 7, the shaded boxes represent earlier versions of a component that are automatically re-created from the most recent component version.
- Deltas are usually stored as lists of changed lines, and, by applying these automatically, one version of a component can be created from another.
- As the most recent version of a component will most likely be the one used, most systems store that version in full. The deltasthen define how to re-create earlier system versions.
- **One of the problems with a delta-based approach to storage management is that it can take a long time to apply all of the deltas.**
- As disk storage is now relatively cheap, Gituses an alternative, faster approach. Git does not use deltas but applies a standard **compression algorithm** to stored files and their associated meta-information. It does not store duplicate copies of files.

System Building

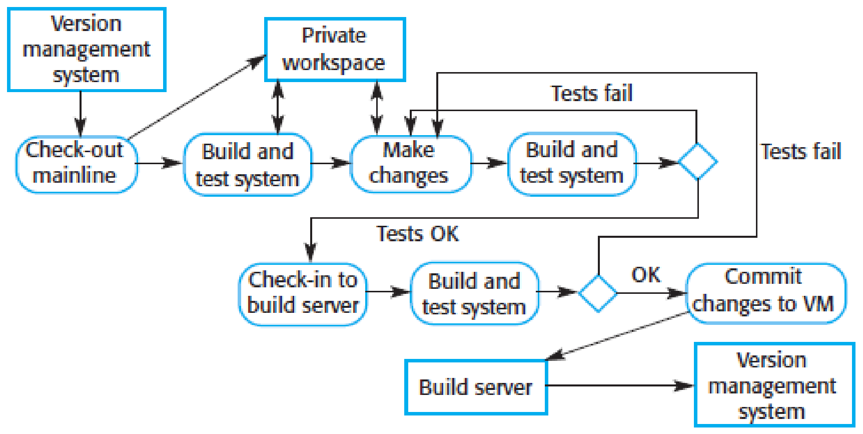
System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, and other information.
System-building tools and version control tools must be integrated as the build process takes component versions from the repository managed by the version control system.



Tools for system integration and building include some or all of the following features:

1. *Build script generation* The build system should analyze the program that is being built, identify dependent components, and automatically generate a build script (configuration file). The system should also support the manual creation and editing of build scripts.
2. *Version control system integration* The build system should check out the required versions of components from the version control system.
3. *Minimal recompilation* The build system should work out what source code needs to be recompiled and set up compilations if required.
4. *Executable system creation* The build system should link the compiled object code files with each other and with other required files, such as libraries and configuration files, to create an executable system.
5. *Test automation* Some build systems can automatically run automated tests using test automation tools such as JUnit. These check that the build has not been “broken” by changes.
6. *Reporting* The build system should provide reports about the success or failure of the build and the tests that have been run.
7. *Documentation generation* The build system may be able to generate release notes about the build and system help pages.

CONTINOUS INTEGRATION



In keeping with the agile methods notion of making many small changes, continuous integration involves rebuilding the mainline frequently, after small source code changes have been made. The steps in continuous integration are:

1. Extract the mainline system from the VC system into the developer's private workspace.
2. Build the system and run automated tests to ensure that the built system passes all tests. If not, the build is broken, and you should inform whoever checked in the last baseline system. He or she is responsible for repairing the problem.
3. Make the changes to the system components.
4. Build the system in a private workspace and rerun system tests. If the tests fail, continue editing.
5. Once the system has passed its tests, check it into the build system server but do not commit it as a new system baseline in the VC system.
6. Build the system on the build server and run the tests. Alternatively, if you are using Git, you can pull recent changes from the server to your private workspace. You need to do this in case others have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
7. If the system passes its tests on the build system, then commit the changes you have made as a new baseline in the system mainline.

CHANGE MANAGEMENT

Change is a fact of life for large software systems. Organizational needs and requirements change during the lifetime of a system, bugs have to be repaired, and systems have to adapt to changes in their environment.

•To ensure that the changes are applied to the system in a controlled way, you need a set of tool-supported, change management processes.

•Change management is intended to ensure that the evolution of the system is controlled and that the most urgent and cost-effective changes are prioritized.

•Change management is the process of analyzing the costs and benefits of proposed changes, approving those changes that are cost-effective, and tracking which components in the system have been changed.

•Figure 11 is a model of a change management process that shows the main change management activities. This process should come into effect when the software is handed over for release to customers or for deployment within an organization

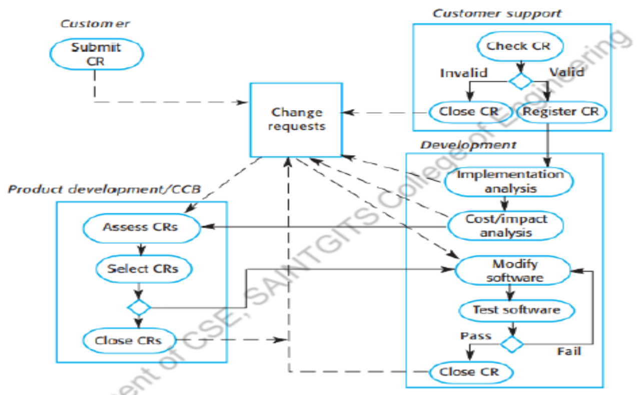


Figure 11: The change management process

System developers decide how to implement the change and estimate the time required to complete the change implementation.
•After a change request has been submitted, it is checked to ensure that it is valid.
•The checker may be from a customer or application support team or, for internal requests, may be a member of the development team. **The change request may be rejected at this stage**.
•If the change request is a bug report, the bug may have already been reported and repaired.
•Sometimes, what people believe to be problems are actually misunderstandings of what the system is expected to do.
•On occasions, people request features that have already been implemented but that they don't know about.
•If any of these features are true(ie. the change is not valid), the issue is closed and the form is updated with the reason for closure.
•If it is a valid change request, it is then logged as an outstanding request for subsequent analysis.
•For valid change requests, the next stage of the process is change assessment and costing.
•This function is usually the responsibility of the development or maintenance team as they can work out what is involved in implementing the change.

Tools to support change management may be relatively simple issue or bug tracking systems or software that is integrated with a configuration management package for large-scale systems, such as Rational Clearcase.
•Issue tracking systems allow anyone to report a bug or make a suggestion for a system change, and they keep track of how the development team has responded to the issues.
•More complex systems are built around a process model of the change management process. They automate the entire process of handling change requests from the initial customer proposal to final change approval and change submission to the development team.

RELEASE MANAGEMENT

•Release creation is the process of creating the collection of files and documentation that include all components of the system release.
•This process involves several steps:

1.The executable code of the programs and all associated data files must be identified in the version control system and tagged with the release identifier.
2.Configuration descriptions may have to be written for different hardware and operating systems.
3.Updated instructions may have to be written for customers who need to configure their own systems.
4.Scripts for the installation program may have to be written.
5.Web pages have to be created describing the release, with links to system documentation.
6.Finally, when all information is available, an executable master image of the software must be prepared and handed over for distribution to customers or sales outlets.

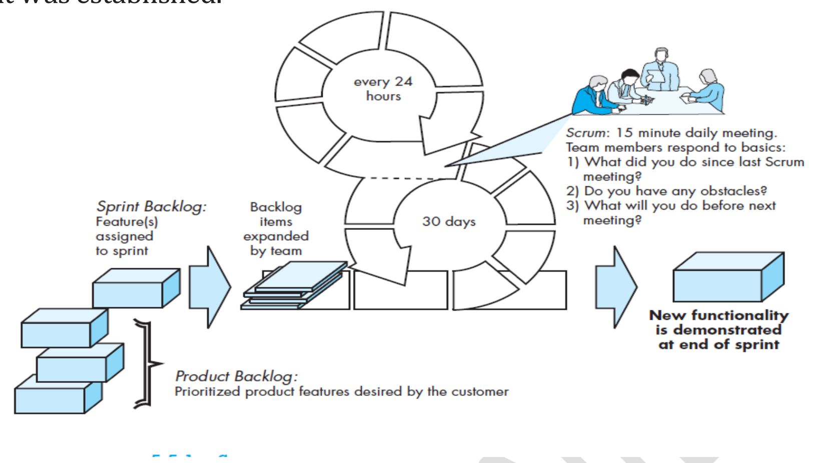
When planning the installation of new system releases, you cannot assume that customers will always install new system releases. Some system users may be happy an existing system and may not consider it worthwhile to absorb the cost of changing to a new release.
•New releases of the system cannot, therefore, rely on the installation of previous releases.
•One benefit of delivering software as a service (SaaS) is that it avoids all of these problems.
•It simplifies both release management and system installation for customers.
•The software developer is responsible for replacing the existing release of a system with a new release, which is made available to all customers at the same time.
•However, this approach requires that all servers running the services be updated at the same time. To support server updates, specialized distribution management tools such as Puppet have been developed for "pushing" new software to servers.

SCRUM FRAMEWORK

Scrum (the name is derived from an activity that occurs during a rugby match) is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s.
Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery. Within each framework activity, work tasks occur within a process pattern (discussed in the following paragraph) called a ***sprint.*** The work conducted within a sprint (the number of sprints required for each framework activity will vary depending on product complexity and size) is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.

Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality. Each of these process patterns defines a set of development activities:
Backlog a prioritized list of project requirements or features that providbusiness value for the customer. Items can be added to the backlog at any time(this is how changes are introduced). The product manager assesses the backlog
and updates priorities as required.
Sprints —consist of work units that are required to achieve a requirement defined in the backlog that must be fi t into a predefi ned time-box 10 (typically30 days). Changes (e.g., backlog work items) are not introduced during the sprint. Hence, the sprint allows team members to work in a short-term, butstable environment.
Scrum meetings —are short (typically 15-minute) meetings held daily by theScrum team. Three key questions are asked and answered by all team members
[Noy02]:
• What did you do since the last team meeting?

• What obstacles are you encountering?
• What do you plan to accomplish by the next team meeting?
A team leader, called a *Scrum master,* leads the meeting and assesses the responses from each person. The Scrum meeting helps the team to uncover potential problems as early as possible. Also, these daily meetings lead to "knowledgesocialization" and thereby promote a self-organizing team structure.
Demos —deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.
It is important to note that the demo may not contain all planned functionality,but rather those functions that can be delivered within the time-box
that was established.


every 24 hours
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
30 days
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

KANBAN APPROACH
Kanban is a workflow management method for defining, managing and improving services that deliver knowledge work. It aims to help you visualize your work, maximize efficiency, and improve continuously.

What is a Kanban Card?
In Kanban, work items are represented by cards. You can imagine these as sticky notes on the whiteboard.
KANBAN PRACTICES

1. Visualize work
By creating a visual model of your work and process, you can observe the flow of work moving through the Kanban system. Making the work visible, along with visual indications of blockers, bottlenecks, and queues, instantly leads to increased communication and collaboration. This helps teams see how fast their work is moving through the system and where they can focus their efforts .

2. Limit work-in-process

By limiting how much unfinished work is in process, you can reduce the time it takes an item to travel through the Kanban system. You can also avoid problems caused by task switching and reduce the need to constantly reprioritize items. WIP limits unlock the full potential of Kanban, enabling teams to deliver quality work faster than ever in a healthier, more sustainable environment.

3. Focus on flow

Using work-in-process limits and team-driven policies, you can optimize your Kanban system to:

- Improve the flow of work
- Collect metrics to analyze flow
- Get leading indicators of future problems

A consistent flow of work is essential for faster and more reliable delivery, bringing greater value to your customers, team, and organization.

4. Continuous improvement

Once your Kanban system is in place, it becomes the cornerstone for a culture of continuous improvement. Teams measure their effectiveness by tracking flow, quality, throughput, lead times, and more.

Experiments and analysis can change the system to improve the team's effectiveness. Continuous improvement is a Lean improvement technique that helps streamline workflows, saving time and money across the enterprise.