Module 4 Software Project Management

Projects need to be managed because professional software engineering is always subject to organizational budget and schedule constraints. The project manager's job is to ensure that the software project meets and overcomes these constraints as well as delivering high-quality software. Good management cannot guarantee project success.

The success criteria for project management obviously vary from project to project, but, for most projects, important goals are:

- To deliver the software to the customer at the agreed time;
- To keep overall costs within budget;
- To deliver software that meets the customer's expectations;
- To maintain a coherent and well-functioning development team.

Software Management Distinction

1. **The product is intangible:** Software is intangible. It cannot be seen or touched. Software project managers cannot see progress by looking at the artifact that is being constructed.
2. **Large software projects are often "one-off" projects**: Every large software development project is unique because every environment where software is developed is, in some ways, different from all others. Even managers who have a large body of previous experience may find it difficult to anticipate problems.
3. **Software processes are variable and organization-specific:** The engineering process for some types of system, such as bridges and buildings, is well understood. However, different companies use quite different software development processes. We cannot reliably predict when a particular software process is likely to lead to development problems.

Factors influencing the project management:

- **Company size:** Small companies can operate with informal management and team communications and do not need formal policies and management structures. They have less management overhead than larger organizations. In larger organizations, management hierarchies, formal reporting and budgeting, and approval processes must be followed.

- **Software customers:** If the customer is an internal then customer communications can be informal and there is no need to fit in with the customer's ways of working. If custom software is being developed for an external customer, agreement has to be reached on more formal communication channels. If the customer is a government agency, the software company must operate according to the agency's policies and procedures, which are likely to be bureaucratic.

- **Software size:** Small systems can be developed by a small team, which can get together in the same room to discuss progress and other management issues. Large systems usually need multiple development teams that may be geographically distributed and in different companies. The project manager has to coordinate the activities of these teams and arrange for them to communicate with each other.

- **Software type:** If the software being developed is a consumer product, formal records of project management decisions are unnecessary. On the other hand, if a safety-critical system is being developed, all project management decisions should be recorded and justified as these may affect the safety of the system.

- **Organizational culture:** Some organizations have a culture that is based on supporting and encouraging individuals, while others are group focused. Large organizations are often bureaucratic. Some organizations have a culture of taking risks, whereas others are risk averse.

- **Software development processes:** Agile processes typically try to operate with "lightweight" management. More formal processes require management monitoring to ensure that the development team is following the defined process.

Fundamental project management activities:

- **Project planning:** Project managers are responsible for planning, estimating, and scheduling project development and assigning people to tasks. They supervise the work to ensure that it is carried out to the required standards, and they monitor progress to check that the development is on time and within budget.
- **Risk management:** Project managers have to assess the risks that may affect a project, monitor these risks, and take action when problems arise.
- **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.
- **Reporting** Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software. They have to be able to communicate at a range of levels, from detailed technical information to management summaries. They have to write concise, coherent documents that abstract critical information from detailed project reports. They must be able to present this information during progress reviews.
- **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 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.).

- **Project risks** affect the project schedule or resources. An example of a project risk is the loss of an experienced system architect. Finding a replacement architect with appropriate skills and experience may take a long time; consequently, it will take longer to develop the software design than originally planned.
- **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. This may affect the overall performance of the system so that it is slower than expected.
- **Business risks** affect the organization developing or procuring the software. For example, a competitor introducing a new product is a business risk. The introduction of a competitive product may mean that the assumptions made about sales of existing software products may be unduly optimistic.

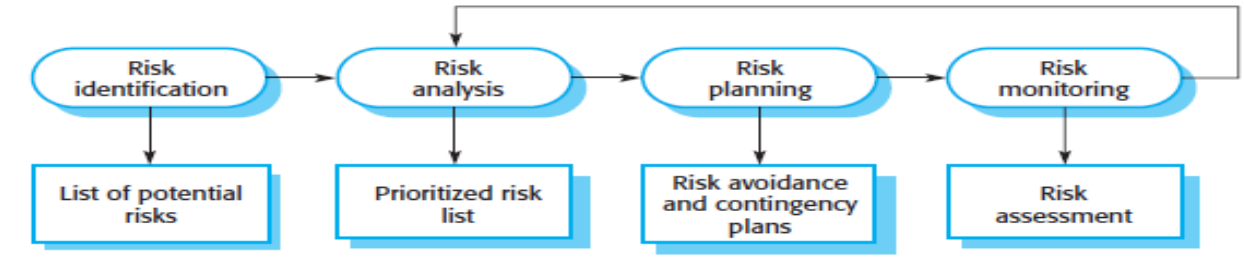
- Software risk management is important because of the inherent uncertainties in software development. These uncertainties stem from loosely defined requirements, requirements changes due

to changes in customer needs, difficulties in estimating the time and resources required for software development, and differences in individual skills.

- You have to anticipate risks, understand their impact on the project, the product, and the business, and take steps to avoid these risks. You may need to draw up contingency plans so that, if the risks do occur, you can take immediate recovery action.

Risk Management Process

- **Risk identification:** You should identify possible project, product, and business risks.
- **Risk analysis:** You should assess the likelihood and consequences of these risks.
- **Risk planning:** You should make plans to address the risk, either by avoiding it or by minimizing its effects on the project.
- **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. Once you have drawn up an initial risk management plan, you monitor the situation to detect emerging risks. As more information about the risks becomes available, you have to re-analyze the risks and decide if the risk priority has changed. You may then have to change your plans for risk avoidance and contingency management.

1. Risk Identification:

- 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. Alternatively, project managers may identify risks based on their experience of what went wrong on previous projects.
- Six types of risk may be included in a risk checklist:

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.

2. Risk Analysis

- During the risk analysis process, you have to consider each identified risk and make a judgment about the probability and seriousness of that risk.. You have to rely on your judgment and experience of previous projects and the problems that arose in them. It is not possible to make precise, numeric

assessment of the probability and seriousness of each risk. Rather, you should assign the risk to one of a number of bands:

- The probability of the risk might be assessed as insignificant, low, moderate, high, or very high.
- 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.

- Once the risks have been analyzed and ranked, you should assess which of these risks are most significant.
- Your judgment must depend on a combination of the probability of the risk arising and the effects of that risk.
- In general, catastrophic risks should always be considered, as should all serious risks that have more than a moderate probability of occurrence.

3. 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.
- In risk planning, you have to ask "what-if" questions that consider both individual risks, combinations of risks, and external factors that affect these risks.
- Based on the answers to these "what-if" questions, you may devise strategies for managing the risks. These strategies fall into three categories:
  - **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
  - **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
  - **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.

4. 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.
- You should also think about whether or not the effects of the risk have changed.
- To do this, you have to look at other factors, such as the number of requirements change requests, which give you clues about the risk probability and its effects.
- You should monitor risks regularly at all stages in a project. At every management review, you should consider and discuss each of the key risks separately.
- You should decide if the risk is more or less likely to arise and if the seriousness and consequences of the risk have changed.

Managing people

- The people working in a software organization are its greatest assets.
- There are four critical factors that influence the relationship between a manager and the people that he or she manages:
  - **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.
  - **Respect**: Different people have different skills, and managers should respect these differences. All members of the team should be given an opportunity to make a contribution.
  - **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.
  - **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. If you try to cover up ignorance or problems, you will eventually be found out and will lose the respect of the group.

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.
- If people are not motivated, they will be less interested in the work they are doing.
- They will work slowly, be more likely to make mistakes, and will not contribute to the broader goals of the team or the organization.

- To provide this encouragement, you should understand a little about what motivates people.
- Maslow suggests that people are motivated by satisfying their needs. These needs are arranged in a series of levels, as shown in Figure 22.7.
- 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.

  - **To satisfy social needs,** you need to give people time to meet their co-workers and provide places for them to meet. Social networking systems and teleconferencing can be used for remote communications; you should arrange some face-to-face meetings early in the project so that people can directly interact with other members of the team. Through this direct interaction, people become part of a social group and accept the goals and priorities of that group.

- **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. Obviously, people must also feel that they are paid at a level that reflects their skills and experience.
- 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.

Personality Types:

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, such as career progression, that motivate them, 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. As more and more attention is paid to user interface design, interaction- oriented individuals are becoming more involved in software engineering.

Teamwork

- Most professional software is developed by project teams that range in size from two to several hundred people.
- However, as it is impossible for everyone in a large group to work together on single problem, large teams are usually split into a number of smaller groups.
- Each group is responsible for developing part of the overall system.
- A good group is cohesive and thinks of itself as a strong, single unit.
- The people involved are motivated by the success of the group as well as by their own personal goals.
- In a cohesive group, members think of the group as more important than the individuals who are group members.
- Members of a well-led, cohesive group 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:
    - **The group can establish its own quality standards**: Because these standards are established by consensus, they are more likely to be observed than external standards imposed on the group.
    - **Individuals learn from and support each other**: Group members learn by working together. Inhibitions caused by ignorance are minimized as mutual learning is encouraged.
    - **Knowledge is shared:** Continuity can be maintained if a group member leaves. Others in the group can take over critical tasks and ensure that the project is not unduly disrupted.
    - **Refactoring and continual improvement is encouraged:** Group members work collectively to deliver high-quality results and fix problems, irrespective of the individuals who originally created the design or program.

- Given a stable organizational and project environment, the three factors that have the biggest effect on team working are:
    - **The people in the group:** You need a mix of people in a project group as software development involves diverse activities such as negotiating with clients, programming, testing, and documentation.

- **The way the group is organized:** A group should be organized so that individuals can contribute to the best of their abilities and tasks can be completed as expected.
- **Technical and managerial communications:** Good communication between group members, and between the software engineering team and other project stakeholders, is essential.

Selecting team members:

- 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.
- Sometimes people are hired from outside the organization; more often, software engineering groups are put together from current employees who have experience on other projects.
- Technical knowledge and ability should not be the only factor used to select group members.
- The "competing engineers" problem can be reduced if the people in the group have complementary motivations.
- 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
- It is sometimes impossible to choose a group with complementary personalities. If this is the case, the project manager has to control the group so that individual goals do not take precedence over organizational and group objectives

Group Organization

- The way a group is organized affects the group's decisions, the ways information is exchanged, and the interactions between the development group and external project stakeholders.

- Small programming groups are usually organized in an informal way. The group leader gets involved in the software development with the other group members. In an informal group, the group as a whole discusses the work to be carried out, and tasks are allocated according to ability and experience.

- Agile development teams are always informal groups. Agile enthusiasts claim that formal structure inhibits information exchange. Many decisions that are usually seen as management decisions (such as decisions on schedule) may be devolved to group members.

- In hierarchical groups the group leader is at the top of the hierarchy. He or she has more formal authority than the group members and so can direct their work. There is a clear organizational structure, and decisions are made toward the top of the hierarchy and implemented by people lower down

Group Communication

- It is absolutely essential that group members communicate effectively and efficiently with each other and with other project stakeholders. Group members must exchange information on the status of their work, the design decisions that have been made, and changes to previous design decisions. Good communication also helps strengthen group cohesiveness

- The effectiveness and efficiency of communications are influenced by:
  - **Group size:** As a group gets bigger, it gets harder for members to communicate effectively.
  - **Group structure:** People in informally structured groups communicate more effectively than people in groups with a formal, hierarchical structure. In hierarchical groups, communications tend to flow up and down the hierarchy. People at the same level may not talk to each other

- **Group composition:** People with the same personality types may clash, and, as a result, communications can be inhibited.
- **The physical work environment:** The organization of the workplace is a major factor in facilitating or inhibiting communications. While some companies use standard open-plan offices for their staff, others invest in providing a workspace that includes a mixture of private and group working areas.
- **The available communication channels:** There are many different forms of communication— face to face, email messages, formal documents, telephone, and technologies such as social networking and wikis. As project teams become increasingly distributed, with team members working remotely, you need to make use of interaction technologies, such as conferencing systems, to facilitate group communications.

- Effective communication is achieved when communications are two-way and the people involved can discuss issues and information and establish a common\ understanding of proposals and problems.

PROJECT PLANNING

- Project planning involves break down the work into parts and assign them to project team members, anticipate problems that might arise, and prepare tentative solutions to those problems. The project plan, which is created at the start of a project and updated as the project progresses, is used to show how the work will be done and to assess progress on the project.

Planning stages

- At the proposal stage, when you are bidding for a contract to develop or provide a software system. You need a plan at this stage to help you decide if you have the resources to complete the work and to work out the price that you should quote to a customer.

- During the project startup phase, when you have 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, and so on.

- Periodically throughout the project, when you update your plan to reflect new information about the software and its development. You learn more about the system being implemented and the capabilities of your development team

Proposal planning: Planning at the proposal stage is inevitably speculative, as you do not have a complete set of requirements for the software to be developed. A plan is often a required part of a proposal, so you have to produce a credible plan for carrying out the work.

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 .You need to think about organizational concerns, the risks associated with the project, and the type of contract that will be used

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.

Pricing to win: It means that a company has some idea of the price that the customer expects to pay and makes a bid for the contract based on the customer's expected price. This may seem unethical and unbusiness like, but it does have advantages for both the customer and the system provider

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.
- Plan-driven development is based on engineering project management techniques and can be thought of as the "traditional" way of managing large software development projects.
- Managers use the plan to support project decision making and as a way of measuring progress.
- 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.
- 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.

Project Plans

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

    - **Introduction:** Briefly describes the objectives of the project and sets out the constraints (e.g., budget, time) that affect the management of the project.
    - **Project organization:** Describes the way in which the development team is organized, the people involved, and their roles in the team.
    - **Risk analysis:** Describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed.
    - **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.
    - **Work breakdown:** Sets out the breakdown of the project into activities and identifies the inputs to and the outputs from each project activity.

- **Project schedule:** Shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities.
- **Monitoring and reporting mechanisms:** Defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

Planning Process

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

Assumptions:

- You should make realistic rather than optimistic assumptions when you are defining a project plan.
- Problems of some description always arise during a project, and these lead to project delays.
- Your initial assumptions and scheduling should therefore be pessimistic and take unexpected problems into account.
- You should include contingency in your plan so that if things go wrong, then your delivery schedule is not seriously disrupted.

Risk Mitigation:

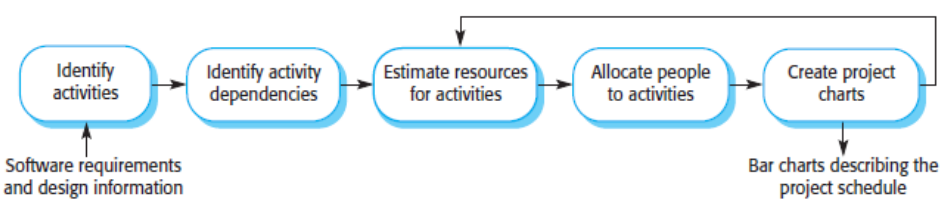
- If there are serious problems with the development work that are likely to lead to significant delays, you need to initiate risk mitigation actions to reduce the risks of project failure.
- In conjunction with these actions, you also have to re-plan the project.
- This may involve renegotiating the project constraints and deliverables with the customer. A new schedule of when work should be completed also has to be established and agreed to with the customer.

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.

Project Scheduling Activities

- 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.



Schedule Presentation

- Project schedules may simply be documented in a table or spreadsheet showing the tasks, estimated effort, duration, and task dependencies.
- Two types of visualization are commonly used:
    - 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. Bar charts are also called Gantt charts, after their inventor, Henry Gantt
    - 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:
    1. A duration in calendar days or months;
    2. An effort estimate, which shows the number of person-days or person-months to complete the work;
    3. A deadline by which the activity should be complete; and
    4. A defined endpoint, which might be a document, the holding of a review meeting, the successful execution of all tests, or the like

Milestones and deliverables:

- Milestone is a logical end to a stage of the project where the progress of the work can be reviewed. Each milestone should be documented by a brief report (often simply an email) that summarizes the work done and whether or not the work has been completed as planned. Milestones may be associated with a single task or with groups of related activities.

- Some activities create project deliverables—outputs that are delivered to the software customer. Usually, the deliverables that are required are specified in the project contract, and the customer's view of the project's progress depends on these deliverables. Milestones and deliverables are not the same thing. Milestones are short reports that are used for progress reporting, whereas deliverables are more substantial project outputs such as a requirements document or the initial implementation of a system.

Agile Planning

- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development.

Stages:

1. Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system. 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.
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.

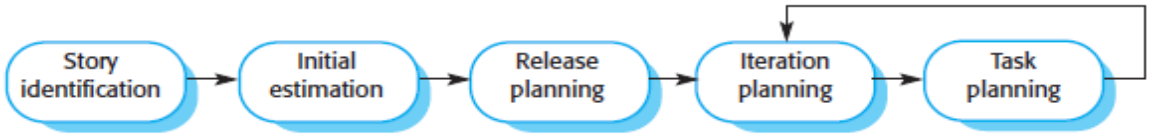
Approaches:

- Planning in Scrum

- Planning game: developed as part of Extreme Programming, is based on user stories. The so-called planning game can be used in both release planning and iteration planning.

Story Based Planning:

- The basis of the planning game is a set of user stories that cover all of the functionality to be included in the final system.
- The team members read and discuss the stories and rank them based on the amount of time they think it will take to implement the story.



Release Planning and Iteration Planning

- 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

- 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.

Task Allocation

- During 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 tasks that they will implement.
- Benefits include:
  - The whole team gets an overview of the tasks to be completed in iteration. They therefore have an understanding of what other team members are doing and who to talk to if task dependencies are identified.
  - 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.

Agile Planning Difficulties

- 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 not available 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.

Agile Planning Applicability: Agile planning works well with small, stable development teams that can get together and discuss the stories to be implemented.

- However, where teams are large and/or geographically distributed, or when team membership changes frequently, it is practically impossible for everyone to be involved in the collaborative planning that is essential for agile project management.

Estimation Techniques

Two types of techniques can be used for making estimates:

1. **Experience-based techniques:** The estimate of future effort requirements is based on the manager's experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.

2. **Algorithmic cost modeling:** In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, process characteristics, and experience of staff involved.

Experience based techniques:

- Experience-based techniques rely on the manager's experience of past projects and the actual effort expended in these projects on activities that are related to software development.
- Typically, you identify the deliverables to be produced in a project and the different software components or systems that are to be developed.
- You document these in a spreadsheet, estimate them individually, and compute the total effort required.
- It usually helps to get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate.
- The difficulty with experience-based techniques is that a new software project may not have much in common with previous projects. Software development changes very quickly, and a project will often use unfamiliar techniques such as web services, application system configuration, or HTML5. If you have not worked with these techniques, your previous experience may not help you to estimate the effort required, making it more difficult to produce accurate costs and schedule estimates

Algorithmic cost Modeling:

- Algorithmic cost modeling uses a mathematical formula to predict project costs based on estimates of the project size, the type of software being developed, and other team, process, and product factors.
- Most algorithmic models for estimating effort in a software project are based on a simple formula:
  Effort = $A * Size^B * M$

  A: a constant factor, which depends on local organizational practices and the type of software that is developed.

  Size: an assessment of the code size of the software or a functionality estimate expressed in function or application points.

  B: represents the complexity of the software and usually lies between 1 and 1.5.

  M: is a factor that takes into account process, product and development attributes, such as the dependability requirements for the software and the experience of the development team.

- The idea of using a scientific and objective approach to cost estimation is an attractive one, but all algorithmic cost models suffer from two key problems:
  1. It is practically impossible to estimate Size accurately at an early stage in a project, when only the specification is available. Function-point and application point estimates (see later) are easier to produce than estimates of code size but are also usually inaccurate.
  2. The estimates of the complexity and process factors contributing to B and M are subjective. Estimates vary from one person to another, depending on their background and experience of the type of system that is being developed.

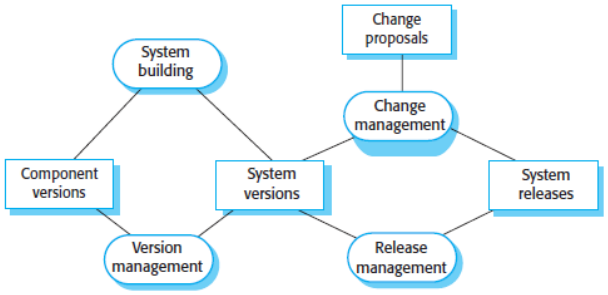
Effectiveness of this Model:

- Algorithmic cost models are a systematic way to estimate the effort required to develop a system. However, these models are complex and difficult to use.
- There are many attributes and considerable scope for uncertainty in estimating their values.
- This complexity means that the practical application of algorithmic cost modeling has been limited to a relatively small number of large companies, mostly working in defense and aerospace systems engineering.
- Another barrier that discourages the use of algorithmic models is the need for calibration. Model users should calibrate their model and the attribute values using their own historical project data, as this reflects local practice and experience.

Configuration Management

- Configuration management (CM) is concerned with the policies, processes, and tools for managing changing software systems.
- The configuration management of a software system product involves four closely related activities:

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.

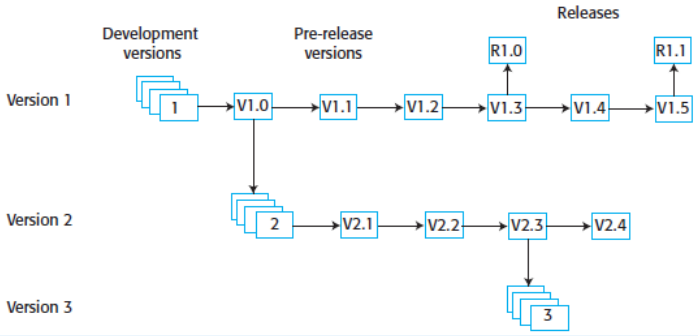


The development of a software product or custom software system takes place in three distinct phases:

1. *A* **development phase** where the development team is responsible for managing the software configuration and new functionality is being added to the software. The development team decides on the changes to be made to the system.
2. *A* **system testing phase** where a version of the system is released internally for testing. This may be the responsibility of a quality management team or an individual or group within the development team. At this stage, no new functionality is added to the system. The changes made at this stage are bug fixes, performance improvements, and security vulnerability repairs. There may be some customer involvement as beta testers during this phase.
3. *A* **release phase** where the software is released to customers for use. After the release has been distributed, customers may submit bug reports and change requests. New versions of the released system may be developed to repair bugs and vulnerabilities and to include new features suggested by customers.

Multi – version system

For large systems, there is never just one "working" version of a system; there are always several versions of the system at different stages of development. Several teams may be involved in the development of different system versions.



Term	Explanation
Baseline	A collection of component versions that make up a system. Baselines are controlled, which means that the component versions used in the baseline cannot be changed. It is always possible to re-create a baseline from its constituent components.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Codeline	A set of versions of a software component and other configuration items on which that component depends.
Configuration (version) control	The process of ensuring that versions of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system.
Configuration item or software configuration item (SCI)	Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. Configuration items always have a unique identifier.
Mainline	A sequence of baselines representing different versions of a system.
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Repository	A shared database of versions of software components and meta-information about changes to these components.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions should always have a unique identifier.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.

1. Version Management

- Version management is the process of keeping track of different versions of software components and the systems in which these components are used. It also involves ensuring that changes made by different developers to these versions do not interfere with each other. In other words, version management is the process of managing code lines and baselines.

Code lines and base lines

- A codeline is a sequence of versions of source code, with later versions in the sequence derived from earlier versions. Code lines normally 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 Systems

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.

Key features of Version Control Systems

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.

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. In some systems, these changes may be used to select a particular system version. This involves tagging components with keywords describing the changes made.

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.

Centralized version control

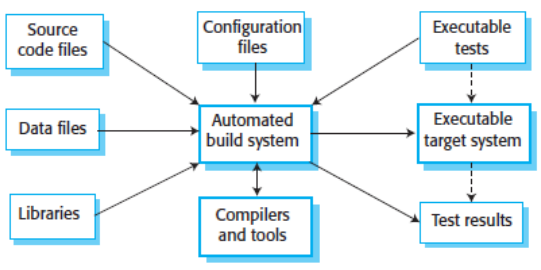
- In centralized systems, developers check out components or directories of components from the project repository into their private workspace and work on these copies in their private workspace. When their changes are complete, they check-in the components back to the repository. This creates a new component version that may then be shared.

Distributed version control

- 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.

2. **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 involves assembling a large amount of information about the software and its operating environment.



Build System functionality

- **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).
- **Version control system integration**: The build system should check out the required versions of components from the version control system.
- **Minimal recompilation**: The build system should work out what source code needs to be recompiled and set up compilations if required.
- **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.
- **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.
- **Reporting:** The build system should provide reports about the success or failure of the build and the tests that have been run.
- **Documentation generation**: The build system may be able to generate release notes about the build and system help pages.

System Platforms:

- *The development system*, which includes development tools such as compilers and source code editors.

- *The build server*: This is used to build definitive, executable versions of the system. This server maintains the definitive versions of a system.

- *The target environment*, which is the platform on which the system executes. This may be the same type of computer that is used for the development and build systems.

Continuous Integration

Steps in Continuous Integration:

- Extract the mainline system from the VC system into the developer's private workspace.
- Build the system and run automated tests to ensure that the built system passes all tests
- Make the changes to the system components.
- Build the system in a private workspace and rerun system tests. If the tests fail, continue editing
- 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.
- Build the system on the build server and run the tests
- 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.

- The advantage of continuous integration is that it allows problems caused by the interactions between different developers to be discovered and repaired as soon as possible.

- If the system is very large, it may take a long time to build and test, especially if integration with other application systems is involved.
- If the development platform is different from the target platform, it may not be possible to run system tests in the developer's private workspace.

Change Management:

- 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.

- 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

Factors in change analysis:

- **The consequences of not making the change:** When assessing a change request, you have to consider what will happen if the change is not implemented. If the change is associated with a reported system failure, the seriousness of that failure has to be taken into account

- **The benefits of the change:** Will the change benefit many users of the system, or will it only benefit the change proposer?

- **The number of users affected by the change**: If only a few users are affected, then the change may be assigned a low priority.

- **The costs of making the change:** If making the change affects many system components and/or takes a lot of time to implement, then the change may be rejected

- **The product release cycle:** If a new version of the software has just been released to customers, it may make sense to delay implementation of the change until the next planned release.

Release Management:

- A system release is a version of a software system that is distributed to customers. For mass-market software, it is usually possible to identify two types of release: major releases, which deliver significant new functionality, and minor releases, which repair bugs and fix customer problems that have been reported.

Release components:

A software product release is not just the executable code of the system. The release may also include:

■ configuration files defining how the release should be configured for particular installations;

■ data files, such as files of error messages in different languages, that are needed successful system operation;

■ an installation program that is used to help install the system on target hardware;

■ electronic and paper documentation describing the system;

■ packaging and associated publicity that have been designed for that release.

Factors affecting system release planning:

Factor	Description
Competition	For mass-market software, a new system release may be necessary because a competing product has introduced new features and market share may be lost if these are not provided to existing customers.
Marketing requirements	The marketing department of an organization may have made a commitment for releases to be available at a particular date. For marketing reasons, it may be necessary to include new features in a system so that users can be persuaded to upgrade from a previous release.
Platform changes	You may have to create a new release of a software application when a new version of the operating system platform is released.
Technical quality of the system	If serious system faults are reported that affect the way in which many customers use the system, it may be necessary to correct them in a new system release. Minor system faults may be repaired by issuing patches, distributed over the Internet, which can be applied to the current release of the system.

- 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.

SCRUM FRAMEWORK

- Scrum is an agile software development.
- Scrum principles are consistent with agile manifesto and are used to guide development activities within a process that incorporates the following framework actives: requirements, analysis, design, evolution and delivery.
- Within each process activities task work within a process pattern called **sprint.**
- The work conducted within a sprint is adapted to the problem at hand and is defined and often modified by the Scrum team.
- Scrums emphasize the use of set of software patterns that have been proven effective for projects with tight timelines, changing requirements and business criticality.
- Each of these process patterns define a set of development activities:
  - **i. Backlog:** a prioritized list of project requirements or features that provide business value for the customer. The project manager access backlogs and updates are made.
  - **ii. Sprints:** consists of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time box.
  - **iii. Scrum meetings:** are usually short meetings held daily by the scrum team. Three questions are asked and answered by team members: what did you do since last meeting? what obstacles are you encountering? What do you plan to accomplish by next meeting? A team leader called Scrum Master leads the meeting and assess each responses. This meeting helps to uncover potential problems as early as possible.
  - **iv. Demos:** deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

KANBAN METHODOLOGY AND LEAN APPROACHES

Seven principles of Lean Methodology

1. **Eliminate waste:** Lean philosophy regards everything not adding value to the customer as waste. Such waste may include:[5]Partially done work, Extra features, Relearning, Task switching, Waiting, Handoffs, Defects, Management activities. In order to eliminate waste, one should be able to recognize it. If some activity could be bypassed or the result could be achieved without it, it is waste.
2. **Amplify learning**: Software development is a continuous learning process based on iterations when writing code. Software design is a problem-solving process involving the developers writing the code and what they have learned. Software value is measured in fitness for use and not in conformance to requirements. Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building.
3. **Decide as late as possible:** As software development is always associated with some uncertainty, better results should be achieved with a *set-based* or *options-based* approach, delaying decisions as

much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle – the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

4. **Deliver as fast as possible**: In the era of rapid technology evolution, it is not the biggest that survives, but the fastest. The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team. With speed, decisions can be delayed**.**
5. **Empower the team:** The lean approach follows the agile principle[6] "build projects around motivated individuals [...] and trust them to get the job done",[7] encouraging progress, catching errors, and removing impediments, but not micro-managing.
6. **Build integrity in**: The customer needs to have an overall experience of the system. This is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, its price and how well it solves problems.
7. **Optimize the whole:**