SOFTWARE MANAGEMENT

Prepared By: Natabar Khatri

New Summit College

Unit 10

SOFTWARE PROJECT MANAGEMENT

- Software project management is a fundamental discipline within software engineering, essential for navigating the organizational budget and schedule constraints inherent in professional software development. The project manager's primary role is to ensure that the project not only adheres to these constraints but also delivers high-quality software that meets customer expectations.
- While good management does not guarantee success, poor management often leads to project failure - manifested as delays, cost overruns, or unsatisfactory outcomes.
- Key success criteria for most projects include:
  - Delivering software on time,
  - Staying within budget,
  - Meeting customer expectations,
  - Maintaining a cohesive and effective development team.

CHALLENGES IN SOFTWARE PROJECT MANAGEMENT

Software engineering presents distinct challenges that complicate project management:

- **Intangibility of the Product:** Unlike physical engineering projects, software progress cannot be directly observed. Managers must rely on reports and documentation to gauge progress.
- **"One-off" Nature of Projects:** Large software projects are often unique, making it difficult to apply lessons from past projects directly. Rapid technological change further complicates this.
- **Variable and Organization-Specific Processes:** There is no single, universally effective software development process. The variability between organizations and projects makes it hard to predict issues, especially in innovative or complex systems.

Factors Influencing Project Management

The role of a project manager varies significantly based on several factors:

- **Company Size**
- **Software Customers**
- **Software Size**
- **Software Type** (e.g., consumer product vs. safety-critical system)
- **Organizational Culture**
- **Software Development Processes** (e.g., Agile vs. formal processes)

Project Management Activities

Despite these variations, all project managers typically engage in five core activities:

- **Project Planning**
- **Risk Management**
- **People Management**
- **Reporting**
- **Proposal Writing**

RISK MANAGEMENT

Risk management is a critical function, involving the anticipation of potential threats to the project schedule, software quality, or the organization, and taking proactive steps to avoid or mitigate them.

Risk Categories

Risks can be categorized based on what they affect:

- **Project Risks:** Threats to the project schedule or resources (e.g., loss of a key architect).
- **Product Risks:** Threats to the quality or performance of the software (e.g., an underperforming purchased component).
- **Business Risks:** Threats to the organization developing or procuring the software (e.g., a competitor launching a new product).

These categories often overlap. For example, the departure of an experienced engineer is simultaneously a project, product, and business risk.

RISK MANAGEMENT PROCESS

- The process in risk management is outlined in **Figure 10.1**
- The Risk management process is iterative and consists of four key stages:
  - **Risk Identification**
  - **Risk Analysis**
  - **Risk Planning**
  - **Risk Monitoring**

For large projects, the outcomes are documented in a **Risk Management Plan**.

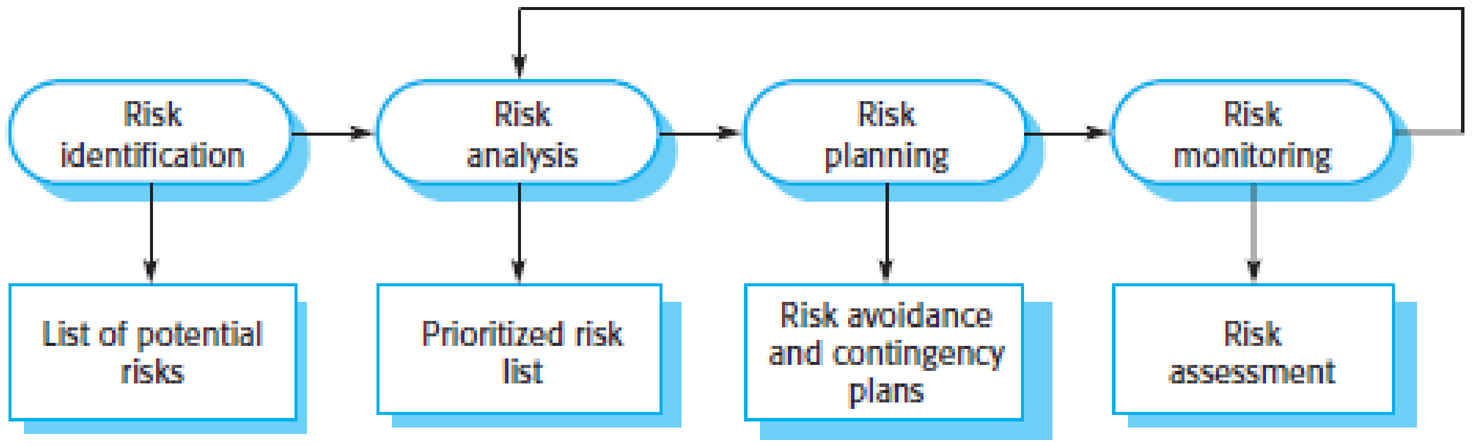


Figure 10.1: The risk management process

RISK MANAGEMENT PROCESS

Risk Identification

This initial stage involves brainstorming and identifying potential risks. A common approach is to use a checklist of risk types:

- **Estimation Risks** (e.g., underestimating time or size)
- **Organizational Risks** (e.g., restructuring or budget cuts)
- **People Risks** (e.g., recruitment problems, staff illness)
- **Requirements Risks** (e.g., numerous changes or customer misunderstandings)
- **Technology Risks** (e.g., database performance issues)
- **Tools Risks** (e.g., inefficient or incompatible software tools)

The goal of risk identification is to create a comprehensive list, which is then pruned to a manageable number of key risks.

RISK MANAGEMENT PROCESS

Risk Analysis

- In this stage, each identified risk is assessed for its **probability** (e.g., Low, Moderate, High) and **effects** (e.g., Insignificant, Serious, Catastrophic).
- Risks are then tabulated and ranked based on this assessment.
- The most significant risks - typically those with high probability and serious or catastrophic consequences are selected for active management.

Risk	**Probability**	**Effects**
It is impossible to recruit staff with the skills required.	High	Catastrophic
Organizational financial problems force reductions in the project budget.	Low	Catastrophic
The time required to develop the software is underestimated.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable

RISK MANAGEMENT PROCESS

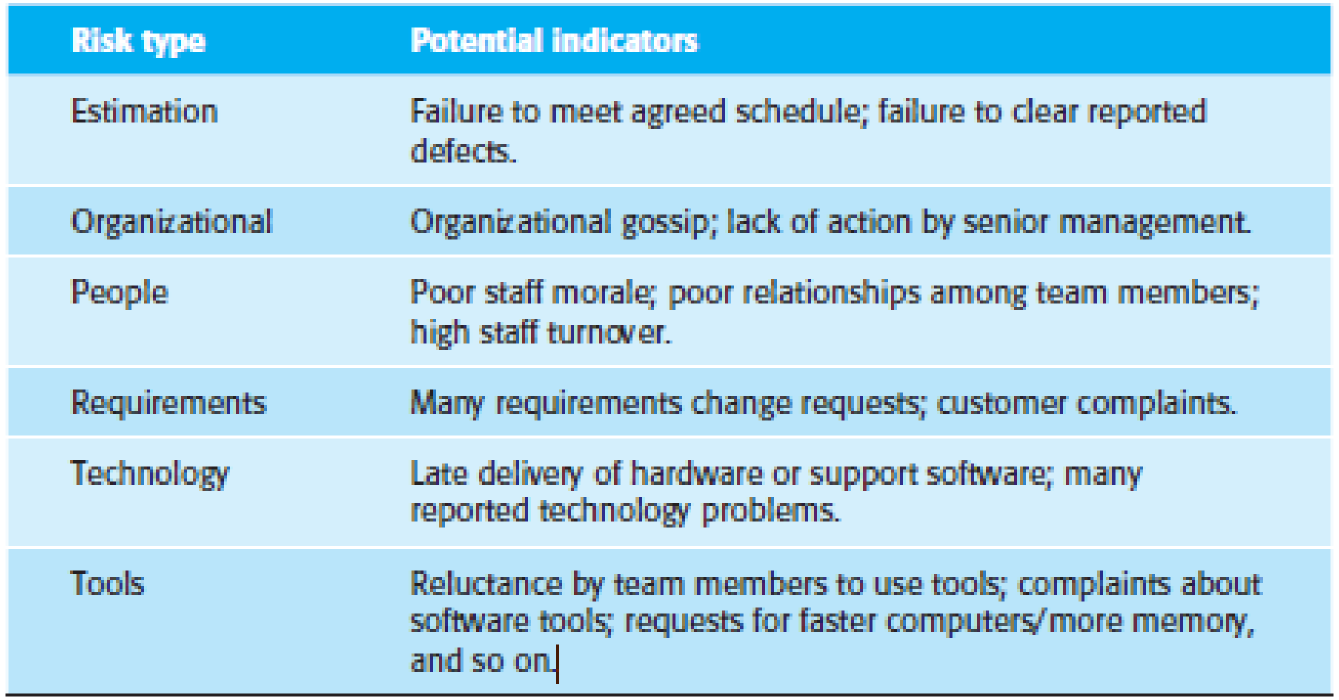
Risk Planning

- This stage involves developing strategies to manage the key risks. Strategies fall into three categories, analogous to critical systems engineering:
- **Avoidance Strategies:** Reduce the probability of the risk occurring (e.g., replacing potentially defective components).
- **Minimization Strategies:** Reduce the impact of the risk if it occurs (e.g., reorganizing teams to handle staff illness).
- **Contingency Plans:** Prepare a plan to deal with the risk if it materializes (e.g., preparing a briefing document to counter potential budget cuts).

RISK MANAGEMENT PROCESS

Risk Monitoring

- Risk monitoring is an ongoing process of checking whether the assumptions about risks have changed. Project managers should regularly review the key risks, assessing if their probability or impact has shifted.
- **Figure 10.2** lists potential **risk indicators** for different risk types (e.g., "Many requirements change requests" for Requirements risks, "Poor staff morale" for People risks). These indicators help in proactively detecting emerging problems.



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.

Fig 10.2: Risk Indicators

PEOPLE MANAGEMENT

In a software organization, people are the most valuable asset. The primary goal of a project manager is to ensure engineers are as productive as possible. This is achieved by respecting individuals and assigning responsibilities that match their skills and experience.

Effective people management is built on four critical pillars:

- **Consistency:** All team members should be treated in a comparable and fair manner.
- **Respect:** A manager must respect the different skills and backgrounds of each team member.
- **Inclusion:** Create an environment where everyone feels heard and their proposals are considered.
- **Honesty:** Be transparent about both successes and failures, as well as your own knowledge limitations.

PEOPLE MANAGEMENT

Motivating People

Motivation is about organizing work and the work environment to encourage people to contribute to the best of their abilities. A lack of motivation leads to slower work, more mistakes, and a lack of contribution to team goals.

- A classic model for understanding motivation is **Maslow's hierarchy of needs**. For software engineers, whose basic physiological and security needs are usually met, the higher-level needs are most relevant:

- **Social Needs:** Satisfied by providing time and space for social interaction. For co-located teams, this means social spaces; for distributed teams, it requires tools like social networking and teleconferencing, ideally supplemented by initial face-to-face meetings to build rapport.
- **Esteem Needs:** Satisfied by showing individuals they are valued through public recognition and fair compensation.
- **Self-Realization Needs:** Satisfied by giving people responsibility, challenging tasks, and opportunities for training and skill development.

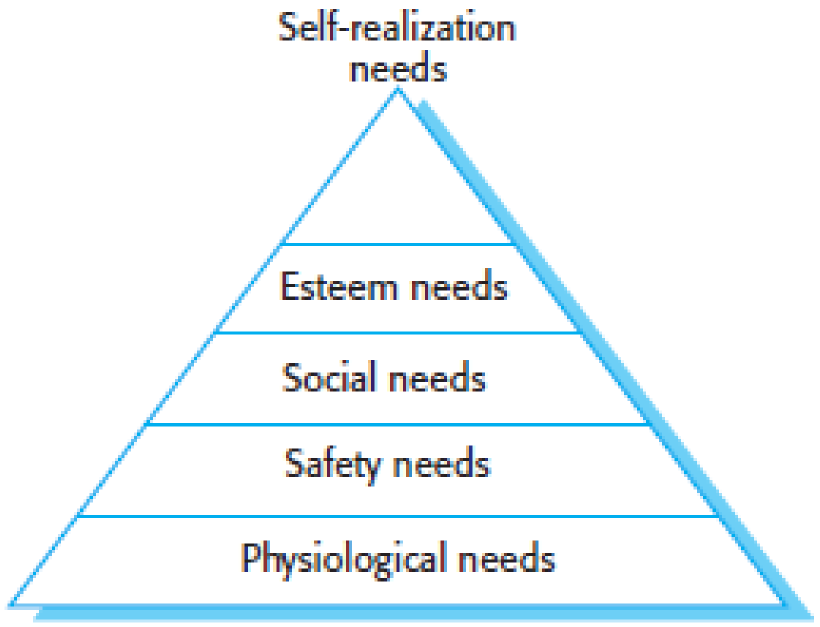


Figure 10.3: Maslow's hierarchy of needs

TEAMWORK

- Most software is developed by teams, with the ideal size being 4 to 6 members to minimize communication overhead. A manager's task is to form a **cohesive group** where members are loyal to the group and its goals, viewing the group's success as more important than individual success.

- **Benefits of a cohesive team include:**
  - The group sets and adheres to its own high-quality standards.
  - Team members learn from and support each other.
  - Knowledge is shared, ensuring continuity if someone leaves.
  - The team collectively takes responsibility for refactoring and improvement.

PEOPLE MANAGEMENT

Three major factors affect team effectiveness:

1. **The People in the Group**

   Managers must select a team with a balance of technical skills and personalities. While technical skill is crucial, it should not be the only factor.

2. **The Group Organization**

   How a team is organized impacts decision-making and communication. Key questions for a manager include defining the technical leader, how decisions are made, and how to handle external communications.

3. **Technical and Managerial Communications**

   Effective communication is the lifeblood of a project. It is essential for exchanging information, resolving problems, and strengthening group cohesion.

REPORTING

- Reporting is a fundamental responsibility of a project manager, acting as the critical communication bridge between the project team, company management, and the customer. Effective reporting ensures transparency, manages stakeholder expectations, and supports informed decision-making.
- The core aspects of reporting involve:
  - **Diverse Audiences:** Project managers must tailor their communication to different stakeholders. They need to provide:
    - **Detailed technical information** for the development team or technical leads.
    - **High-level management summaries** for company executives and customers, focusing on progress against goals, budget, and schedule.
  - **Concise and Coherent Documentation:** The ability to distill complex, detailed project data into clear, concise, and actionable documents is crucial. Reports must abstract the most critical information—such as milestones reached, risks identified, and resource status—without overwhelming the reader.
  - **Effective Presentation:** Reporting is not just written; it also involves verbal communication. Project managers must be prepared to present their findings and summaries confidently during formal **progress reviews**, answering questions and providing clarification to stakeholders.

PROPOSAL WRITING

- *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.
- Proposal writing is a critical task as the survival of many software companies depends on having enough proposals accepted and contracts awarded.

PROJECT PLANNING

- Project planning is a fundamental management activity that involves breaking down work, assigning tasks, anticipating problems, and preparing solutions. The resulting **project plan** is a dynamic document, created at the project's start and updated throughout, used to guide execution and assess progress.
- Planning occurs at three key stages:
  - **Proposal Stage:** A speculative plan is created for bidding, to determine resource availability and calculate a customer quote.
  - **Project Startup Phase:** The initial plan is refined with more detailed information on team structure, work breakdown, and resource allocation.
  - **Throughout the Project:** The plan is continuously updated to reflect new information, requirement changes, and a better understanding of the team's capabilities.

COST ESTIMATION

- A fundamental part of planning, especially at the proposal stage, is working out the project's cost. This involves estimating the effort required to complete each activity and calculating the total cost. It is crucial to calculate costs objectively to predict expenses accurately.
- The primary parameters for computing software development costs are:
  - Effort Costs (Salaries)
  - Hardware and Software Costs
  - Travel and Training Costs
- The two distinct metrics used to measure the **size of software** for project estimation and productivity comparison are **Lines of Code (LOC)** and **Function Points (FP).**
- Functional points are derived using an empirical relationship based on countable (direct) measures of software information domain and assessments of software complexity.
- Size of software product is directly dependent on the number of different functions or features it supports.
- Advantage of functional point is that it can be used to easily estimate the size of software product directly from problem specification
- This is in contrast to LOC metric, where the size can be accurately determined only after the product has fully been developed.

SOFTWARE PRICING

- The price quoted to a customer is not simply the cost of development plus profit. It is influenced by broader business and strategic factors.
- These factors include:
  - **Contractual Terms:** Ownership of source code can affect the price.
  - **Cost Estimate Uncertainty:** A contingency may be added if estimates are unsure.
  - **Financial Health:** A company in trouble may lower its price to secure cash flow.
  - **Market Opportunity:** A low price might be quoted to enter a new market segment.
  - **Requirements Volatility:** A low initial price can be offered with the expectation of charging more for subsequent changes.
- A common strategy is **"pricing to win,"** where a company bases its bid on the customer's expected budget rather than its own precise cost calculation. This can help secure a contract and maintain a skilled workforce for future, more profitable projects.

PLAN-DRIVEN DEVELOPMENT

- In plan-driven development, the entire process is planned in detail beforehand. This traditional approach is beneficial for managing organizational resources and identifying dependencies and risks early. It is often essential for large, safety-critical systems or projects involving multiple companies.
- A **project plan** in this context typically includes:
    1. Introduction
    2. Project Organization
    3. Risk Analysis
    4. Hardware and Software Resource Requirements
    5. Work Breakdown
    6. Project Schedule
    7. Monitoring and Reporting Mechanisms
- Supplementary plans for aspects like quality, validation, and deployment (Table 10.1) are also common for large projects.

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.

Table 10.1: Project plan supplements

THE PLANNING PROCESS

Project planning is an iterative process, as depicted in Figure 10.4. It involves:

1. Identifying constraints, risks, milestones, and deliverables.
2. Defining a project schedule.
3. Doing the work and monitoring progress.
4. Replanning as necessary to handle slippage or serious problems.
5. The process should be based on **pessimistic assumptions** and include **contingency** to account for unforeseen issues.

THE PLANNING PROCESS

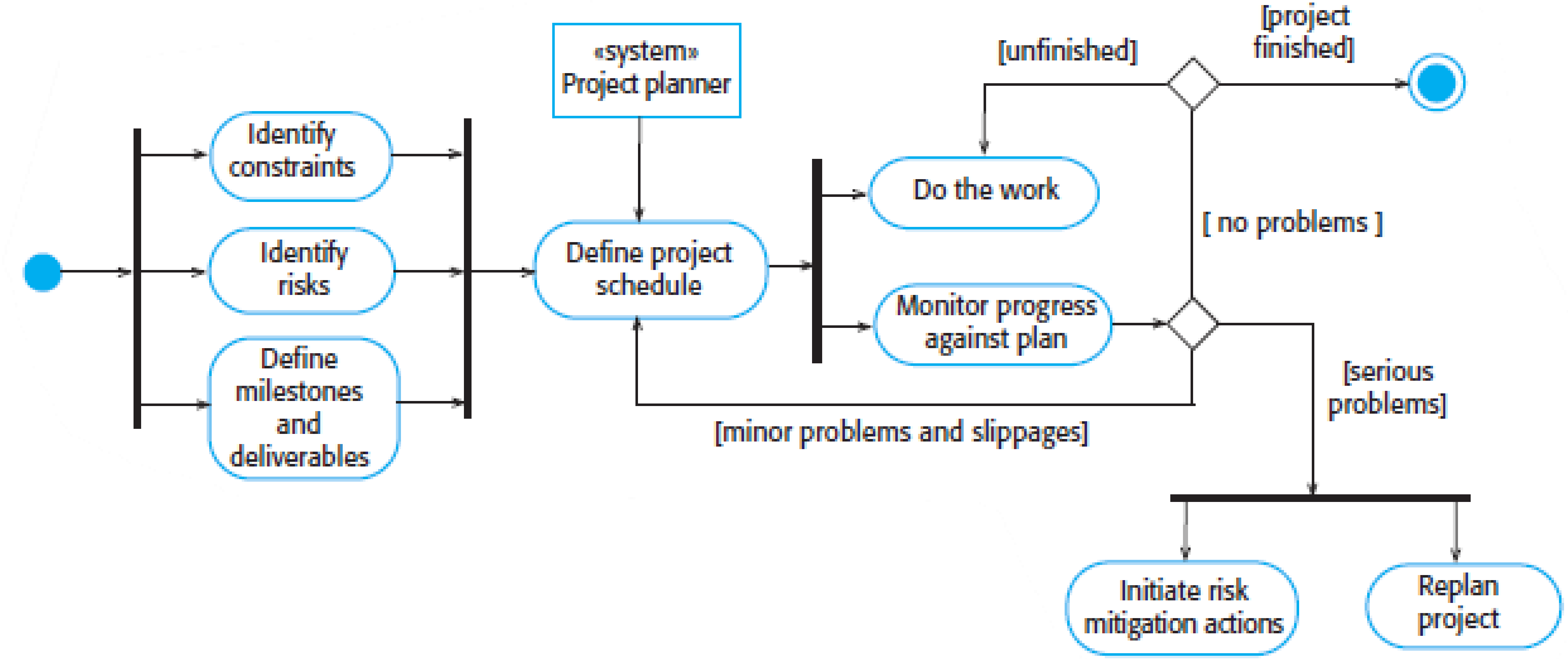


Figure 10.4: The project planning process

PROJECT SCHEDULING

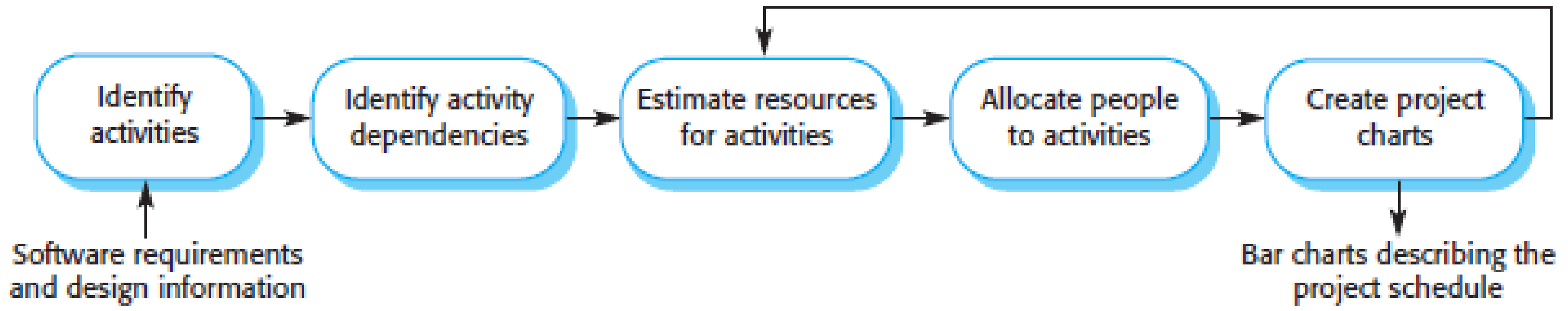
Project scheduling involves deciding how work is organized into tasks, and estimating the time, effort, and resources for each. In plan-driven projects, the process follows the steps in Figure.

- **Tasks** should last between one week and two months.
- Schedules must account for tasks being carried out in parallel and their dependencies.
- A good rule of thumb is to estimate as if nothing will go wrong and then add a **contingency** (e.g., 30-50%) for anticipated and unanticipated problems.

Schedule Presentation

Schedules are often presented visually for clarity. Two common graphical visualizations are:

- **Bar Charts (Gantt Charts):** Show tasks against a calendar, their start/end dates, and milestones. They also help in resource allocation, showing who is working on what and when.
- **Activity Networks:** Show the dependencies between tasks.


Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software requirements and design information
Bar charts describing the project schedule

ESTIMATION TECHNIQUES

- Estimating project effort is challenging due to early-stage uncertainties. The accuracy of estimates improves as the project progresses. There are two primary techniques:
  1. **Experience-based Techniques:** Managers use their judgment and experience from past projects to estimate effort.
  2. **Algorithmic Cost Modeling:** Uses a mathematical formula to predict effort based on estimates of project size, complexity, and other factors.
- **Algorithmic Cost Modeling**

  The general formula is: $\textbf{Effort} = \textbf{A} \times \textbf{Size}^{\textbf{B}} \times \textbf{M}$

  A: A constant factor dependent on the organization.

  Size: The size of the software (e.g., in Lines of Code or Function Points).

  B: An exponent reflecting the project's complexity.

  M: A multiplier based on product, project, and team attributes.
- These models are systematic but complex. Their accuracy is limited because:
  - It is hard to estimate **Size** accurately early on.
  - The assessments of factors for **B** and **M** are subjective.
  - They require **calibration** with historical project data, which many organizations lack.

COCOMO COST MODELING

- COCOMO (Constructive Cost Model) is a well-known, freely available algorithmic cost model developed by **Barry W. Boehm** in **1981**. It consists of a hierarchy of sub models for different stages of a project (**Figure 10.6**):
  - **Application Composition Model:** For projects built from reusable components, scripts, or database programming. Effort is based on **Application Points** (counting screens, reports, etc.).
  - **Early Design Model:** Used after requirements are established. It uses a simplified formula with seven multipliers. Size is often estimated in **Function Points** and converted to lines of code.
  - **Reuse Model:** Estimates the effort required to integrate and adapt reusable components.
  - **Post-Architecture Model:** The most detailed model, used after the system architecture is designed. It uses a full formula with **17 cost drivers** (e.g., personnel capability, reliability, schedule pressure) to refine the estimate.

COCOMO COST MODELING

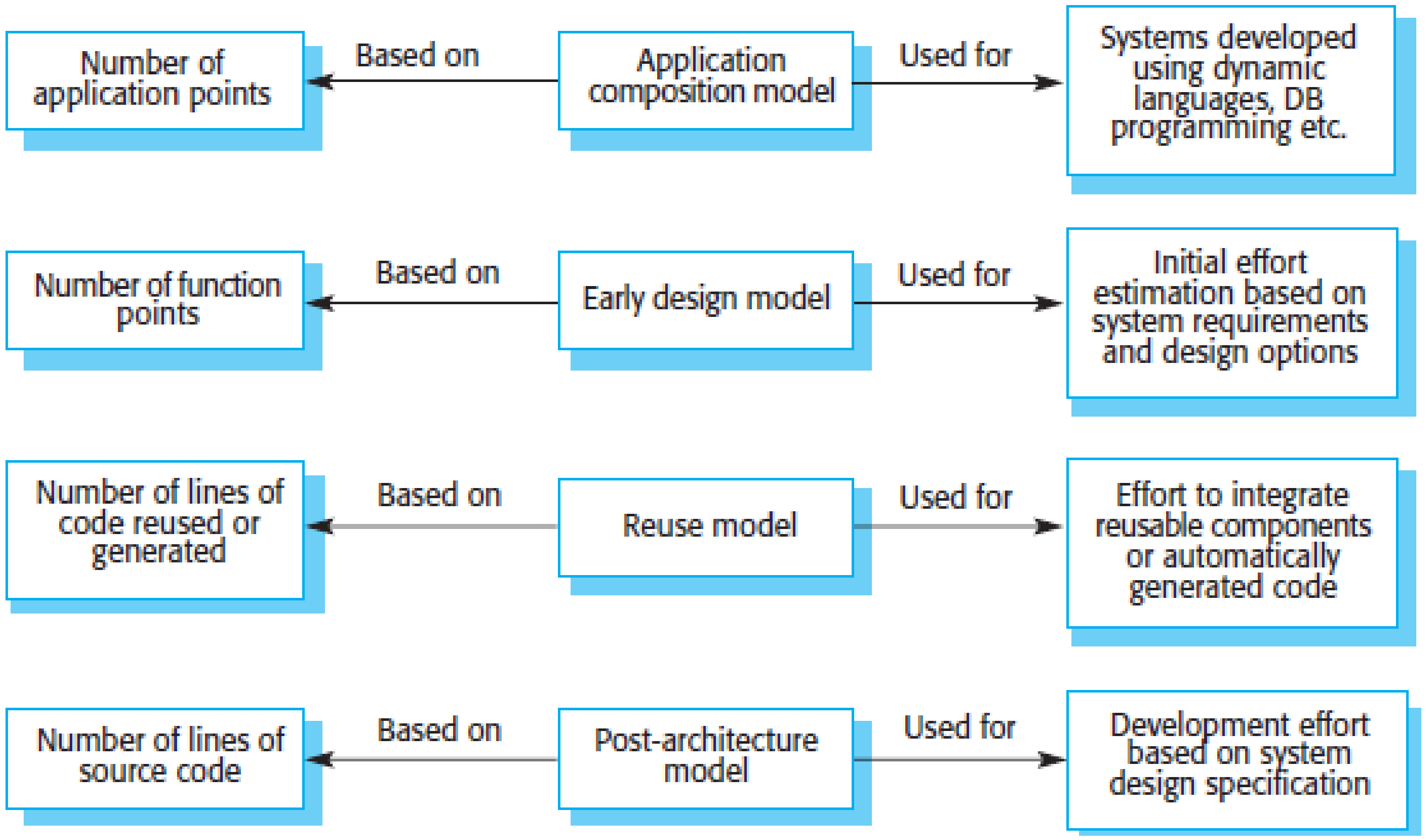

Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

Figure 10.6: COCOMO estimation models

COCOMO COST MODELING

- COCOMO has three different models that reflect the complexity. Any of these three form can be adopted according to requirement
  - Basic Model
  - Intermediate Model
  - Detailed Model

Basic COCOMO

- Can be used for quick and slightly rough calculations of software costs.
- Accuracy is somewhat restricted due to absence of sufficient factor considerations.
  - $\varepsilon = a(KLOC)^{b}$ [Person Month]
  - Time = c(*Effort)d months
  - Person required = $\frac{Effort}{Time}$ r
- The constant values a, b, c and for Basic model are different according to organic, semi-detached and embedded.
- Effort is Total effort required to develop the software product/expressed in person months (PMs)
  - Cost = time × Rate [Cost = time + rate]

COCOMO COST MODELING

Intermediate COCOMO

- The basic COCOMO model assumes that efforts is only a function of number of liens of code and some constant evaluated to different software system. Basic COCOMO assume effort and development time are function of product size also. However, cost of other project parameter beside product size also should be considered
- Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken in account.
- Various other factor such as reliability, experience, capability, size of database, complexity of product, hardware attributes, runtime performance, constrains, memory constrains, programming language experience, use of software tools etc utilizes is such derivers for cost estimation.
- The project manager is to rate these 15 different parameter for particular project on scale of one to three.
- These 15 values are multiplied to calculate EAF (Effort Adjustment Factor)
- The Intermediate COCOMO formula now takes the form

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

COCOMO COST MODELING

Complete/Detailed COCOMO

- The major shortcoming of basic and intermediate COCOMO is they consider a software project a single homogenous entity.
- However, most large system are made up several smaller sub system. These subsystem may have widely different characteristics.
- Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step of software engineering process.
- In detailed COCOMO the whole software is divided into different modules and then we apply COCOMO in the different modules to estimate effort and then sum the effort.
- The effort is calculated as function of program size and set of cost drivers are given according to each phase of the software lifecycle.

INTRODUCTION TO QUALITY MANAGEMENT

- Software Quality Management (SQM) is a critical discipline concerned with ensuring that software systems are **"fit for purpose."** This means the final product must meet user needs, perform efficiently and reliably, and be delivered on time and within budget. The application of SQM techniques, alongside advancements in technology and testing, has led to significant improvements in software quality.

The Scope of Quality Management

- Formalized QM is especially vital for large, complex systems with long lifetimes, developed for external clients using plan-based processes. It operates at two distinct levels:
- **Organizational Level:** Establishes a framework of organizational processes and standards designed to consistently produce high-quality software. The QM team defines the software development processes and the standards for all deliverables, including requirements, design, and code.
- **Project Level:** Involves the application of specific quality processes for a particular project. This includes checking that planned processes are followed and ensuring that project outputs comply with the defined standards. At this level, a **Quality Plan** is often created, which sets out the project's quality goals and defines the processes and standards to be used.

QUALITY ASSURANCE VS. QUALITY CONTROL

- SQM techniques are derived from manufacturing, where two key concepts are used:
  - **Quality Assurance (QA):** The proactive definition of processes and standards that should lead to high-quality products. It is about *preventing* defects by establishing a good process.
  - **Quality Control (QC):** The reactive application of these quality processes to identify and weed out products that do not meet the required quality standards. It is about *finding* defects in the final product.
- Both QA and QC are integral components of overall Quality Management.

INTRODUCTION TO CONFIGURATION MANAGEMENT

- Software systems are dynamic entities that undergo constant change throughout their lifecycle. Bugs are fixed, requirements evolve, and systems must adapt to new hardware and competitive pressures. Each change creates a new **version** of the system, and managing these multiple, concurrent versions is a critical challenge.
- **Configuration Management (CM)** is the discipline of managing the policies, processes, and tools for controlling changing software systems. Its primary purpose is to prevent chaos by ensuring that you can always identify which changes are in which version, avoid modifying the wrong version, deliver the correct version to customers, and maintain a reliable record of all system components.

Why Configuration Management is Essential

- **For Individual Projects:** It helps a single developer keep track of changes.
- **For Team Projects:** It is indispensable for coordinating the work of multiple developers, especially in geographically distributed teams. The CM system provides controlled access to the code and manages concurrent changes.
- **For Agile Development:** Agile methods, which involve multiple daily changes, are impossible without automated CM tools. They rely on a shared project repository and system-building tools to integrate changes smoothly.

INTRODUCTION TO CONFIGURATION MANAGEMENT

The Four Core Activities of Configuration Management

- CM involves four closely related activities, as illustrated in Figure 10.7:
  - **Version Management:** Tracks different versions of system components to ensure that changes made by different developers do not interfere with each other.
  - **System Building:** The process of assembling program components, data, and libraries, then compiling and linking them to create a working executable system.
  - **Change Management:** Tracks requests for changes from customers and developers, assesses the cost and impact of these changes, and decides if and when they should be implemented.
  - **Release Management:** Prepares software for external release and keeps track of the different system versions that have been delivered to customers.

INTRODUCTION TO CONFIGURATION MANAGEMENT

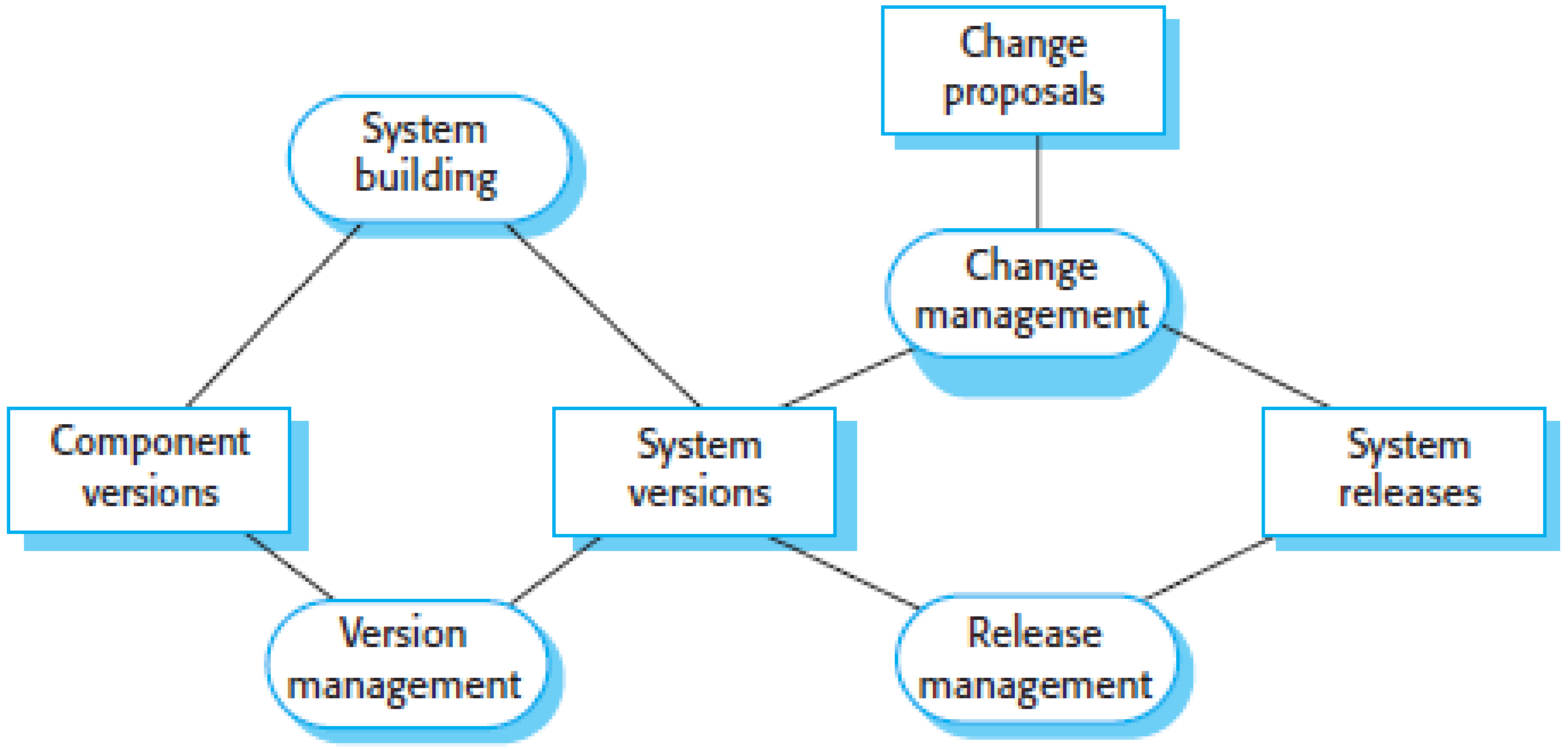


Figure 10.7: Configuration management activities