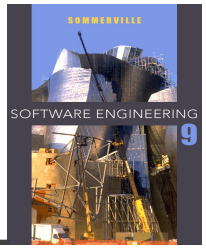
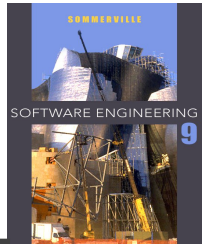


Chapter 22 – Project Management

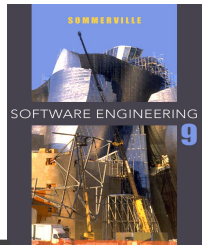
Lecture 1

Topics covered



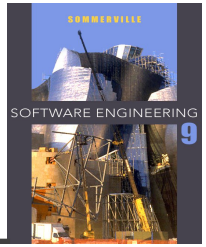
- Risk management
- Managing people
- Teamwork

Software project management



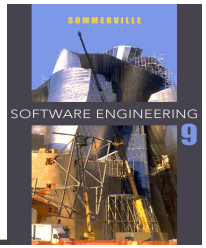
- Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Success criteria



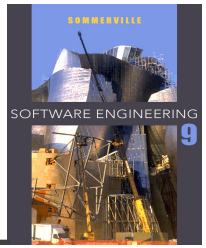
- Deliver the software to the customer at the agreed time.
- Keep overall costs within budget.
- Deliver software that meets the customer's expectations.
- Maintain a happy and well-functioning development team.

Software management distinctions



- The product is intangible.
  Software cannot be seen or touched. Software project managers cannot see progress by simply looking at the artefact that is being constructed.
- Many software projects are 'one-off' projects.
  Large software projects are usually different in some ways from previous projects. Even managers who have lots of previous experience may find it difficult to anticipate problems.
- Software processes are variable and organization specific.
  We still cannot reliably predict when a particular

Management activities



- *Project planning*

  Project managers are responsible for planning. estimating and scheduling project development and assigning people to tasks.

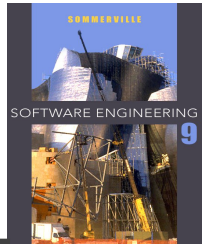
- *Reporting*

  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.

- *Risk management*

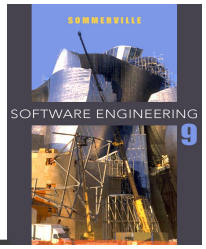
  Project managers assess the risks that may affect a project, monitor these risks and take action when

Management activities



- *People management*
  - Project managers have to choose people for their team and establish ways of working that leads to effective team performance
- *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.

Risk management



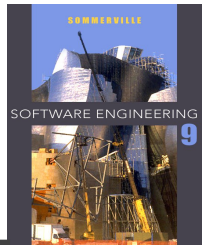
- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- A risk is a probability that some adverse circumstance will occur

  Project risks affect schedule or resources;

  Product risks affect the quality or performance of the software being developed;

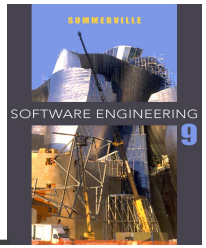
  Business risks affect the organisation developing or procuring the software.

Examples of common project, product, and business risks



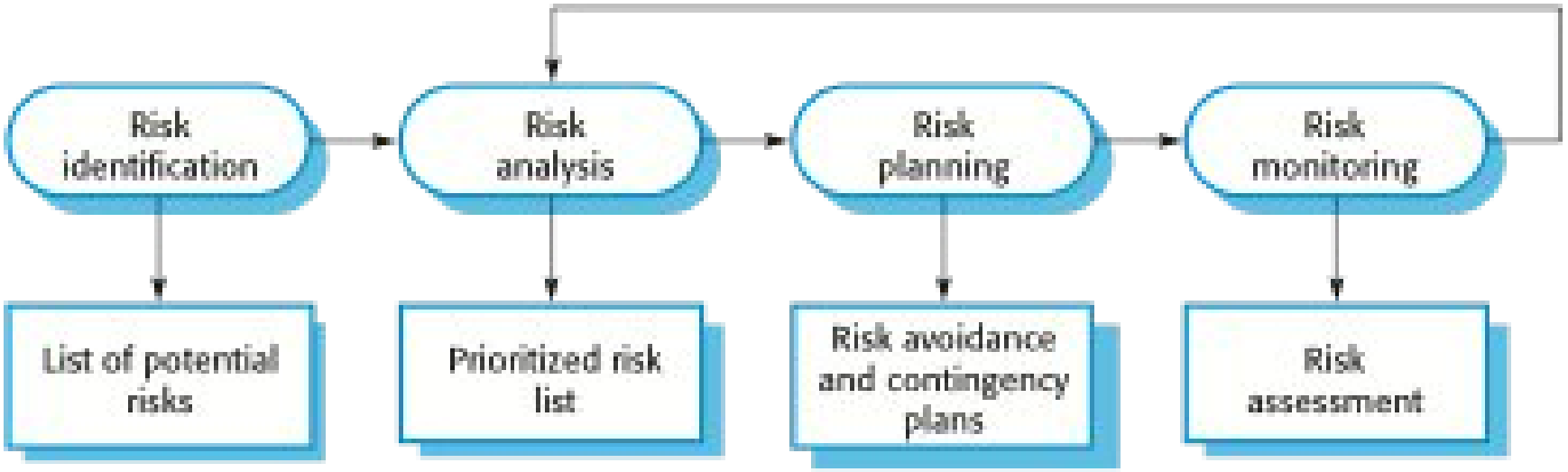
Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not 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.
CASE tool underperformance	Product	CASE tools, which 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.

The risk management process

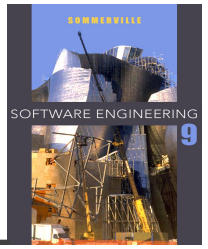


- Risk identification
  Identify project, product and business risks;
- Risk analysis
  Assess the likelihood and consequences of these risks;
- Risk planning
  Draw up plans to avoid or minimise the effects of the risk;
- Risk monitoring
  Monitor the risks throughout the project;

The risk management process



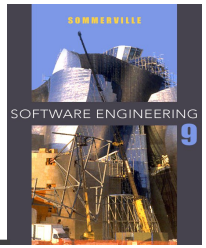
Risk identification



- May be a team activities or based on the individual project manager's experience.
- A checklist of common risks may be used to identify risks in a project

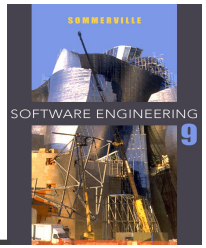
  Technology risks.
  People risks.
  Organisational risks.
  Requirements risks.
  Estimation risks.

Examples of different risk types



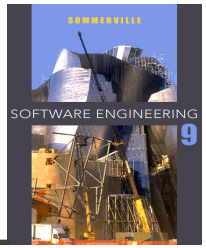
Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. (1) Reusable software components contain defects that mean they cannot be reused as planned. (2)
People	It is impossible to recruit staff with the skills required. (3) Key staff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

Risk analysis



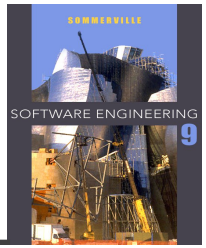
- Assess probability and seriousness of each risk.
- Probability may be very low, low, moderate, high or very high.
- Risk consequences might be catastrophic, serious, tolerable or insignificant.

Risk types and examples



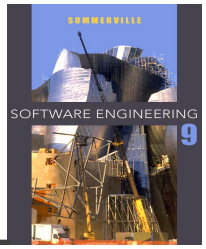
[2]Risk	[2]Probability	[2]Effects
[2]Organizational financial problems force reductions in the project budget (7).	[2]Low	[2]Catastrophic
[2]It is impossible to recruit staff with the skills required for the project (3).	[2]High	[2]Catastrophic
[2]Key staff are ill at critical times in the project (4).	[2]Moderate	[2]Serious
[2]Faults in reusable software components have to be repaired before these components are reused. (2).	[2]Moderate	[2]Serious
[2]Changes to requirements that require major design rework are proposed (10).	[2]Moderate	[2]Serious
[2]The organization is restructured so that different management are responsible for the project (6).	[2]High	[2]Serious
[2]The database used in the system cannot process as many transactions per second as expected (1).	[2]Moderate	[2]Serious

Risk types and examples



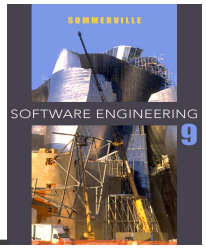
[2]Risk	[2]Probability	[2]Effects
[2]The time required to develop the software is underestimated (12).	[2]High	[2]Serious
[2]Software tools cannot be integrated (9).	[2]High	[2]Tolerable
[2]Customers fail to understand the impact of requirements changes (11).	[2]Moderate	[2]Tolerable
[2]Required training for staff is not available (5).	[2]Moderate	[2]Tolerable
[2]The rate of defect repair is underestimated (13).	[2]Moderate	[2]Tolerable
[2]The size of the software is underestimated (14).	[2]High	[2]Tolerable
[2]Code generated by code generation tools is inefficient (8).	[2]Moderate	[2]Insignificant

Risk planning



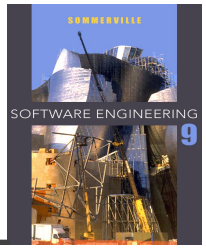
- Consider each risk and develop a strategy to manage that risk.
- Avoidance strategies
  The probability that the risk will arise is reduced;
- Minimisation strategies
  The impact of the risk on the project or product will be reduced;
- Contingency plans
  If the risk arises, contingency plans are plans to deal with that risk;

Strategies to help manage risk



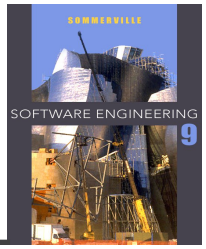
[2]Risk	[2]Strategy
[2]Organizational financial problems	[2]Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost-effective.
[2]Recruitment problems	[2]Alert customer to potential difficulties and the possibility of delays; investigate buying-in components.
[2]Staff illness	[2]Reorganize team so that there is more overlap of work and people therefore understand each other's jobs.
[2]Defective components	[2]Replace potentially defective components with bought-in components of known reliability.
[2]Requirements changes	[2]Derive traceability information to assess requirements change impact; maximize information hiding in the design.

Strategies to help manage risk



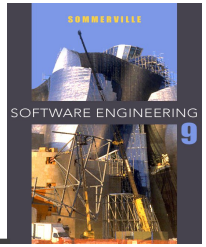
[2]Risk	[2]Strategy
[2]Organizational restructuring	[2]Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
[2]Database performance	[2]Investigate the possibility of buying a higher-performance database.
[2]Underestimated development time	[2]Investigate buying-in components; investigate use of a program generator.

Risk monitoring



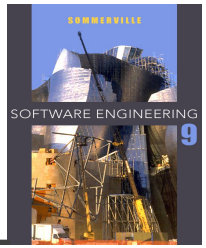
- Assess each identified risks regularly to decide whether or not it is becoming less or more probable.
- Also assess whether the effects of the risk have changed.
- Each key risk should be discussed at management progress meetings.

Risk indicators

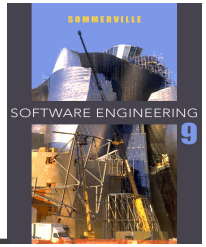


[2]Risk type	[2]Potential indicators
[2]Technology	[2]Late delivery of hardware or support software; many reported technology problems.
[2]People	[2]Poor staff morale; poor relationships amongst team members; high staff turnover.
[2]Organizational	[2]Organizational gossip; lack of action by senior management.
[2]Tools	[2]Reluctance by team members to use tools; complaints about CASE tools; demands for higher-powered workstations.
[2]Requirements	[2]Many requirements change requests; customer complaints.
[2]Estimation	[2]Failure to meet agreed schedule; failure to clear reported defects.

Key points



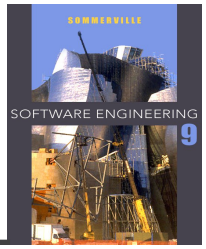
- Good project management is essential if software engineering projects are to be developed on schedule and within budget.
- Software management is distinct from other engineering management. Software is intangible. Projects may be novel or innovative with no body of experience to guide their management. Software processes are not as mature as traditional engineering processes.
- Risk management is now recognized as one of the most important project management tasks.
- Risk management involves identifying and assessing project risks to establish the probability that they will occur and the consequences for the project if that risk does arise. You should make plans to avoid, manage or deal with likely risks if or when they arise.



Chapter 22 – Project Management

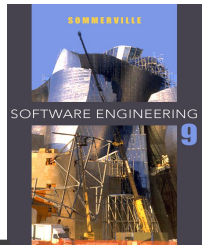
Lecture 2

Managing people



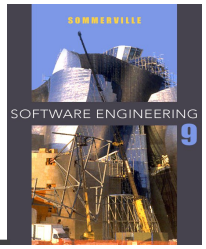
- People are an organisation's most important assets.
- The tasks of a manager are essentially people-oriented. Unless there is some understanding of people, management will be unsuccessful.
- Poor people management is an important contributor to project failure.

People management factors



- Consistency
  Team members should all be treated in a comparable way without favourites or discrimination.
- Respect
  Different team members have different skills and these differences should be respected.
- Inclusion
  Involve all team members and make sure that people's views are considered.
- Honesty
  You should always be honest about what is going well and what is going badly in a project.

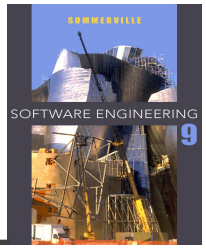
Motivating people



- An important role of a manager is to motivate the people working on a project.
- Motivation means organizing the work and the working environment to encourage people to work effectively.
  If people are not motivated, they will not be 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.
- Motivation is a complex issue but it appears that their are different types of motivation based on:

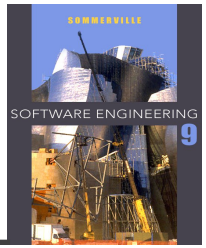
Human needs hierarchy

Need satisfaction



- In software development groups, basic physiological and safety needs are not an issue.
- Social
  - Provide communal facilities;
  - Allow informal communications e.g. via social networking
- Esteem
  - Recognition of achievements;
  - Appropriate rewards.
- Self-realization
  - Training - people want to learn more;
  - Responsibility.

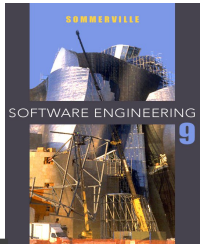
Individual motivation



Alice is a software project manager working in a company that develops alarm systems. This company wishes to enter the growing market of assistive technology to help elderly and disabled people live independently. Alice has been asked to lead a team of 6 developers than can develop new products based around the company's alarm technology.

Alice's assistive technology project starts well. Good working relationships develop within the team and creative new ideas are developed. The team decides to develop a peer-to-peer messaging system using digital televisions linked to the alarm network for communications. However, some months into the project, Alice notices that Dorothy, a hardware design expert, starts coming into work late, the quality of her work deteriorates and, increasingly, that she does not appear to be communicating with other members of the team.
Alice talks about the problem informally with other team members to try to find out if Dorothy's personal circumstances have changed, and if this might be affecting her work. They don't know of anything, so Alice decides to talk with Dorothy to try to understand the problem.

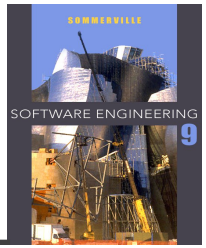
Individual motivation



After some initial denials that there is a problem, Dorothy admits that she has lost interest in the job. She expected that she would be able to develop and use her hardware interfacing skills. However, because of the product direction that has been chosen, she has little opportunity for this. Basically, she is working as a C programmer with other team members.

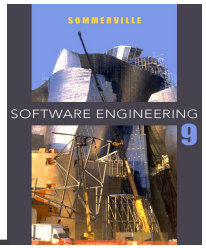
Although she admits that the work is challenging, she is concerned that she is not developing her interfacing skills. She is worried that finding a job that involves hardware interfacing will be difficult after this project. Because she does not want to upset the team by revealing that she is thinking about the next project, she has decided that it is best to minimize conversation with them.

Personality types



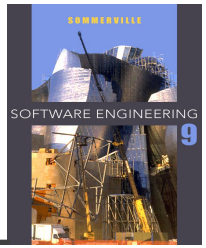
- The needs hierarchy is almost certainly an over-simplification of motivation in practice.
- Motivation should also take into account different personality types:
  Task-oriented;
  Self-oriented;
  Interaction-oriented.

Personality types



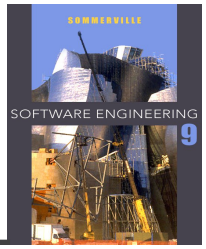
- Task-oriented.
  The motivation for doing the work is the work itself;
- Self-oriented.
  The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.;
- Interaction-oriented
  The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work.

Motivation balance



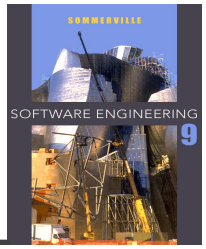
- Individual motivations are made up of elements of each class.
- The balance can change depending on personal circumstances and external events.
- However, people are not just motivated by personal factors but also by being part of a group and culture.
- People go to work because they are motivated by the people that they work with.

Teamwork



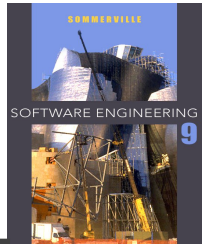
- Most software engineering is a group activity
  The development schedule for most non-trivial software projects is such that they cannot be completed by one person working alone.
- A good group is cohesive and has a team spirit. The people involved are motivated by the success of the group as well as by their own personal goals.
- Group interaction is a key determinant of group performance.
- Flexibility in group composition is limited
  Managers must do the best they can with available people

Group cohesiveness



- In a cohesive group, members consider the group to be more important than any individual in it.
- The advantages of a cohesive group are:
  - Group quality standards can be developed by the group members.
  - Team members learn from each other and get to know each other's work; Inhibitions caused by ignorance are reduced.
  - Knowledge is shared. Continuity can be maintained if a group member leaves.
  - Refactoring and continual improvement is encouraged. Group members work collectively to deliver high quality

Team spirit

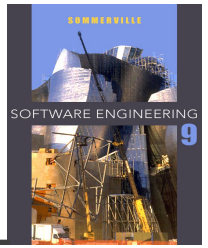


Alice, an experienced project manager, understands the importance of creating a cohesive group. As they are developing a new product, she takes the opportunity of involving all group members in the product specification and design by getting them to discuss possible technology with elderly members of their families. She also encourages them to bring these family members to meet other members of the development group.

Alice also arranges monthly lunches for everyone in the group. These lunches are an opportunity for all team members to meet informally, talk around issues of concern, and get to know each other. At the lunch, Alice tells the group what she knows about organizational news, policies, strategies, and so forth. Each team member then briefly summarizes what they have been doing and the group discusses a general topic, such as new product ideas from elderly relatives.

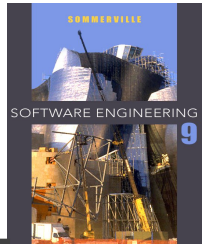
Every few months, Alice organizes an ‘away day’ for the group where the team spends two days on ‘technology updating’. Each team member prepares an update on a relevant technology and presents it to the group. This is an off-site meeting in a good hotel and plenty of time is scheduled for discussion and social interaction.

The effectiveness of a team



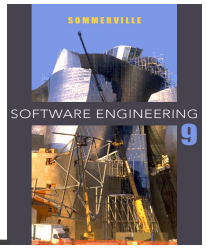
- The people in the group
  - You need a mix of people in a project group as software development involves diverse activities such as negotiating with clients, programming, testing and documentation.
- The group organization
  - A group should be organized so that individuals can contribute to the best of their abilities and tasks can be completed as expected.
- Technical and managerial communications
  - Good communications between group members, and between the software engineering team and other project stakeholders, is essential.

Selecting group members



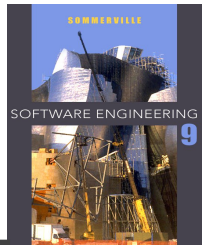
- A manager or team leader's job is to create a cohesive group and organize their group so that they can work together effectively.
- This involves creating a group with the right balance of technical skills and personalities, and organizing that group so that the members work together effectively.

Assembling a team



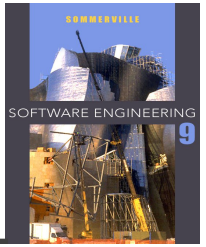
- May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff;
  - Staff with the appropriate experience may not be available;
  - An organisation may wish to develop employee skills on a software project.
- Managers have to work within these constraints especially when there are shortages of trained staff.

Group composition



- Group composed of members who share the same motivation can be problematic
  - Task-oriented - everyone wants to do their own thing;
  - Self-oriented - everyone wants to be the boss;
  - Interaction-oriented - too much chatting, not enough work.
- An effective group has a balance of all types.
- This can be difficult to achieve software engineers are often task-oriented.
- Interaction-oriented people are very important as they can detect and defuse tensions that arise.

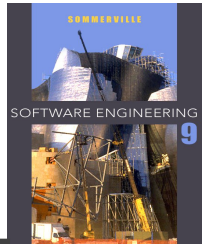
Group composition



In creating a group for assistive technology development, Alice is aware of the importance of selecting members with complementary personalities. When interviewing potential group members, she tried to assess whether they were task-oriented, self-oriented, or interaction-oriented. She felt that she was primarily a self-oriented type because she considered the project to be a way of getting noticed by senior management and possibly promoted. She therefore looked for one or perhaps two interaction-oriented personalities, with task-oriented individuals to complete the team. The final assessment that she arrived at was:

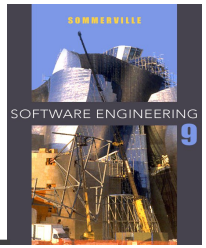
Alice—self-oriented
Brian—task-oriented
Bob—task-oriented
Carol—interaction-oriented
Dorothy—self-oriented
Ed—interaction-oriented
Fred—task-oriented

Group organization



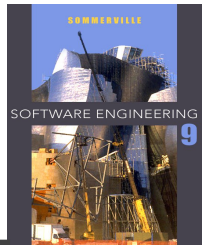
- The way that a group is organized affects the decisions that are made by that group, the ways that information is exchanged and the interactions between the development group and external project stakeholders.
  - Key questions include:
    - Should the project manager be the technical leader of the group?
    - Who will be involved in making critical technical decisions, and how will these be made?
    - How will interactions with external stakeholders and senior company management be handled?
    - How can groups integrate people who are not co-located?
    - How can knowledge be shared across the group?

Group organization



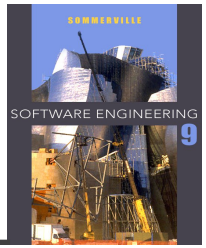
- Small software engineering groups are usually organised informally without a rigid structure.
- For large projects, there may be a hierarchical structure where different groups are responsible for different sub-projects.
- Agile development is always based around an informal group on the principle that formal structure inhibits information exchange

Informal groups



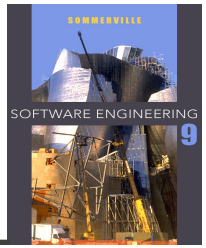
- The group acts as a whole and comes to a consensus on decisions affecting the system.
- The group leader serves as the external interface of the group but does not allocate specific work items.
- Rather, work is discussed by the group as a whole and tasks are allocated according to ability and experience.
- This approach is successful for groups where all members are experienced and competent.

Group communications



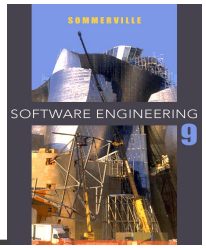
- Good communications are essential for effective group working.
- Information must be exchanged on the status of work, design decisions and changes to previous decisions.
- Good communications also strengthens group cohesion as it promotes understanding.

Group communications

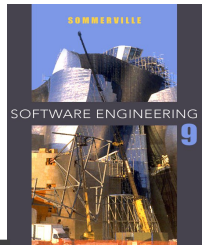


- Group size
  - The larger the group, the harder it is for people to communicate with other group members.
- Group structure
  - Communication is better in informally structured groups than in hierarchically structured groups.
- Group composition
  - Communication is better when there are different personality types in a group and when groups are mixed rather than single sex.
- The physical work environment
  - Good workplace organisation can help encourage communications.

Key points



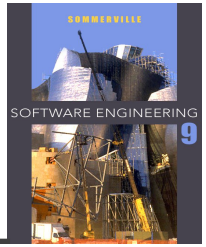
- People are motivated by interaction with other people, the recognition of management and their peers, and by being given opportunities for personal development.
- Software development groups should be fairly small and cohesive. The key factors that influence the effectiveness of a group are the people in that group, the way that it is organized and the communication between group members.
- Communications within a group are influenced by factors such as the status of group members, the size of the group, the gender composition of the group, personalities and available communication channels.



Chapter 23 – Project planning

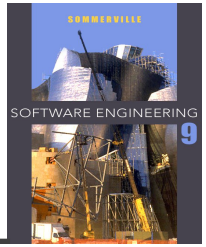
Lecture 1

Topics covered



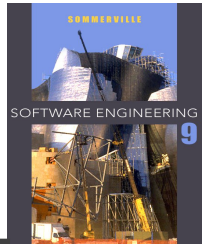
- **Software pricing**
- **Plan-driven development**
- **Project scheduling**
- Agile planning
- **Estimation techniques**

Project planning



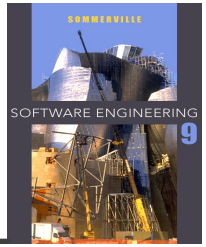
- Project planning involves breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Planning stages



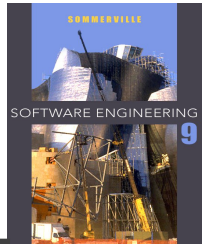
- **At the proposal stage**, when you are bidding for a contract to develop or provide a software system.
- **During the project startup phase**, when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
- **Periodically throughout the project**, when you modify your plan in the light of experience gained and information from monitoring the progress of the work.

Proposal planning



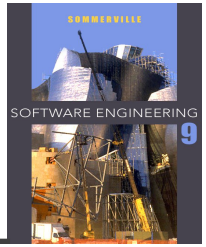
- Planning may be necessary with only outline software requirements.
- The aim of planning at this stage is to provide information that will be used in setting a price for the system to customers.

Software pricing



- Estimates are made to discover the cost, to the developer, of producing a software system.
  - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

Cont…



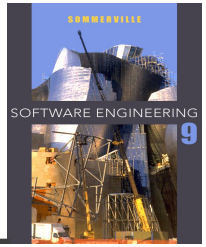
- ✧ pricing of a software project—it is not simply cost + profit.
- ✧ There are three main parameters that you should use when computing the costs of a software development project:

• effort costs (the costs of paying software engineers and managers);

• hardware and software costs, including maintenance;

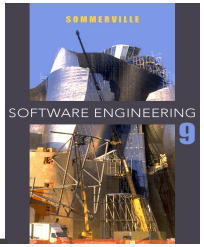
• travel and training costs.

Factors affecting software pricing



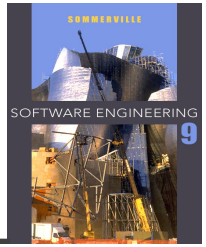
Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
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.
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 may then be less than if the software source code is handed over to the customer.

Factors affecting software pricing



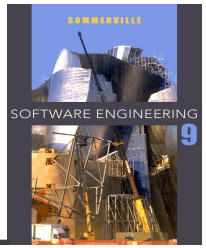
Factor	Description
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price 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.

Plan-driven development



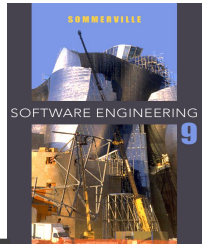
- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
  - Plan-driven development is based on engineering project management techniques and is the ‘traditional’ way of managing large software development projects.
- A project 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.

Plan-driven development – pros and cons



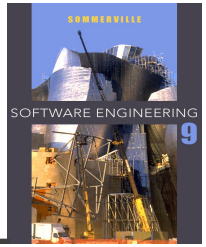
- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
- The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

Project plans



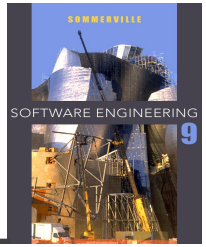
- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.
- Plan sections
  - Introduction
  - Project organization
  - Risk analysis
  - Hardware and software resource requirements
  - Work breakdown
  - Project schedule
  - Monitoring and reporting mechanisms

Project plan supplements



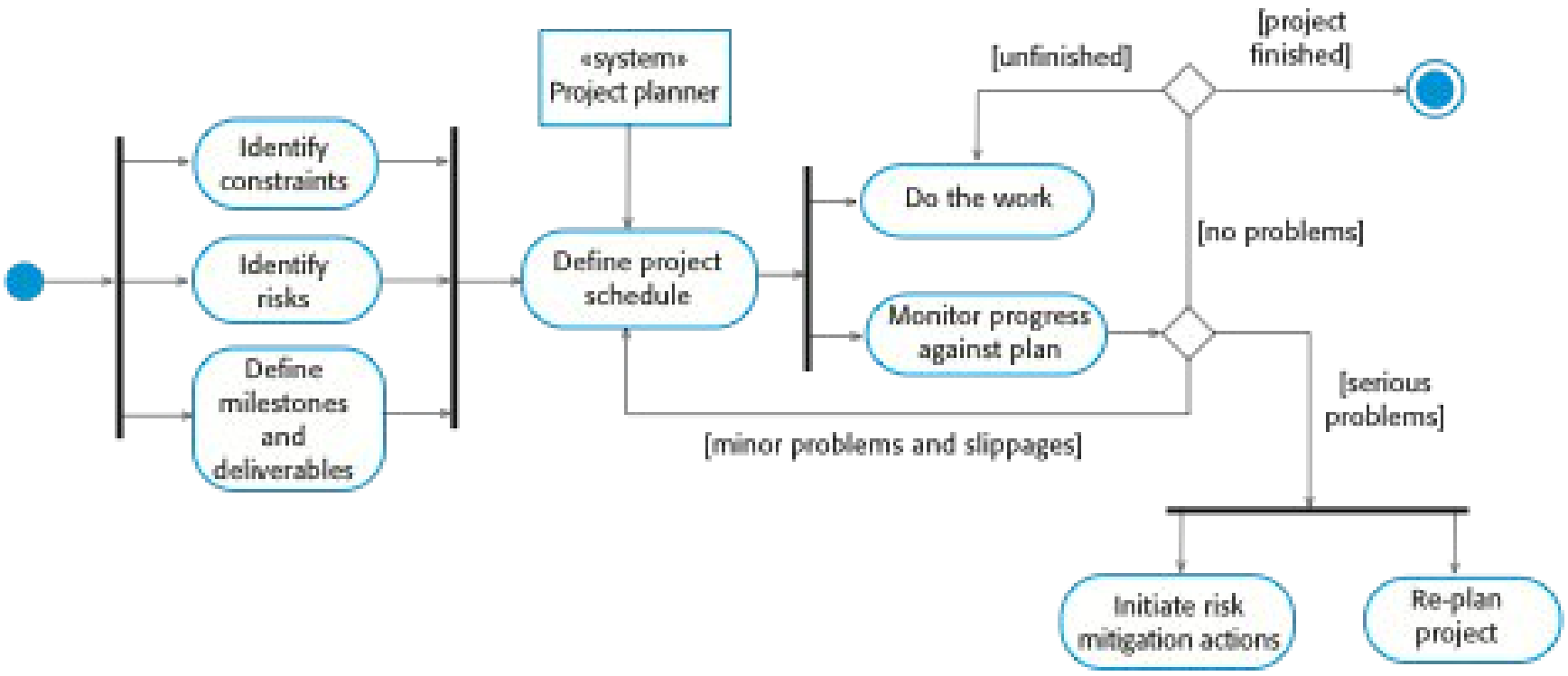
Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

The planning process

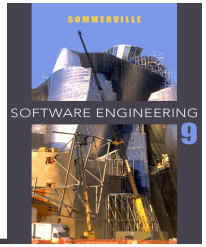


- Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.
- Plan changes are inevitable.
  - As more information about the system and the project team becomes available during the project, you should regularly revise the plan to reflect requirements, schedule and risk changes.
  - Changing business goals also leads to changes in project plans. As business goals change, this could affect all projects, which may then have to be re-planned.

The project planning process

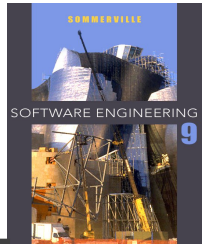


Project scheduling



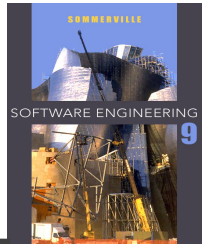
- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- You estimate the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified.
- You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

Project scheduling activities



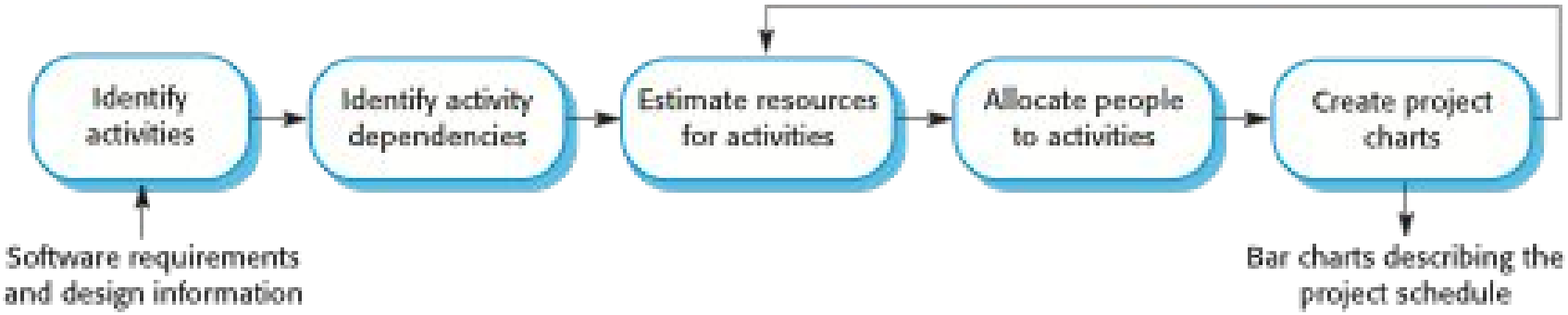
- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

Milestones and deliverables

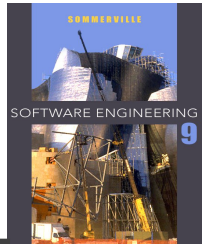


- Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- Deliverables are work products that are delivered to the customer, e.g. a requirements document for the system.

The project scheduling process

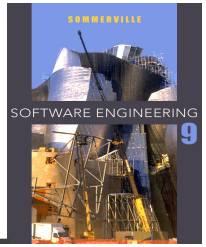


Scheduling problems



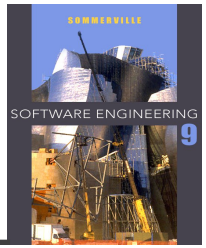
- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

Schedule representation



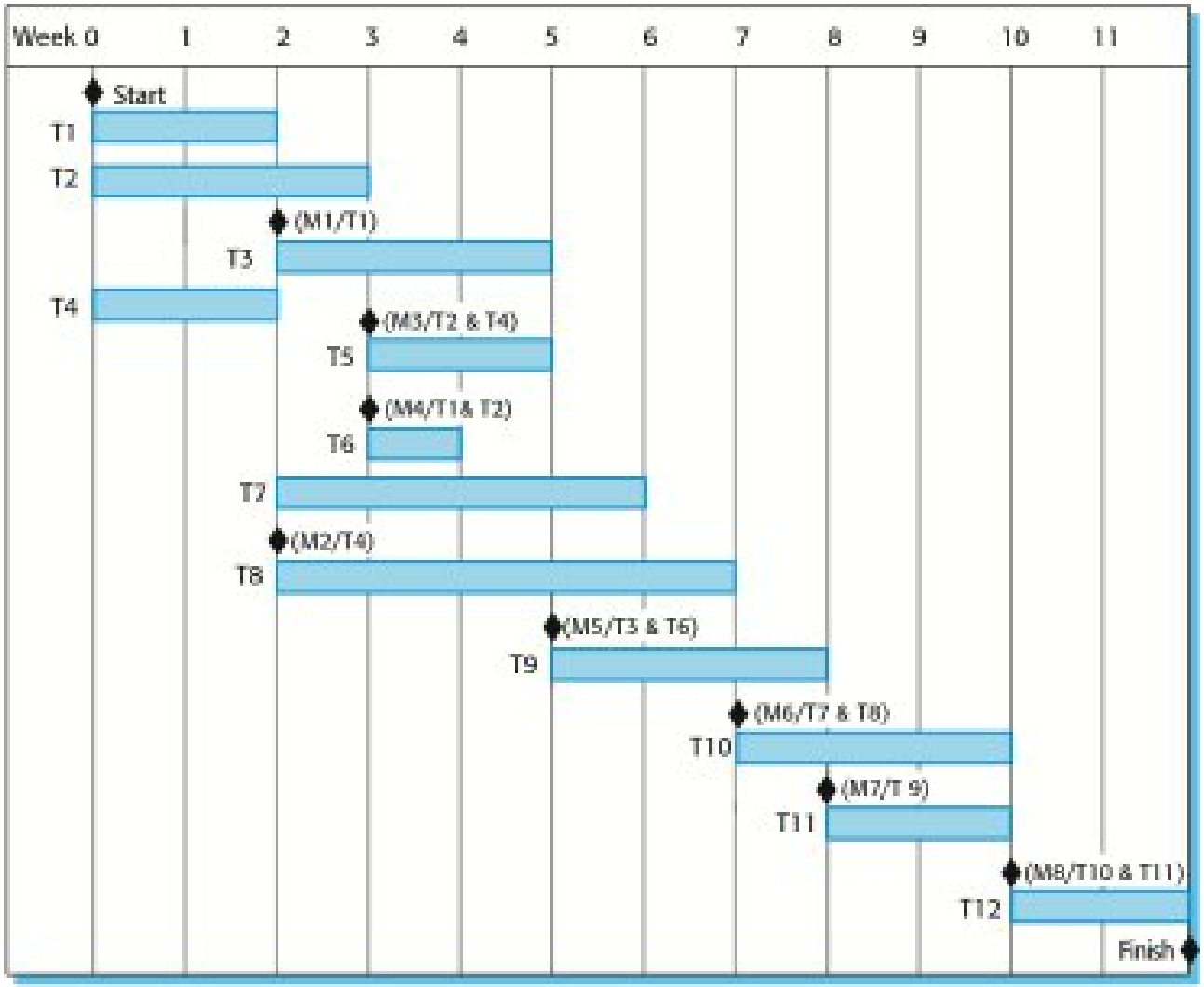
- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Bar charts are the most commonly used representation for project schedules. They show the schedule as activities or resources against time.

Tasks, durations, and dependencies

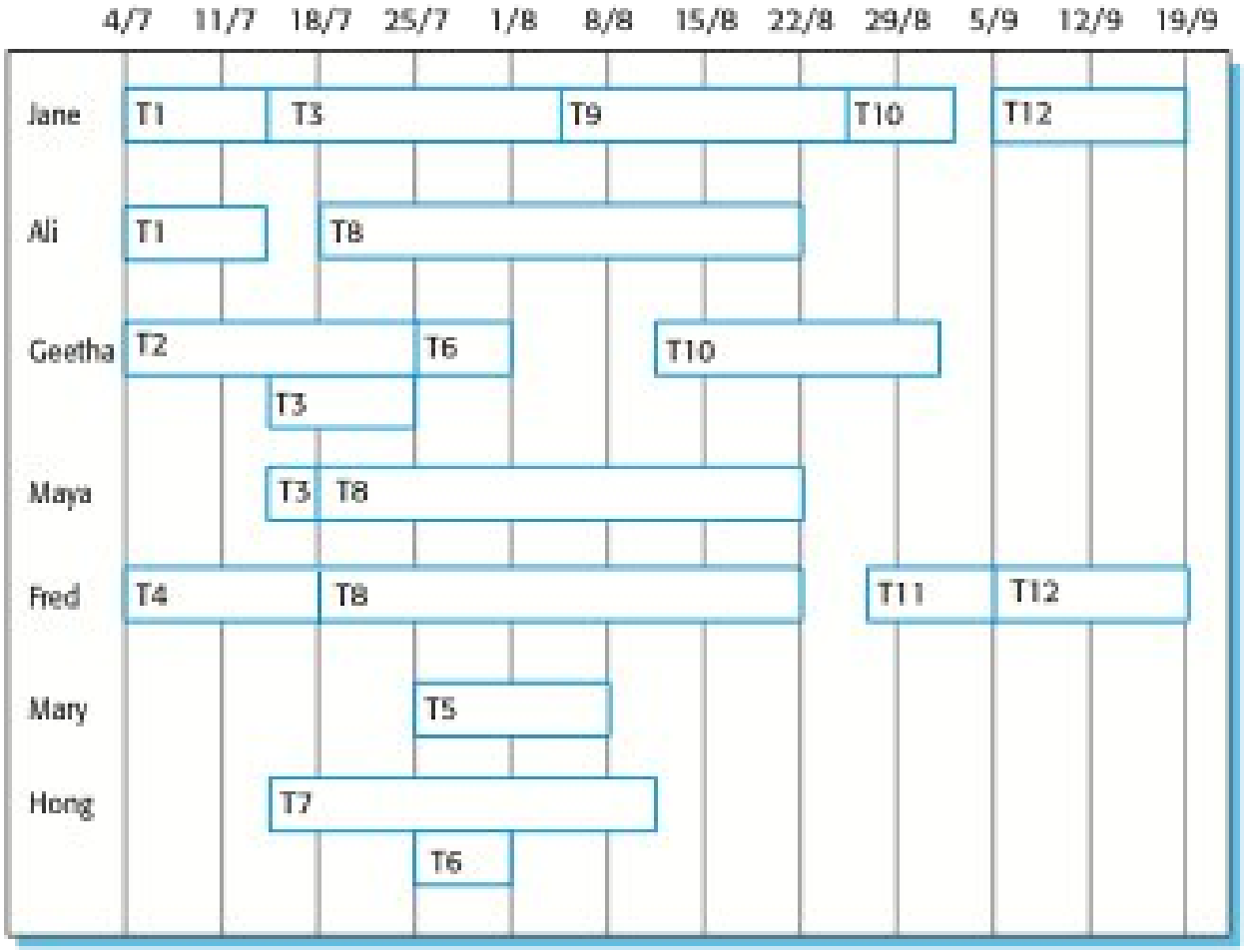


Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

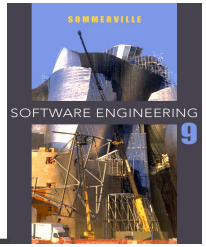
Activity bar chart



Staff allocation chart

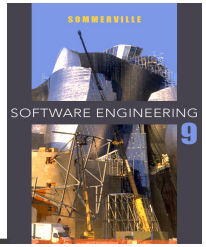


Agile planning



- 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.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.

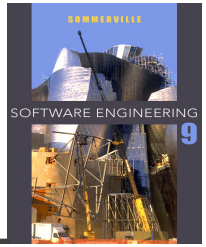
Agile planning stages



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

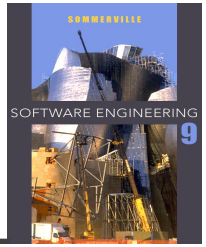
Planning in XP

Story-based planning

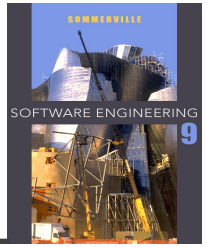


- The system specification in XP is based on user stories that reflect the features that should be included in the system.
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.
- Release planning involves selecting 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.
- Stories to be implemented in each iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks).

Key points



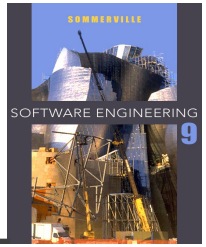
- The price charged for a system does not just depend on its estimated development costs; it may be adjusted depending on the market and organizational priorities.
- Plan-driven development is organized around a complete project plan that defines the project activities, the planned effort, the activity schedule and who is responsible for each activity.
- Project scheduling involves the creation of graphical representations the project plan. Bar chartsshow the activity duration and staffing timelines, are the most commonly used schedule representations.
- The XP planning game involves the whole team in project planning. The plan is developed incrementally and, if problems arise, is adjusted. Software functionality is reduced instead of delaying delivery of an increment.



Chapter 23 – Project planning

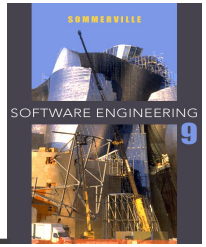
Lecture 2

Estimation techniques



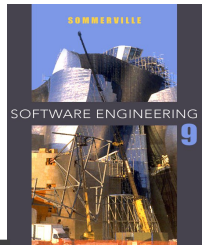
- Organizations need to make software effort and cost estimates. There are two types of technique that can be used to do this:
  - *Experience-based techniques* The estimate of future effort requirements is based on the manager's experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.
  - *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, and process characteristics, such as experience of staff involved.

Experience-based approaches



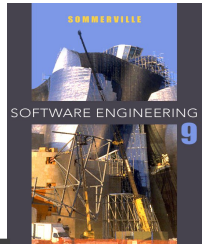
- Experience-based techniques rely on judgments based on experience of past projects and the effort expended in these projects on software development activities.
- Typically, you identify the deliverables to be produced in a project and the different software components or systems that are to be developed.
- 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.

Algorithmic cost modelling



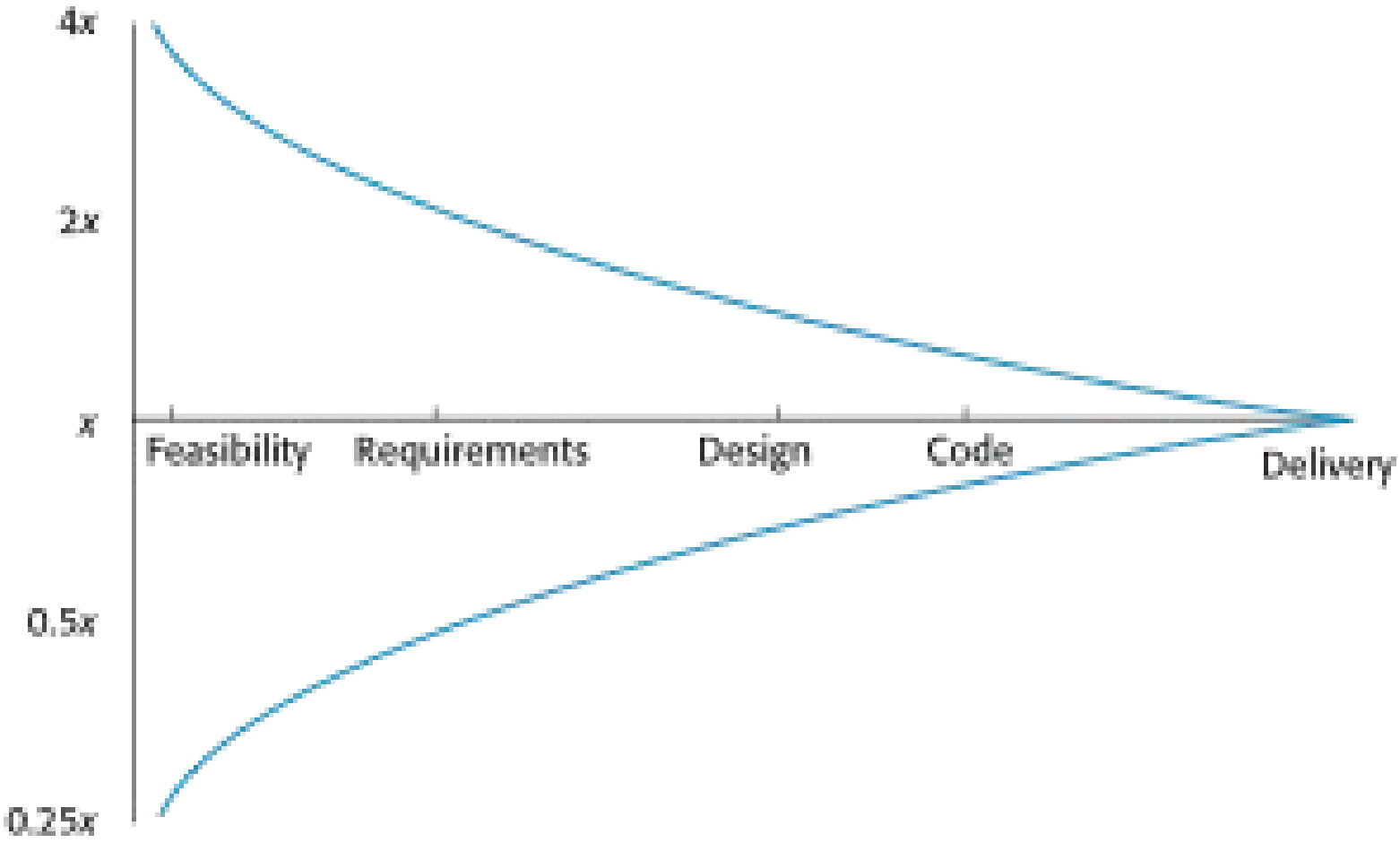
- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - Effort = A × Size^{B} × M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes. Size is an code size.
- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.

Estimation accuracy

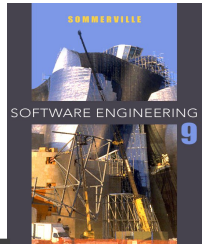


- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of COTS and components;
  - Programming language;
  - Distribution of system.
- As the development process progresses then the size estimate becomes more accurate.
- The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.

Estimate uncertainty

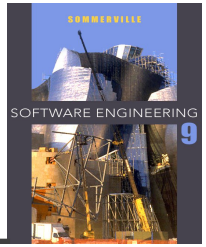


The COCOMO 2 model



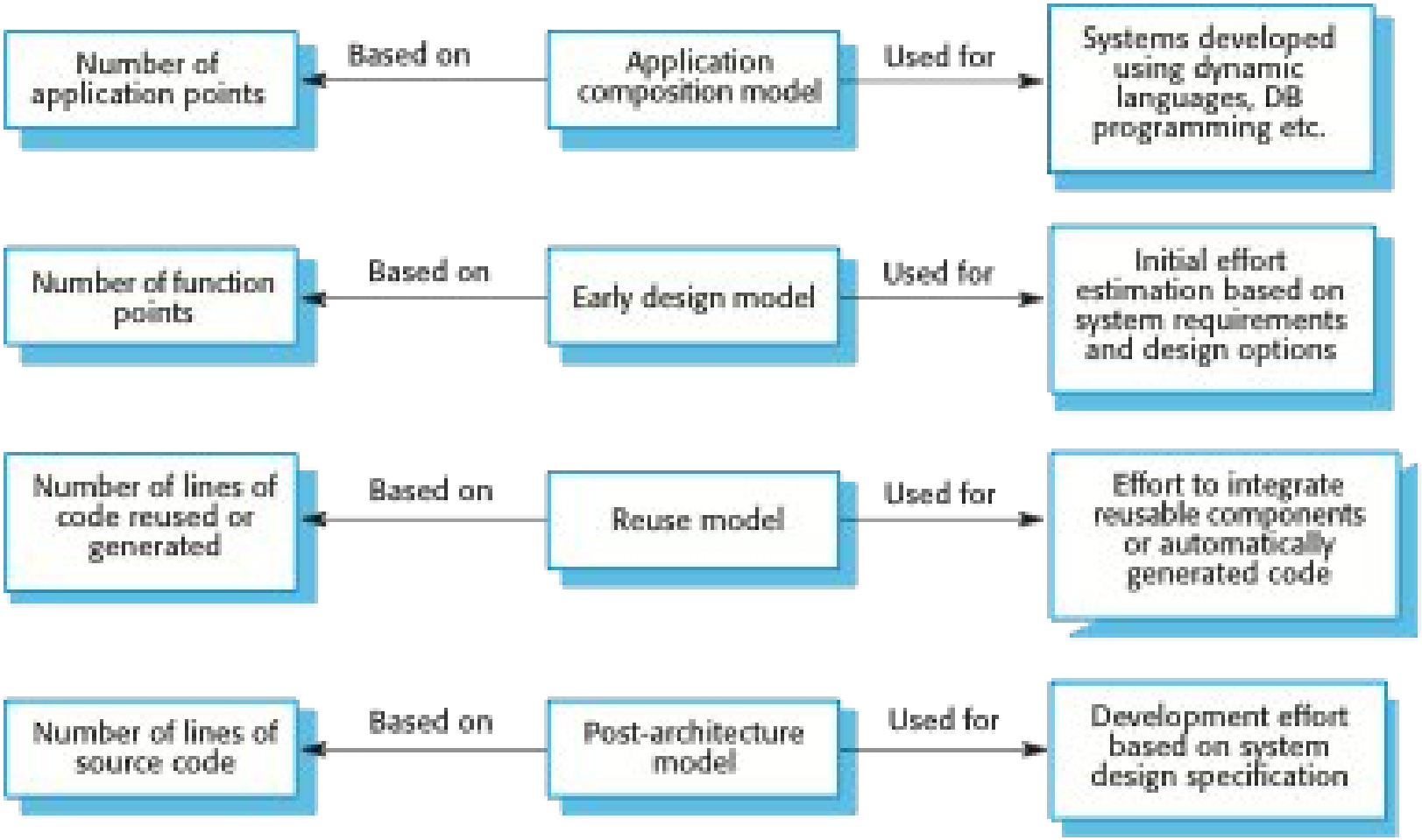
- An empirical model based on project experience.
- Well-documented, ‘independent’ model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 2 models

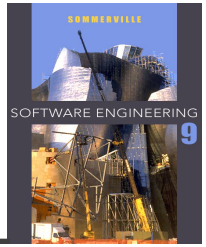


- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the effort of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

COCOMO estimation models

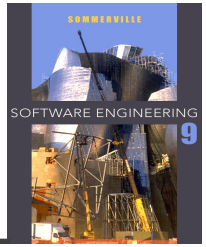


Application composition model



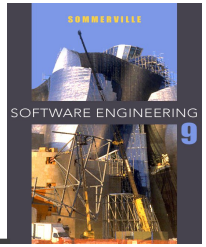
- Supports prototyping projects and projects where there is extensive reuse.
- Based on standard estimates of developer productivity in application (object) points/month.
- Takes CASE tool use into account.
- Formula is
  - PM = (NAP × (1 - %reuse/100)) / PROD
  - PM is the effort in person-months, NAP is the number of application points and PROD is the productivity.

Application-point productivity



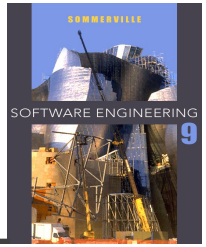
Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

Early design model



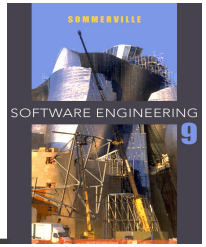
- Estimates can be made after the requirements have been agreed.
- Based on a standard formula for algorithmic models
  - PM = A × Size^{B} × M where
  - M = PERS × RCPX × RUSE × PDIF × PREX × FCIL × SCED;
  - A = 2.94 in initial calibration, Size in KLOC, B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

Multipliers



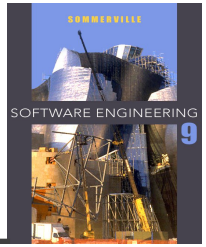
- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity;
  - RUSE - the reuse required;
  - PDIF - platform difficulty;
  - PREX - personnel experience;
  - PERS - personnel capability;
  - SCED - required schedule;
  - FCIL - the team support facilities.

The reuse model



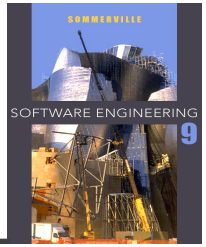
- Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code.
- There are two versions:
  - Black-box reuse where code is not modified. An effort estimate (PM) is computed.
  - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates 1



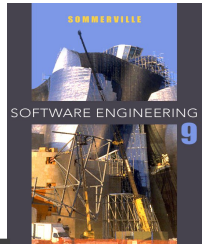
- For generated code:
  - PM = (ASLOC * AT/100)/ATPROD
  - ASLOC is the number of lines of generated code
  - AT is the percentage of code automatically generated.
  - ATPROD is the productivity of engineers in integrating this code.

Reuse model estimates 2



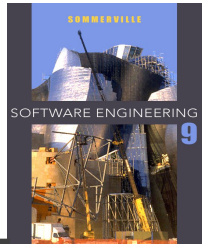
- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - ASLOC and AT as before.
  - AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.

Post-architecture level



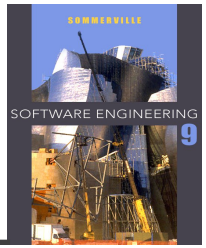
- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.

The exponent term



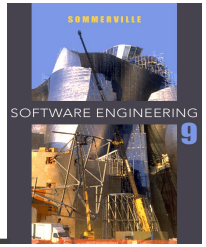
- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.
  - Precedenteness - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low .(5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)
- Scale factor is therefore 1.17.

Scale factors used in the exponent computation in the post-architecture model



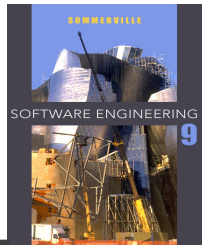
Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organization. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Multipliers



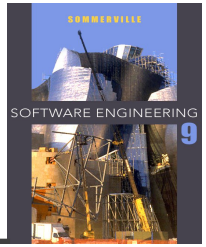
- Product attributes
  - Concerned with required characteristics of the software product being developed.
- Computer attributes
  - Constraints imposed on the software by the hardware platform.
- Personnel attributes
  - Multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes
  - Concerned with the particular characteristics of the software development project.

The effect of cost drivers on effort estimates



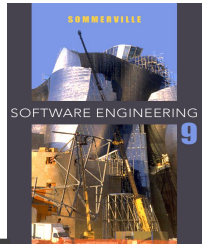
Exponent value	**1.17**
System size (including factors for reuse and requirements volatility)	128,000 DSI
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low, multiplier = 1.12
Schedule	Accelerated, multiplier = 1.29
Adjusted COCOMO estimate	**2,306 person-months**

The effect of cost drivers on effort estimates



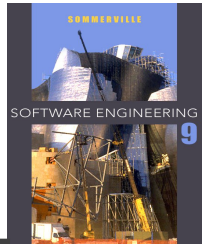
Exponent value	**1.17**
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
Adjusted COCOMO estimate	**295 person-months**

Project duration and staffing



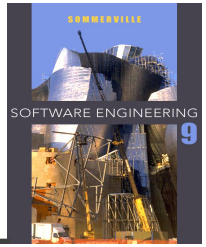
- As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required.
- Calendar time can be estimated using a COCOMO 2 formula
  - $TDEV = 3 \times (PM)^{(0.33+0.2*(B-1.01))}$
  - PM is the effort computation and B is the exponent computed as discussed above (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project.
- The time required is independent of the number of people working on the project.

Staffing requirements

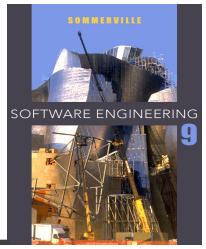


- Staff required can’t be computed by diving the development time by the required schedule.
- The number of people working on a project varies depending on the phase of the project.
- The more people who work on the project, the more total effort is usually required.
- A very rapid build-up of people often correlates with schedule slippage.

Key points



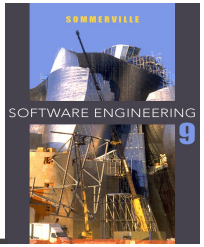
- Estimation techniques for software may be experience-based, where managers judge the effort required, or algorithmic, where the effort required is computed from other estimated project parameters.
- The COCOMO II costing model is an algorithmic cost model that uses project, product, hardware and personnel attributes as well as product size and complexity attributes to derive a cost estimate.



Chapter 24 - Quality Management

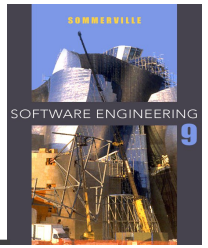
Lecture 1

Topics covered



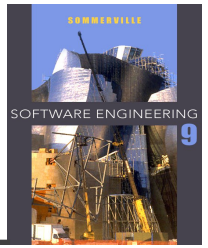
- Software quality
- Software standards
- Reviews and inspections
- Software measurement and metrics

Software quality management



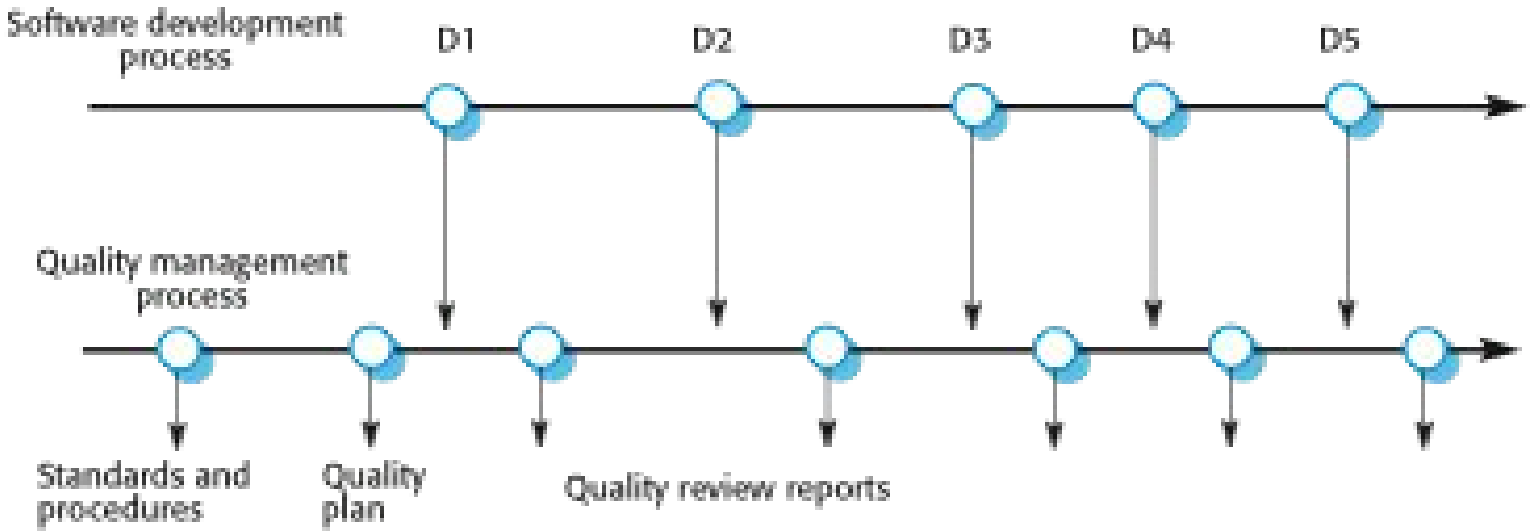
- Concerned with ensuring that the required level of quality is achieved in a software product.
- Three principal concerns:
  - At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.

Quality management activities

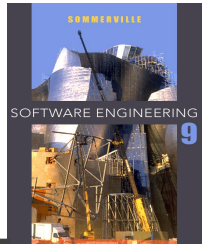


- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals
- The quality team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

Quality management and software development

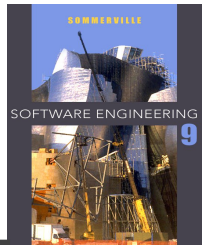


Quality planning



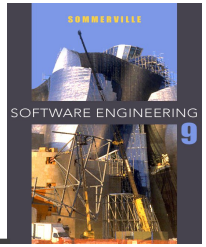
- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.

Quality plans



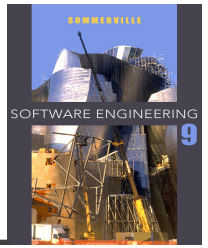
- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them.

Scope of quality management



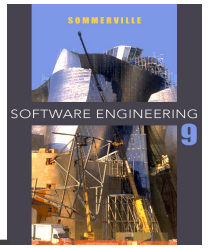
- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Software quality



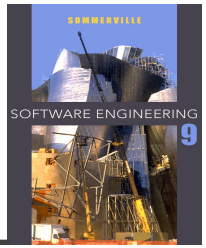
- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- The focus may be ‘fitness for purpose’ rather than specification conformance.

Software fitness for purpose



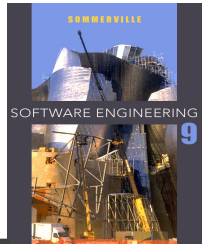
- Have programming and documentation standards been followed in the development process?
- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?

Software quality attributes



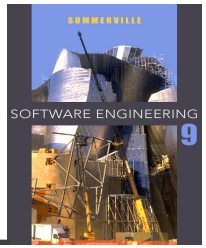
2	Safety	2	Understandability	2	Portability
2	Security	2	Testability	2	Usability
2	Reliability	2	Adaptability	2	Reusability
2	Resilience	2	Modularity	2	Efficiency
2	Robustness	2	Complexity	2	Learnability

Quality conflicts



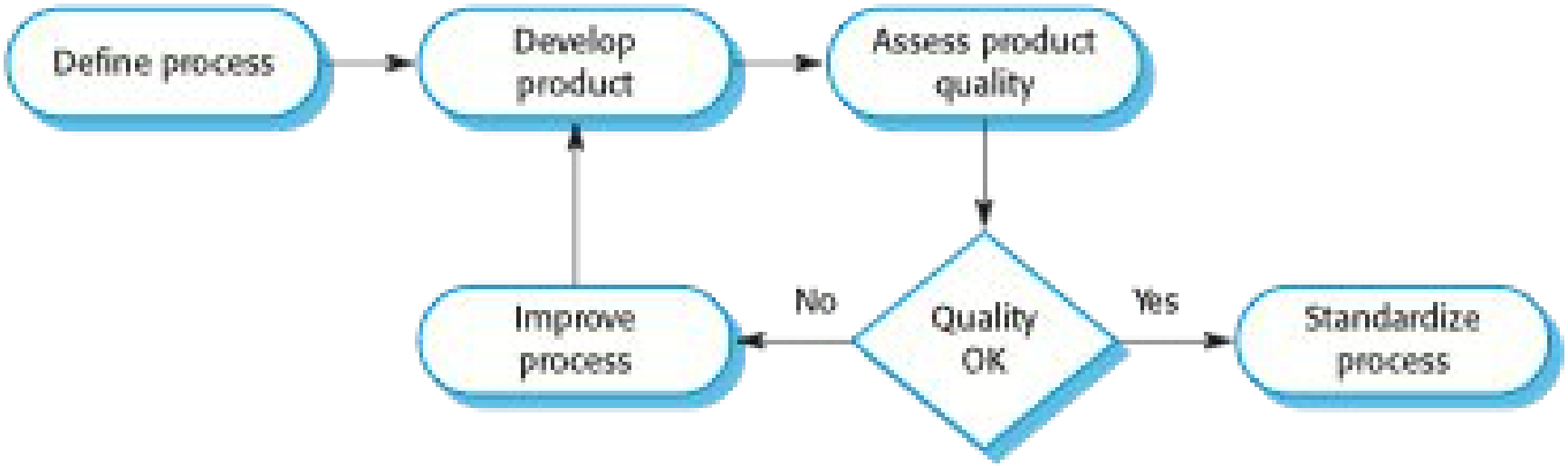
- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

Process and product quality

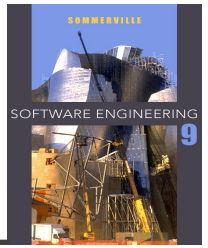


- The quality of a developed product is influenced by the quality of the production process.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

Process-based quality

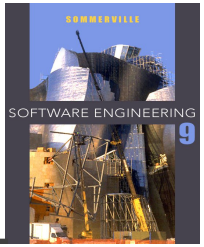


Software standards



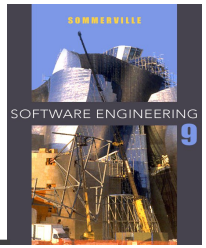
- Standards define the required attributes of a product or process. They play an important role in quality management.
- Standards may be international, national, organizational or project standards.
- Product standards define characteristics that all software components should exhibit e.g. a common programming style.
- Process standards define how the software process should be enacted.

Importance of standards



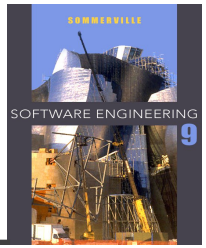
- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.

Product and process standards



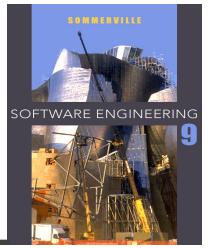
Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards



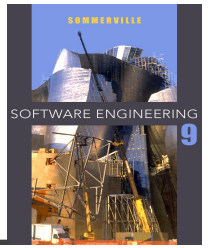
- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.

Standards development



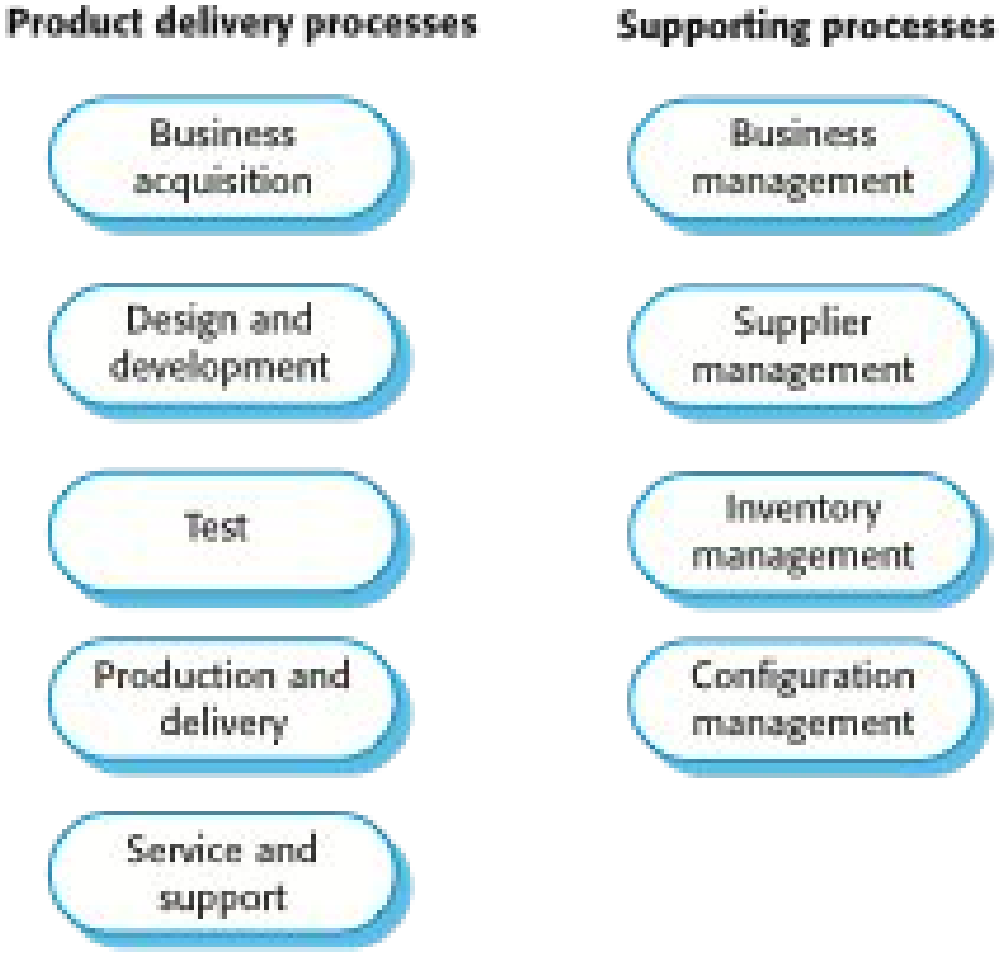
- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have specialized tool support. Excessive clerical work is the most significant complaint against standards.
  - Web-based forms are not good enough.

ISO 9001 standards framework

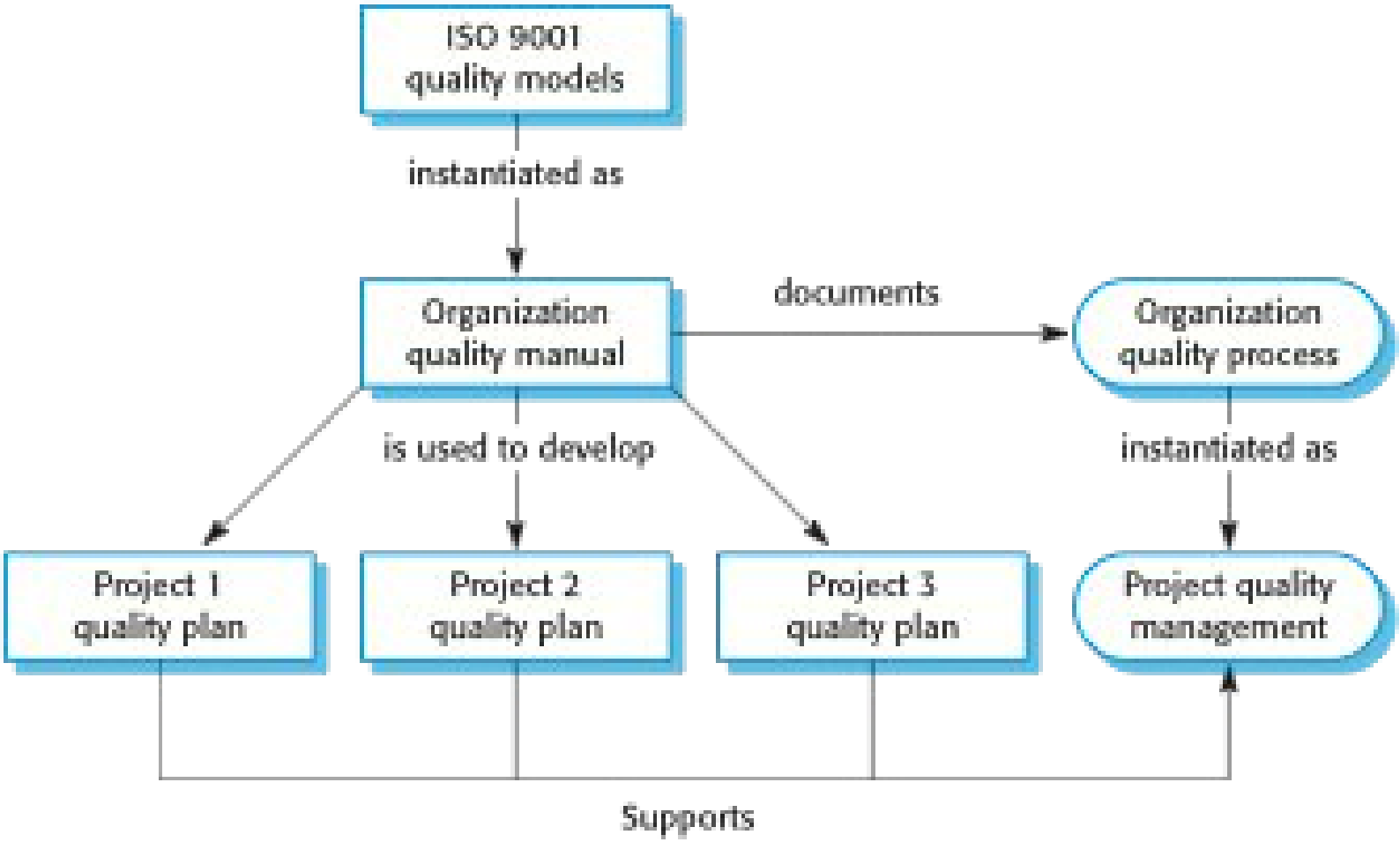


- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
  - It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.

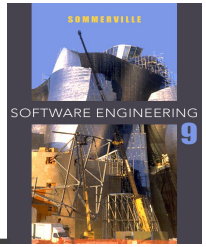
ISO 9001 core processes



ISO 9001 and quality management

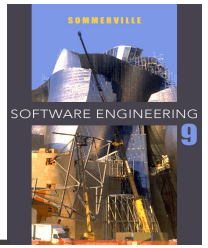


ISO 9001 certification

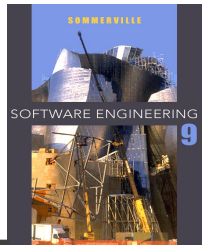


- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.

Key points



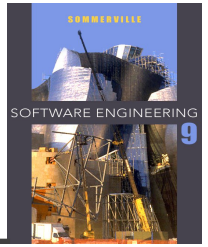
- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability and so on.
- SQM includes defining standards for processes and products and establishing processes to check that these standards have been followed.
- Software standards are important for quality assurance as they represent an identification of ‘best practice’.
- Quality management procedures may be documented in an organizational quality manual, based on the generic model for a quality manual suggested in the ISO 9001 standard.



Chapter 24 - Quality Management

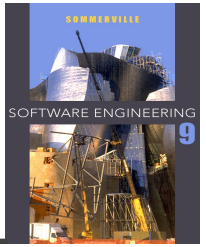
Lecture 2

Reviews and inspections



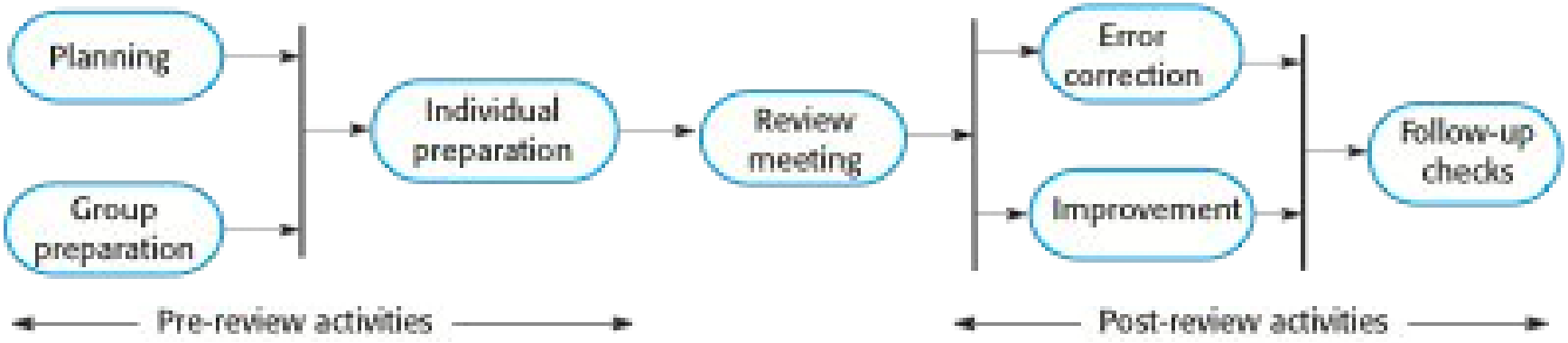
- A group examines part or all of a process or system and its documentation to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

Quality reviews

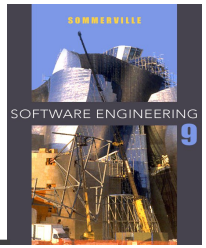


- A group of people carefully examine part or all of a software system and its associated documentation.
- Code, designs, specifications, test plans, standards, etc. can all be reviewed.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

The software review process

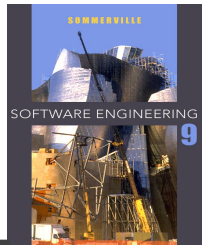


Reviews and agile methods



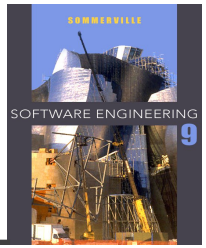
- The review process in agile software development is usually informal.
  - In Scrum, for example, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member.
- XP relies on individuals taking the initiative to improve and refactor code. Agile approaches are not usually standards-driven, so issues of standards compliance are not usually considered.

Program inspections



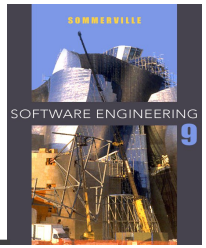
- These are peer reviews where engineers examine the source of a system with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design,configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

Inspection checklists



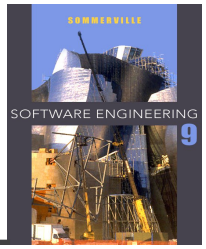
- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

An inspection checklist (a)



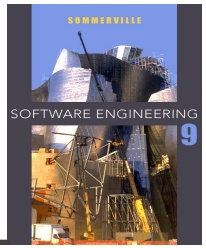
[2]Fault class	[2]Inspection check
[2]Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
[2]Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
[2]Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output?

An inspection checklist (b)



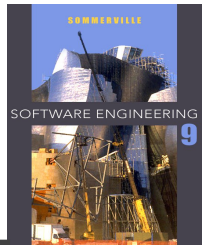
[2]Fault class	[2]Inspection check
[2]Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
[2]Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
[2]Exception management faults	• Have all possible error conditions been taken into account?

Agile methods and inspections



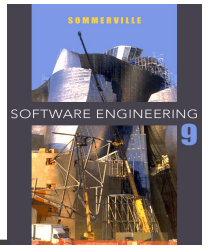
- Agile processes rarely use formal inspection or peer review processes.
- Rather, they rely on team members cooperating to check each other's code, and informal guidelines, such as 'check before check-in', which suggest that programmers should check their own code.
- Extreme programming practitioners argue that pair programming is an effective substitute for inspection as this is, in effect, a continual inspection process.

Software measurement and metrics



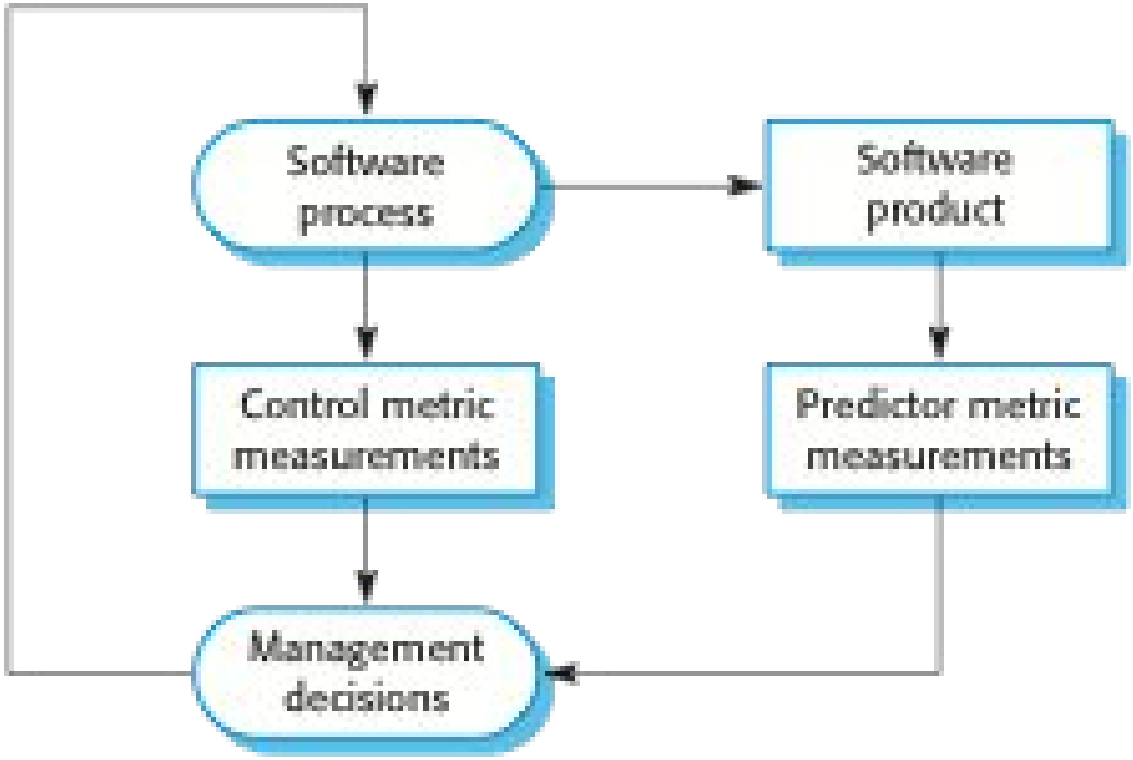
- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don't make systematic use of software measurement.
- There are few established standards in this area.

Software metric

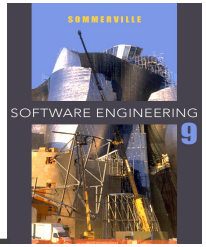


- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Predictor and control measurements

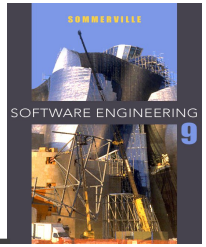


Use of measurements



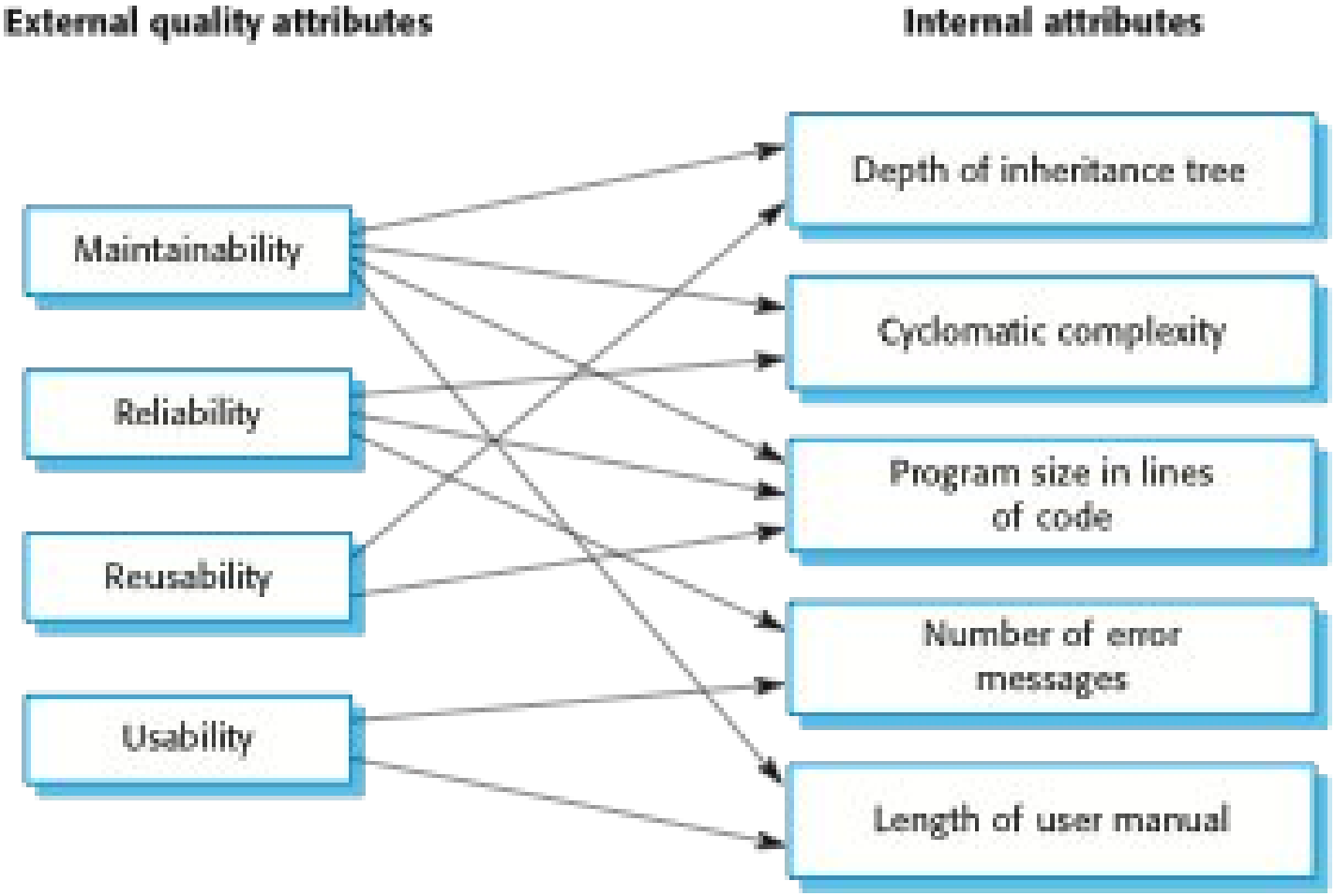
- To assign a value to system quality attributes
  - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
- To identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.

Metrics assumptions

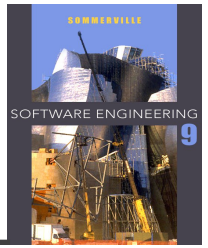


- A software property can be measured.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Relationships between internal and external software

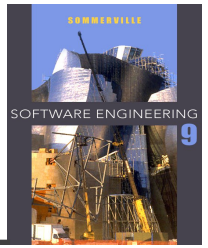


Problems with measurement in industry



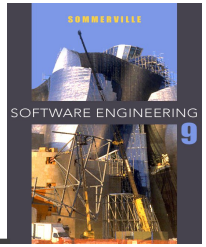
- It is impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.

Product metrics



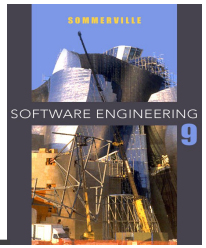
- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.

Dynamic and static metrics



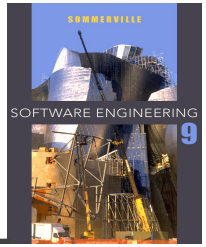
- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Static software product metrics



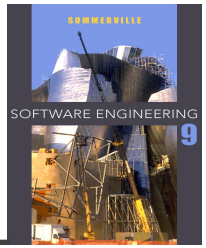
[2]Software metric	[2]Description
[2]Fan-in/Fan-out	[2]Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
[2]Length of code	[2]This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.

Static software product metrics



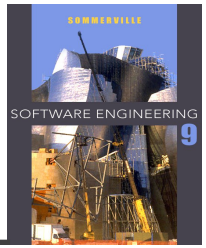
[2]Software metric	[2]Description
[2]Cyclomatic complexity	[2]This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
[2]Length of identifiers	[2]This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
[2]Depth of conditional nesting	[2]This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
[2]Fog index	[2]This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite



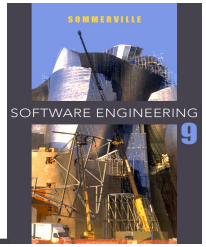
[2]Object-oriented metric	[2]Description
[2]Weighted methods per class (WMC)	[2]This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
[2]Depth of inheritance tree (DIT)	[2]This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
[2]Number of children (NOC)	[2]This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

The CK object-oriented metrics suite



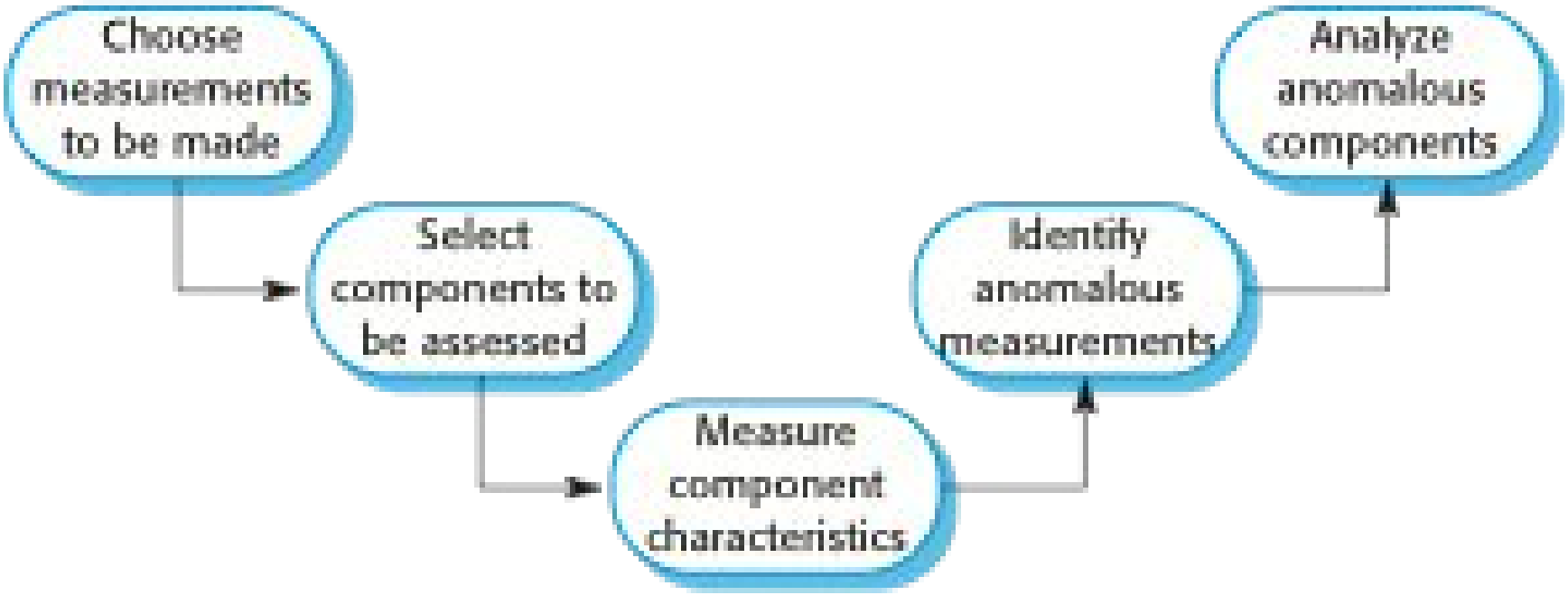
[2]Object-oriented metric	[2]Description
[2]Coupling between object classes (CBO)	[2]Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
[2]Response for a class (RFC)	[2]RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
[2]Lack of cohesion in methods (LCOM)	[2]LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics.

Software component analysis

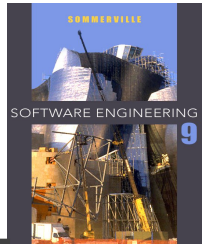


- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.

The process of product measurement

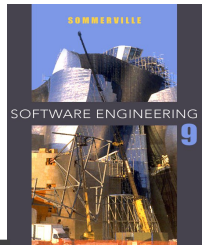


Measurement surprises



- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Key points



- Reviews of the software process deliverables involve a team of people who check that quality standards are being followed.
- In a program inspection or peer review, a small team systematically checks the code. They read the code in detail and look for possible errors and omissions
- Software measurement can be used to gather data about software and software processes.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems.