ICT-5405

PROJECT MANAGEMENT AND QUALITY ASSURANCE

03 PROJECT PLANNING

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

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

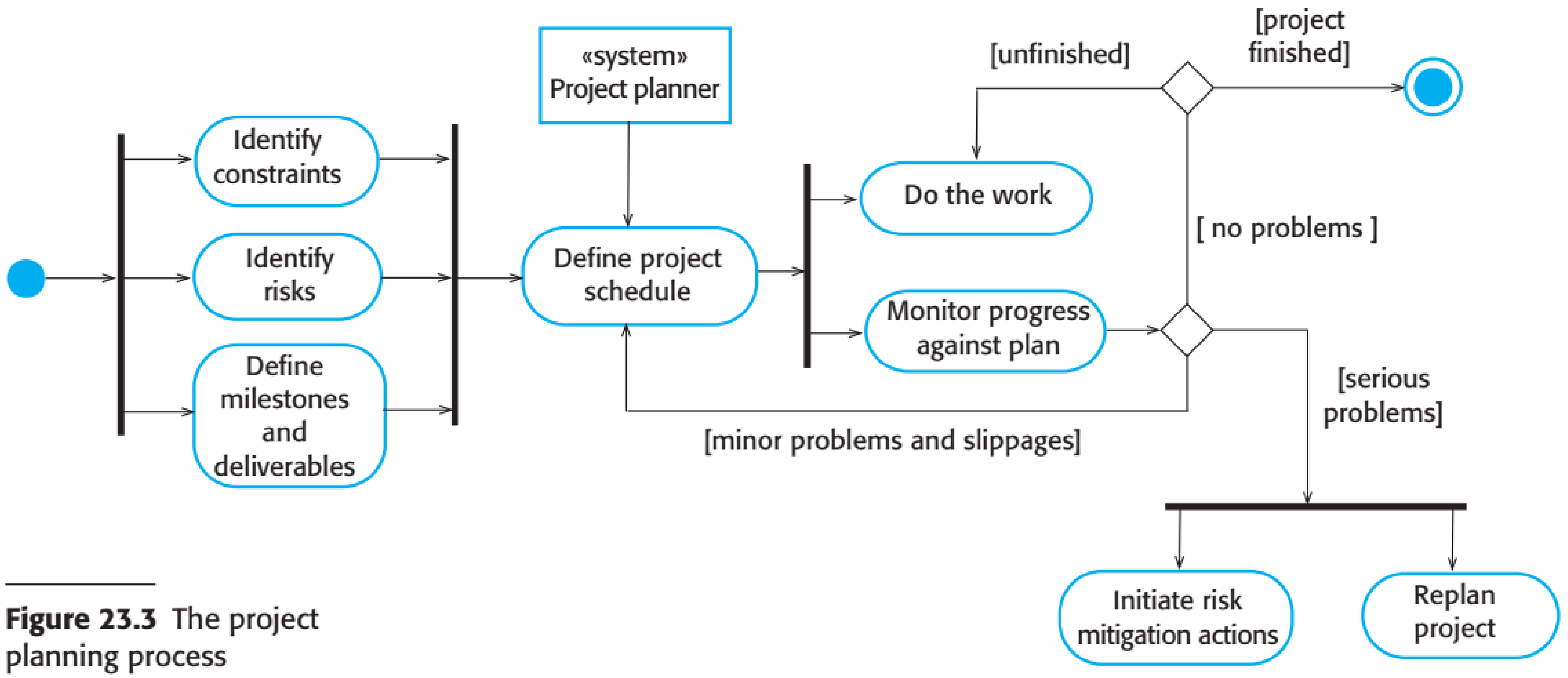


Figure 23.3 The project planning process

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.

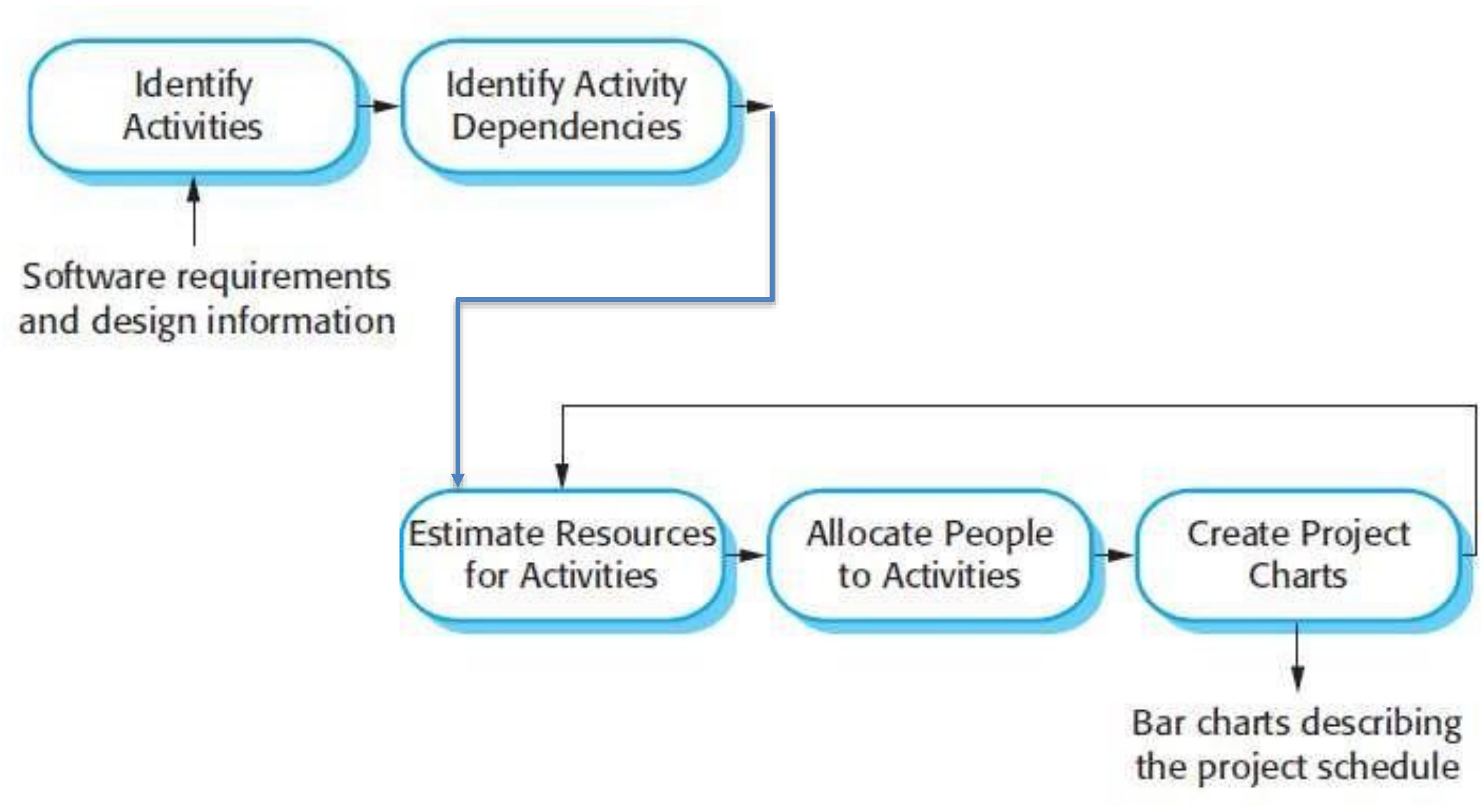
Project scheduling activities

- ✧ 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



Identify Activities
Identify Activity Dependencies
Software requirements and design information
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Bar charts describing the project schedule

Scheduling problems

- ✧ Assessing the complexity of problems and hence determining the expense of producing a solution is challenging.
- ✧ Productivity is not proportional to the number of people working on a task.
- ✧ The inclusion of additional individuals in a project that is already behind schedule results in further delays due to the increased burden of communication.
- ✧ The unexpected always happens. Always allow contingency in planning.

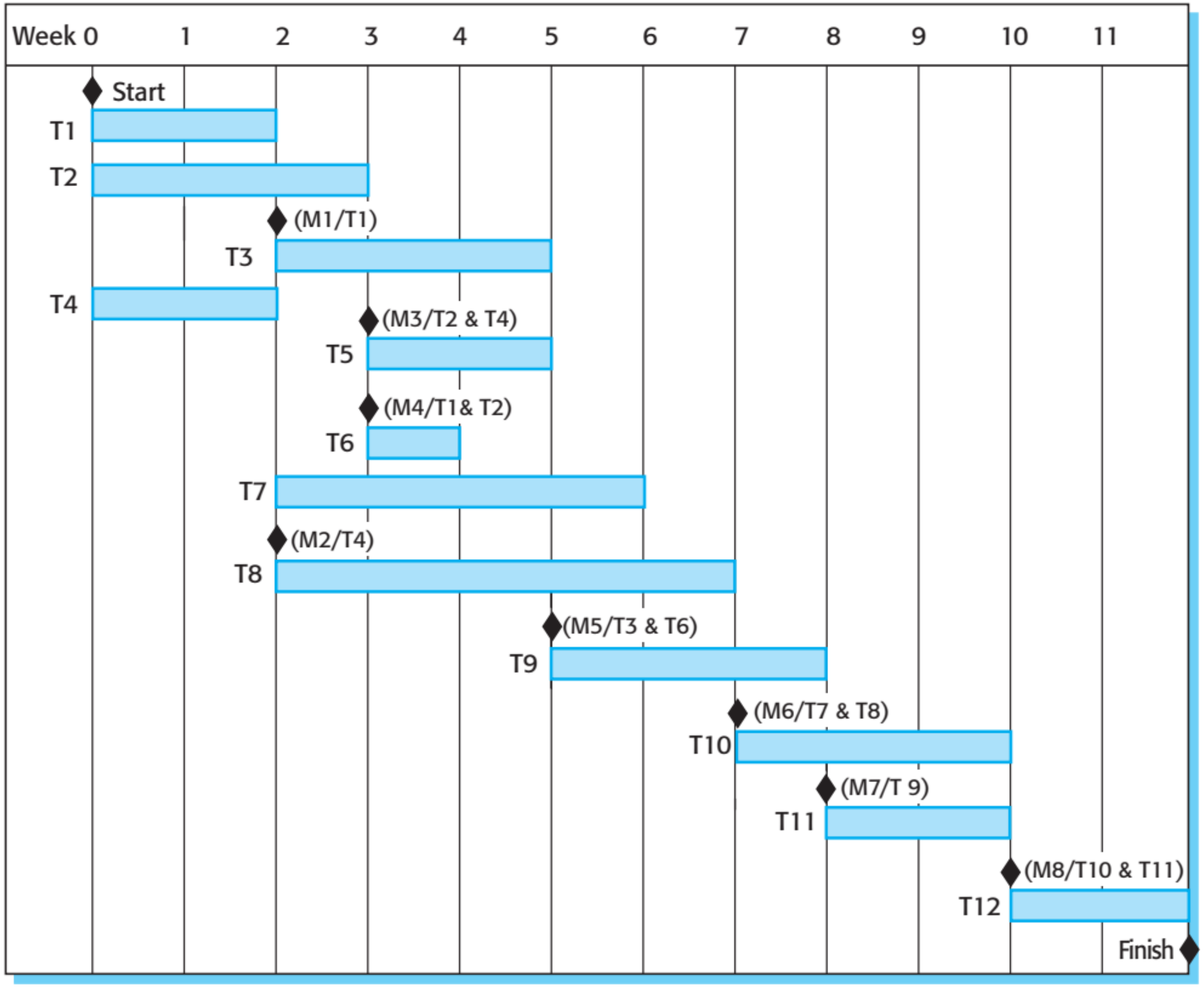
Schedule representation

- 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.

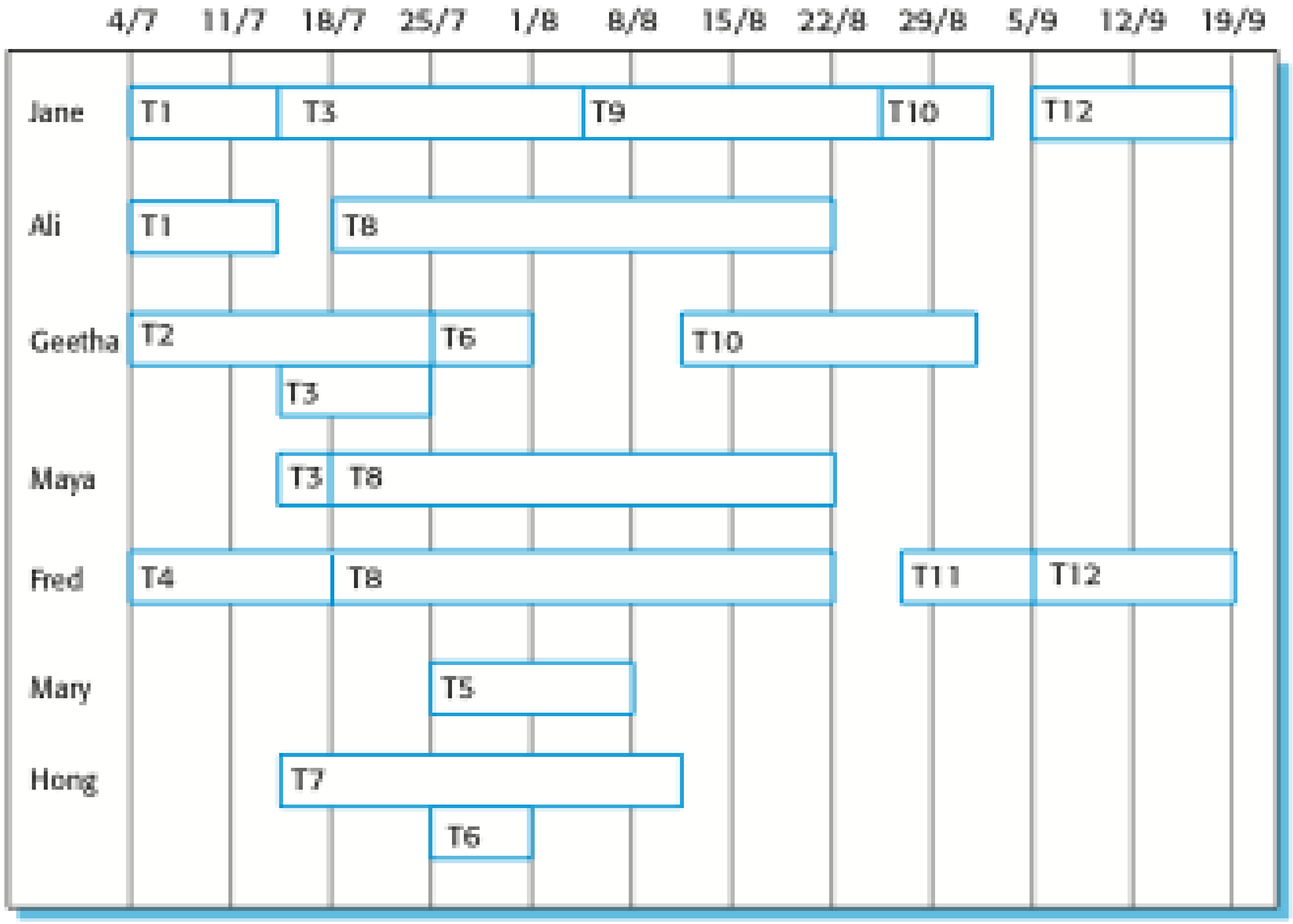
Tasks, durations, and dependencies

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



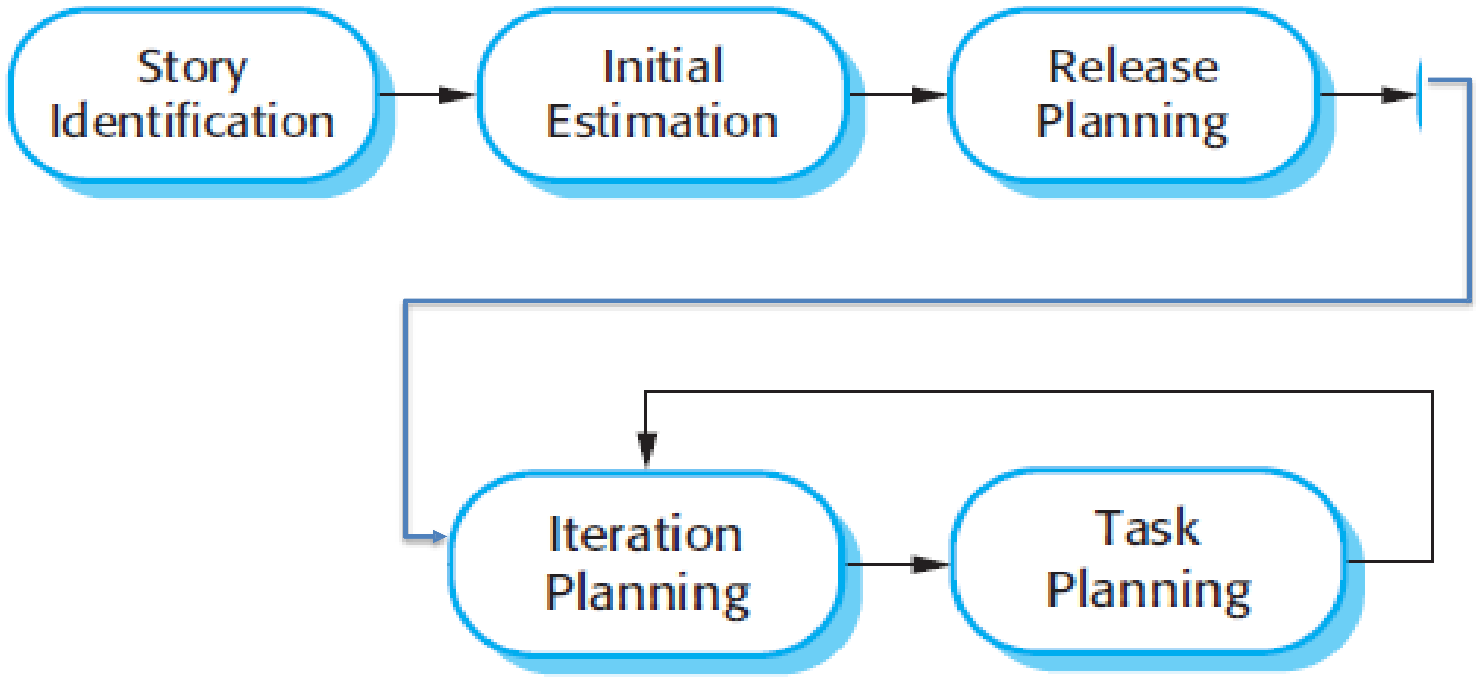
Agile planning

- ✧ 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).

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.

Effort estimation is a process in which project managers evaluate how much time and money they need for completing a project

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

- Algorithmic cost modeling uses a mathematical formula that predict project costs based on estimates of the project size, the type of software being developed, and other team, process, and product factors.
- Most algorithmic models for estimating effort in a software project are based on a simple formula:

$\text{Effort} = A \times \text{Size}^B \times M$

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

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

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

M: is a factor that takes into account process, product and development attributes, such as the dependability requirements for the software and the experience of the development team. These attributes may increase or decrease the overall difficulty of developing the system.

Algorithmic cost modelling Challenges

- Complexity Hinders Widespread Use: Algorithmic cost models offer a structured approach to estimating development efforts, yet their complex nature and numerous variables pose challenges. This complexity restricts their practical application, mostly confined to larger firms in sectors like defense and aerospace engineering.

- Uncertainty and Attribute Variation: Estimating attributes within these models involves multiple factors, leading to considerable uncertainty. Attributes' values are challenging to ascertain, limiting the broader adoption of these models across industries.

- Calibration and Data Limitations: Model calibration demands historical project data, aligning with an organization's practices. However, insufficient data collection practices in many companies hinder calibration efforts, relying instead on published parameter values that may not reflect the organization's specifics.

- Estimation Approach: To mitigate inaccuracies, using algorithmic models necessitates creating a range of estimates (worst, expected, best) rather than a single prediction. Accurate estimates often result from a deep understanding of the software type being developed and calibrated models based on local data or predetermined language and hardware choices

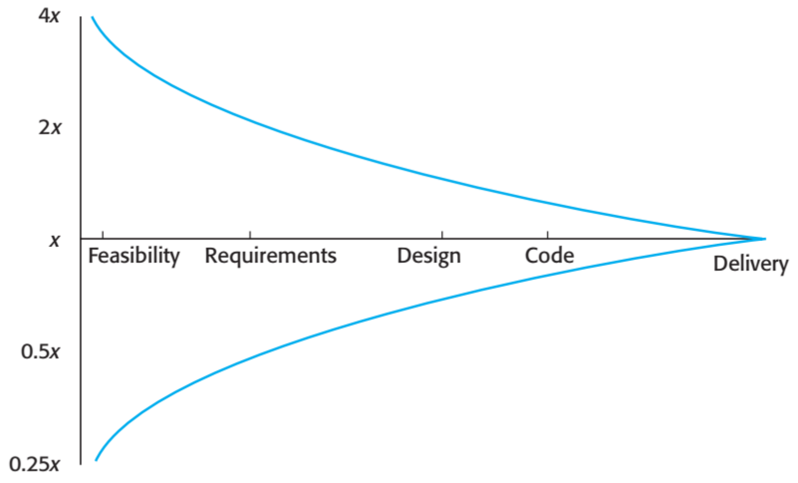
Estimation accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of COTS* (**Commercial off-the-shelf**) 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.

* It refers to hardware or software that is ready-made and available for purchase

Estimate uncertainty

- Based on data collected from a large number of projects, Boehm et al. (**B. Boehm et al. 1995**) discovered that startup estimates vary significantly.
- If the initial estimate of effort required is **x** months of effort, they found that the range may be from **0.25x to 4x** of the actual effort as measured when the system was delivered. During development planning, estimates become more and more accurate as the project progresses.



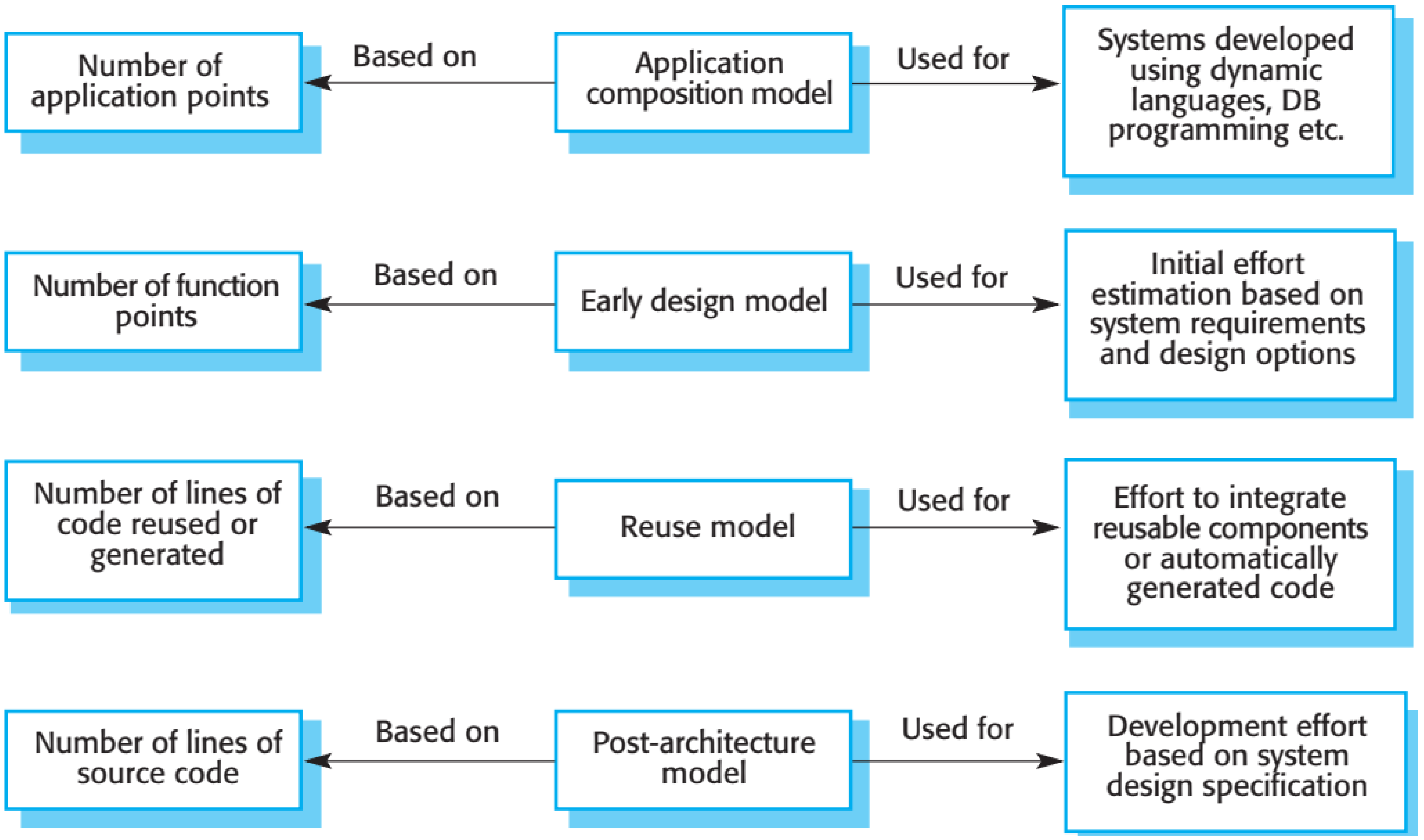
The COCOMO 2 model

- 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. Once the system architecture has been designed, a more accurate estimate of the software size can be made. Again, this model uses the standard formula for cost estimation discussed above. However, it includes a more extensive set of 17 multipliers reflecting personnel capability, product, and project characteristics.

COCOMO estimation models


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

Application composition model

- ✧ The application composition model was introduced into COCOMO II to support the estimation of effort required for prototyping projects and for projects where the software is developed by composing existing components.
- ✧ Based on standard estimates of developer productivity in application (object) points/month.
- ✧ Takes CASE tool use into account.
- ✧ Formula is : PM =(NAP X (1 - %reuse/100)) / PROD

✓ **PM: the effort estimate in person-months.**

✓ **NAP: the total number of application points in the delivered system.**

✓ **%reuse: an estimate of the amount of reused code in the development.**

✓ **PROD: the application-point 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

- This model may be used during the early stages of a project, before a detailed architectural design for the system is available.
- The early design model assumes that user requirements have been agreed and initial stages of the system design process are underway. Your goal at this stage should be to make a quick and approximate cost estimate.
- Based on a standard formula for algorithmic models
  - Effort = 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.
  1. RCPX - product reliability and complexity;
  2. RUSE - the reuse required;
  3. PDIF - platform difficulty;
  4. PREX - personnel experience;
  5. PERS - personnel capability;
  6. SCED - required schedule;
  7. FCIL - the team support facilities.

  The multiplier **M** is based on seven project and process attributes that increase or decrease the estimate. You estimate values for these attributes using a six-point scale, where 1 corresponds to “very low” and 6 corresponds to “very high”; for example, PERS = 6 means that expert staff are available to work on the project.

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** code is code that can be reused without understanding the code or making changes to it. Examples of black-box code are components that are automatically generated from UML models or application libraries such as graphics libraries. It is assumed that the development effort for black- box code is zero. Its size is not taken into account in the overall effort computation.
  - **White-box** code is reusable code that has to be adapted to integrate it with new code or other reused components. *Development effort is required* for reuse because the code has to be understood and modified before it can work correctly in the system.

The reuse model

Three factors contribute to the effort involved in reusing white-box code components:

1. The effort involved in assessing whether or not a component could be reused in a system that is being developed.
2. The effort required to understand the code that is being reused.
3. The effort required to modify the reused code to adapt it and integrate it with the system being developed.

Reuse model estimates

- The development effort in the reuse model is calculated using the COCOMO early design model and is based on the total number of lines of code in the system
- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - **ESLOC:** the equivalent number of lines of new source code.
  - **ASLOC**: an estimate of the number of lines of code in the reused components that have to be changed.
  - **AT**: the percentage of reused code that can be modified automatically
  - **AAM**: an Adaptation Adjustment Multiplier that reflects the additional effort required to reuse components.

Reuse model estimates

Once you have calculated a value for ESLOC, you apply the standard estimation formula to calculate the total effort required, where the Size parameter = ESLOC.

Therefore, the formula to estimate the reuse effort is:

Effort =A x $ESLOC^B$ x M

where A, B, and M have the same values as used in the early design model.

Post-architecture level

- The post-architecture model is the most detailed of the COCOMO II models. It is used when you have an initial architectural design for the system. The starting point for estimates produced at the post-architecture level is the same basic formula used in the early design estimates:
- **$PM = A \times Size^{B} \times M$**
- By this stage in the process, you should be able to make a more accurate estimate of the project size, as you know how the system will be decomposed into subsystems and components.
- ***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 (B)

- ✧ The exponent term (B) in the effort computation formula is related to the levels of project complexity. As projects become more complex, the effects of increasing system size become more significant.
- ✧ The value of the exponent B is based on five factors, as shown in next slide.

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(Capability Maturity Model) Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

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
  - $TDEV = 3 \times (PM)^{(0.33+0.2*(B-1.01))}$
  - **TDEV**: the nominal schedule for the project, in calendar months, ignoring any multiplier that is related to the project schedule.
  - **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 that means “An accelerated increase in project personnel has been found to be associated with delays in the project timeline”.