includes static validation techniques.

Goals of software testing:

- To **demonstrate** to the developer and the customer that the **software meets its requirements**.
  - Leads to **validation testing**: you expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
  - A successful test shows that the system operates as intended.
- To **discover** situations in which the behavior of the software is **incorrect, undesirable or does not conform to its specification**.
  - Leads to **defect testing**: the test cases are designed to expose defects; the test cases can be deliberately obscure and need not reflect how the system is normally used.
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

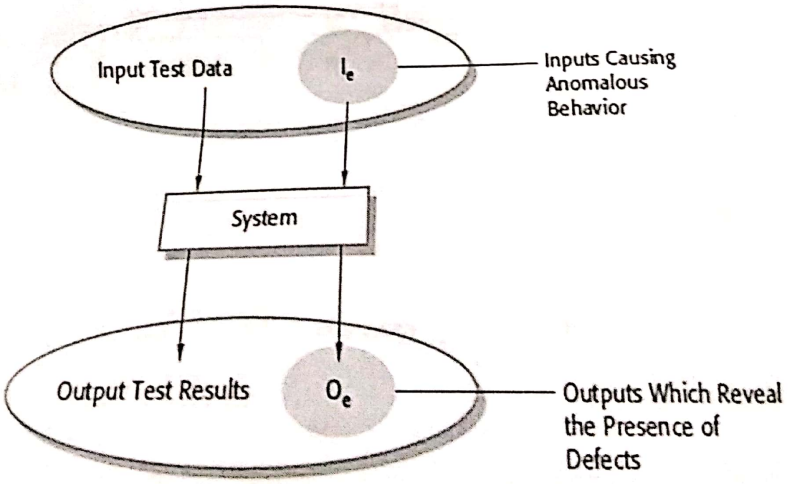

Fig: An input-output model of program testing

Verification and validation

Testing is part of a broader process of software **verification and validation** (V & V).

1. **Verification: Are we building the product right?**
   The software should conform to its specification.
2. **Validation: Are we building the right product?**
   The software should do what the user really requires.

The **goal of V & V** is to establish confidence that the system is **good enough for its intended use**, which depends on:

1. **Software purpose**: the level of confidence depends on how critical the software is to an organization.
2. **User expectations**: users may have low expectations of certain kinds of software.
3. **Marketing environment**: getting a product to market early may be more important than finding defects in the program.

Inspections and Testing

- This accelerates the process of software ageing so that future changes become progressively more difficult and maintenance costs increase

Program Evolution Dynamics

- Program evolution dynamics is the study of system change.
- Lehman and Belady carried out several empirical studies of system change with a view to understanding more about characteristics of software evolution
- Lehman and Belady claim these laws are likely to be true for all types of large organizational software systems (what they call E-type systems).
- These are systems in which the requirements are changing to reflect changing business needs.
- New releases of the system are essential for the system to provide business value

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multi agent, multi loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Fig: Lehman's laws

- The **first law** states that system maintenance is an inevitable process.
- As the system's environment changes, new requirements emerge and the system must be modified.
- When the modified system is reintroduced to the environment, this promotes more environmental changes, so the evolution process starts again
- The **second law** states that, as a system is changed, its structure is degraded.
- The only way to avoid this happening is to invest in preventative maintenance.
- You spend time improving the software structure without adding to its functionality.
- Obviously, this means additional costs, over and above those of implementing required system changes
- The **third law** suggests that large systems have a dynamic of their own that is established at an early stage in the development process.
- This determines the gross trends of the system maintenance process and limits the number of possible system changes
- Lehman's **fourth law** suggests that most large programming projects work in a 'saturated' state.
- That is, a change to resources or staffing has imperceptible effects on the long-term evolution of the system.
- Lehman's **fifth law** is concerned with the change increments in each system release.
- Adding new functionality to a system inevitably introduces new system faults.
- The more functionality added in each release, the more faults there will be.
- Therefore, a large increment in functionality in one system release means that this will have to be followed by a further release in which the new system faults are repaired.
- Relatively little new functionality should be included in this release. This law suggests that you should not budget for large functionality increments in each release without taking into account the need for fault repair.
- The **sixth and seventh laws** are similar and essentially say that users of software will become increasingly unhappy with it unless it is maintained and new functionality is added to it.

- The **final law** reflects the most recent work on feedback processes, although it is not yet clear how this can be applied in practical software development

Software Maintenance

Software maintenance focuses on **modifying a program after it has been put into use**. The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions. Maintenance does not normally involve major changes to the system's architecture. Changes are implemented by modifying existing components and adding new components to the system.

Types of software maintenance include:

1. **Fault repairs:** Coding errors are usually relatively cheap to correct; design errors are more expensive as they may involve rewriting several program components. Requirements errors are the most expensive to repair because of the extensive system redesign which may be necessary.
2. **Environmental adaptation:** This type of maintenance is required when some aspect of the system's environment such as the hardware, the platform operating system, or other support software changes. The application system must be modified to adapt it to cope with these environmental changes.
3. **Functionality addition:** This type of maintenance is necessary when the system requirements change in response to organizational or business change. The scale of the changes required to the software is often much greater than for the other types of maintenance

- In practice, there is not a clear-cut distinction between these types of maintenance.
- When you adapt the system to a new environment, you may add functionality to take advantage of new environmental features.
- Software faults are often exposed because users use the system in unanticipated ways.
- Changing the system to accommodate their way of working is the best way to fix these faults
- Different people sometimes give them different names
- **'Corrective maintenance'** is universally used to refer to maintenance for fault repair.
- **'Adaptive maintenance'** sometimes means adapting to a new environment and sometimes means adapting the software to new requirements.
- **'Perfective maintenance'** sometimes means perfecting the software by implementing new requirements; in other cases it means maintaining the functionality of the system but improving its structure and its performance
- The surveys broadly agree that software maintenance takes up a higher proportion of IT budgets than new development (roughly two-thirds maintenance, one-third development).
- They also agree that more of the maintenance budget is spent on implementing new requirements than on fixing bugs

Software Pricing

- In principle, the price of a software product to a customer is simply the cost of development plus profit for the developer.
- In practice, however, the relationship between the project cost and the price quoted to the customer is not usually so simple.
- When calculating a price, you should take broader organizational, economic, political, and business considerations into account, such as those shown in the next figure.

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.
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.

Fig: Factors affecting Software pricing

- You need to think about organizational concerns, the risks associated with the project, and the type of contract that will be used.
- These may cause the price to be adjusted upwards or downwards.
- Because of the organizational considerations involved, deciding on a project price should be a group activity involving marketing and sales staff, senior management, and project managers.

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 replanned
- The following is a UML activity diagram that shows a typical workflow for a project planning process

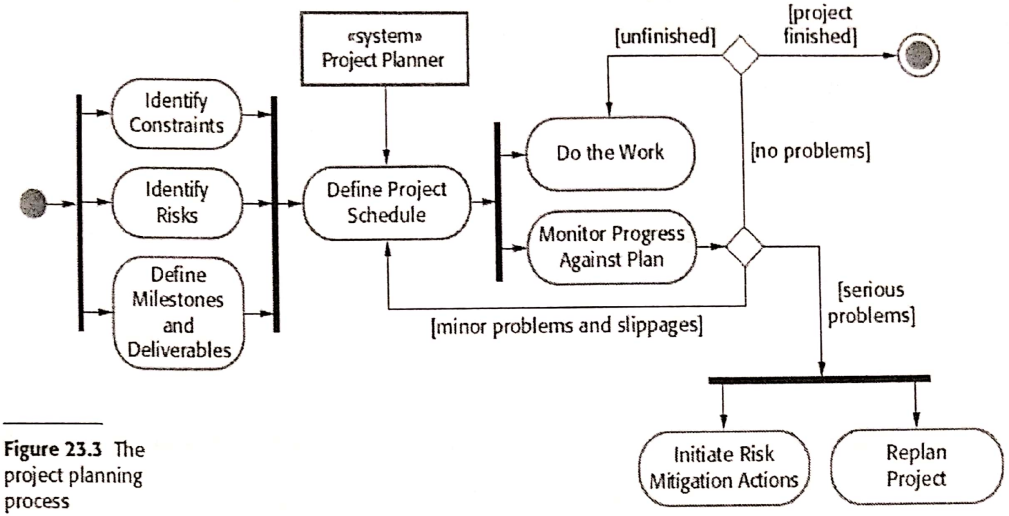

Figure 23.3 The project planning process

Fig: the project planning process

- At the beginning of a planning process, you should assess the constraints affecting the project.
- The required delivery date, staff available, overall budget, available tools, and so on
- You should also identify the project milestones and deliverables.
- Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- Deliverables are work products that are delivered to the customer (e.g., a requirements document for the system).
- The process then enters a loop.
- You draw up an estimated schedule for the project and the activities defined in the schedule are initiated or given permission to continue.
- After some time (usually about two to three weeks), you should review progress and note discrepancies from the planned schedule
- It is important to be realistic when you are creating a project plan

Project scheduling

Fig: staff allocation chart

Estimation techniques

- Project schedule estimation is difficult
- You may have to make initial estimates on the basis of a high-level user requirements definition.
- The software may have to run on unfamiliar computers or use new development technology.
- The people involved in the project and their skills will probably not be known
- There are so many uncertainties that it is impossible to estimate system development costs accurately during the early stages of a project.
- **There are two types of technique that can be used to do this:**

1. ***Experience-based techniques*:**The estimate of future effort requirements is based on the manager's experience of past projects and the application domain
2. ***Algorithmic cost modeling*:**In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

Algorithmic cost modeling

- Algorithmic cost modeling uses a mathematical formula to predict project costs based on estimates of the project size; the type of software being developed; and other team, process, and product factors.
- An algorithmic cost model can be built by analyzing the costs and attributes of completed projects, and finding the closest-fit formula to actual experience.
- Algorithmic models for estimating effort in a software project are mostly based on a simple formula:
- **Effort = A x Size^{B} x M**
- A is a constant factor which depends on local organizational practices and the type of software that is developed
- Size may be either an assessment of the code size of the software or a functionality estimate expressed in function or application points
- The value of exponent B usually lies between 1 and 1.5.
- M is a multiplier made by combining process, product, and development attributes, such as the dependability requirements for the software and the experience of the development team.
- **All algorithmic models have similar problems:**
    1. It is often difficult to estimate Size at an early stage in a project, when only the specification is available
    2. The estimates of the factors contributing to B and M are subjective. Estimates vary from one person to another, depending on their background and experience of the type of system that is being developed
- Accurate code size estimation is difficult at an early stage in a project because the size of the final program depends on design decisions that may not have been made when the estimate is required
- The programming language used for system development also affects the number of lines of code to be developed. A language like Java might mean that more lines of code are necessary than if C (say) was used
- Algorithmic cost models are a systematic way to estimate the effort required to develop a system.

Legacy system management

- There are still many legacy systems that are critical business systems.
- These have to be extended and adapted to changing e-business practices
- Most organizations usually have a portfolio of legacy systems that they use, with a limited budget for maintaining and upgrading these systems.
- They have to decide how to get the best return on their investment.
- This involves making a realistic assessment of their legacy systems and then deciding on the most appropriate strategy for evolving these systems
- There are four strategic options:

1. ***Scrap the system completely:*** *This option should be chosen when the system is* not making an effective contribution to business processes. This commonly occurs when business processes have changed since the system was installed and are no longer reliant on the legacy system
2. ***Leave the system unchanged and continue with regular maintenance:*** *This* option should be chosen when the system is still required but is fairly stable and the system users make relatively few change requests.
3. ***Reengineer the system to improve its maintainability*** *:This option should be* chosen when the system quality has been degraded by change and where a new change to the system is still being proposed
4. ***Replace all or part of the system with a new system*** *:This option should be chosen* when factors, such as new hardware, mean that the old system cannot continue in operation or where off-the-shelf systems would allow the new system to be developed at a reasonable cost

Organizations that rely on legacy systems must choose a strategy for evolving these systems. The chosen strategy should depend on the system quality and its business value:

- **Low quality, low business value**: should be scrapped.
- **Low-quality, high-business value**: make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- **High-quality, low-business value**: replace with COTS, scrap completely, or maintain.
- **High-quality, high business value**: continue in operation using normal system maintenance.

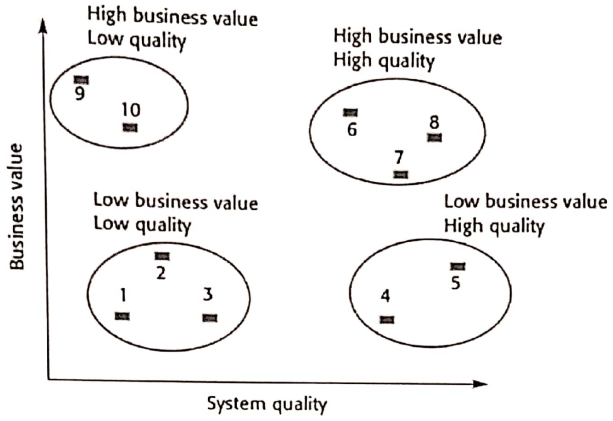

Fig: An example of a legacy system assessment

- To assess the business value of a system, you have to identify system stakeholders, such as end-users of the system and their managers, and ask a series of questions about the system.
- There are four basic issues that you have to discuss:
- ***The use of the system***
- If systems are only used occasionally or by a small number of people, they may have a low business value.
- You have to be careful, however, about occasional but important use of systems.
- For example, in a university, a student registration system may only be used at the beginning of each academic year. However, it is an essential system with a high business value
- ***The business processes that are supported***
- *When a system is introduced, business* processes are designed to exploit the system's capabilities. If the system is inflexible, changing these business processes may be impossible
- ***The system dependability***
- *System dependability is not only a technical problem* but also a business problem.

- If a system is not dependable and the problems directly affect the business customers or mean that people in the business are diverted from other tasks to solve these problems, the system has a low business value
- ***The system outputs***
- *The key issue here is the importance of the system outputs to* the successful functioning of the business.
- If the business depends on these outputs, then the system has a high business value.
- Conversely, if these outputs can be easily generated in some other way or if the system produces outputs that are rarely used, then its business value may be low