Unit 5

- Software testing
- Development testing
- Test-driven development
- Release testing
- User testing
- Dependability properties
- Availability and reliability
- Safety Security.

Prof. Rinku Sharma
Assistant Professor
COMPUTER DEPARTMENT

Software Testing

- Development testing
- Test-driven development
- Release testing
- User testing

Software Testing

- Software testing in software engineering is the process of verifying and validating that a software product meets its specified requirements, ensuring it is functional, reliable, secure, and performs as expected by identifying and fixing bugs.

Testing goals

Verification and Validation:
- Testing verifies that the software is built correctly (verification) and validates that it meets the user's needs (validation).

Bug/Defect Identification:
- The primary goal is to find errors, faults, and missing requirements so they can be corrected before release, saving time and money.

Quality Assurance:
- It helps ensure the final software product is high-quality, performing well, and reliable for users.

Improving Software:
- Testing provides objective information about the software's quality, helping to improve its accuracy, efficiency, and usability.

Validation and defect testing

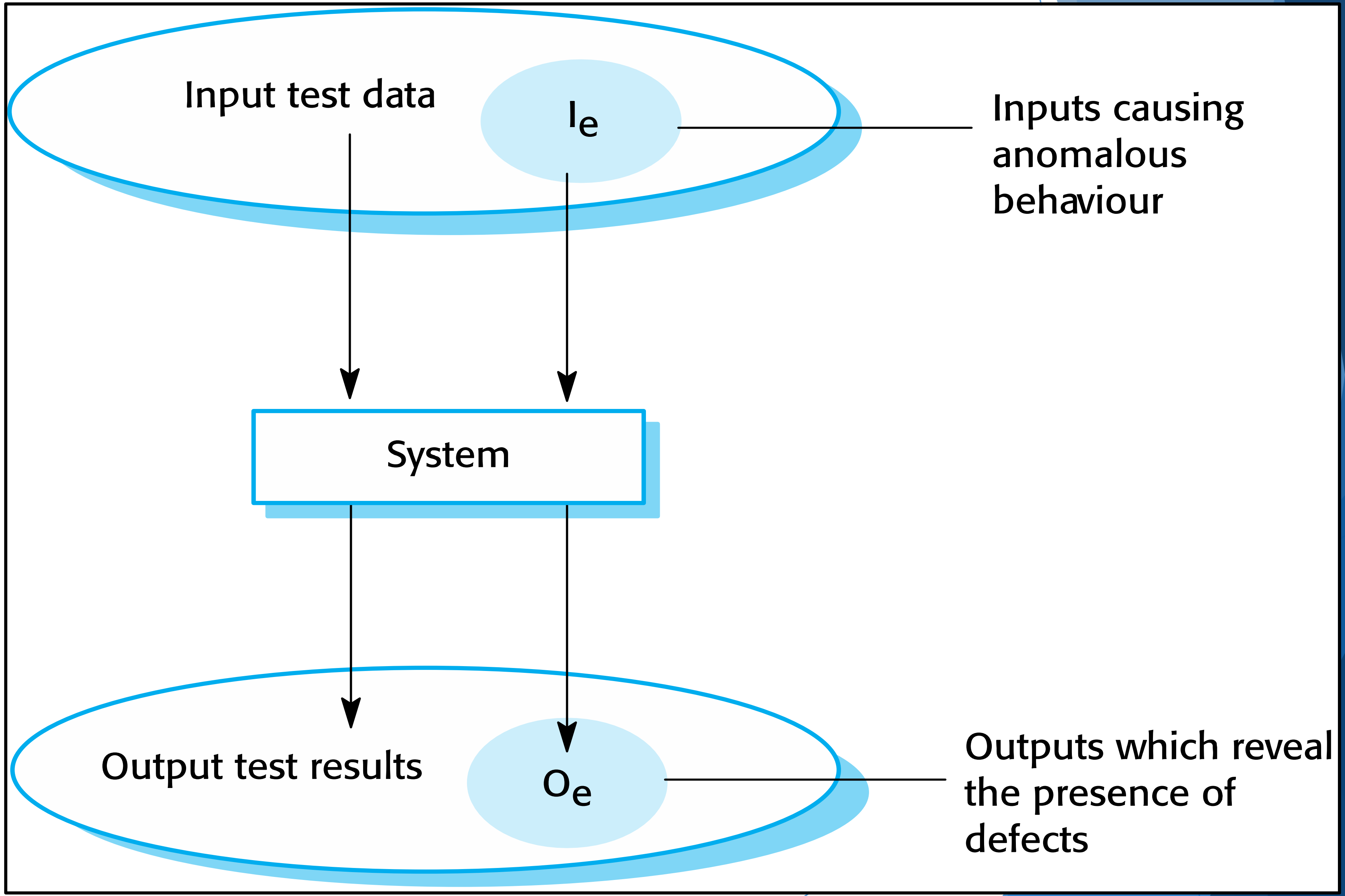
- **The first goal leads to**

  validation testing:
- You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- To demonstrate to the developer and the system customer that the software meets its requirements
- A successful test shows that the system operates as intended.

- The second goal leads to

  defect testing
  - The test cases are designed to expose defects. They can be deliberately obscure and need not reflect how the system is normally used.
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

An input-output model of program testing


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

Verification vs Validation

Verification:

- "**Are we building the product right**”.
- The software should conform to its specification.

Validation:

- "**Are we building the right product**”.
- The software should do what the user really requires.

V & V confidence : “Fit for purpose” ultimate Goal

- Depends on system’s purpose, user expectations and marketing environment

Software purpose

The level of confidence depends on how critical the software is to an organisation.

User expectations

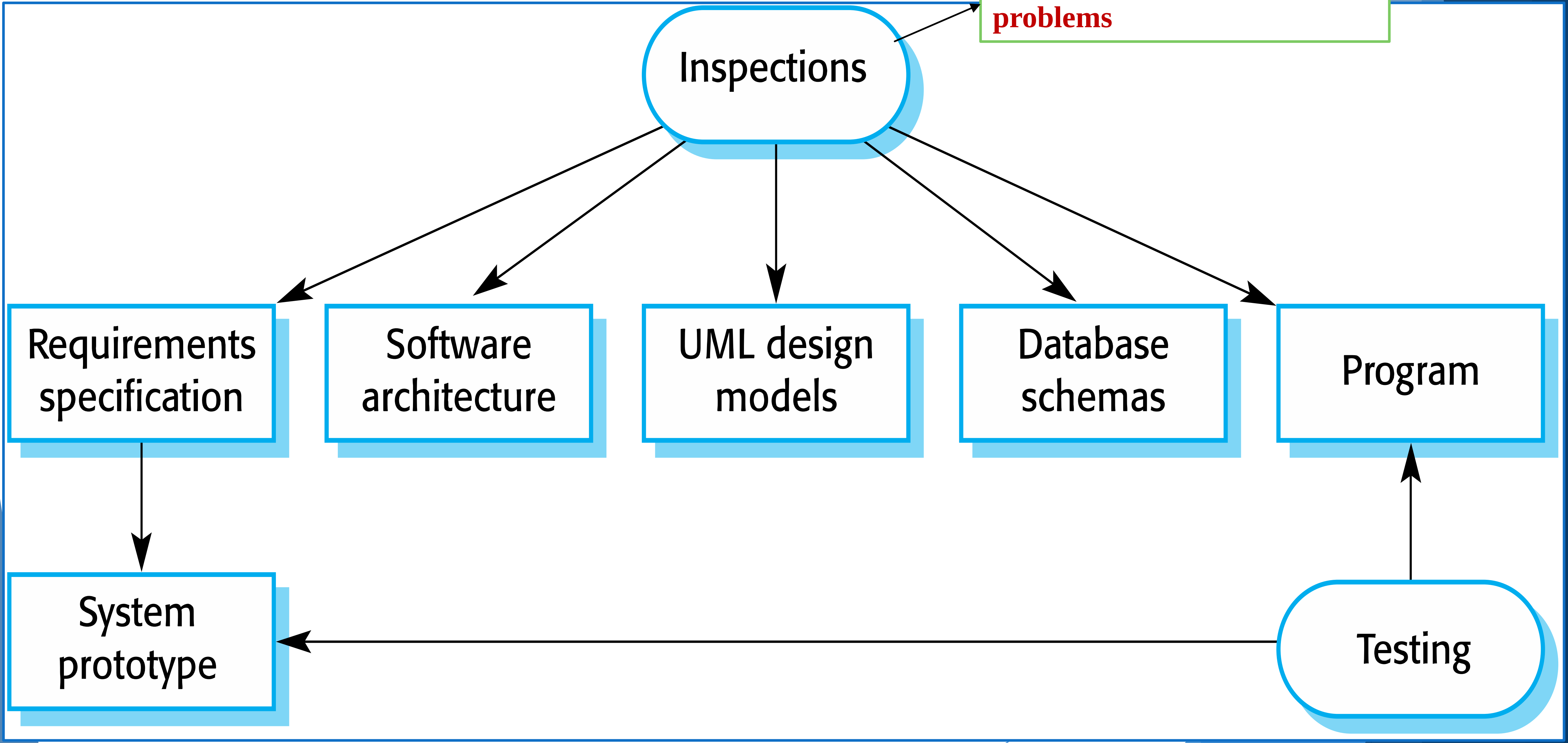
Users may have low expectations of certain kinds of software.

Marketing environment

Getting a product to market early may be more important than finding defects in the program.

Inspections and Testing

Analysis of the static system representation to discover problems



Advantages of inspections

During testing, errors can mask (hide) other errors. Because inspection is a static process, you don’t have to be concerned with interactions between errors.

Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.

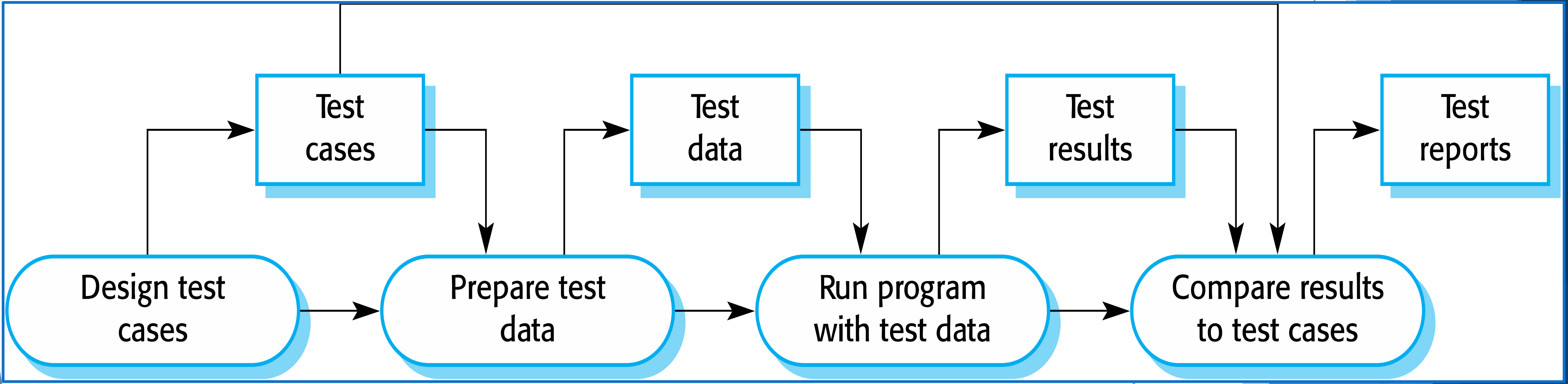
As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

Inspections and testing

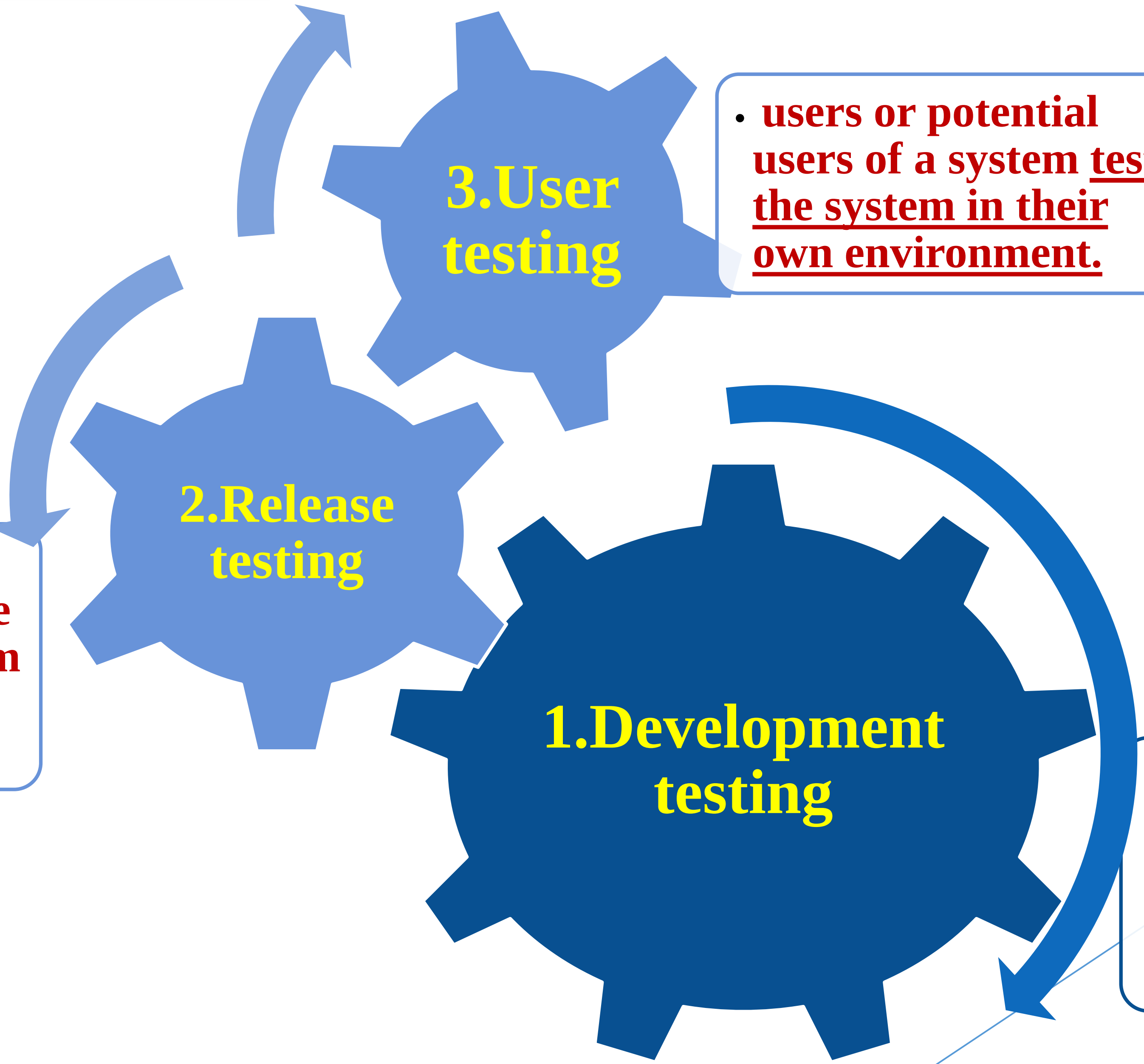
- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer’s real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

A model of the software testing process

- It is an abstract model of the '**traditional' testing process**, as used in plan driven development.
- Test cases are specifications of the inputs to the test and the expected output from the system (the test results), plus a statement of what is being tested.



Three Stages of testing



- users or potential users of a system test the system in their own environment.

- A separate testing team test a complete version of the system before it is released to users.

- The system is tested during development to discover bugs and defects.

1

Development testing

1.Development Testing

- Development testing includes all testing activities that are carried out by the team developing the system.

A.Unit testing

- where individual program units or object classes are tested. **Unit testing should focus on testing the functionality of objects or methods.**

B.Component testing

- where several individual units are integrated to create composite components. **Component testing should focus on testing component interfaces.**

C.System testing

- where some or all of the components in a system are integrated and the system is tested as a whole. **System testing should focus on testing component interactions.**

A. Unit testing

- Unit testing is the process of testing individual components in isolation. It is a defect testing process.
- Units can be : Individual functions or methods within an object

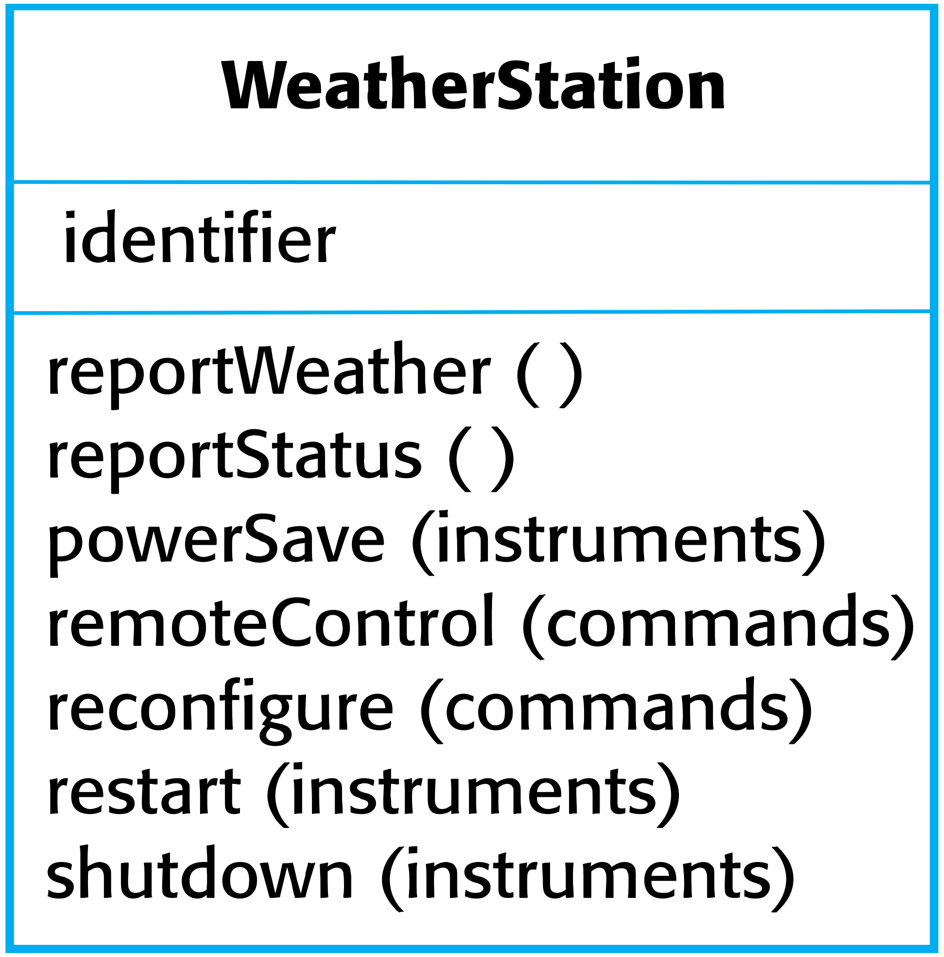
  Object classes with several attributes and methods

  Composite components with defined interfaces used to access their functionality.

Object class testing

- **Complete test coverage of a class involves**
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states.

Ex: The weather station object interface



Weather station testing

- Need to define test cases for reportWeather, calibrate, test, startup and shutdown.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions
- For example:
  - **Shutdown -> Running-> Shutdown**
  - **Configuring-> Running-> Testing -> Transmitting -> Running**
  - **Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running**
- *Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention. In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.*

An automated test has three parts:

1. **<u>A setup part</u>**
   - **where you initialize the system with the test case, namely the inputs and expected outputs.**

2. **<u>A call part</u>**
   - **where you call the object or method to be tested**

3. **<u>An assertion part</u>**
   - **where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful;**
   - **if false,then it has failed.**

Testing strategies

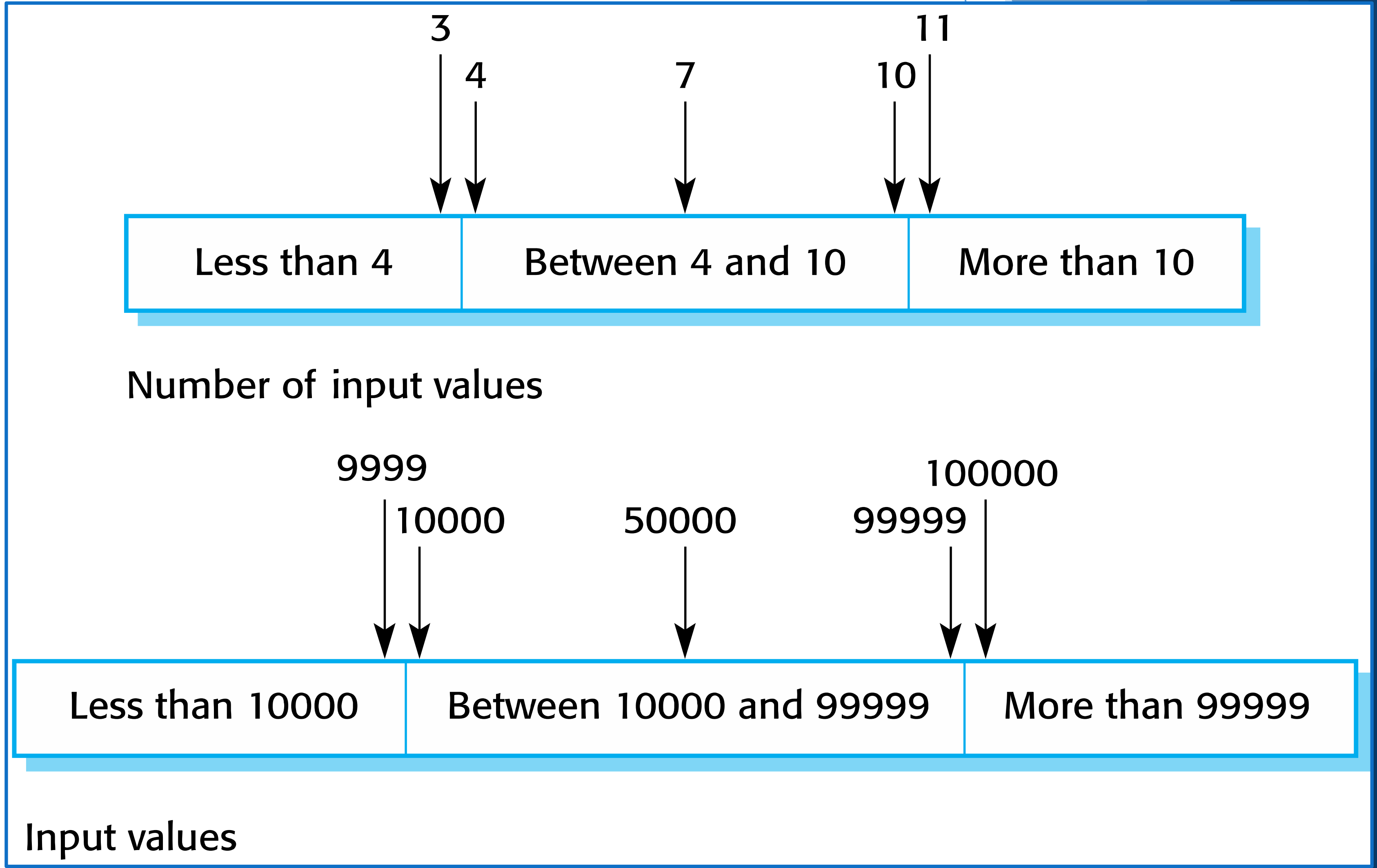
- **Partition testing,**
  - where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.

- **Guideline-based testing**
  - where you use testing guidelines to choose test cases.
  - These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Example: A program specification states that the program accepts 4 to 8 inputs which are five-digit integers greater than 10,000. You use this information to identify the input partitions and possible test input values .This is called as Equivalence partitions .

Testing guidelines :

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Test with sequences of zero length.



General testing Guidelines :

- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

B.Component testing

- Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

Interface testing

Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.

Interface Types

Parameter Interfaces

- Data passed from one method or procedure to another.

Shared memory Interfaces

- Block of memory is shared between procedures or functions.

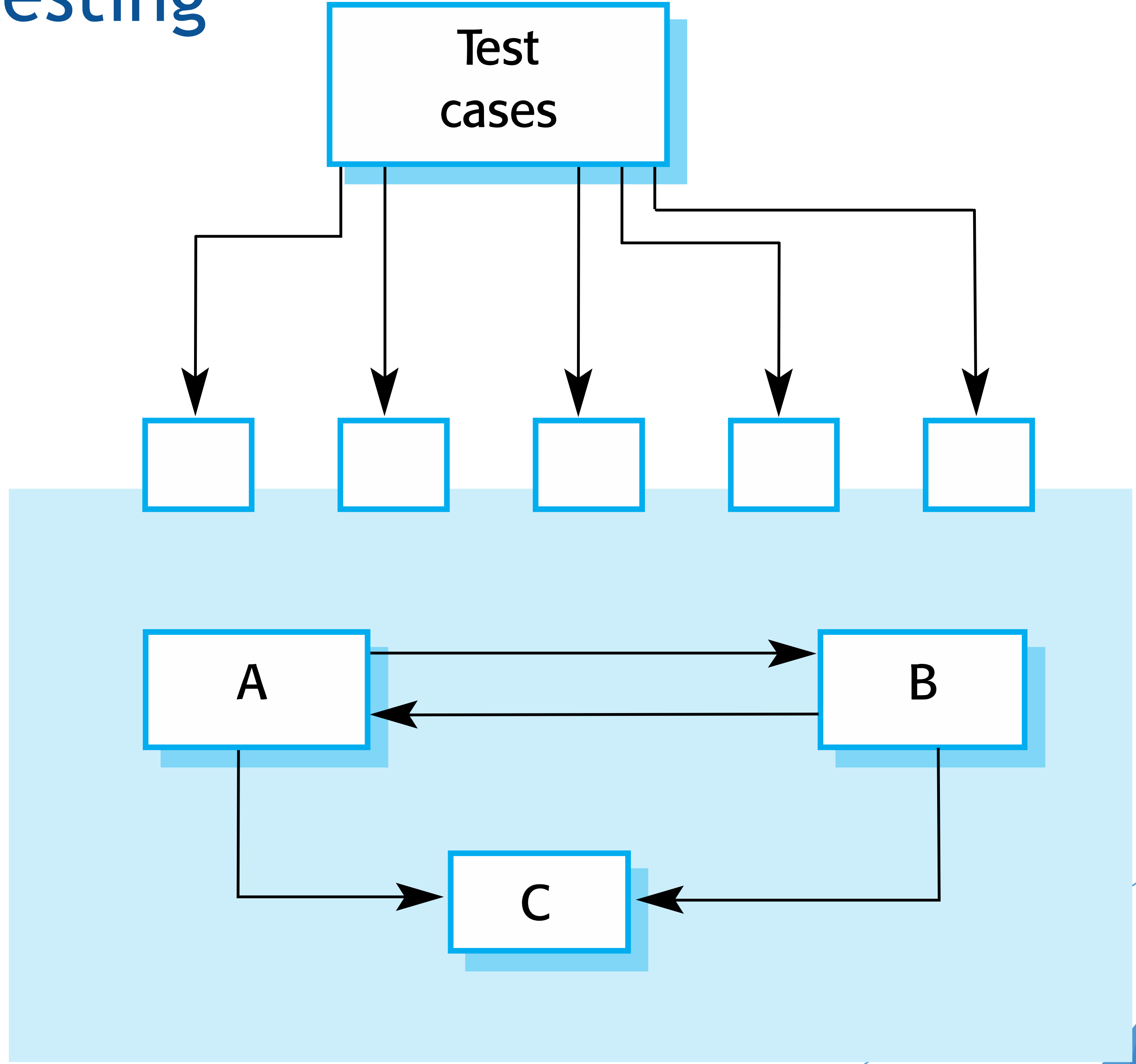
Procedural Interfaces

- Sub-system encapsulates a set of procedures to be called by other sub-systems.

Message passing Interfaces

- Sub-systems request services from other sub-systems

Interface testing



Interface errors

Interface misuse

A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.

Interface misunderstanding

A calling component embeds assumptions about the behaviour of the called component which are incorrect.

Timing errors

The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

C. System testing

- System testing during development **involves integrating components to create a version of the system and then testing the integrated system.**
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- ***System testing tests the emergent behavior of a system.***

Difference between System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.

- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.

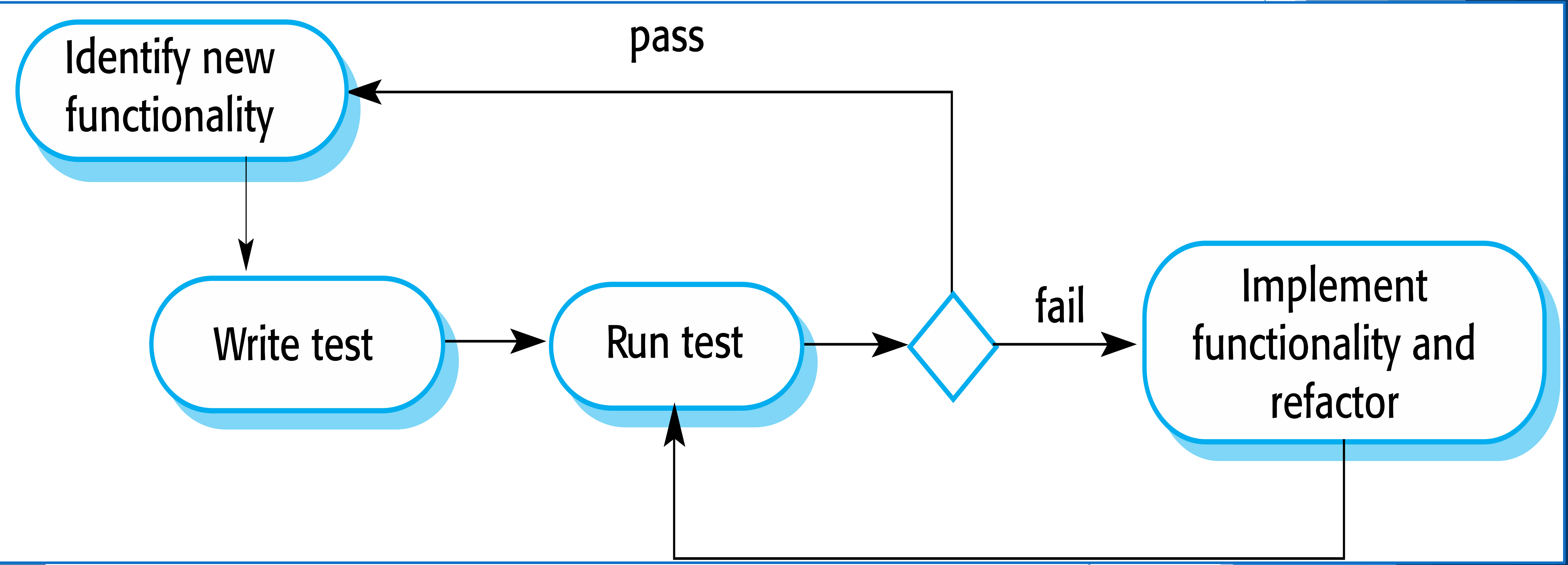
Testing policies

- For most systems, it is difficult to know how much system testing is essential and when you should to stop testing.
- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- ***Examples of testing policies:***
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

Test-driven development

- **Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.**
- Tests are written before code and ‘passing’ the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Test-driven development


pass
Identify new functionality
Write test
Run test
fail
Implement functionality and refactor

Benefits of test-driven development

- **Code coverage** : Every code segment that you write has at least one associated test so all code written has at least one test.
- **Regression testing** :A regression test suite is developed incrementally as a program is developed. Regression testing is testing the system to check that changes have not 'broken' previously working code.
- **Tests must run 'successfully**' before the change is committed.
- **Simplified debugging** :When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- **System documentation** :The tests themselves are a form of documentation that describe what the code should be doing.

2

Release testing

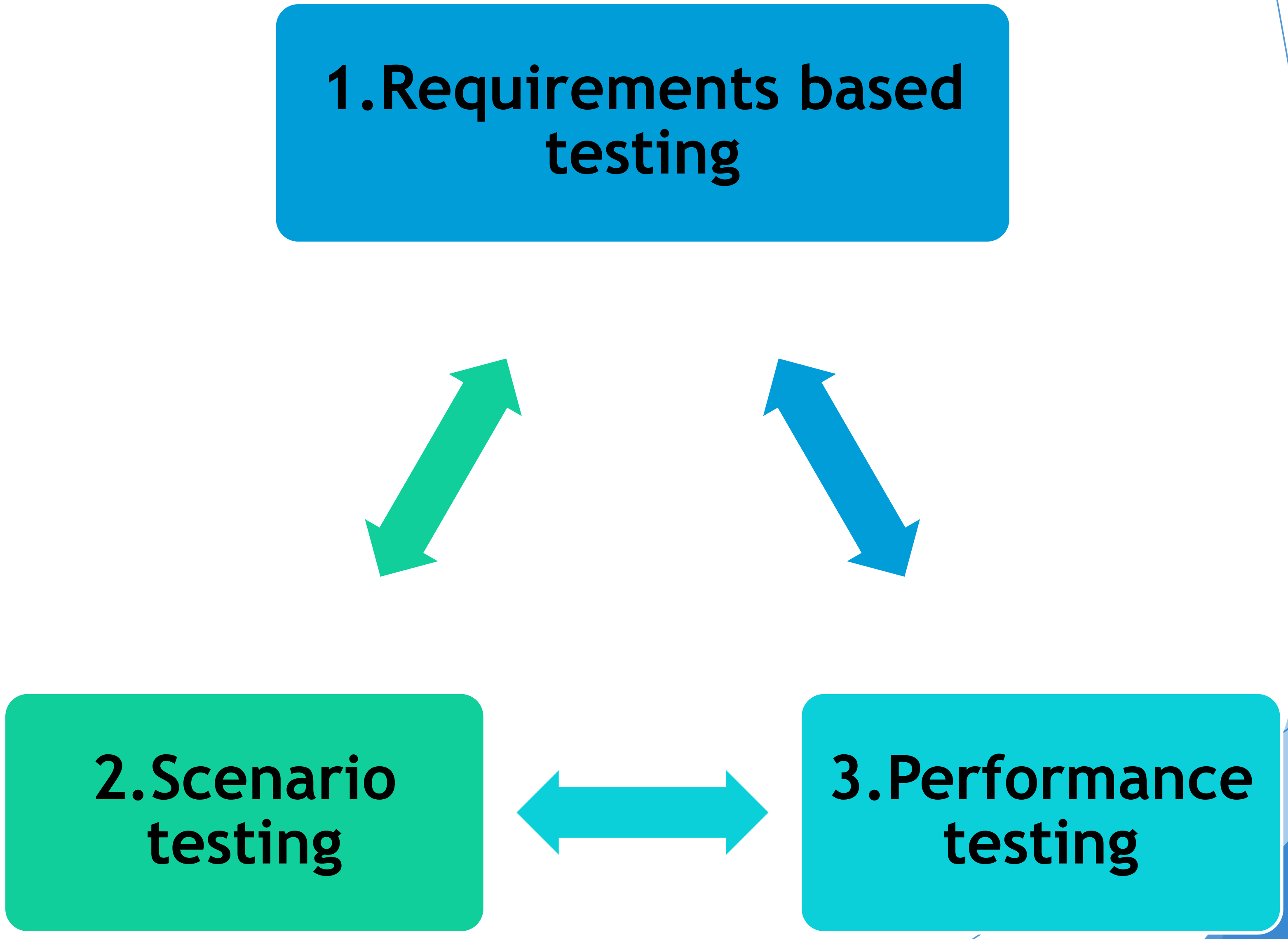
2. Release Testing

- **Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.**
- ***The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.***
  - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Difference :Release testing and system testing

- *Release testing is a form of system testing.*
- **Important differences:**
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing).
  - The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Release testing



1.Requirements based testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.
- **Ex: Mentcare system requirements**:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

The Requirements tests can be as follows:

- Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
- Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
- Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
- Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

2.Scenario testing

Features tested by scenario

- Authentication by logging on to the system.
- Downloading and uploading of specified patient records to a laptop.
- Home visit scheduling.
- Encryption and decryption of patient records on a mobile device.
- Record retrieval and modification.
- Links with the drugs database that maintains side-effect information.
- The system for call prompting.

3.Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

3

User testing

3. User Testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user’s working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of user testing

Alpha testing

- Users of the software work with the development team to test the software at the developer’s site.

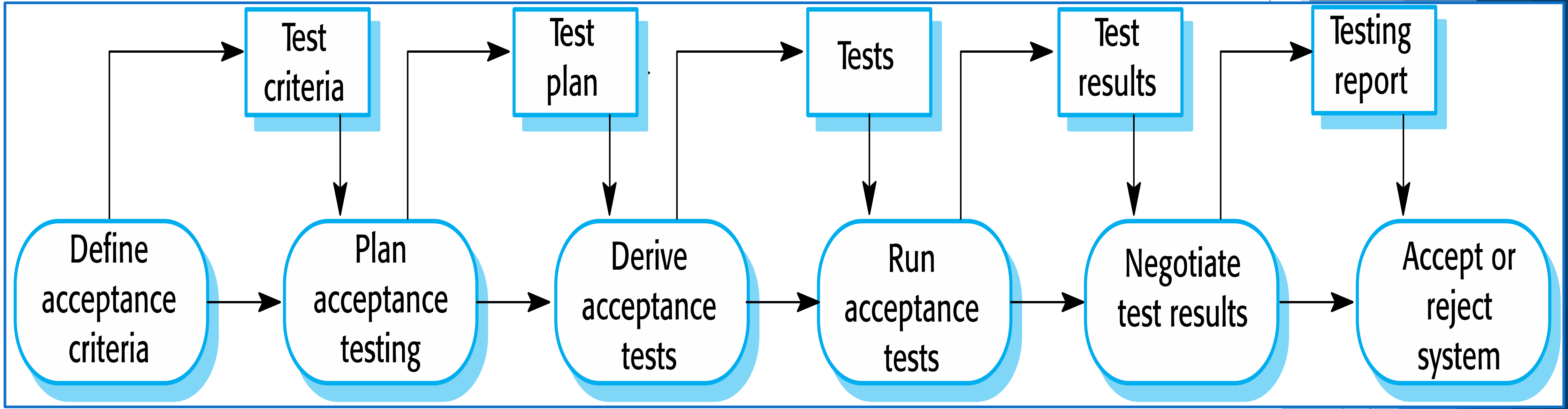
Beta testing

- A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

Acceptance testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

The acceptance testing process has 6 stages:



Agile methods and acceptance testing

- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is 'typical' and can represent the interests of all system stakeholders.

Dependable systems

- Dependability properties
- Sociotechnical systems
- Redundancy and diversity
- Dependable processes
- Formal methods and dependability

System dependability

- For many computer-based systems, the most important system property is the dependability of the system.
- The dependability of a system reflects the user's degree of trust in that system.
- It reflects the extent of the user's confidence that it will operate as users expect and that it will not 'fail' in normal use.
- Dependability covers the related systems attributes of reliability, availability and security.
- These are all inter-dependent.

Importance of dependability

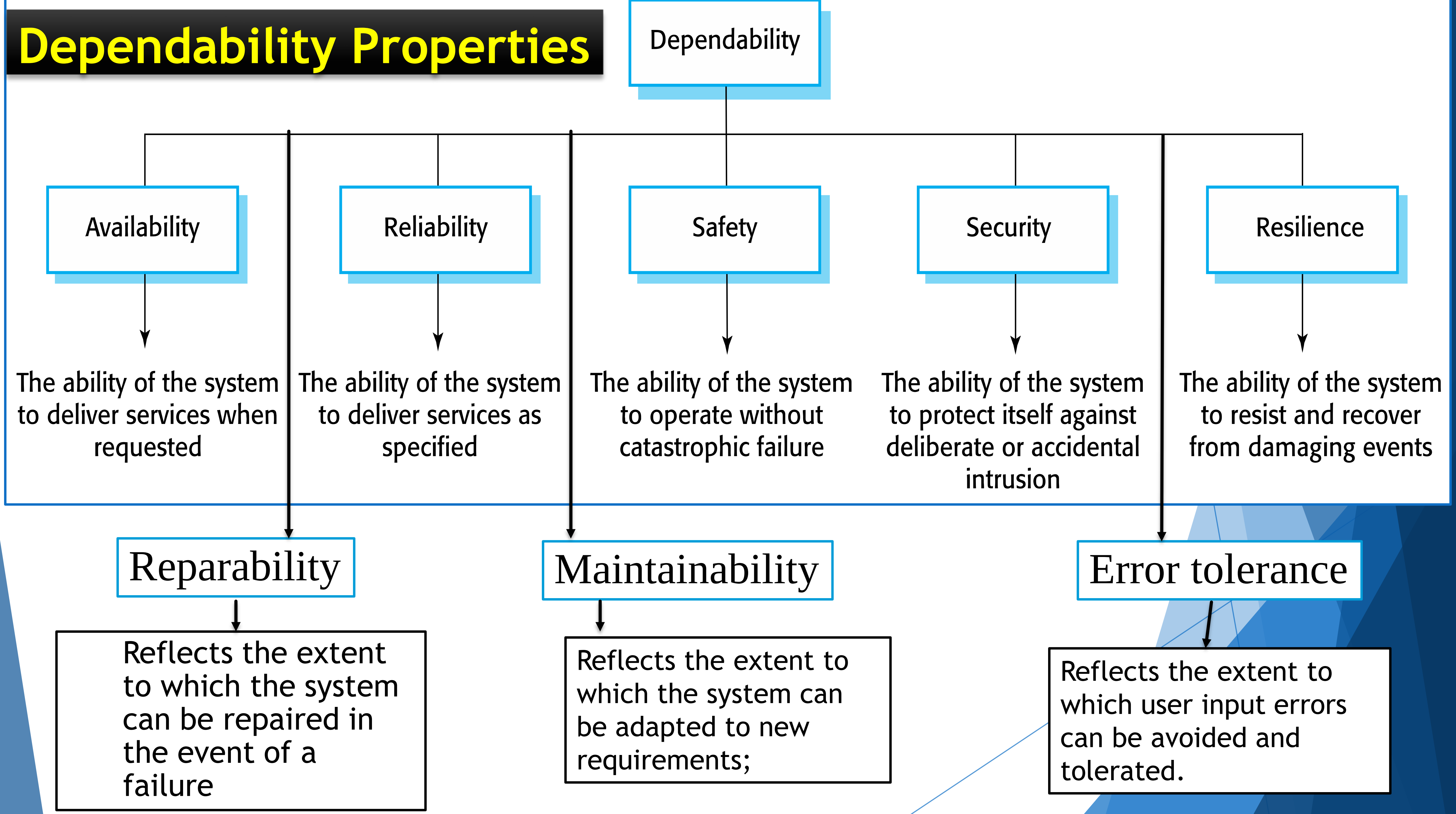
- System failures may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high if the failure leads to economic losses or physical damage.
- Undependable systems may cause information loss with a high consequent recovery cost.

Causes of failure

- **Hardware failure**
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- **Software failure**
  - Software fails due to errors in its specification, design or implementation.
- **Operational failure**
  - Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

Dependability properties

Dependability Properties



Dependability

Availability

The ability of the system to deliver services when requested

Reliability

The ability of the system to deliver services as specified

Safety

The ability of the system to operate without catastrophic failure

Security

The ability of the system to protect itself against deliberate or accidental intrusion

Resilience

The ability of the system to resist and recover from damaging events

Reparability

Reflects the extent to which the system can be repaired in the event of a failure

Maintainability

Reflects the extent to which the system can be adapted to new requirements;

Error tolerance

Reflects the extent to which user input errors can be avoided and tolerated.

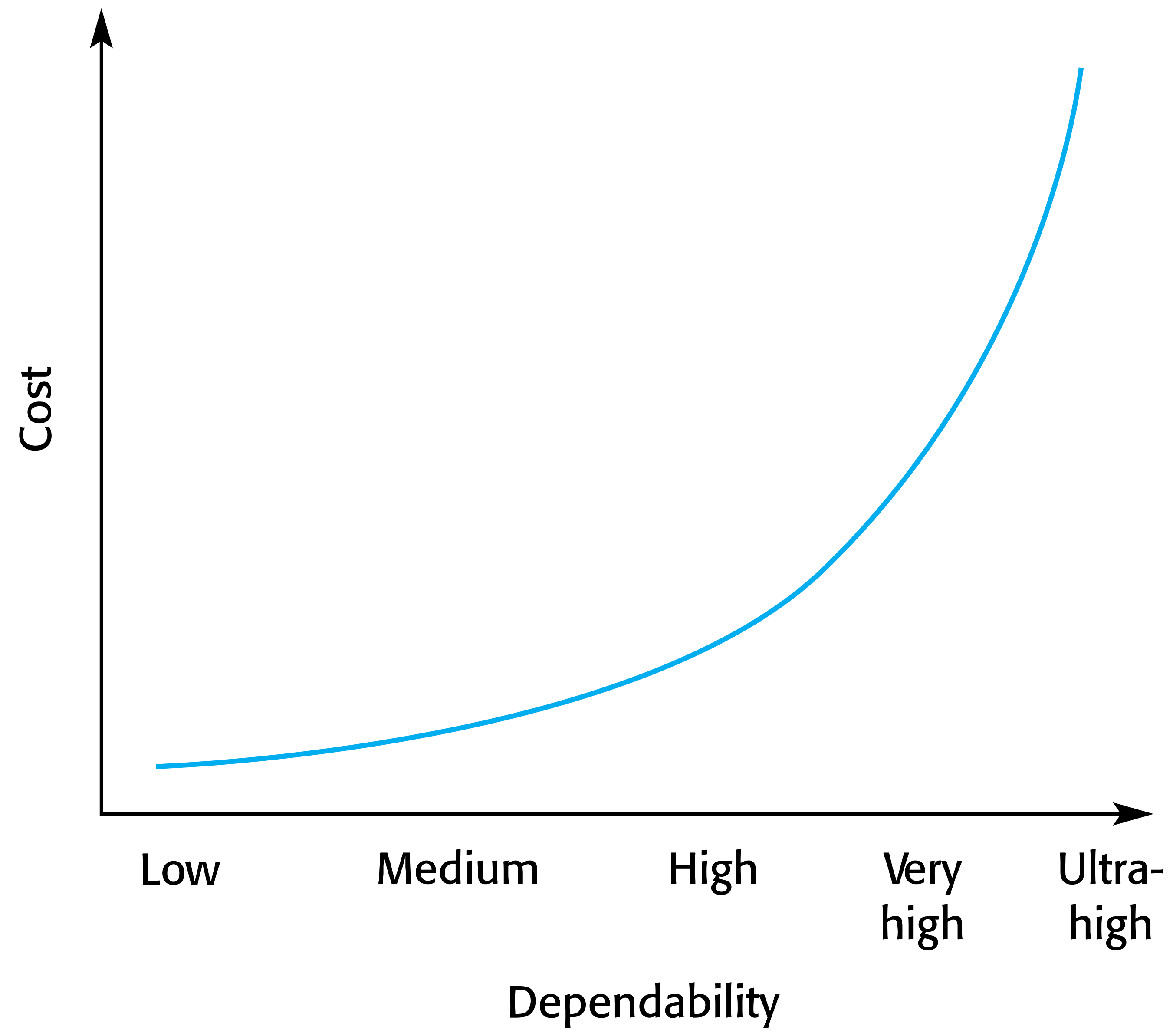
To develop dependable software, you therefore need to ensure that:

- Avoid the introduction of accidental errors when developing the system.
- Design V & V processes that are effective in discovering residual errors in the system.
- Design systems to be fault tolerant so that they can continue in operation when faults occur
- Design protection mechanisms that guard against external attacks.

Dependability costs

- ***Dependability costs tend to increase exponentially as increasing levels of dependability are required.***
- **There are two reasons for this**
  - The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability.
  - The increased testing and system validation that is required to convince the system client and regulators that the required levels of dependability have been achieved.

Cost/dependability curve



Dependable processes

- To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process.
- A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people.
- Regulators use information about the process to check if good software engineering practice has been used.
- For fault detection, it is clear that the process activities should include significant effort devoted to verification and validation.

Dependable process characteristics

- **Explicitly defined**
  - A process that has a defined process model that is used to drive the software production process. Data must be collected during the process that proves that the development team has followed the process as defined in the process model.
- **Repeatable**
  - A process that does not rely on individual interpretation and judgment. The process can be repeated across projects and with different team members, irrespective of who is involved in the development.

Attributes of dependable processes

Process characteristic	Description
Auditable	The process should be understandable by people apart from process participants, who can check that process standards are being followed and make suggestions for process improvement.
Diverse	The process should include redundant and diverse verification and validation activities.
Documentable	The process should have a defined process model that sets out the activities in the process and the documentation that is to be produced during these activities.
Robust	The process should be able to recover from failures of individual process activities.
Standardized	A comprehensive set of software development standards covering software production and documentation should be available.

Reliability Engineering

- Availability and reliability
- Reliability requirements
- Fault-tolerant architectures
- Programming for reliability
- Reliability measurement

Software reliability

- In general, software customers expect all software to be dependable.
- However, for non-critical applications, they may be willing to accept some system failures.
- Some applications (critical systems) have very high reliability requirements and special software engineering techniques may be used to achieve this.
  - Medical systems
  - Telecommunications and power systems
  - Aerospace systems

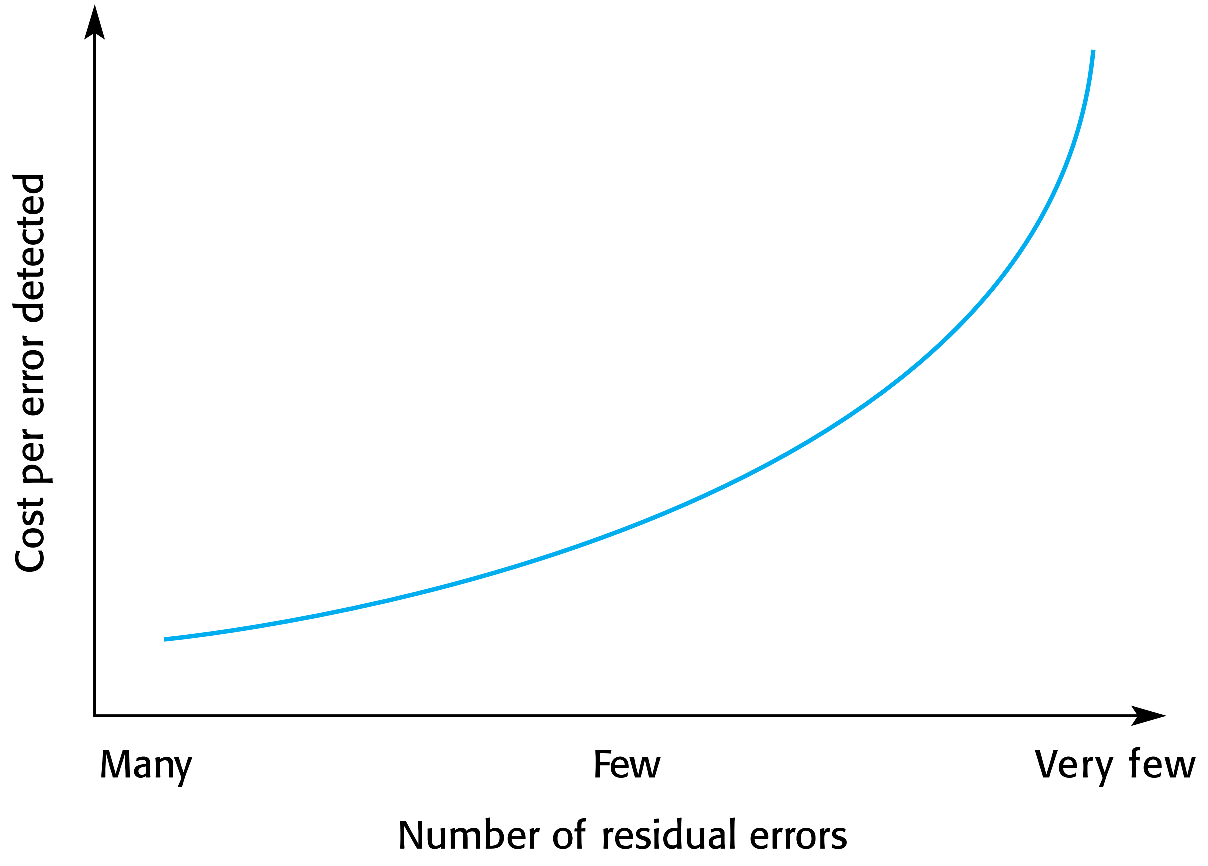
Faults, errors and failures

Term	Description
Human error or mistake	Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.

Reliability achievement

- **Fault avoidance**
  - Development technique are used that either minimise the possibility of mistakes or trap mistakes before they result in the introduction of system faults.
- **Fault detection and removal**
  - Verification and validation techniques are used that increase the probability of detecting and correcting errors before the system goes into service are used.
- **Fault tolerance**
  - Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures.

The increasing costs of residual fault removal



Availability and reliability

Availability and reliability

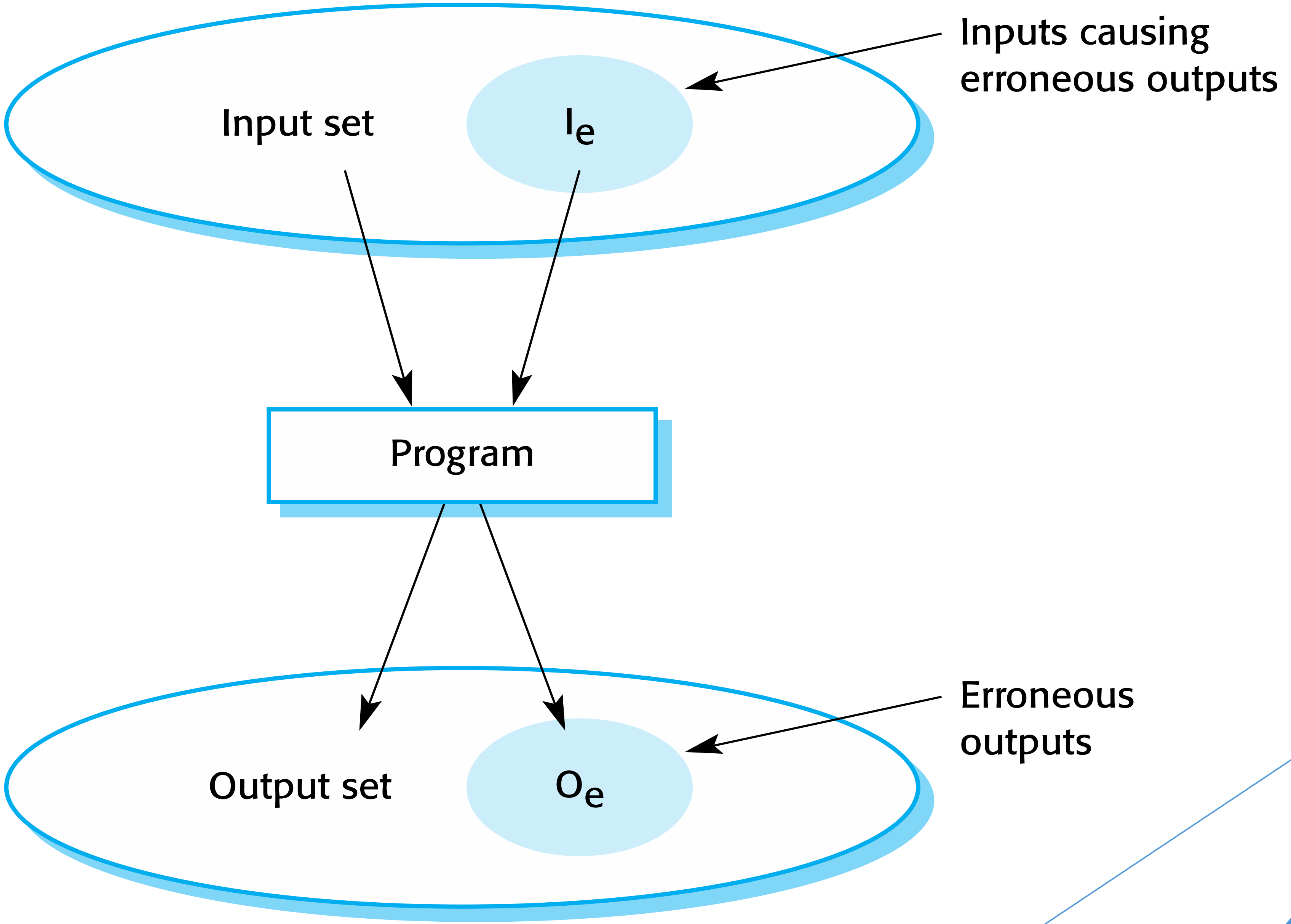
- **Reliability**
  - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- **Availability**
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services

- Both of these attributes can be expressed quantitatively e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.

Reliability and specifications

- ***Reliability can only be defined formally with respect to a system specification i.e. a failure is a deviation from a specification.***
- However, many specifications are incomplete or incorrect – hence, a system that conforms to its specification may ‘fail’ from the perspective of system users.
- Furthermore, users don’t read specifications so don’t know how the system is supposed to behave.
- Therefore perceived ***reliability is more important in practice.***

A system as an input/output mapping



Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account two factors:
  - The number of users affected by the service outage. Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage. The longer the outage, the more the disruption. Several short outages are less likely to be disruptive than 1 long outage. Long repair times are a particular problem.

Reliability in use

- Removing X% of the faults in a system will not necessarily improve the reliability by X%.
- Program defects may be in rarely executed sections of the code so may never be encountered by users.
- Removing these does not affect the perceived reliability.
- A program with known faults may therefore still be perceived as reliable by its users.

Reliability requirements

Reliability metrics

- Reliability metrics are units of measurement of system reliability.
- System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational.
- A long-term measurement programme is required to assess the reliability of critical systems.

- Metrics
  1. **Probability of failure on demand**
  2. **Rate of occurrence of failures/Mean time to failure**
  3. **Availability**

1.Probability of failure on demand (POFOD)

- This is the probability that the system will fail when a service request is made.
- Useful when demands for service are intermittent and relatively infrequent.
- Appropriate for protection systems where services are demanded occasionally and where there are serious consequence if the service is not delivered.
- Relevant for many safety-critical systems with exception management components
  - Emergency shutdown system in a chemical plant.

2.Rate of fault occurrence (ROCOF)

- Reflects the rate of occurrence of failure in the system.
- ROCOF of 0.002 means 2 failures are likely in each 1000 operational time units e.g. 2 failures per 1000 hours of operation.
- Relevant for systems where the system has to process a large number of similar requests in a short time
  - Credit card processing system, airline booking system.
- **Reciprocal of ROCOF is Mean time to Failure (MTTF)**
  - Relevant for systems with long transactions i.e. where system processing takes a long time (e.g. CAD systems). MTTF should be longer than expected transaction length.

3.Availability

- Measure of the fraction of the time that the system is available for use.
- Takes repair and restart time into account
- Availability of 0.998 means software is available for 998 out of 1000 time units.
- Relevant for non-stop, continuously running systems
  - telephone switching systems, railway signalling systems.

Availability specification

Availability	Explanation
0.9	The system is available for 90% of the time. This means that, in a 24-hour period (1,440 minutes), the system will be unavailable for 144 minutes.
0.99	In a 24-hour period, the system is unavailable for 14.4 minutes.
0.999	The system is unavailable for 84 seconds in a 24-hour period.
0.9999	The system is unavailable for 8.4 seconds in a 24-hour period. Roughly, one minute per week.

Example: Insulin pump reliability specification

- Probability of failure (POFOD) is the most appropriate metric.
- Transient failures that can be repaired by user actions such as recalibration of the machine. A relatively low value of POFOD is acceptable (say 0.002) – one failure may occur in every 500 demands.
- Permanent failures require the software to be re-installed by the manufacturer. This should occur no more than once per year. POFOD for this situation should be less than 0.00002.

Examples of functional reliability requirements

RR1: A pre-defined range for all operator inputs shall be defined and the system shall check that all operator inputs fall within this pre-defined range. (Checking)

RR2: Copies of the patient database shall be maintained on two separate servers that are not housed in the same building. (Recovery, redundancy)

RR3: N-version programming shall be used to implement the braking control system. (Redundancy)

RR4: The system must be implemented in a safe subset of Ada and checked using static analysis. (Process)

Safety terminology

Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment.
Hazard	A condition with the potential for causing or contributing to an accident.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage..
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard.
Hazard probability	The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from 'probable' (say 1/100 chance of a hazard occurring) to 'implausible' (no conceivable situations are likely in which the hazard could occur).
Risk	This is a measure of the probability that the system will cause an accident..

Normal accidents

- Accidents in complex systems rarely have a single cause as these systems are designed to be resilient to a single point of failure
    - Designing systems so that a single point of failure does not cause an accident is a fundamental principle of safe systems design.
- Almost all accidents are a result of combinations of malfunctions rather than single failures.
- It is probably the case that anticipating all problem combinations, especially, in software controlled systems is impossible so achieving complete safety is impossible. Accidents are inevitable.

Safety specification

- The goal of safety requirements engineering is to identify protection requirements that ensure that system failures do not cause injury or death or environmental damage.
- Safety requirements may be ‘shall not’ requirements i.e. they define situations and events that should never occur.
- ***Functional safety requirements define:***
  - Checking and recovery features that should be included in a system
  - Features that provide protection against system failures and external attacks

Security

- ***The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack***.
- Security is essential as most systems are networked so that external access to the system through the Internet is possible.
- **<u>Security is an essential pre-requisite for availability, reliability and safety.</u>**
- If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable.
- These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and/or its data.
- Therefore, ***<u>the reliability and safety assurance is no longer valid</u>***.

Security terminology

Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Attack	An exploitation of a system’s vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Control	A protective measure that reduces a system’s vulnerability.
Exposure	Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Threat	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.

Examples of security terminology (Mentcare)

Term	Example
Asset	The records of each patient that is receiving or has received treatment.
Exposure	Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation.
Vulnerability	A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names.
Attack	An impersonation of an authorized user.
Threat	An unauthorized user will gain access to the system by guessing the credentials (login name and password) of an authorized user.
Control	A password checking system that disallows user passwords that are proper names or words that are normally included in a dictionary.

Threat types

Interception threats

- **It allow an attacker to gain access to an asset.**
- A possible threat to the Mentcare system might be a situation where an attacker gains access to the records of an individual patient.

Interruption threats

- **It allow an attacker to make part of the system unavailable.**
- A possible threat might be a denial of service attack on a system database server so that database connections become impossible.

Modification threats

- **it allow an attacker to tamper with a system asset.**
- In the Mentcare system, a modification threat would be where an attacker alters or destroys a patient record.

Fabrication threats

- **it allow an attacker to insert false information into a system.**
- This is perhaps not a credible threat in the Mentcare system but would be a threat in a banking system, where false transactions might be added to the system that transfer money to the perpetrator’s bank account.

Security assurance

- **Vulnerability avoidance**
  - The system is designed so that vulnerabilities do not occur. For example, if there is no external network connection then external attack is impossible
- **Attack detection and elimination**
  - The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure. For example, virus checkers find and remove viruses before they infect a system
- **Exposure limitation and recovery**
  - The system is designed so that the adverse consequences of a successful attack are minimised. For example, a backup policy allows damaged information to be restored

Security and dependability

- ***Security and reliability***
  - If a system is attacked and the system or its data are corrupted as a consequence of that attack, then this may induce system failures that compromise the reliability of the system.
- ***Security and availability***
  - A common attack on a web-based system is a denial of service attack, where a web server is flooded with service requests from a range of different sources. The aim of this attack is to make the system unavailable.

Security testing

- Testing the extent to which the system can protect itself from external attacks.
- **Problems with security testing**
  - Security requirements are 'shall not' requirements i.e. they specify what should not happen.
  - It is not usually possible to define security requirements as simple constraints that can be checked by the system.
  - The people attacking a system are intelligent and look for vulnerabilities.
  - They can experiment to discover weaknesses and loopholes in the system.

Security validation

Experience-based testing	• The system is reviewed and analysed against the types of attack that are known to the validation team.
Penetration testing	• A team is established whose goal is to breach the security of the system by simulating attacks on the system.
Tool-based analysis	• Various security tools such as password checkers are used to analyse the system in operation.
Formal verification	• The system is verified against a formal security specification.

Examples of entries in a security checklist

Security checklist

1. Do all files that are created in the application have appropriate access permissions? The wrong access permissions may lead to these files being accessed by unauthorized users.

2. Does the system automatically terminate user sessions after a period of inactivity? Sessions that are left active may allow unauthorized access through an unattended computer.

3. If the system is written in a programming language without array bound checking, are there situations where buffer overflow may be exploited? Buffer overflow may allow attackers to send code strings to the system and then execute them.

4. If passwords are set, does the system check that passwords are 'strong'? Strong passwords consist of mixed letters, numbers, and punctuation, and are not normal dictionary entries. They are more difficult to break than simple passwords.

5. Are inputs from the system's environment always checked against an input specification? Incorrect processing of badly formed inputs is a common cause of security vulnerabilities.