



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

ISTQB CTFL

Certified Tester in Foundation Level

Phòng LT & Mạng

<http://csc.edu.vn/kiem-thu-phan-mem>

2019



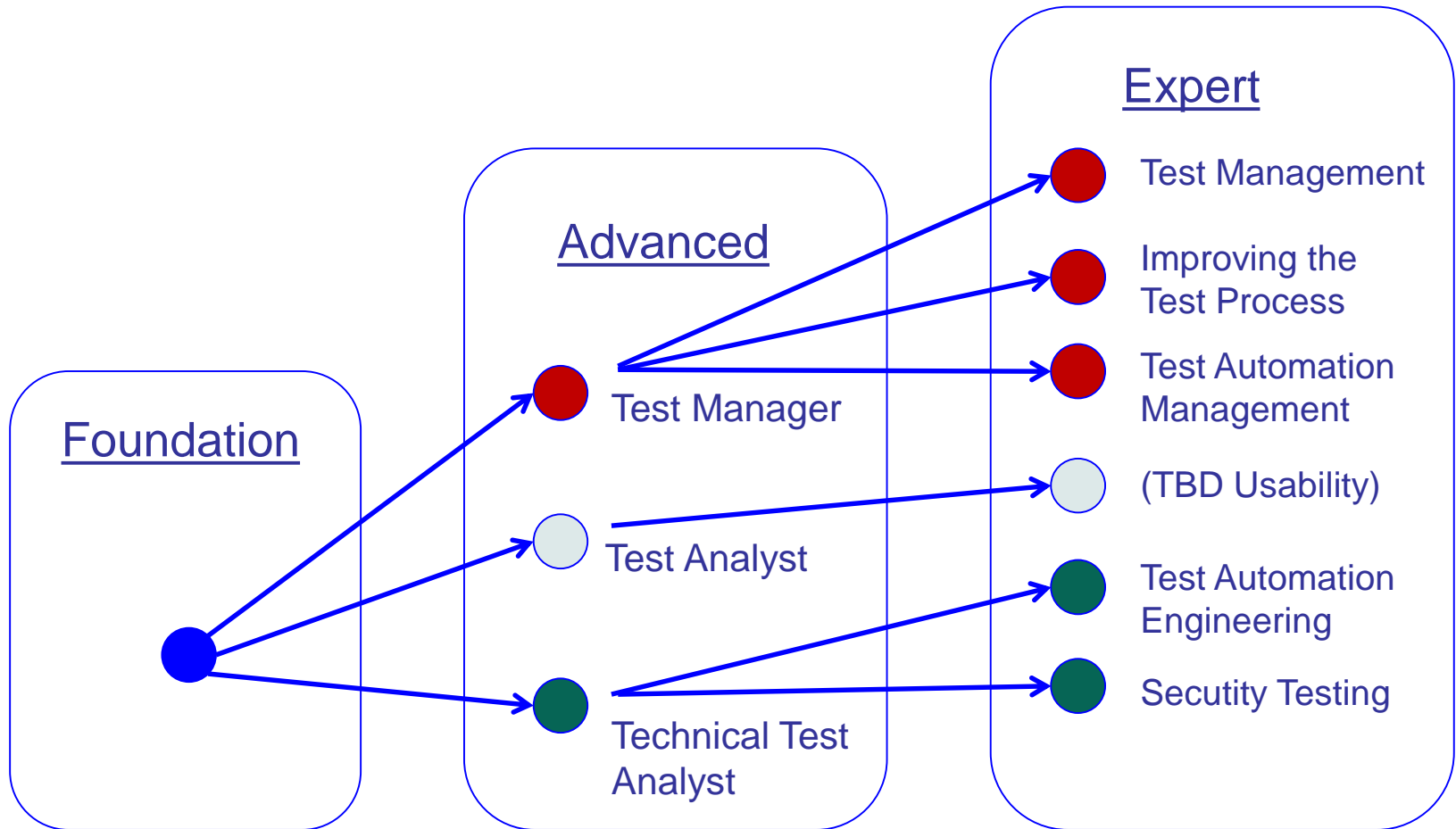
WHAT IS ISTQB?

- 1. The Test Development Process**
2. Categories of Test Design Techniques
3. Specification-based or Black-box Techniques
4. Structure-based or White-box Techniques
5. Experience-based Techniques
6. Choosing Test Techniques

WHAT IS ISTQB?

- ❑ The ISTQB is stands for "International Software Testing Qualification Board"
- ❑ Biggest Software Testing Certification and Education Program in the world.
- ❑ The ISTQB is NON-PROFIT ORGARNIZATION
- ❑ The ISTQB is responsible for the international qualification scheme called "ISTQB Certified Tester".

With the new Expert Level, ISTQB® offers career paths for testers



ISTQB® Syllabus - Foundation Level

①	②	③	④	⑤	⑥
Software testing fundamentals	Testing throughout the software life-cycle	Static testing techniques	Test design techniques	Test management	Tool support for testing
Definition of terms	Software development models	Meaning of static test	Explanation of techniques terms	Organization	Types of testing tool
Reason for testing	The economics of testing	Reviews and the test process	Black and white box tests	Configuration management	Tool selection and use
Fundamental test process	Test plan	Types of review	Functional test techniques	Test estimation, monitoring & control	
The psychology of testing	Component test		Structural test techniques	Defect management	
Re-test and regression test	Integration test		Experience-based techniques	Testing standards	
Expected results	Functional system test				
Prioritizing tests	Non-functional system test				
	Acceptance test				
	Maintenance test				

□ Topics

- Topics within the Foundation syllabus have been categorized by learning objective, defining the level of knowledge a candidate must display for each topic.
- The three categories are as follows:
 - K1 – remember, retrieve, recognize, recall, know
 - K2 - understand, explain, give reasons, compare, classify, summarize
 - K3 - apply in a specific context
 - K4 - analyze and propose appropriate action to solve a problem/ task
- The ISTQB has mandated a weighting for each exam paper.
 - 50% of the questions will be K1 level,
 - 30% will be K2 level and
 - 20% will be K3/K4 level.
- 40 questions/ 60 minutes: Passed: 26/40

Reference Books

❑ Reference Books

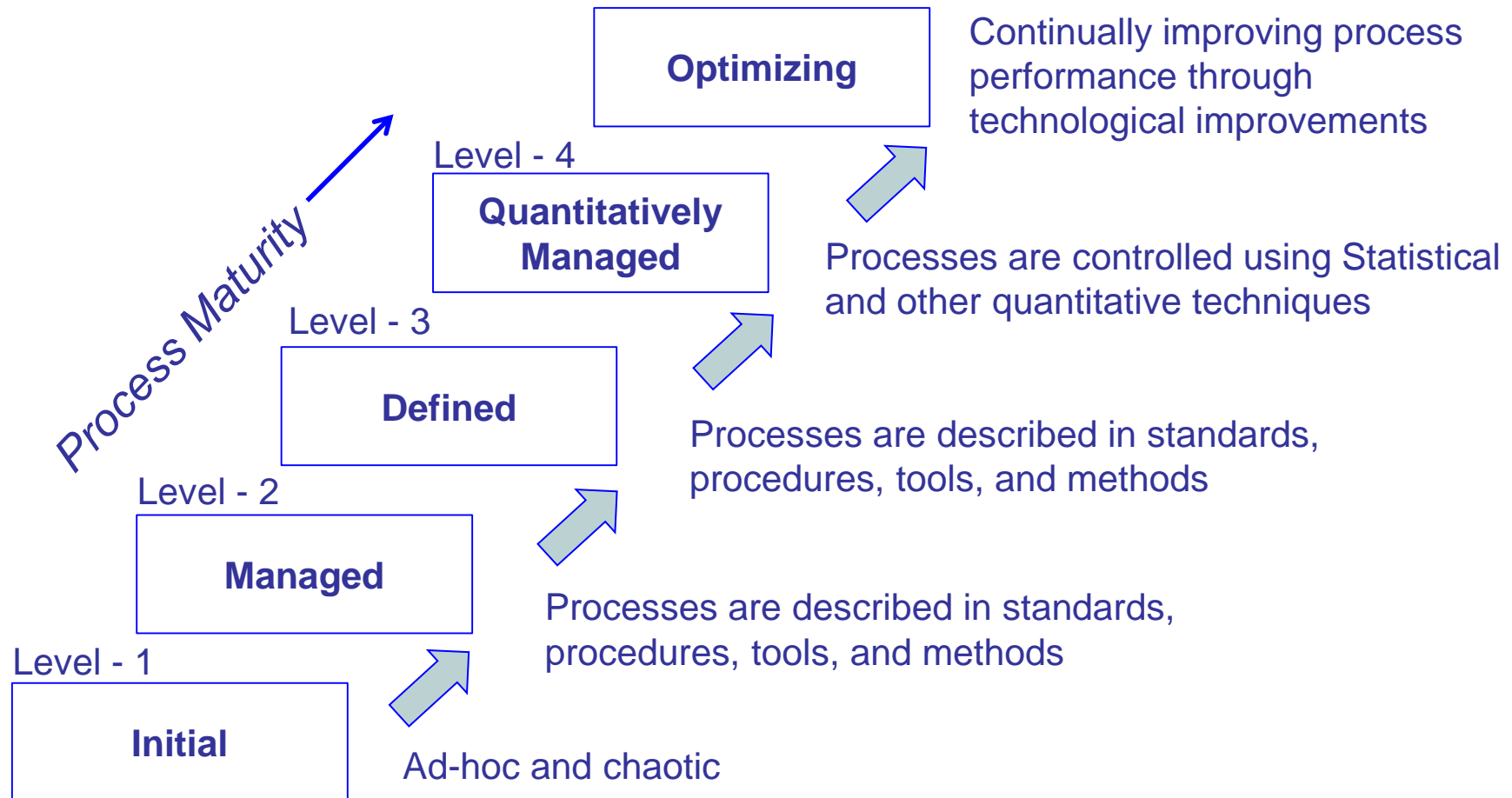
- ISTQB Foundation Level Syllabus (2018).
- ISTQB Glossary (V3.2)
- Foundation of Software Testing - Rex Black

❑ Other helpful books

- The Art of Software Testing - Glenford Myers
- Black Box Testing - Boris Beizer
- Lessons Learned in Software Testing - James Bach, Cem Kaner
- Managing the Testing Process - Rex Black
- Managing Agile Projects - Robert C., Martin S.
- How to break Software: A Practical Guide to Testing - James Whitaker

- ❑ ISO/IEC (International Organization for Standardization / International Electrotechnical Commission)
- ❑ IEEE (Institute of Electrical and Electronics Engineers)
 - Foundation of Software Testing - Rex Black
- ❑ E.g:
 - ISO/IEC/IEEE 29119 replaces IEEE Standard 829
 - Standard for Software Test Documentation
 - ISO/IEC 25010 replaces ISO 9126
 - The Software quality characteristics
 - ISO/IEC 20246 replaces IEEE 1028

❑ CMM (Capability Maturity Model® Integration)



- ❑ Knowledge transferring to the You.
- ❑ Globally recognized Certification means you will be recognized in globe
- ❑ Built strong infrastructure for Vietnam Software Industry.

After Training

- ☐ Learn the basics needed to become a software test professional
- ☐ Understand how testing fits into the software development lifecycle.
- ☐ Find out what it takes to be a successful software test engineer
- ☐ How testing can add significant value to software development.

CHAPTER 1

Fundamentals Of Software Testing

1. Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

1.3 General Testing Principles

1.4 Fundamental Test Process

1.5 The psychology of Testing

1. Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

1.3 General Testing Principles

1.4 Fundamental Test Process

1.5 The psychology of Testing

Failure Examples

❑ Y2K (1999)

- **Cost:** \$500 billion
- **Disaster:** While no significant computer failures occurred, preparation for the Y2K bug had a significant cost and time impact on all industries that use computer technology.
- **Cause:** To save computer storage space, legacy software often stored the year for dates as two digit numbers, such as "99" for 1999. The software also interpreted "00" to mean 1900 rather than 2000.

Software that does not work correctly can lead to loss of money, time, or business reputation, and even injury or death.

1.1 What is testing?

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.

1.1.1 Testing Objectives

- To evaluate work products such as requirements, user stories, design, and code
- To verify whether all specified requirements have been fulfilled
- To validate whether the test object is complete and works as the users and other stakeholders expect
- To build confidence in the level of quality of the test object
- To prevent defects
- To find failures and defects
- To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object
- To reduce the level of risk of inadequate software quality (e.g., previously undetected failures occurring in operation)
- To comply with contractual, legal, or regulatory requirements or standards, and/ or to verify the test object's compliance with such requirements or standards

1.1.1 Testing Objectives

Different viewpoints in testing take different objectives into account:

- ❑ Development testing (e.g. component, integration and system testing):
 - To cause as many failures as possible so that defects in the software are identified and can be fixed.
 - To increase code coverage.

- ❑ Acceptance testing
 - To confirm that the system works as expected, to gain confidence that it has met the requirements.
 - To give information about the risk of releasing the system at a given time.

1.1.1 Testing Objectives

☐ Maintenance testing

- No new errors have been introduced during development of the changes.

☐ Operational testing

- To assess system characteristics such as reliability or availability.

1.1.2 Testing and Debugging

❑ Testing is not debugging

- Testing:
 - Show failures that are caused by defects.
 - Responsibility: Tester
- Debugging:
 - The development activity that identifies the cause of a defect repairs the code and checks that the defect has been fixed correctly.
 - Responsibility: Developer

However, in Agile development and in some other lifecycles, testers may be involved in debugging and component testing.

1.1.3 Testing Terms

❑ Common testing terms

- Test Process
- Test Case
- Test procedure
- Test suite, Test run
- Test scenario
- Test condition, input, expected result, test oracle, actual result

❑ Bug free software

- No known bug free software
- Can not conclude the system is completely safe with no further faults after running all test cases that not reveal any further bugs.

1.1.3 Testing Terms

❑ Test effort

- Complete Test is not possible
- Test effort is between 20% - 50%
- Fault can cause high cost
- Define test intensity and test extent in dependence to the risk
- Select adequate test procedures
- Test of extra functionality
- Limited resources

1.1.3 Testing Terms

❑ Software Quality

- Quality is “fitness for use”

(Joseph Juran)

- Quality is “conformance to requirements”

(Philip B. Crosby)

- Quality of a product or services is its ability to satisfy the needs and expectations of the customer

1.1.3 Testing Terms

❑ Software Quality is measured based on its attributes

Functionality:

- Suitability
- Accuracy
- Interoperability
- Compliance
- Security

Reliability:

- Maturity
- Recoverability
- Fault Tolerance

Usability:

- Learnability
- Understandability
- Operability
- Attractability

Efficiency:

- Time Behavior
- Resource Behavior

Maintainability:

- Analyzability
- Changeability
- Stability
- Testability

Portability:

- Adaptability
- Ease of Installation
- Conformance
- Replaceability

1.1.4 Testing and Quality

- Testing give confidence in the quality of the software if it finds few or no defects.
- A properly designed test that passes reduces the overall level of risk in a system.
- When testing does find defects, the quality of the software system increases when those defects are fixed.
- Lessons should be learned from previous projects.

Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

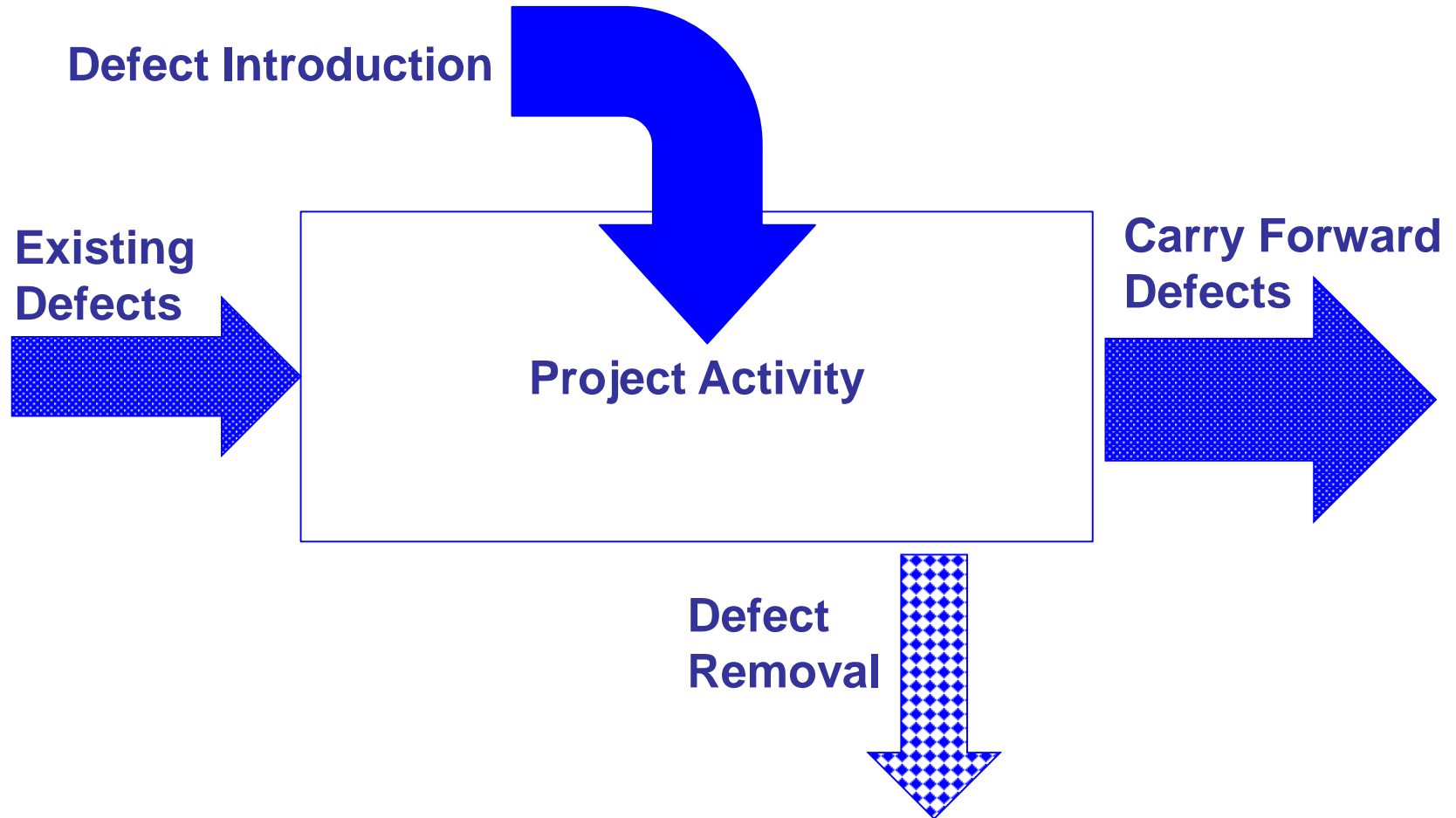
1.3 General Testing Principles

1.4 Fundamental Test Process

1.5 The psychology of Testing

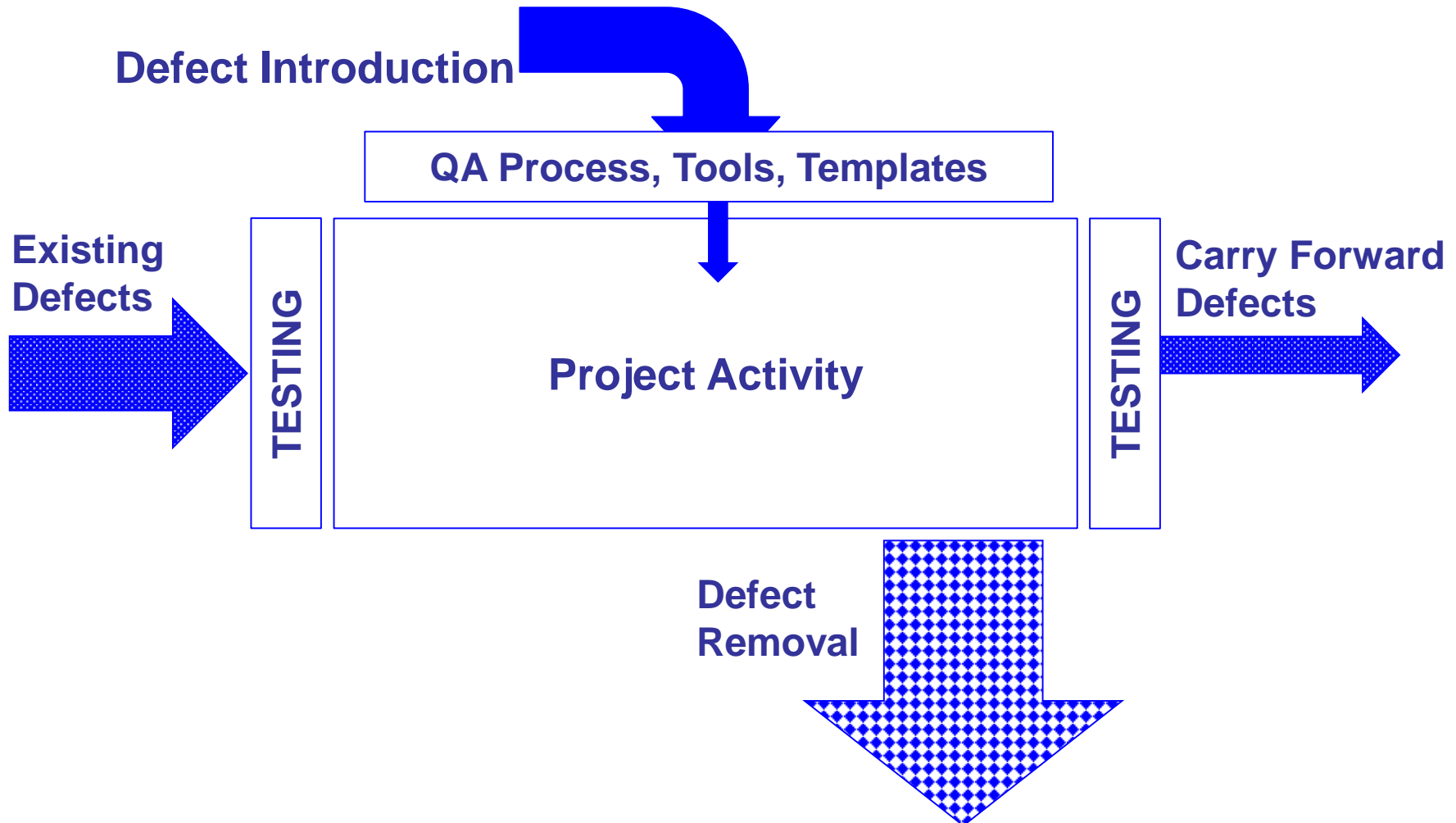
How defect happen?

❑ Without testing:

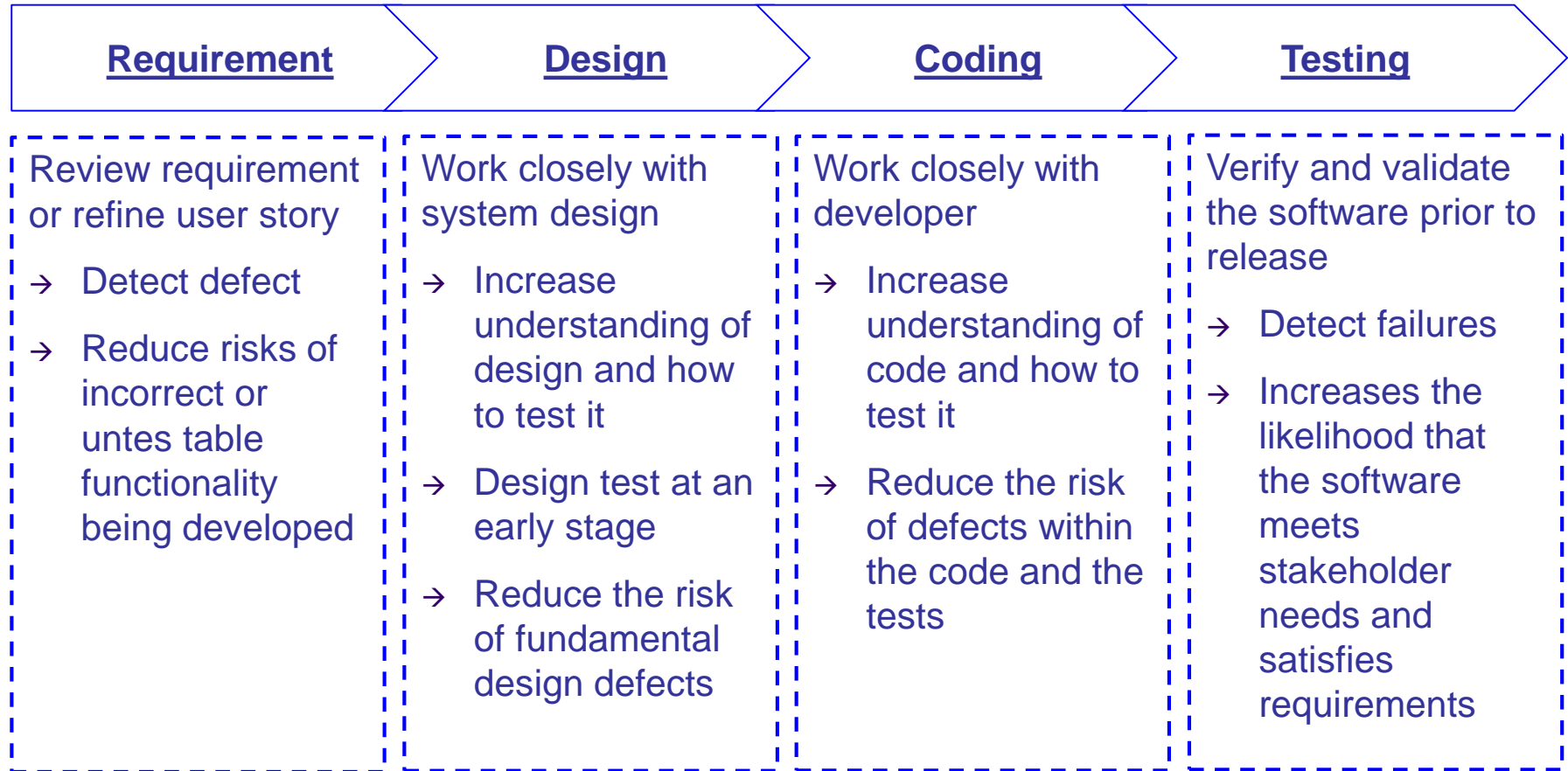


How defect happen?

❑ Having testing:



1.2.1 Testing's Contribution to Success



1.2.2 Quality Assurance and Testing

- ❑ Quality management:
 - Quality Assurance: PREVENT
 - Quality Control: DETECT

1.2.3 Errors, Defects, Failures

❑ Error/ Mistake:

- An human action that produce an incorrect result

❑ Defect/ Bug/ Fault:

- A flaw in a system that cause the sustem to fail to perform it's function.
- Caused by an error or mistake by a person

❑ Failure:

- A non-fulfillment of given requirement; a discrepancy between the actual behavior and user expected behavior.
- Caused by a defect in software or environment conditions

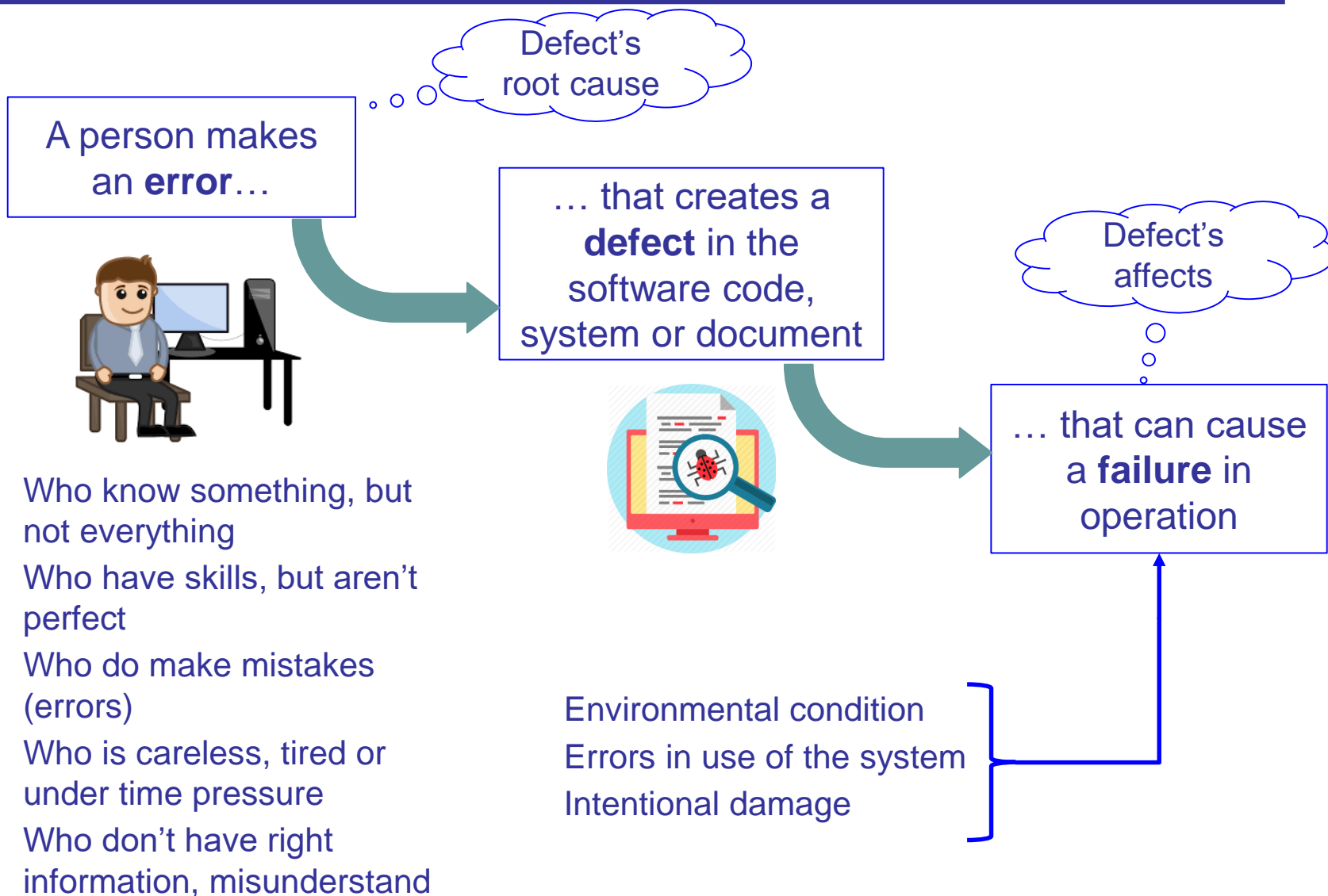
Definition: refer to ISTQB Glossary

1.2.3 Errors, Defects, Failures

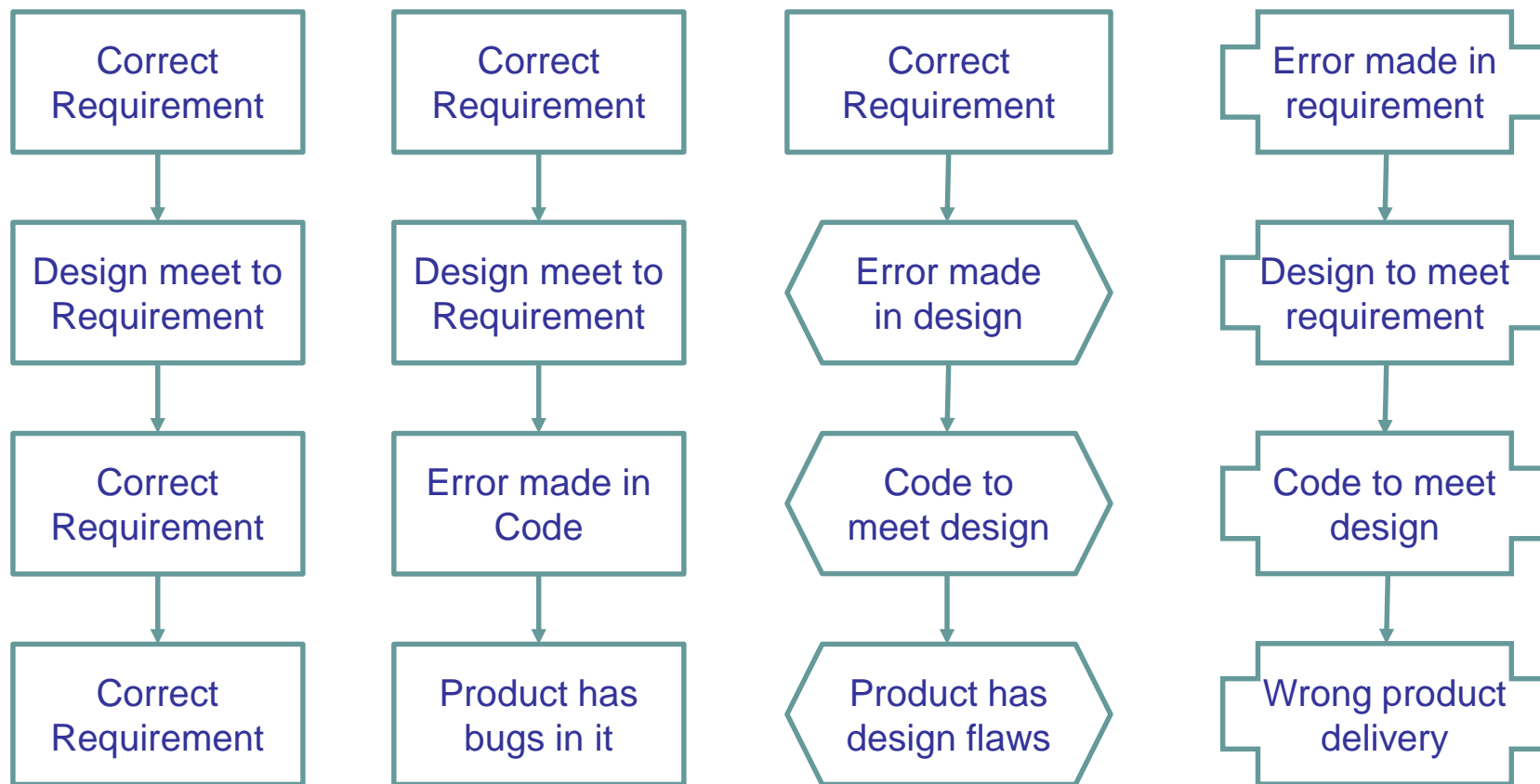
❑ Some Errors:

- Time pressure
- Human fallibility
- Inexperienced or insufficiently skilled project participants
- Miscommunication between project participants, including miscommunication about requirements and design
- Complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used
- Misunderstandings about intra-system and inter-system interfaces, especially when such intra-system and inter-system interactions are large in number
- New, unfamiliar technologies
- ...

1.2.4 Defects, Root Causes and Effects



1.2.4 Defects, Root Causes and Effects



1.2.5 Why testing is necessary?

❑ Example 1: Complexity of infrastructure:

- Contain over million lines of code
- 1 bug every 30 lines of code

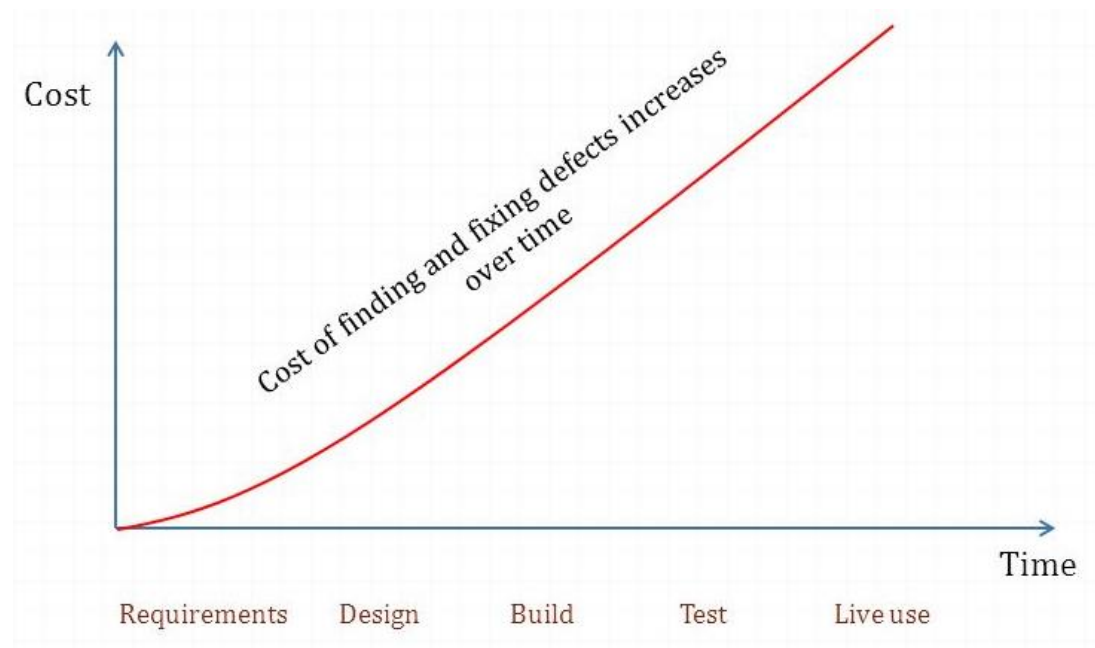
❑ Example 2: Safety critical systems

- Aircraft producer
- Radiation treatment (Therac-25)
- Satellite launch
- Bank systems
 - A bank which makes 5% of its revenue on ATM fees suffers an ATM network outage due to a software defect

1.2.5 Why testing is necessary?

❑ Cost of defect:

- Loss of money
- Loss of time
- Business reputation
- Even cause injury or death



1.2.5 Why testing is necessary?

❑ Impact of defect:

- Minor: Likelihood of customers to demand refunds
- Moderate:
 - Larger impact to sales based on references
 - Downstream effects of poor quality on sales of future version n+1
- Critical Financial Impacts:
 - Automotive software failure (\$7.5 million)
 - Medical product defect (\$7.76 million)
 - Mars orbiter failure (> \$300 million)
 - Defect on Container ship Balance Program (400 million)

1.2.5 Why testing is necessary?

❑ Benefits from testing:

- Save money
 - Save Time
 - Manual Testing: save 50 MD
 - Manual + Automation Tool: save 85 MD
 - Save Resource
 - ROI (Return on Invest) in Practice
- Raise Quality:
 - Find more bugs which are resolved prior to release
 - More Coverage
 - ...

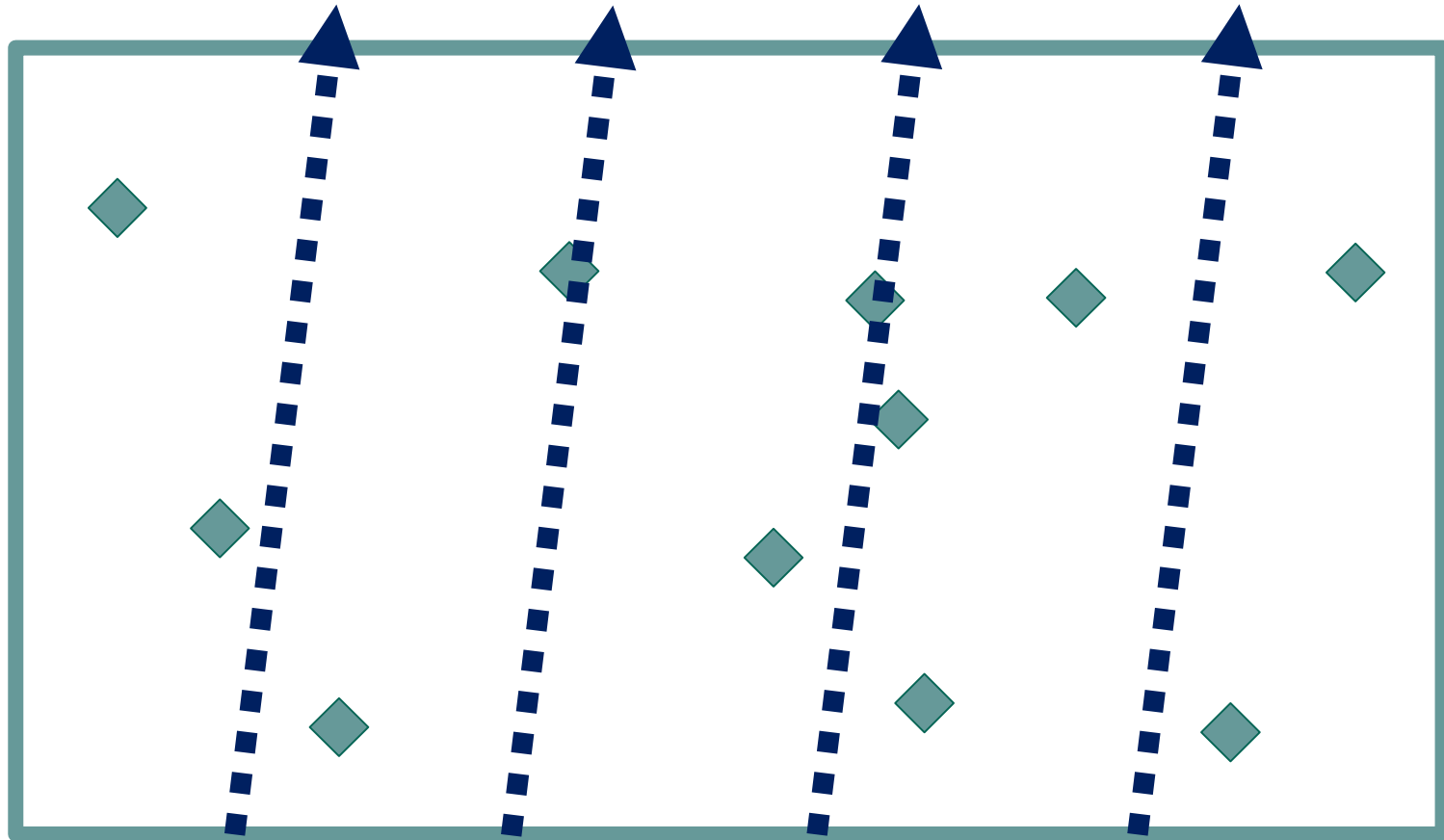
1.2.6 How much testing is enough?

- ❑ Should take account of the level of risk.
 - Technical
 - Business product
 - Project risks
 - Project constraints such as **TIME** and **BUDGET**
- ❑ Use RISK to determine: (e.g: i.e. where to place emphasis)
 - What to test first
 - What to test most
 - How thoroughly to test each item
 - What not to test (this time)
- ❑ Use RISK to
 - Allocate the time available for testing before prioritising testing

Good detect-detection percentage: 90-95% of defects present in system under test (SUT)

1.2.6 How much testing is enough?

□ Testing Analogy: Clearing Mines

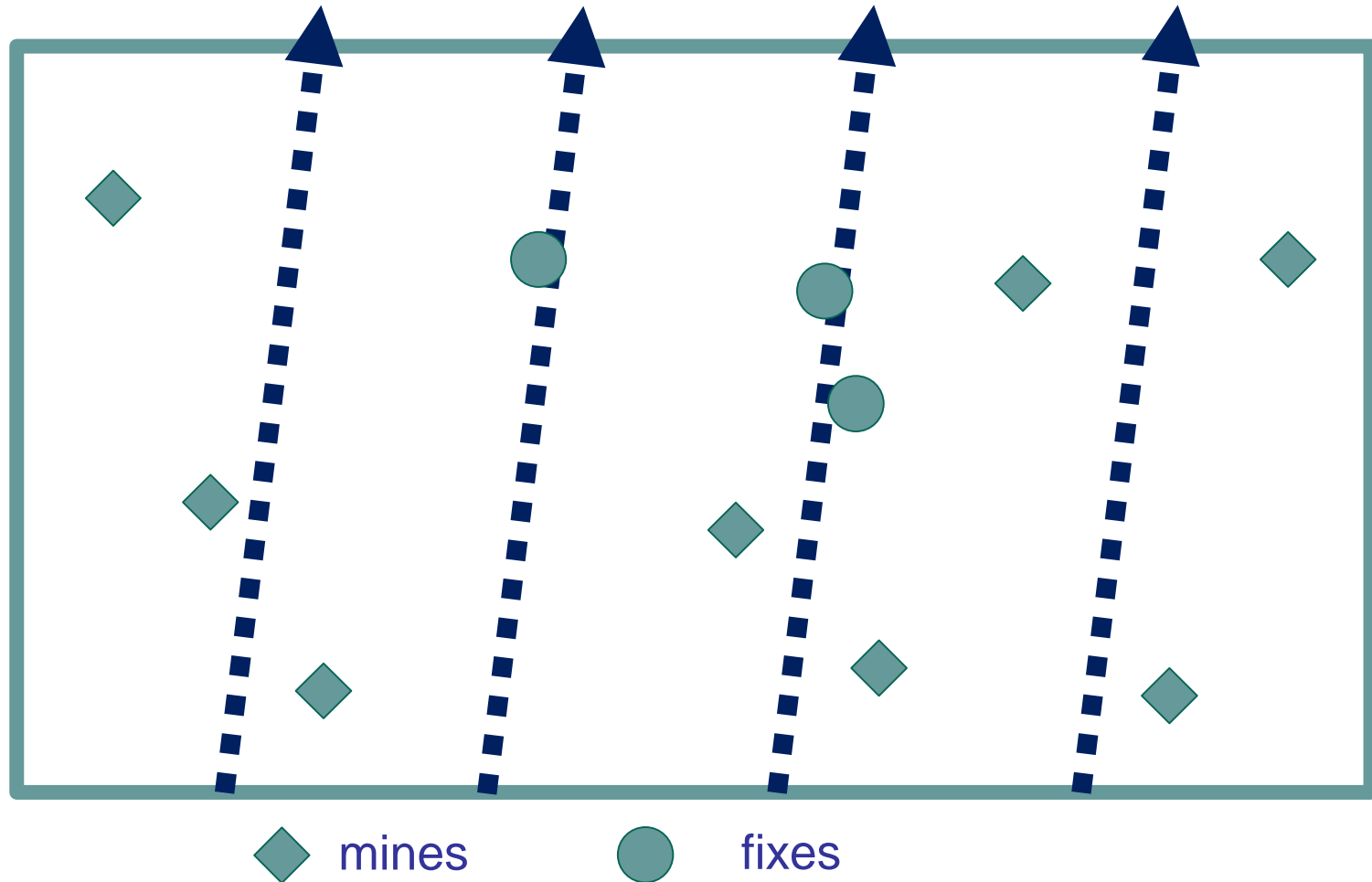


◆ mines

This analogy was first presented by Brian Marick.
These slides are from James Bach..

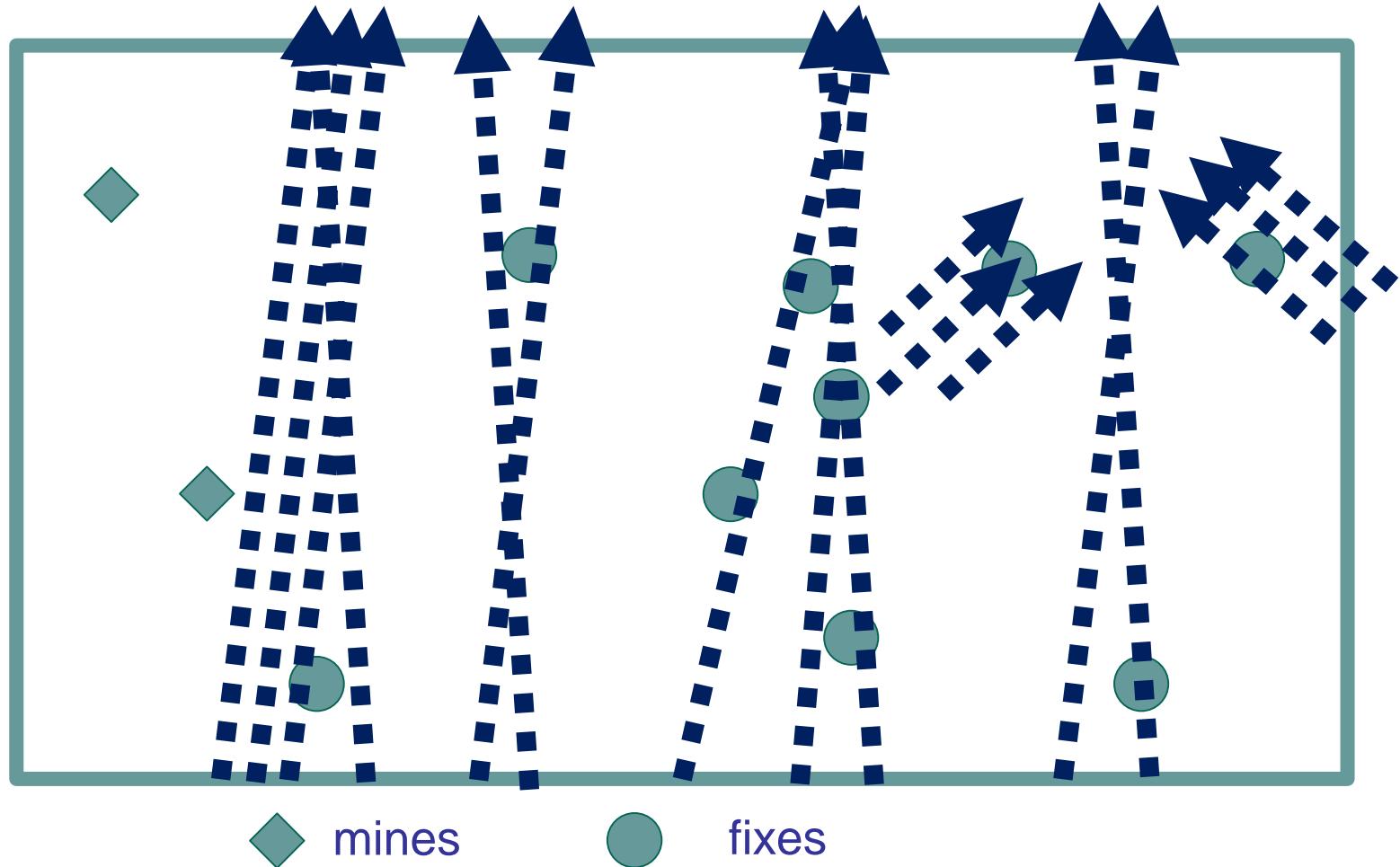
1.2.6 How much testing is enough?

❑ Clearing Mines: Found vs Not Found



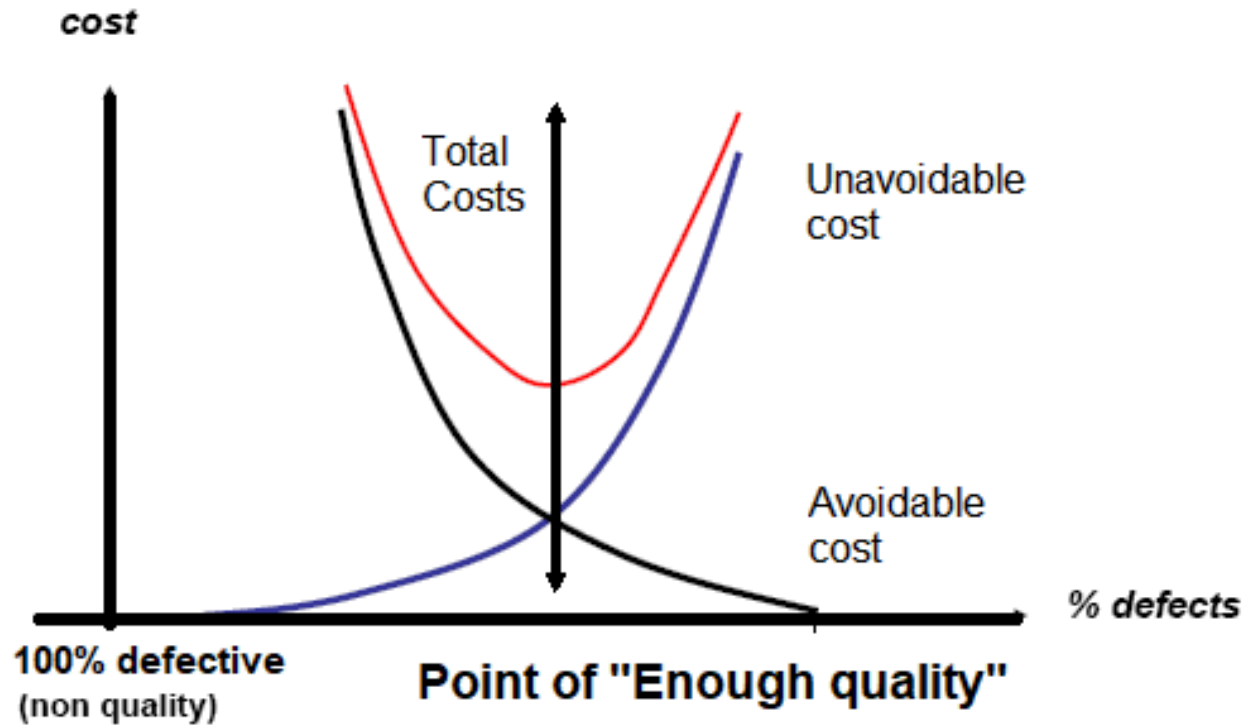
1.2.6 How much testing is enough?

- ❑ Clearing Mines: How many testing is needed?



1.2.6 How much testing is enough?

Cost of Quality



1.2.6 How much testing is enough?

❑ Testing Termination:

- Terminate when an adequacy criterion is met
 - A measure of how well the testing process has been performed
 - Relates a test set to the program, the specification or both
 - Could relate a test set to the program's operational profile

Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

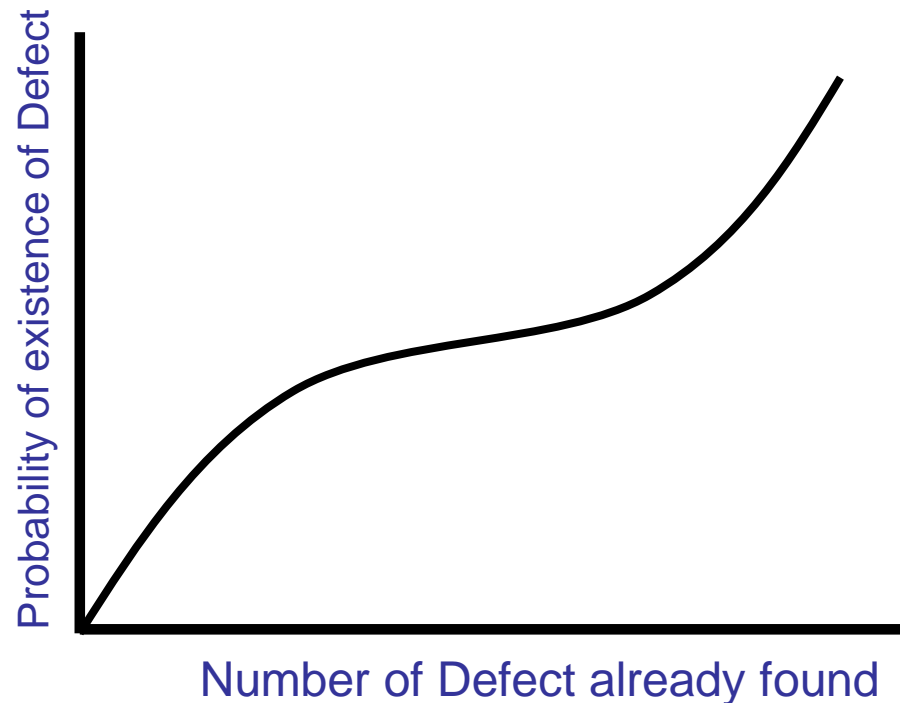
1.3 General Testing Principles

1.4 Fundamental Test Process

1.5 The psychology of Testing

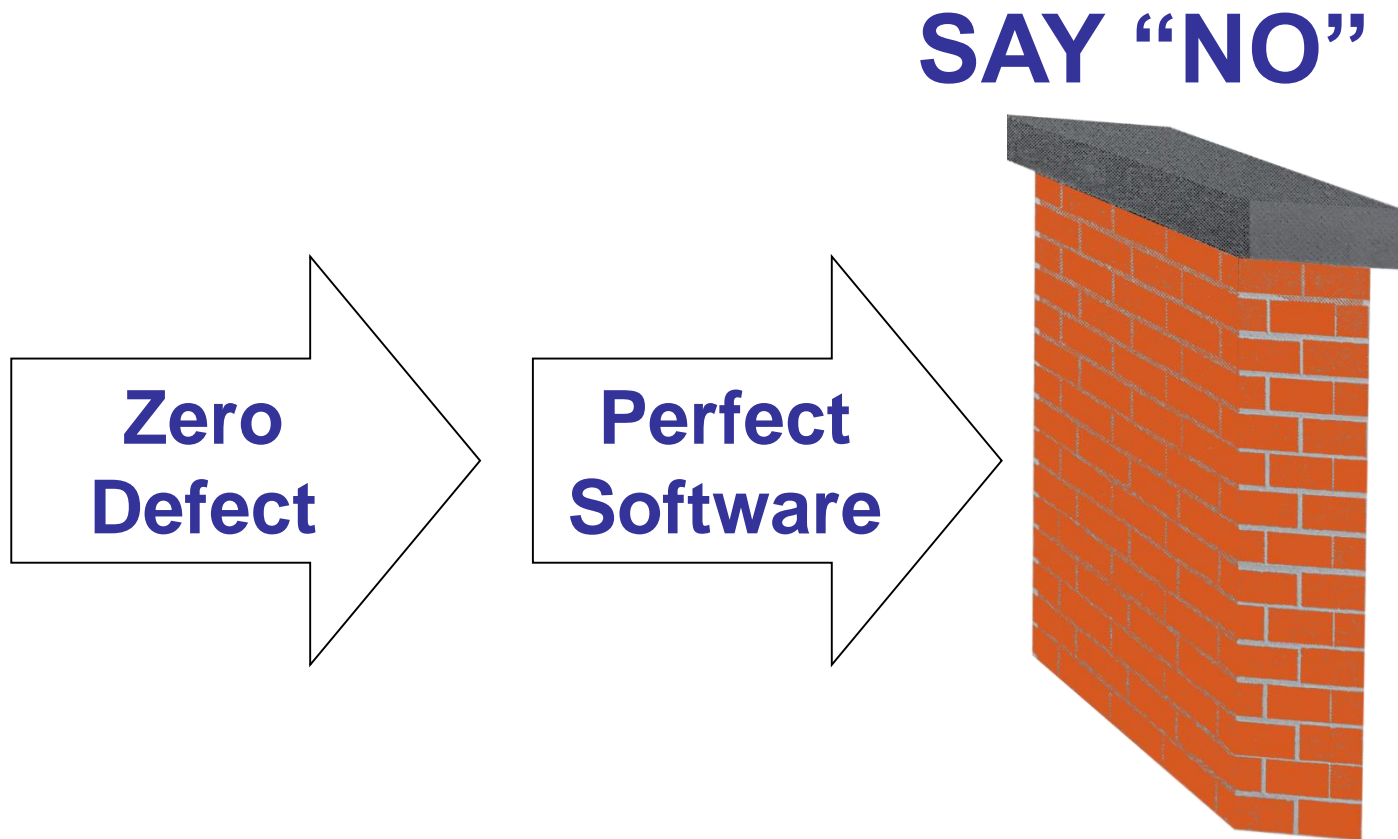
1.3.1 Testing shows presence of defects

- Can show that defects are present
- Cannot prove that there are no defects
- Reduces the probability of undiscovered defects remaining in the software
- Even if no defects are found, it is not a proof of correctness



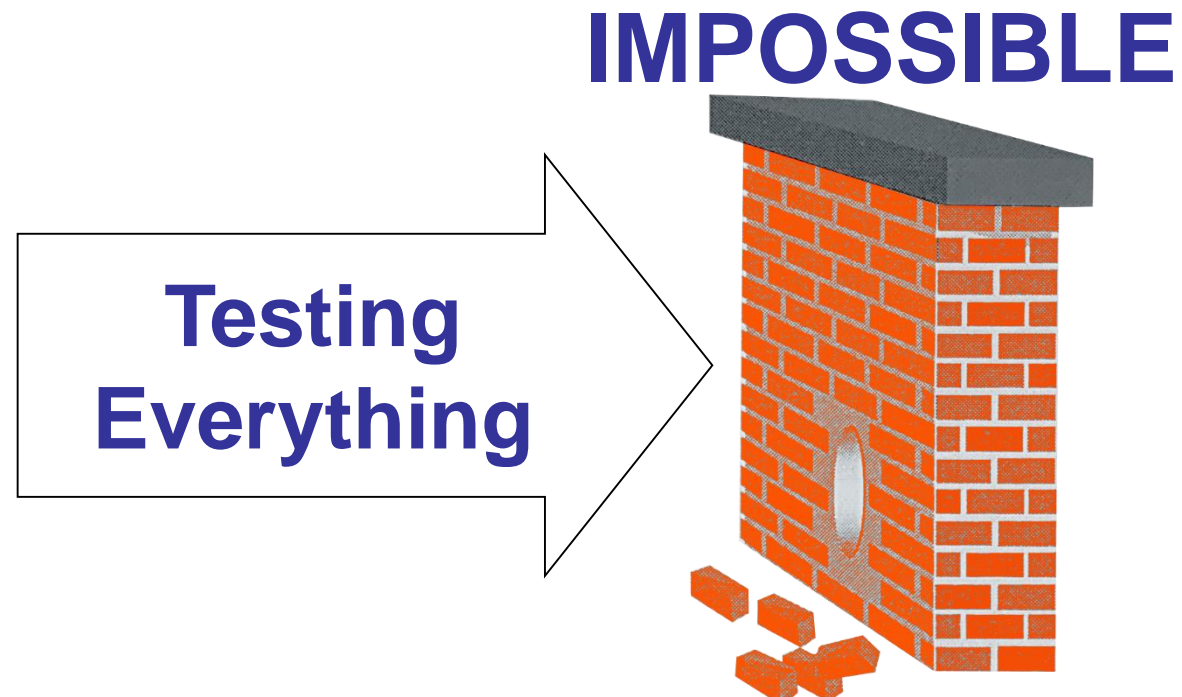
1.3.1 Testing shows presence of defects

- ❑ Zero Defect Policy



1.3.2 Exhaustive testing is impossible

- ❑ Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases
- ❑ The test effort must therefore be controlled, taking into account risk and priorities.



1.3.3 Early testing

□ Testing activities should start

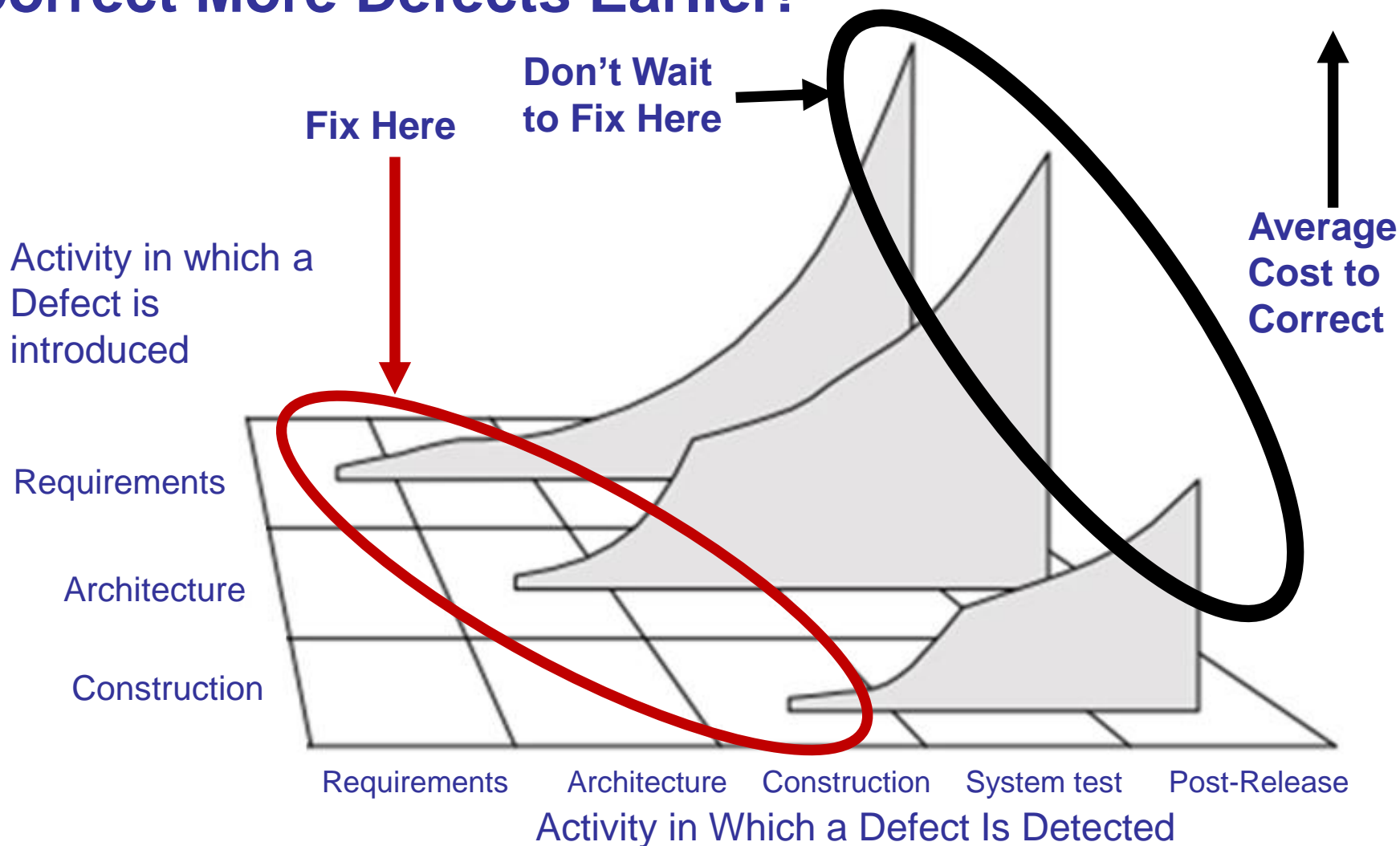
- **As early as possible**
- In the software or system development life cycle, and should be focused on defined objectives.

□ WHY?

- This contributes to finding defects early.
- ...

1.3.3 Early testing

Correct More Defects Earlier!



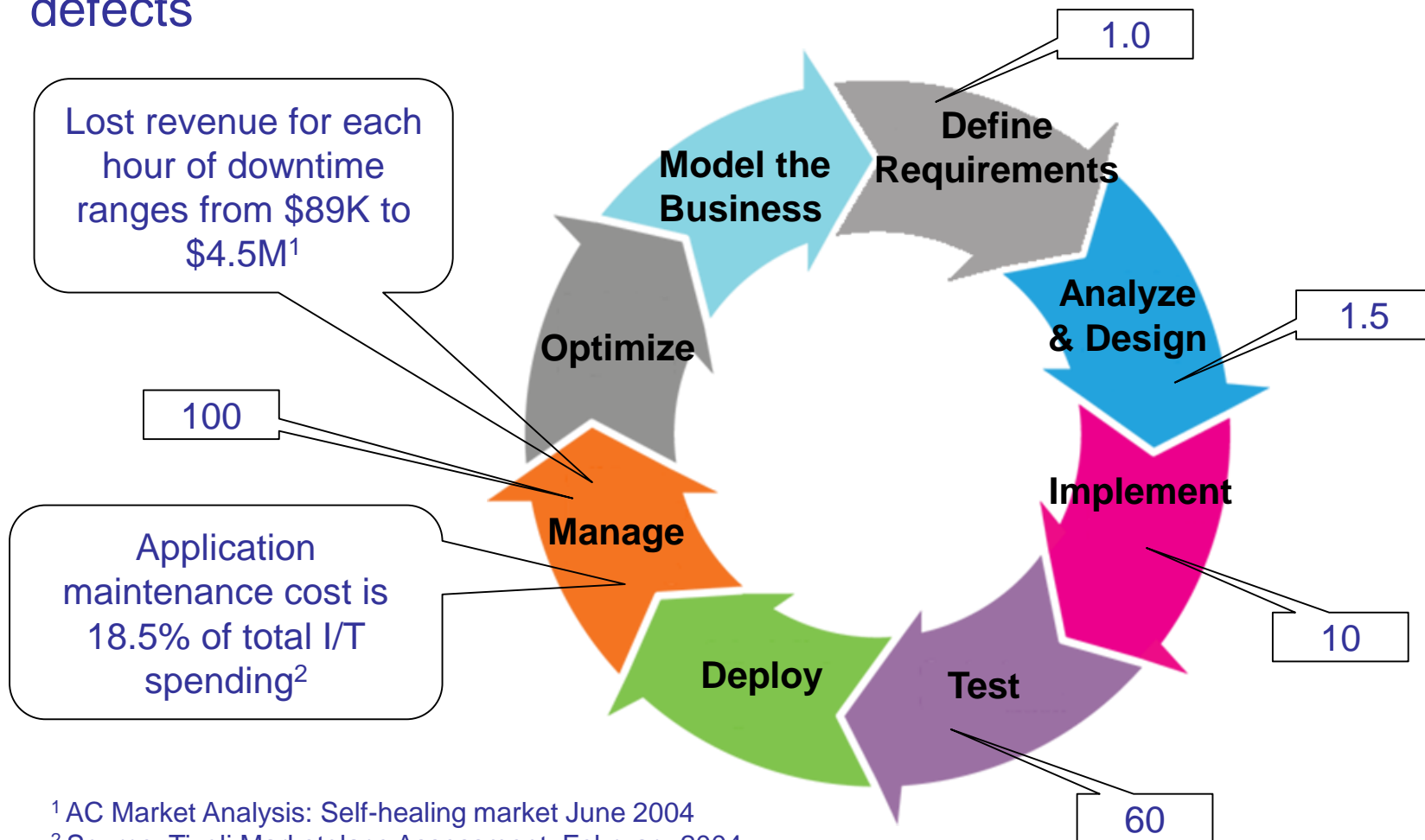
1.3.3 Early testing

❑ Relative cost of fixing a problem found in

	Remus 1983	Kan 1989
Design/coding	1\$	1
Testing	20\$	13
After Release	82\$	92

1.3.3 Early testing

- ❑ An IBM & Gartner "rule of thumb" for the relative costs to fix defects

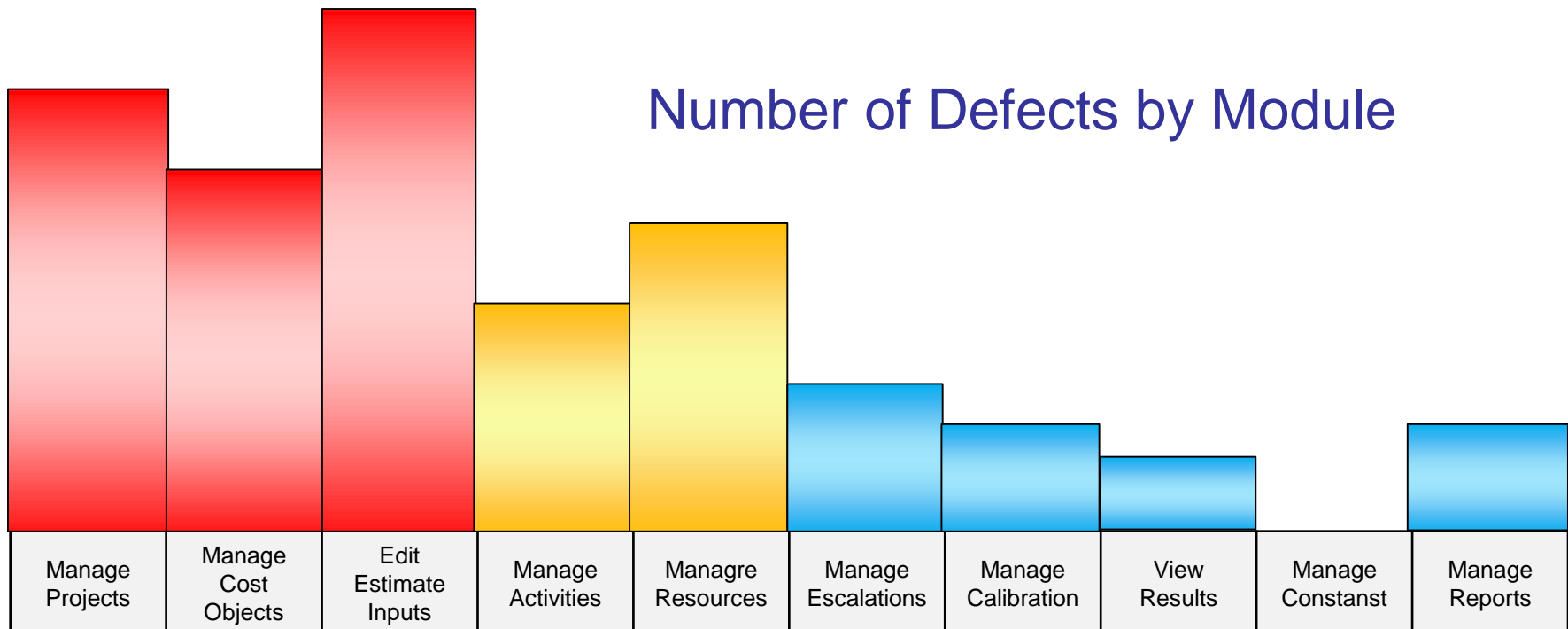


¹ AC Market Analysis: Self-healing market June 2004

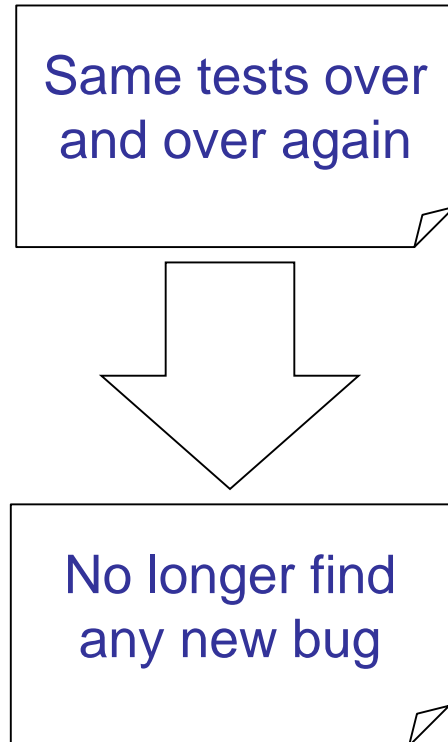
² Source: Tivoli Marketplace Assessment, February 2004

1.3.4 Defect Clustering

- ❑ Defect Clustering : A small number of modules contain most of the defects discovered.
- ❑ It happens a lot.



1.3.5 Pesticide paradox

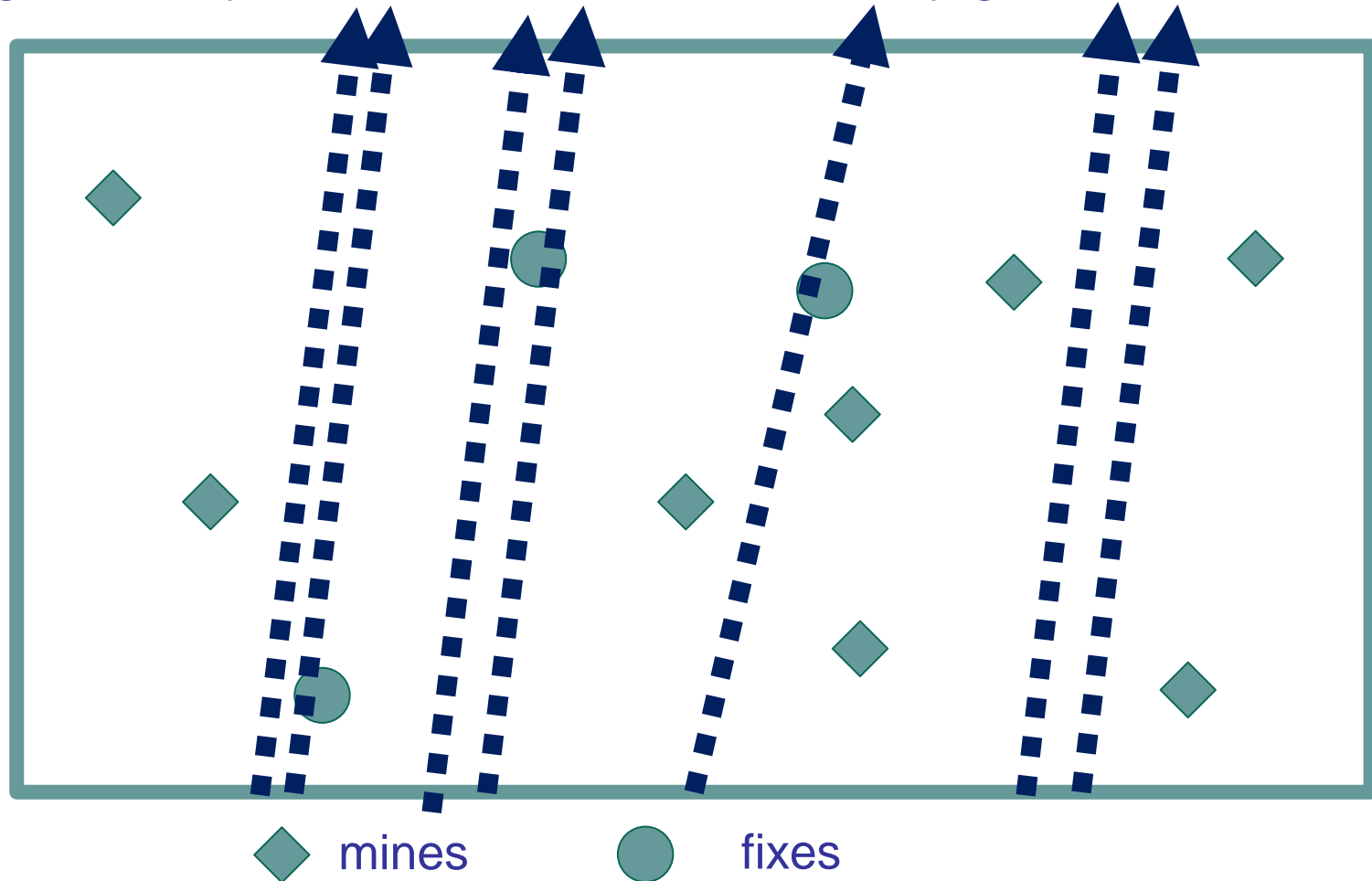


Pesticide Paradox



1.3.5 Beware of the pesticide paradox

- ❑ If you test 1000 times with same way (tests) over and over again, can you find other mines? You only got 3 fixes.



1.3.5 Beware of the pesticide paradox

- ❑ To overcome this “pesticide paradox”:
 - The test cases need to be regularly reviewed and revised,
 - New and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

1.3.6 Testing is context dependent

- ❑ Testing must be adapted to the risk inherent in the use and environment of the application
- ❑ No two system should be tested in the exactly same way
- ❑ For every software system, the test exit criteria,... should be designed upon individually, depending on its usage environment

Ex: Safety critical system different tests than e-commerce applications

1.3.7 Absence-of-errors is a fallacy

- ❑ Fallacy of assuming that no failures means a useful system
 - Finding and fixing defects does not help
 - If the system built is unusable
 - *Does not fulfill the users' needs and expectations*
- ❑ Early involvement of user in the development process and the use prototypes are preventive measures intended to avoid problems.

Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

1.3 General Testing Principles

1.4 Fundamental Test Process

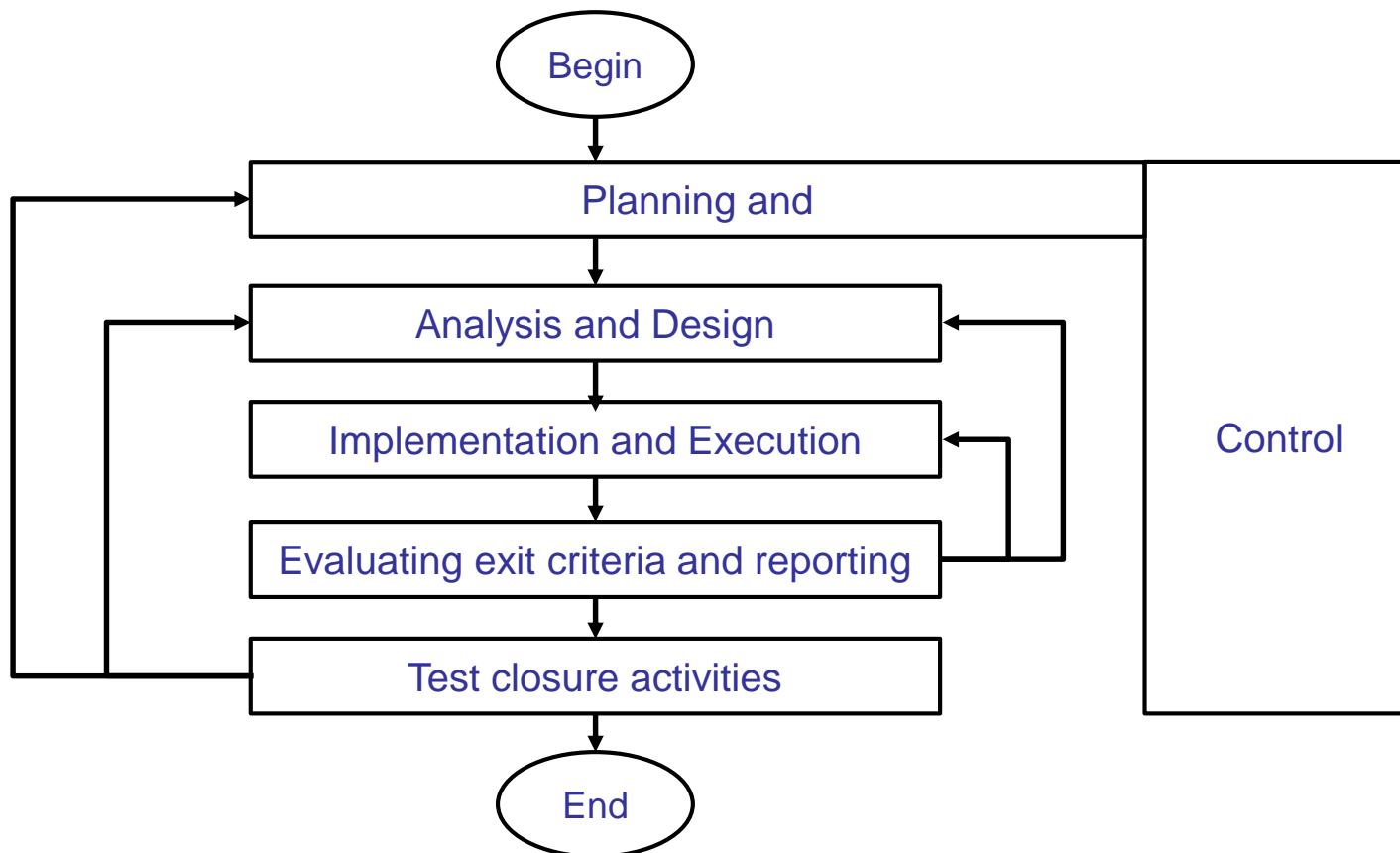
1.5 The psychology of Testing

1.4.1 Test Process in context

- ❑ Contextual factors that influence the test process for an organization:
 - Software development lifecycle model and project methodologies being used
 - Test levels and test types being considered
 - Product and project risks
 - Business domain
 - Operational constraints, including but not limited to:
 - Budgets and resources
 - Timescales
 - Complexity
 - Contractual and regulatory requirements
 - Organizational policies and practices
 - Required internal and external standards

1.4.2 Test activities and Tasks

- Fundamental test process is divided into the following basic steps:



1.4.2.1 Test planning task

□ Test planning

- Set goal and objectives for testing
- Determine the scope and risk
- Determine the test approach (techniques, test items, coverage, identifying and interfacing the teams involved in testing, testware)

1.4.2.2 Test control task

□ Test control major tasks:

- Measuring and analyzing results
- Monitoring and documenting progress, test coverage and exit criteria
- Initiation of corrective actions
- Making decisions

1.4.2.3 Test analysis

- ❑ **Test analyst:** determine 'WHAT' is to be tested:
 - Review test basis
 - Evaluate testability of the test basis and test objects
 - Identify and prioritize test condition based on analysis of test items, the specification, behavior and structure.

1.4.2.4 Test design

- ❑ **Test design:** determine 'HOW' the WHAT' is to be tested
 - Design and prioritize high level test cases:
 - Data input
 - Expected results
 - Design the test environment set-up and identifying any required infrastructure and tools.
 - Create Requirement Traceability Matrix between test basis and test cases

1.4.2.5 Test implementation

□ Test implementation

- Develop and prioritize test cases
- Develop test data
- Develop test procedures
- Prepare test harnesses
- Write automated test scripts
- Create test suite, test scenario
- Implement and verify the test environment

1.4.2.6 Test Execution

□ Test execution

- Execute the test suite and test cases
- Compare actual results with expected results
- Analyze anomalies and report defect
- Log the outcome of test execution (e.g., pass, fail, blocked)
- Execute the most important ones first
- Would not execute all test cases if
 - Testing only fault fixes
 - Too many faults found by early test cases
 - Time pressure
- Can be performed manually or automated

1.4.2.7 Test Completion

❑ Test completion includes major activities:

- Checking whether all defect reports are closed, entering change requests or product backlog items for any defects that remain unresolved at the end of test execution
- Creating a test summary report to be communicated to stakeholders
- Finalizing and archiving the test environment, the test data, the test infrastructure, and other testware for later reuse
- Handing over the testware to the maintenance teams, other project teams, and/or other stakeholders who could benefit from its use
- Analyzing lessons learned from the completed test activities to determine changes needed for future iterations, releases, and projects
- Using the information gathered to improve test process maturity

1.4.3 Test Work Products

- ❑ Test work products that describe:
 - How the system is tested (e.g., test strategies and plans),
 - That actually test the system (e.g., manual and automated tests) or
 - That present test results
- ❑ Types of test work products:
 - Test planning work products
 - Test monitoring and control work products
 - Test analysis work products
 - Test design work products
 - Test implementation work products
 - Test execution work products
 - Test completion work products

1.4.3 Test Work Products

- ❑ Test planning work products
 - Test plan
 - Exit criteria/ Definition of Done
 - Bi-directional Traceability (Test basic, other test work product)
- ❑ Test monitoring and control work products
 - Test progress report
 - Test summary report
- ❑ Test analysis work products
 - Defined and prioritized test conditions
 - Bi-directional Traceability (Test basic, Test condition)

1.4.3 Test Work Products

❑ Test design work products

- High-level test cases, without concrete values for input data and expected results
- Test Data
- Test Environment
- Infrastructure and tools
- Bi-directional Traceability (Test Basic - Test condition, Test case)

❑ Test implementation work products

- Test procedures and the sequencing of those test procedures
- Test suites
- A test execution schedule

1.4.3 Test Work Products

❑ Test execution work products

- Test report (e.g., ready to run, pass, fail, blocked, deliberately skipped, etc.)
- Defect report
- Documentation about which test item(s), test object(s), test tools, and testware were involved in the testing
- Bi-directional traceability (Test basis, Test case result - Defect) Test implementation work products

❑ Test completion work products

- Test Summary report
- Action items for improvement of subsequent projects or iterations
- Change request or Product back log
- Finalized Testware

1.4.4 Traceability between the Test Basis and Test Work Products

- Analyzing the impact of changes
- Making testing auditable
- Meeting IT governance criteria
- Improving the understandability of test progress reports and test summary reports to include the status of elements of the test basis (e.g., requirements that passed their tests, requirements that failed their tests, and requirements that have pending tests)
- Relating the technical aspects of testing to stakeholders in terms that they can understand
- Providing information to assess product quality, process capability, and project progress against business goals.

Fundamentals Of Software Testing

1.1 What is testing?

1.2 Why is testing necessary?

1.3 General Testing Principles

1.4 Fundamental Test Process

1.5 The psychology of Testing

1.5.1 The psychology of testing

- ❑ Design is fun

- ❑ Development is mostly fun
 - If you like problem solving

- ❑ Testing is often not fun
 - Shy away from bad news
 - Deadline pressure (release deadline)
 - Management lack of planning
 - Setup of tests costly

1.5.1 The psychology of testing

- ❑ Some psychological factors that influence testing and its success
 - Clear objective for testing
 - The proper roles (PM, test lead, test designer, tester...)
 - Balance of self-testing and independent testing
 - Clear and courteous communication and feedback on defects

1.5.1 The psychology of testing

❑ Clear Objective → to avoid argument during testing or after release about whether 'enough' testing has been done.

❑ E.g.

1. Want the testing to find as many defects as possible before software is released, which will take longer to do and require time for fixing, re-testing and regression testing

- Does what it shouldn't
- Doesn't do what it should

Goal: find faults

Success: system fails

Fastest achievement: difficult test cases — fewer faults left in

2. Want confirmation that the software works and is good enough if this is seen as a way of delivering as fast as possible

- Does what it shouldn't
- Doesn't do what it should

Goal: show working

Success: system works

Fastest achievement: easy test cases → Faults left in

1.5.1 The psychology of testing

□ Levels of independence

- Self-testing: tests designed by the person who wrote the software
- Tests designed by a different person
- Tests designed by someone from a different department or team (e.g. test team)
- Tests designed by someone from a different organisation (e.g. agency)
- Tests generated by a tool (low quality tests?)

1.5.1 The psychology of testing

- ❑ Need good interpersonal skills to communicate factual information about defects, progress and risks in a constructive way.
 - Without criticizing the person who created it
 - Don't go at - you are not perfect either
 - Don't blame
 - Be constructive critical and discuss in neutral way
 - Emphasize the benefits of testing. Explain we can work round it so the delivered system is better for client
 - Start with collaboration rather than battles

1.5.2 Tester's and Developer's Mindsets

❑ Tester's mindset

- Objective of testing: to verify and validate the product, find defects prior to release

together helps achieve a higher level of product quality

- Curiosity, professional pessimism, a critical eye, attention to detail, and a motivation for good and positive communications and relationships.
- A tester's mindset tends to grow and mature as the tester gains experience

❑ Developer's mindset

- Objective of development: to design and build a product

- Often more interested in designing and building solutions
- Able to test their own code
- Confirmation bias makes it difficult to find mistakes in their own work

Independent Tester

CHAPTER 2

Testing Throughout The Software Life Cycle

2. Testing Throughout The Software Life Cycle

2.1 Software Development Life Cycle (K2)

2.1.1 Sequential Model

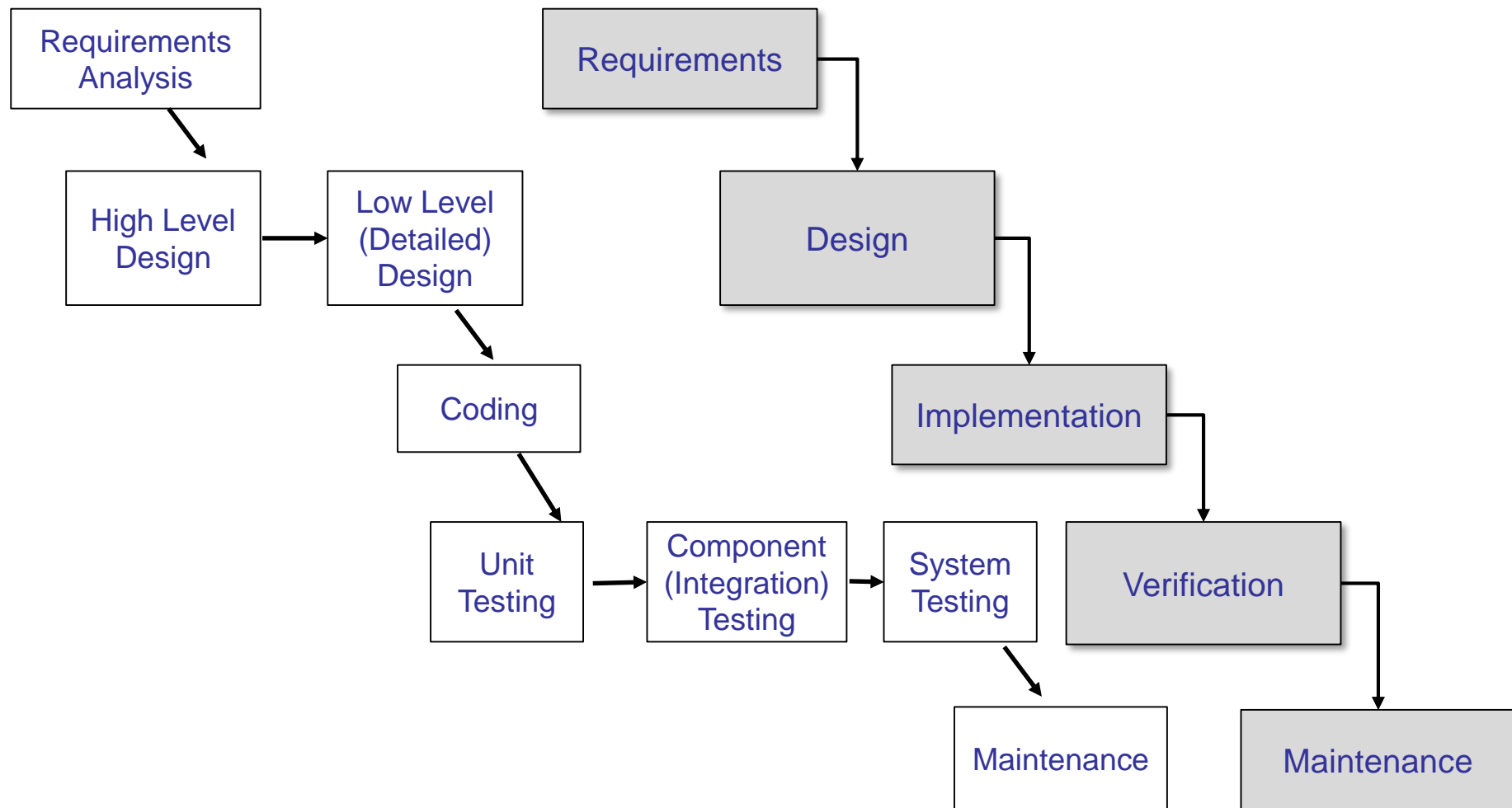
2.1.2 Iterative-Incremental Model

2.2 Test Level (K2)

2.3 Test Types (K2)

2.4 Maintenance Testing (K2)

2.1.1.1 The Waterfall Model



2.1.1.1 The Waterfall Model

❑ Requirements analysis phase

- Basic market research is performed
- Potential customer requirements are identified, evaluated, and refined.
- The result of this phase
 - A marketing requirement
 - Product concept specification (hereafter referred to as a concept specification)
- The requirements definition phase results in a document called the software requirements specification (SRS).

2.1.1.1 The Waterfall Model

❑ Design Phase

- Once the SRS is developed, software engineers should have a complete description of the requirements the software must implement.
- This enables software engineers to begin the design phase.

❑ Phase's task

- The overall software architecture is defined
- The high-level and detailed design work is performed.
- Output: Software design description (SDD).

2.1.1.1 The Waterfall Model

❑ Coding Phase

- The information contained in the SDD should be sufficient to begin to the coding phase.
- The design is transformed or implemented in code.
- If the SDD is complete, the coding phase proceeds smoothly, since all of the information needed by software engineers is contained in the SDD.

2.1.1.1 The Waterfall Model

□ Testing Phase

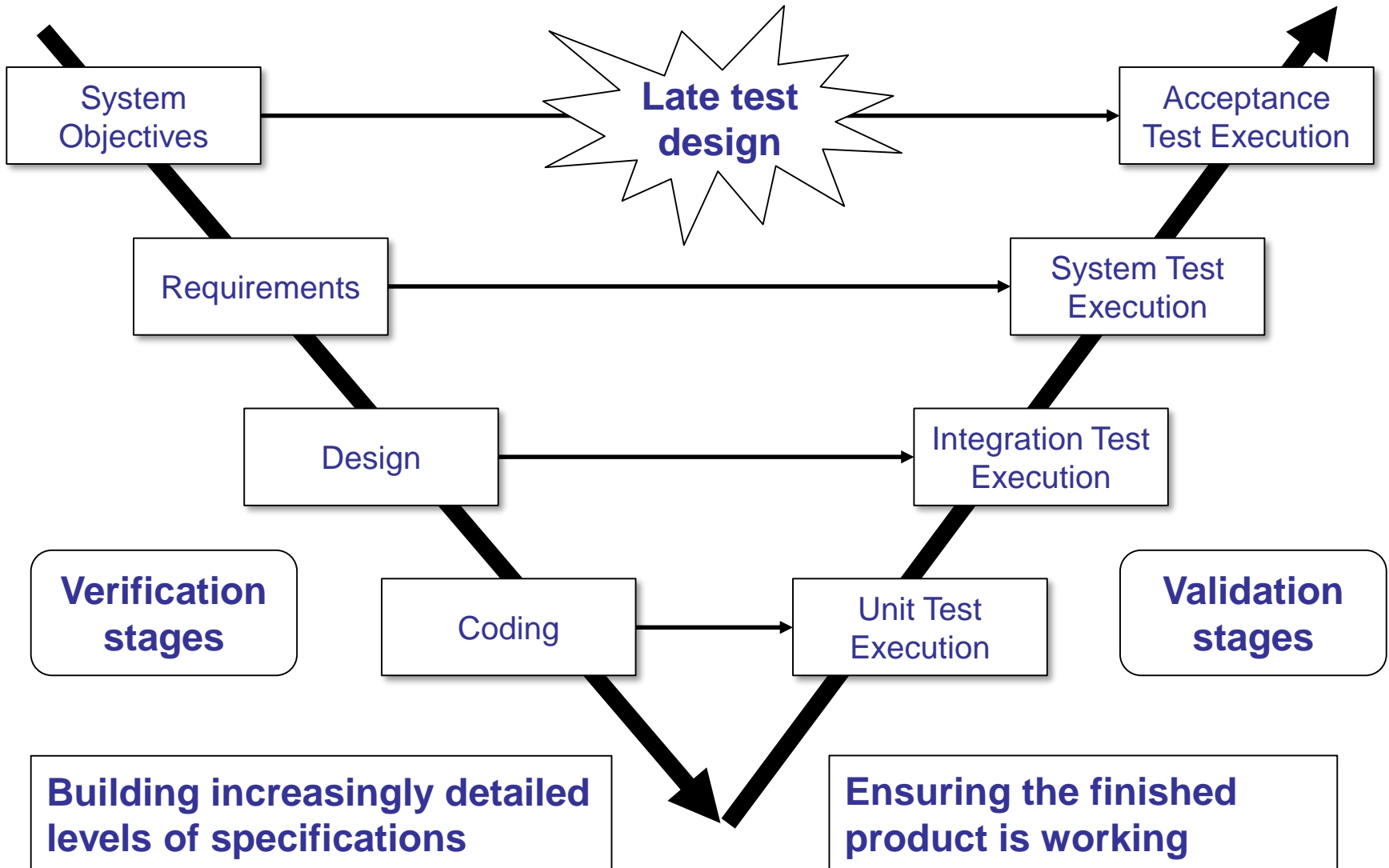
- The testing phase begins when the coding phase is completed.
- Tests are developed based on information contained in the SRS and the SDD.
- These tests determine if the software meets defined requirements.
- A software validation test plan is written which defines the overall validation testing process.
- Individual test procedures are developed based on a logical breakdown of requirements.
- The results of the testing activities are usually documented in a software validation test report.

2.1.1.1 The Waterfall Model

❑ Maintenance Phase

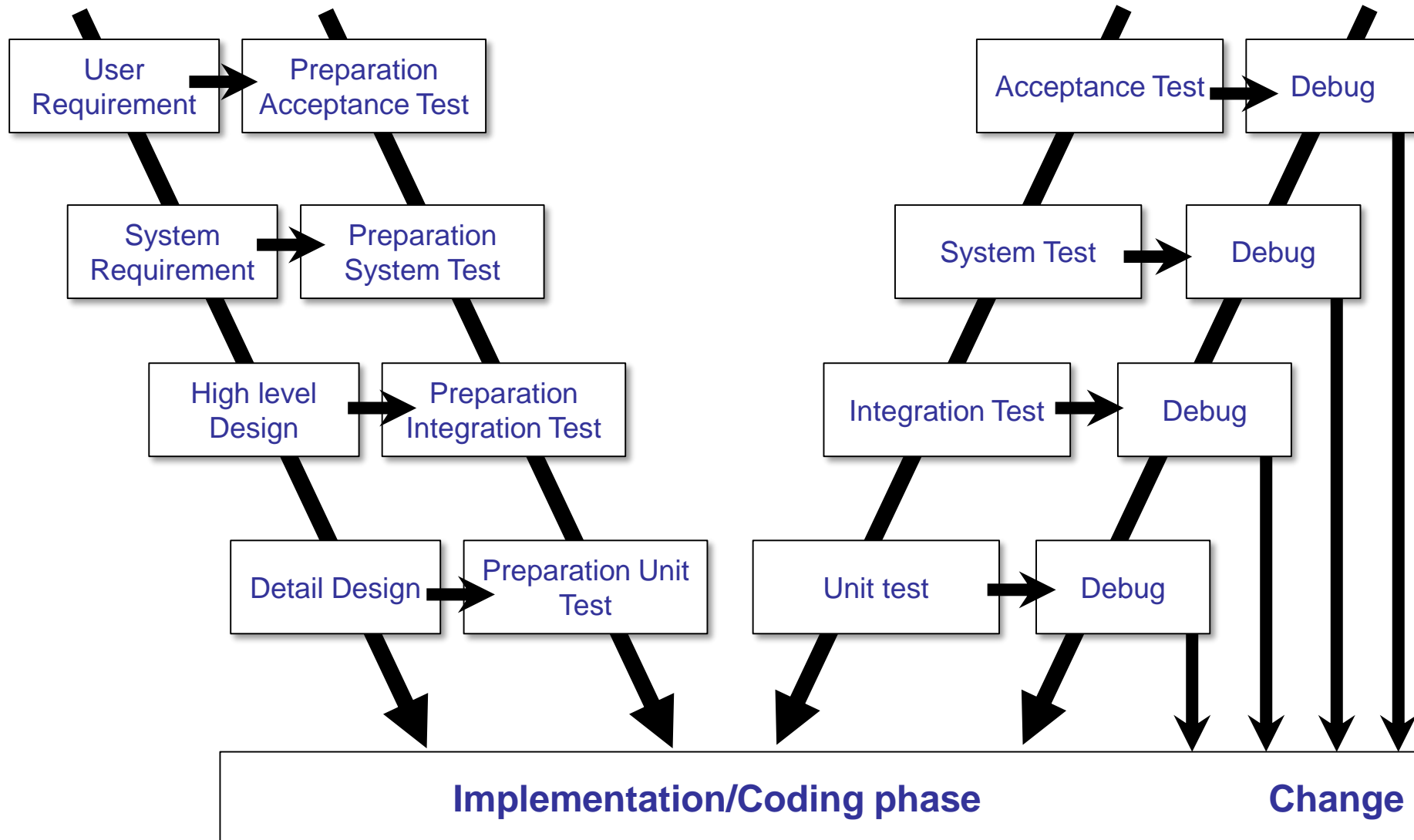
- Once the product is being shipped, the maintenance phase begins.
- This phase lasts until the support for the product is discontinued.
- Many of the same activities performed during the development phases are also performed during the maintenance phase.
- Good to write a software maintenance plan to describe how these activities will be performed.

2.1.1.2 The V-Model



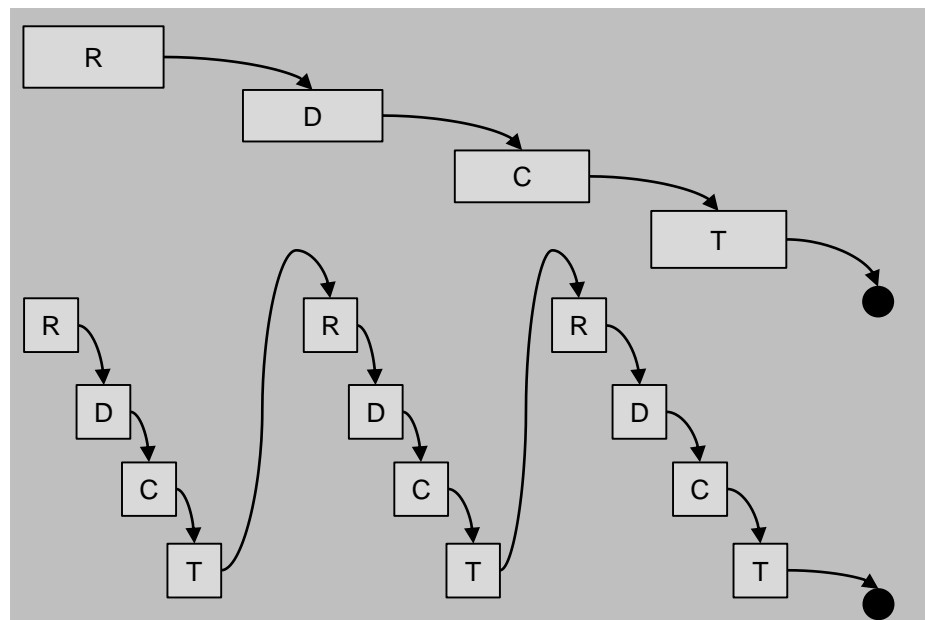
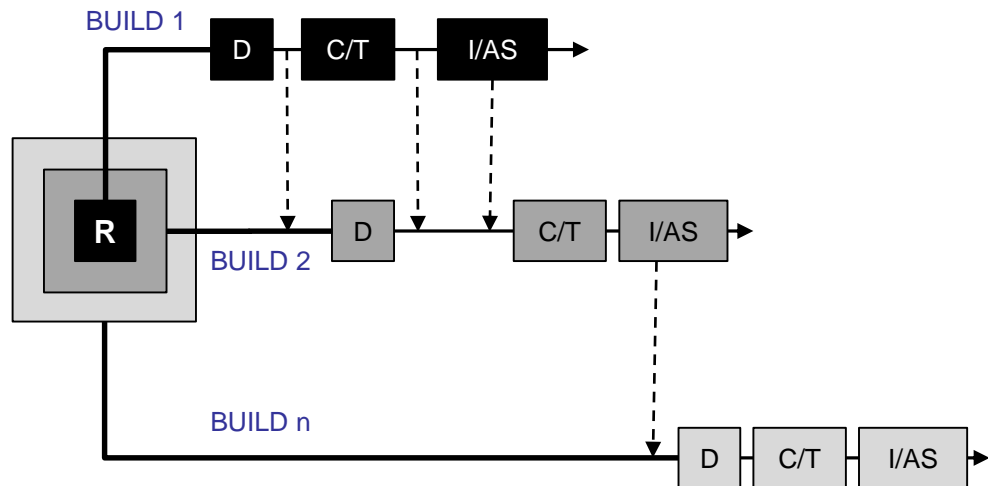


2.1.1.3 The W-Model



2.1.2 Iterative - Incremental development models

- ❑ The process of establishing requirements, designing, building and testing a system, done as a series of smaller developments.
- ❑ Examples
 - Agile approaches:
 - Scrum
 - Kanban
 - Rational Unified Process (RUP)
 - Prototyping
 - Spiral
- ❑ Regression testing: important on all iterations after the first one.
- ❑ Verification and validation can be carried out on each increment.

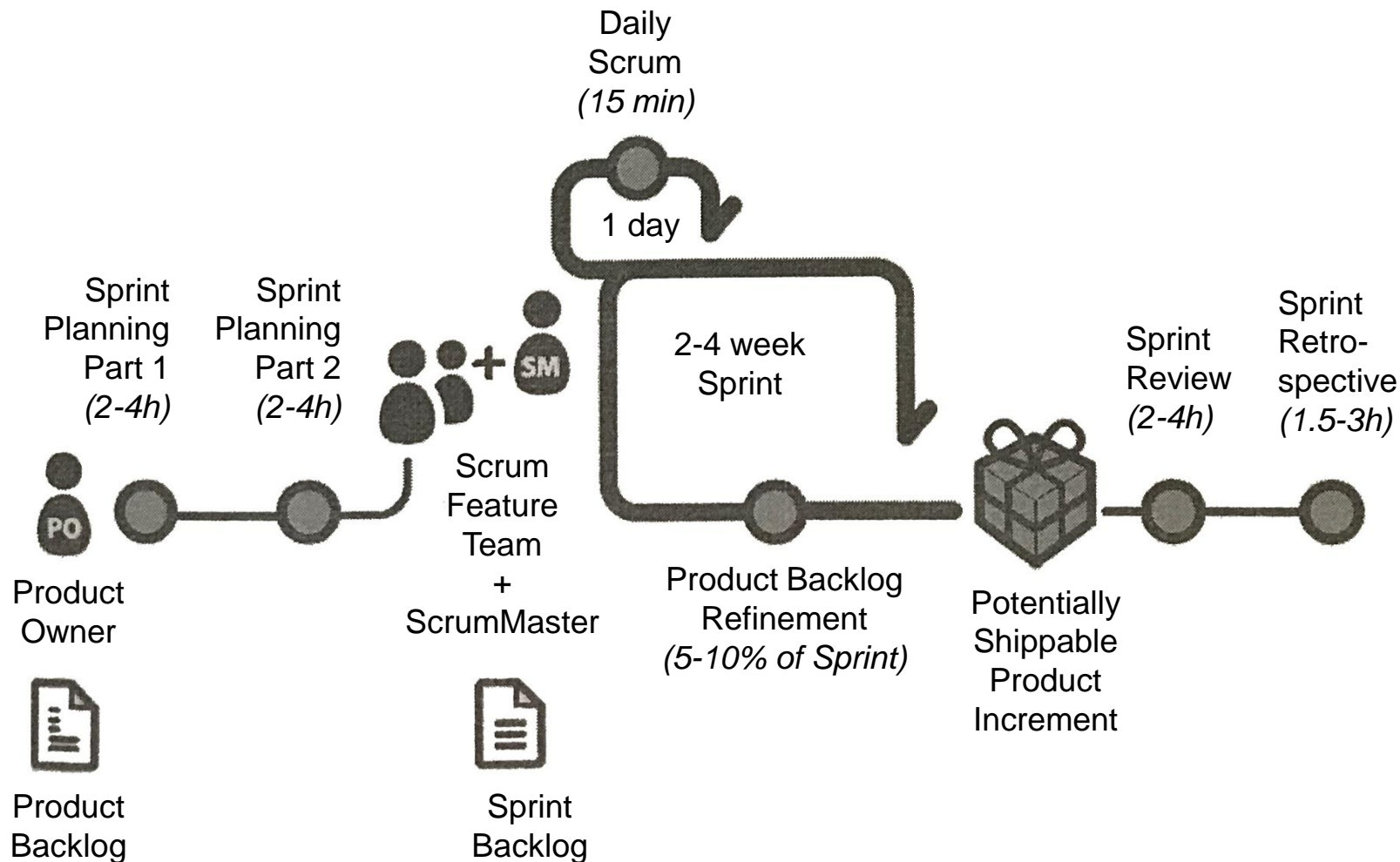


2.1.2.1 Agile approach

- ❑ Agile chooses to do things in small increments with minimal planning, rather than long-term planning
- ❑ Iterations are short time frames which typically last from 1 to 4 weeks.
- ❑ Each iteration is worked on by a team including planning, requirement analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders.
- ❑ This helps to minimize the overall risk, and allows the project to adapt to changes quickly.

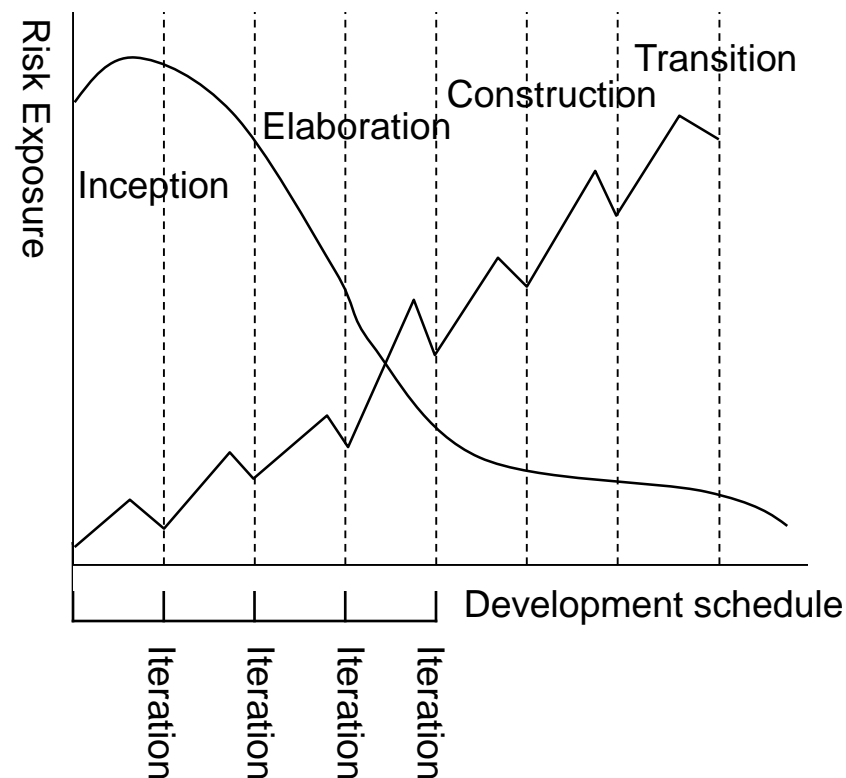
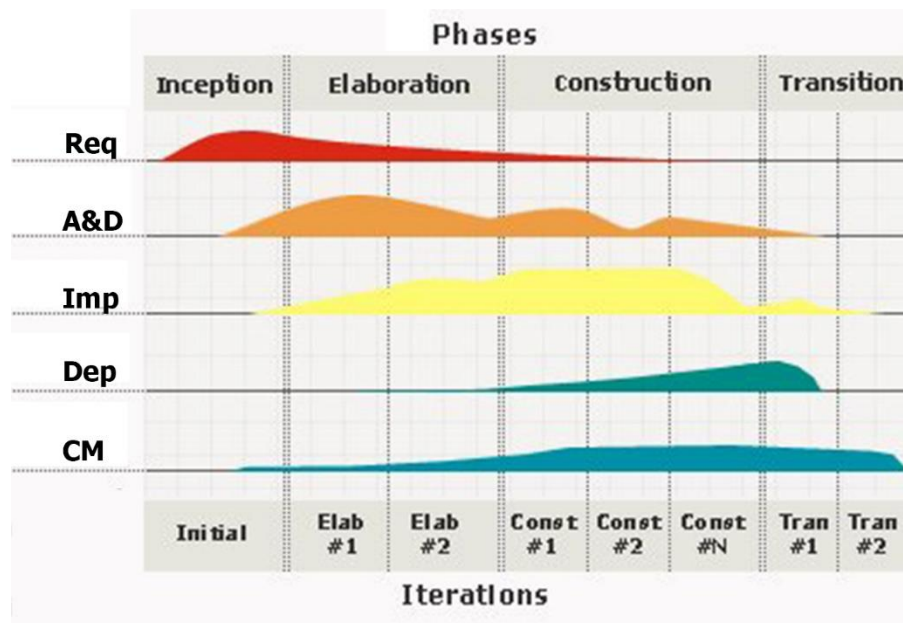
2.1.2.1 Agile approach

❑ Scrum – Agile management framework



2.1.2.2 Rational Unified Process (RUP)

- Rational Unified Process is an iterative and incremental method, aiding Rapid Application Development

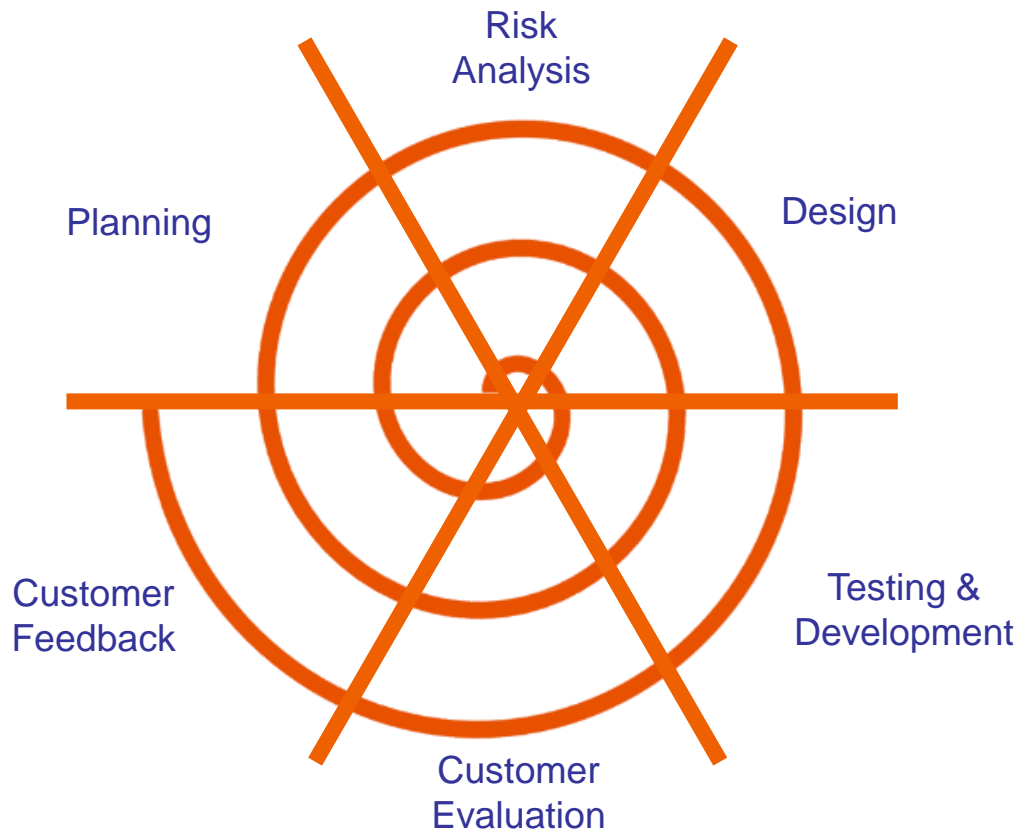


2.1.2.3 Prototyping model

- ❑ The Prototyping Model is a systems development method (SDM) in which a prototype is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.
- ❑ This model works best in scenarios where not all of the project requirements are known in detail ahead of time.
- ❑ It is an iterative, trial-and-error process that takes place between the developers and the users.

2.1.2.4 Spiral model

- ❑ Spiral Model Spiral: Involves creating experimental increments, some of which may be heavily re-worked or even abandoned in subsequent development work



Testing Throughout The Software Life Cycle

2.1 Software Development Life Cycle (K2)

2.1.1 Sequential Model

2.1.2 Iterative-Incremental Model

2.2 Test Level (K2)

2.3 Test Types (K2)

2.4 Maintenance Testing (K2)

2.2 Test Levels

Test Levels



Test Levels in V-Model:

1. Component testing
2. Integration testing
3. System testing
4. Acceptance testing

Test Techniques



Test Techniques:

1. Black-box testing
2. White-box testing
3. Experienced testing

Test Types



Test Types:

1. Functional test
2. Performance test
3. Load Test
4. Usability test...

❑ Test level:

- A group of test activities that are organized and managed together
- A test level is linked to the responsibilities in a project.

2.2.1 Component testing / Unit Testing

- ❑ Test objective: is to verify if each component performs correctly according to its specification
- ❑ Test basis
 - Component requirements
 - Detailed design
 - Code
- ❑ Typical test objects
 - Components
 - Programs
 - Data conversion / migration programs
 - Database modules
- ❑ Searches for defects in, and verifies the functioning of software (e.g. modules, programs, objects, classes, etc.) that are separately testable.
- ❑ Depending on the context of the development life cycle and the system.

2.2.1 Component testing / Unit Testing

- ❑ Component testing may include
 - Testing of functionality
 - Specific non-functional characteristics,
 - Efficiency testing (Resource-behavior (e.g. memory leaks))
 - Robustness testing
 - Structural testing (e.g. branch coverage).
- ❑ New test approach
 - Test-first approach or test-driven development (TDD)

2.2.1 Component testing / Unit Testing

❑ Refer to case study: Virtual Show Room (VSR)

```
double calculate_price (double baseprice, double specialprice, double extraprice, int extras,
                        double discount)
{
    double addon_discount;
    double result;

    if (extras >= 3) addon_discount = 10;
    else if (extras >= 5) addon_discount = 15;
    else addon_discount = 0;

    if (discount > addon_discount)
        addon_discount = discount;
    result = baseprice/100.0* (100-discount) + specialprice + extraprice/100.0* (100-
        addon_discount);
    return result;
}
```

2.2.1 Component testing / Unit Testing

❑ Refer to case study: Virtual Show Room (VSR)

```
bool test_calculate_price()
{
    double price;
    bool test_ok = TRUE;

    // testcase 01
    price = calculate_price (10000.00, 2000.00,1000.00,3,0);
    test_ok = test_ok && (abs (price-12900.00) < 0.01);

    // testcase 02
    price = calculate price (25500.00, 3450.00, 6000.00,6,0);
    test_ok = test ok e (abs (price-34050.00) < 0.01);

    // testcase ...
    // test result
    return test_ok;
}
```

2.2.2 Integration Testing

Unit-Component-Integration:

❑ Unit

- Individual classes or types

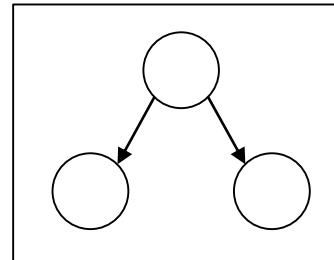


Test Concepts

- **Test harness**
- **Test stub**
- **Test driver**

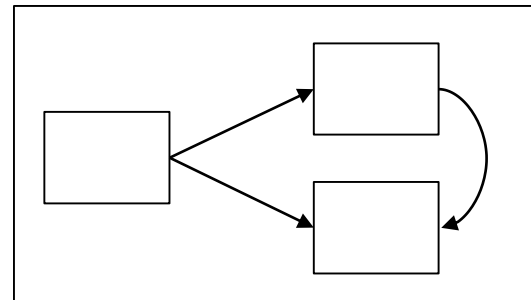
❑ Component

- Group of related classes or types



❑ Integration

- Interaction between classes



2.2.2 Integration Testing

❑ Test Objective

- is to test interface between components, interactions with different parts of a system, such as the operating system, file system, hardware, interfaces between systems.
- to expose faults in interfaces and interaction between integrated components.

❑ Test basis

- Software and system design
- Sequence diagram
- Interface and communication protocol specification
- Architecture at component or system level
- Workflows
- Use cases
- External interface definitions

❑ Typical test objects

- Subsystems
- Database implementation
- Infrastructure
- Interfaces
- APIs
- Microservices
- System configuration and configuration data

2.2.2 Integration Testing

❑ Integration testing may include

- Testing of Functionality
- Testing of Specific non-functional characteristics

❑ Typical defects and failures

- Incorrect data, missing data, or incorrect data encoding
- Incorrect sequencing or timing of interface calls
- Interface mismatch
- Failures in communication between components/ Systems
- Unhandled or improperly handled communication failures between components/ Systems
- Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components
- Failure to comply with mandatory security regulations (for System Integration Test)

❑ Integration testing levels

- Component-Integration: tests interactions between components
- System integration: tests interactions between different systems

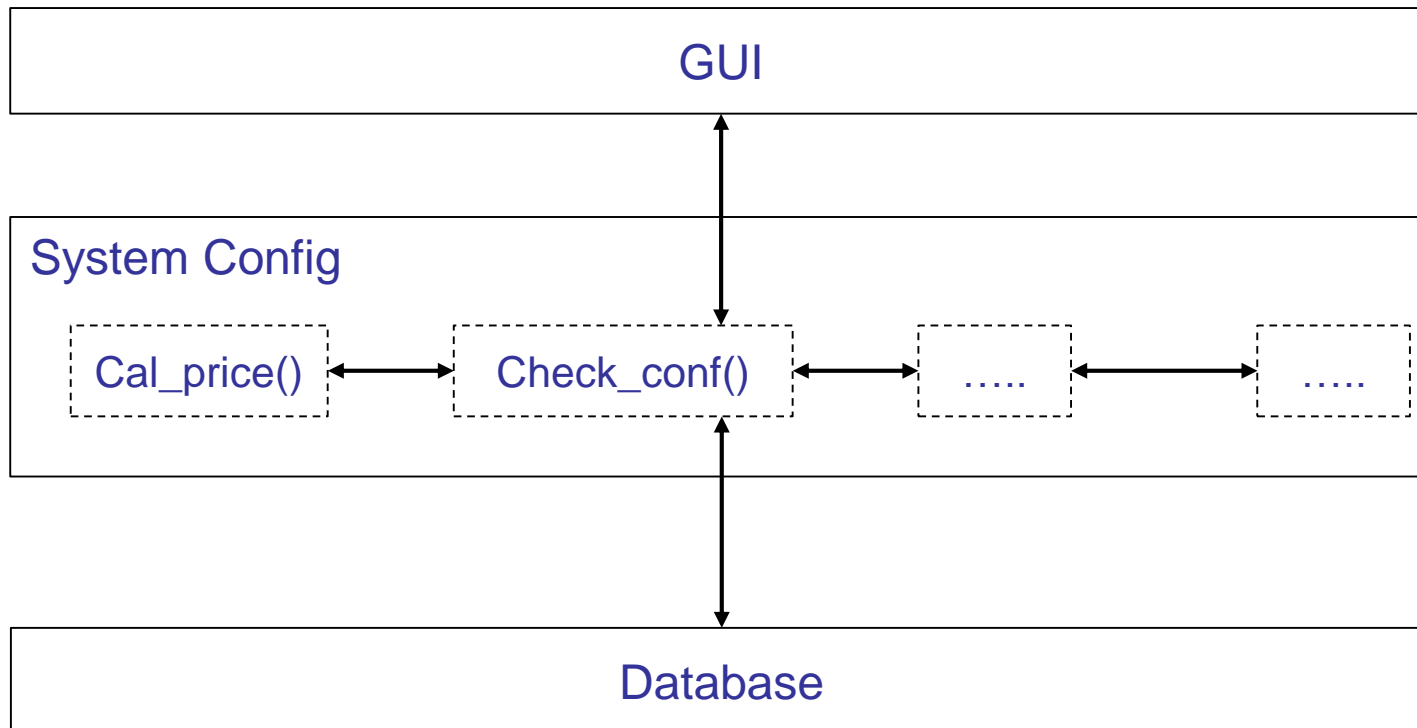
2.2.2 Integration Testing

❑ Test Strategy:

- Top-down Integration:
 - Advantage: Test drivers are not needed or only simple ones required
 - Disadvantage: Lower level components not yet integrated must be replaced by stubs → very costly.
- Bottom-up Integration
 - Advantage: No stubs are needed.
 - Disadvantage: Higher level components must be simulated by test drivers
- Ad-hoc Integration: The test starts with integrated components in the random order in which they are finished.
 - Advantage: Time saving as every finished component is integrated as early as possible.
 - Disadvantage: Stubs as well as test drivers are required.
- Backbone Integration Strategy (Functional Incremental): Integration and testing take place on the basis of the functions or functionality, as documented in the functional specs.

2.2.2 Integration Testing

- ❑ Refer to case study: Virtual Show Room (VSR)



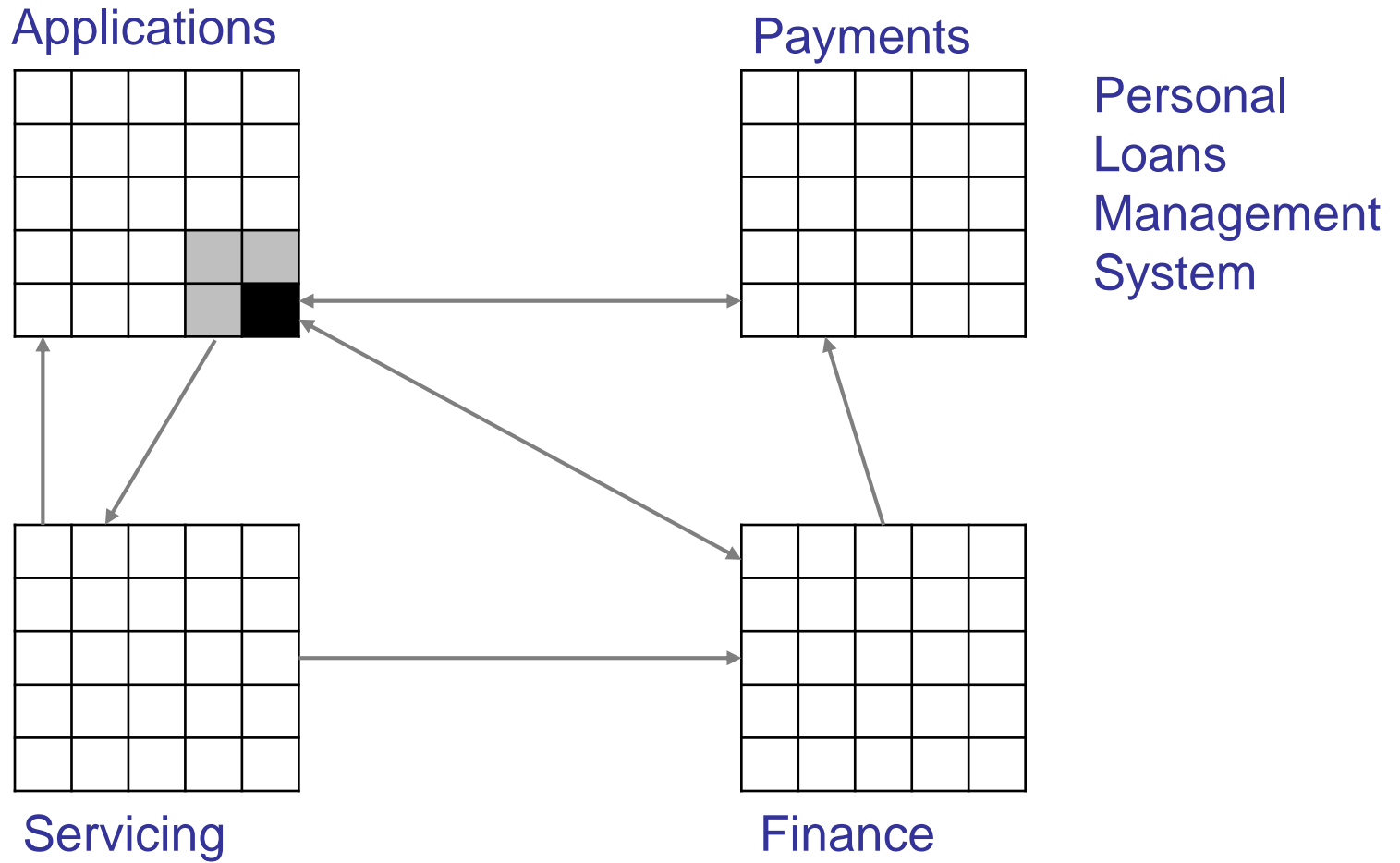
2.2.3 System Testing

- ❑ Test Objective: Verifies system as a whole meets the specified requirements.
- ❑ Test basis
 - System and software requirement specification
 - Risk analysis reports
 - Epics and User stories
 - Use cases
 - Models of system behavior
 - State diagrams
 - System and user manuals
- ❑ Typical test objects
 - Applications, System under test (SUT)
 - Hardware/ software system
 - User and operation manuals
 - System configuration and configuration data

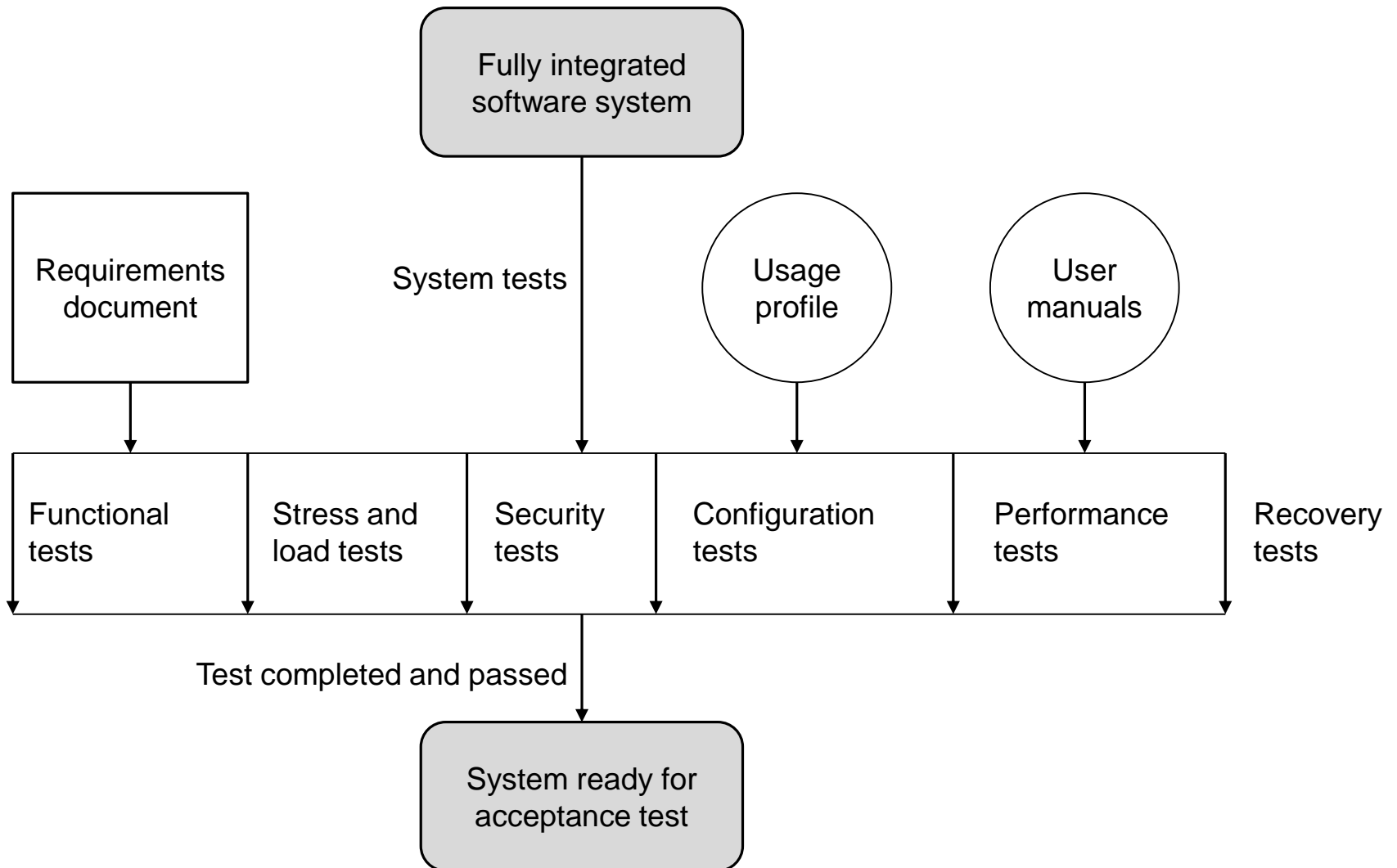
2.2.3 System Testing

- ❑ System testing may include
 - Functional test (Business confirmation test)
 - Non-functional test (Performance, Reliability, Security, ...)
- ❑ Typical defects and failures:
 - Incorrect calculations
 - Incorrect or unexpected system functional or non-functional behavior
 - Incorrect control and/or data flows within the system
 - Failure to properly and completely carry out end-to-end functional tasks
 - Failure of the system to work properly in the production environment(s)
- ❑ Test environment:
 - Should be separated - production-like environment
- ❑ Difficulties:
 - Unclear system requirements
 - Missed decisions from earlier phases will cause trouble in determining failure and root cause. Bugs now cost too much because it's late to change design

2.2.3 System Testing

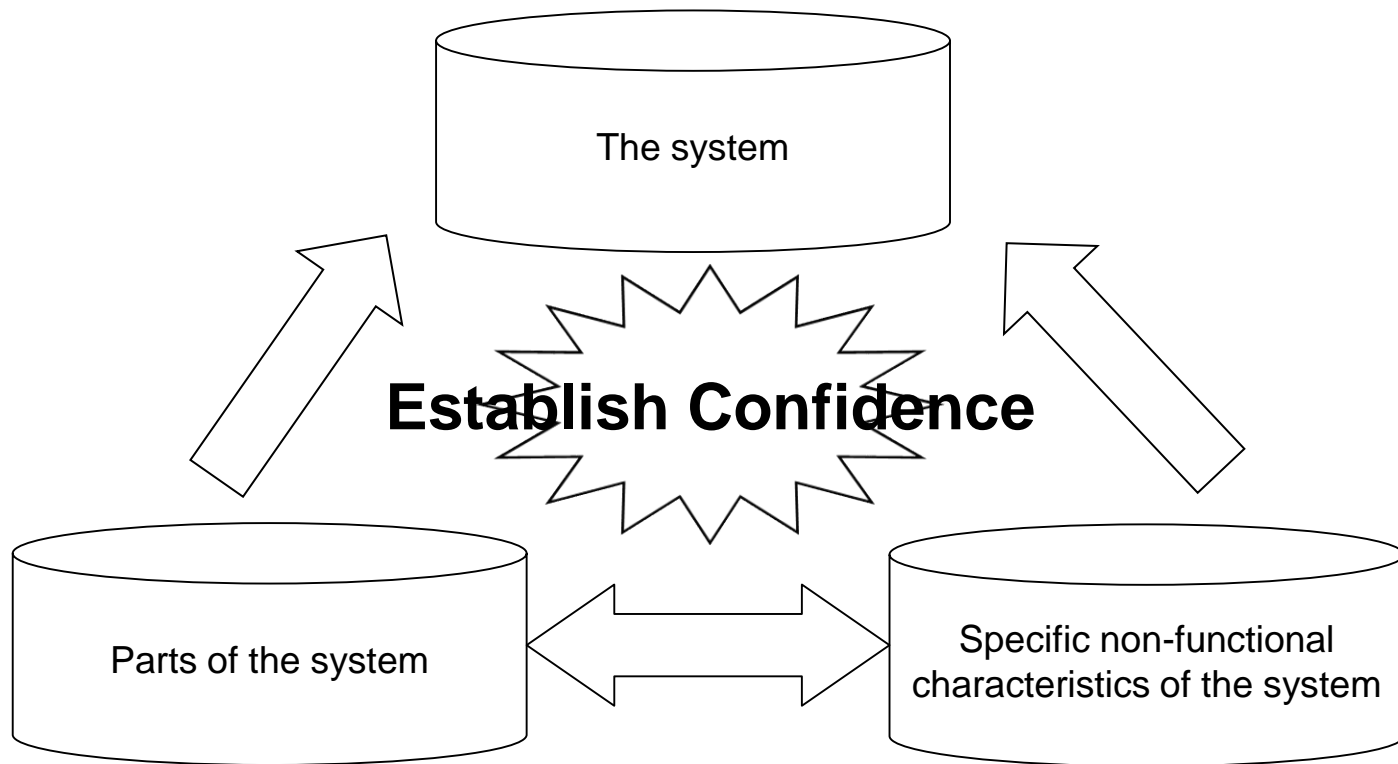


2.2.3 System Testing



2.2.4 Acceptance testing

- ❑ Test Objective is to establish confidence in the system, part of the system or specific non-functional characteristics, e.g. usability, of the system.



2.2.4 Acceptance testing

- ❑ **Responsibility:** customers or users of a system; Other stakeholders may be involved as well.
- ❑ Finding defects is not the main focus in acceptance testing.
- ❑ Acceptance testing may assess the system's readiness for deployment and use, although it is not necessarily the final level of testing.
 - For example, a large-scale system integration test may come after the acceptance test for a system.
- ❑ Acceptance test can perform on more than a single test level:
 - Component test: Usability of a component
 - Integration test: A COTS software product is integrated
 - Before system test: New functional enhancement

2.2.4.1 User Acceptance testing

- ❑ Typically verifies the fitness for use of the system by business users .
- ❑ The main objective is building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost and risk.

Business Processes	Operational Processing	Exceptions
System Functionality		Management Information
Training	Data	Performance

2.2.4.2 Operational acceptance testing

- ❑ The acceptance of the system by the system administrators, including:
 - Testing of backup/restore
 - Installing, uninstalling and upgrading
 - Disaster recovery
 - User management
 - Maintenance tasks
 - Data load and migration tasks
 - Periodic checks of security vulnerabilities
 - Performance testing

2.2.4.3 Contract/regulation acceptance testing

☐ Contract acceptance testing

- Perform against a contract's acceptance criteria for producing custom-developed software.

☐ Acceptance criteria should be defined when the contract is agreed.

2.2.4.4 Field testing

- ❑ Testing implemented by pre-selected customers/users who represent the market for software under test.
 - Alpha Testing: carried out at the developer's site
 - Beta Testing: carried out at the customer's site

Testing Throughout The Software Life Cycle

2.1 Software Development Life Cycle (K2)

2.1.1 Sequential Model

2.1.2 Iterative-Incremental Model

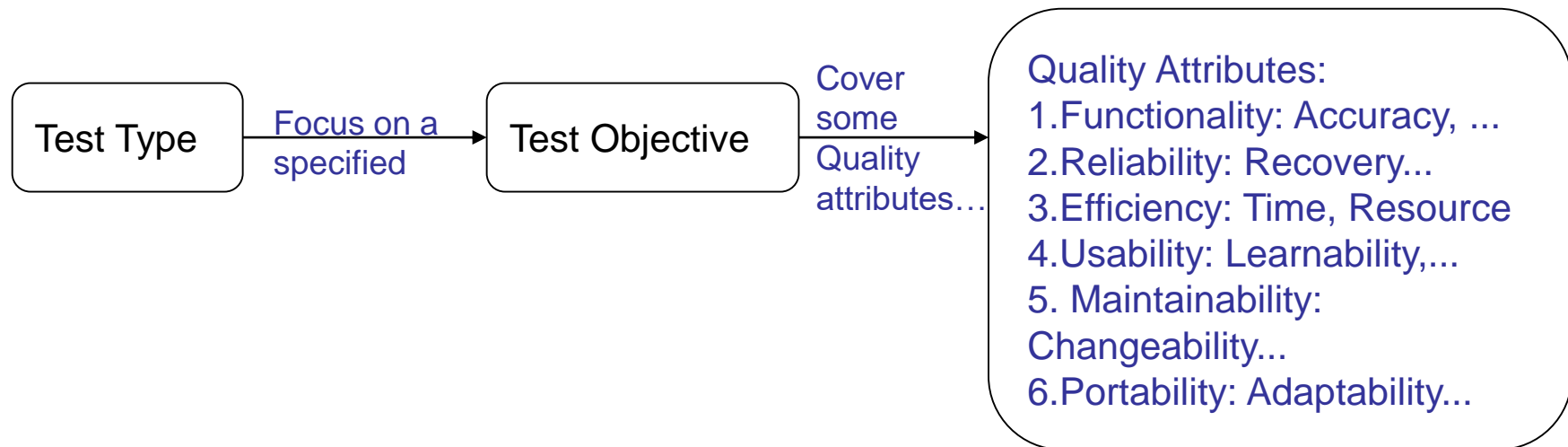
2.2 Test Level (K2)

2.3 Test Types (K2)

2.4 Maintenance Testing (K2)

2.3 Test Types – Target of Testing

- ❑ A group of test activities aimed at testing a component or system focused on specific test objective



- ❑ The following types of testing can be:
 - Functional testing
 - Non-functional testing
 - Structure testing
 - Confirmation & regression testing

2.3.1 Functional Testing

- ❑ The functions that a system, subsystem or component are to perform may be described in work products such as
 - a requirements specification
 - use cases
 - a functional specification
 - may be undocumented.
- ❑ The functions are "what" the system does.
- ❑ **Functionality** is the capability of the software product to provide functions that meet stated and implied needs.
- ❑ **Functionality testing**: The process of testing to determine the functionality of a software product.

2.3.2 Non-functional Testing

- ❑ Non-functional attributes: Quality characteristics
- ❑ It is the testing of "how" the system works.
- ❑ Non-functional testing includes, but is not limited to,
 - performance testing
 - load testing
 - stress testing
 - scalability testing
 - usability testing
 - interoperability testing
 - maintainability testing
 - reliability testing
 - portability testing.

2.3.3 Structural Testing

- ❑ Structural testing = white-box or glass-box
- ❑ Structural testing: Testing based on an analysis of the internal structure of the component or system
- ❑ Performed at all test levels.
- ❑ Best used after specification-based techniques, in order to help measure the thoroughness of testing through assessment of coverage of a type of structure.
- ❑ Coverage: expressed as a percentage of the items being covered.
 - If coverage is not 100%, then more tests may be designed to test those items that were missed and, therefore, increase coverage.

2.3.4 Testing related to changes

❑ Confirmation Testing

- When a defect is detected and fixed then the software should be retested
- To confirm that the original defect has been successfully removed or to verify the success of corrective action.

❑ Regression testing

- The repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the change(s).

2.3.4 Testing related to changes

- ❑ These defects may be either
 - in the software being tested
 - in another related or unrelated software component.
- ❑ The extent of regression testing is based on the risk of not finding defects in software that was working previously.

Testing Throughout The Software Life Cycle

2.1 Software Development Life Cycle (K2)

2.1.1 Sequential Model

2.1.2 Iterative-Incremental Model

2.2 Test Level (K2)

2.3 Test Types (K2)

2.4 Maintenance Testing (K2)

2.4 Maintenance testing

- ❑ Maintenance: Modification of a software product after delivery.
- ❑ Maintenance testing is done on an existing operational system
- ❑ Maintenance is triggered by
 - modifications
 - migration
 - Retirement (Data migration or archiving)

Test Concepts:

- **Maintenance testing**
- **Maintainability testing**

2.4 Maintenance testing

❑ Modifications include

- planned enhancement changes (e.g. release-based)
- corrective and emergency changes,
- changes of environment,
 - planned operating system
 - database upgrades,
 - patches to newly exposed
 - discovered vulnerabilities of the operating system.
- Ad-hoc corrective modifications
 - Defects requiring an immediate solution
 - Risk analysis should be performed.

❑ Impact Analysis

- Maintenance testing consist of two parts:
 - Testing the changes
 - Regression Testing
- Impact Analysis: A decision is made on what parts need careful regression testing

CHAPTER 3

Static Techniques

3. Static Techniques

3.1 Static Testing Basics

3.2 Review process

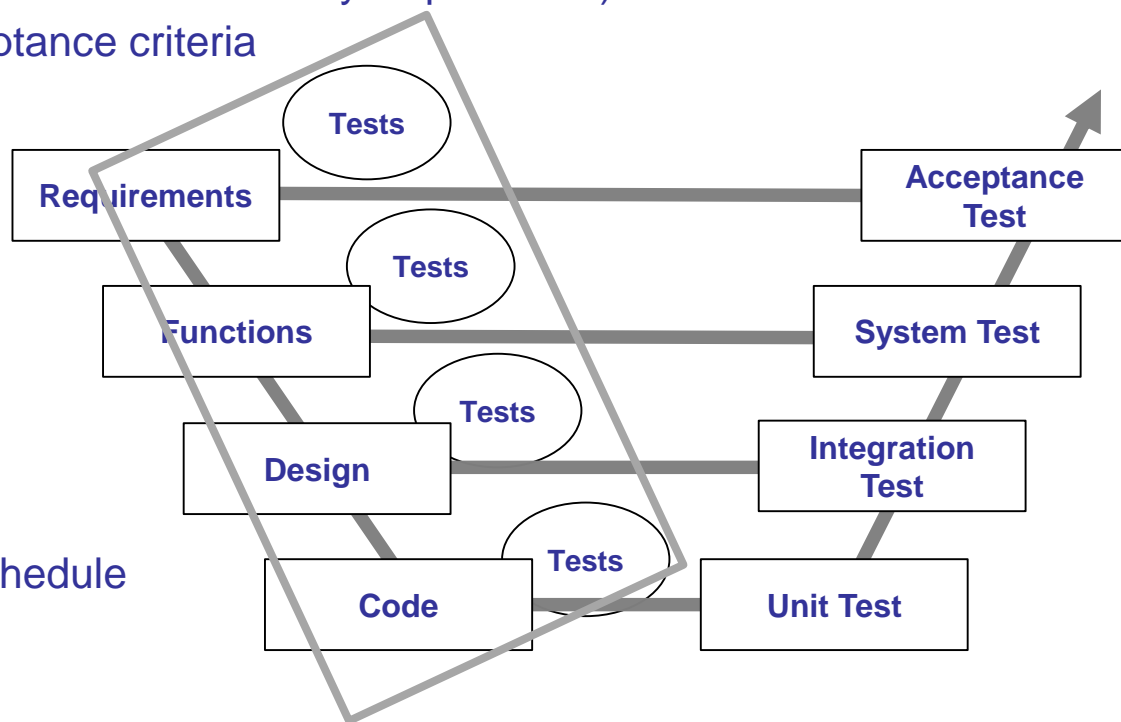
3.1.1 Work products can be examined

❑ Static testing:

- Software work products are examined manually, or with a set of tools, but not executed.

❑ Review Objective: Any software work product:

- Specifications (business/ functional/ security requirement)
- Epics, user stories, acceptance criteria
- Design specifications
- Code
- Test plan
- Test specifications
- Test cases
- Test scripts
- User guides
- Web pages
- Contract, project plan, schedule
- ...



3.1.2 Benefits

❑ Benefits of reviews

- early defect detection And correction
- development productivity improvements
- reduced development timescales
- reduced testing cost and time
- lifetime cost reductions
- fewer defects
- improved communication.

25% reduction in schedules, remove 80% - 95% of faults at each stage, 28 times reduction in maintenance cost, many others

Reviews are cost-effective

3.1.3 Static Testing vs Dynamic Testing

❑ Static testing:

- Finds defects in work products
- Types of defects that are easier to find during static testing are:
 - Requirements defects
 - Design defects
 - Coding defects
 - Deviations from standards
 - Inconsistent interface specifications
 - Security vulnerabilities
 - Gaps or inaccuracies in test basis traceability or coverage

❑ Dynamic testing:

- Find failures caused by defects when the software is run

3.1.3 Static Testing - Defects

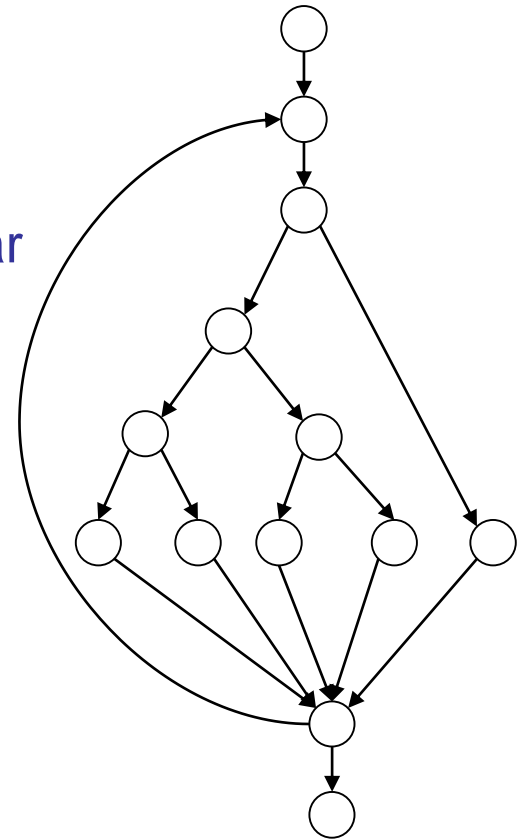
❑ Coding defects

- syntax violations of code and software models
- incorrect data type usage by data and variables
- undeclared variables, variables that are never used
- unreachable (dead) code
- overflow or underflow of field boundaries, array bound violations
- inconsistent interface between modules and components
- all labels as jump start or jump target
- security problems: vulnerabilities, lack of buffer overflow protection

3.1.3 Static Testing - Defects

- ❑ Coding defects - **control flow anomalies**:

- nodes not accessible from start node
- infinite loops
- multiple entry to loops
- whether code is well structured, i.e. reducible
- whether code conforms to a flowchart grammar
- any jumps to undefined labels
- any labels not jumped to
- cyclomatic complexity and other metrics



3.1.3 Static Testing - Defects

❑ Coding defects - **data flow anomalies**:

- 3 types of data flow anomalies:
 - -ur-anomaly: an undefined value (u) of a variable is read/used (r).
 - du-anomaly: The variable is assigned a value (d) that become invalid/undefined (u) without having been used in the meantime
 - dd-anomaly: The variable receives a value for the second time (d) and the first value had not been used (d)

Note: d: Define, r: reference, u: undefined

● Sample

```
function swap (int Min, int Max)
{
```

```
    int help;
```

```
    if (Min > Max) {
```

```
        Max := help;  —————> ur: "help" is used before it has been defined
```

```
        Max := Min;   —————> dd: "Max" is re-defined without being used the last value
```

```
        help := Min; —————> du: "help" is defined without being used
```

```
    }
```

```
}
```


3.1.3 Static Testing - Defects

- ❑ Deviations from standards against:
 - Programming guideline
 - Programming regulation
 - Convention/ standards

3.1 Static Testing Basics

3.2 Review process

3.2.1 Work Product Review Process

3.2.2 Role and Responsibilities

3.2.3 Review Types

3.2.4 Applying Review Techniques

3.2.5 Success Factors for Review

3.2.1 Work Product Review Process

- ❑ Reviews vary from very informal to very formal (i.e. well structured and regulated).
- ❑ The formality of a review process is related to factors such as
 - The maturity of the development process
 - Any legal or regulatory requirements
 - The need for an audit trail.
- ❑ The way a review is carried out depends on the agreed objective of the review (e.g. find defects, gain understanding, or discussion and decision by consensus).

3.2.1 Work Product Review Process

1. Planning

- Defining the scope, including purpose, document, quality characteristics
- Estimating effort, timeframe
- Identifying review types with roles, activities, checklists
- Selecting people/ allocating roles > Defining the entry and exit criteria
- Checking that entry criteria are met

2. Initial review

- Distributing work products
- Explaining the scope, objectives, process, role and work product
- Answer any questions about the review

3. Individual Preparation

- Participants review all or part of the work product
- Note potential defects, questions and comments

3.2.1 Work Product Review Process

4. Review meeting (Issue communication and analysis)

- 3 phases: logging phase, discussion phase and decision phase
- Communicating identified potential defects
- Analyzing potential defects, assigning ownership and status to them
- Evaluating and documenting quality characteristics
- Evaluating the review findings against the exit criteria to make a review decision (reject; major changes needed; accept, possibly with minor changes)
- Checking that entry criteria are met

5. Fixing and reporting:

- Creating defect reports for those findings that require changes
- Fixing defects found, typically done by the author
- Communicating defects to the appropriate person or team
- Recording updated status of defects (in formal reviews)
- Gathering metrics
- Checking on exit criteria (for more formal review types)
- Accepting the work product when the exit criteria are reached.

3.2.2 Roles and Responsibilities

☐ Author:

- Create the work product
- Fix defects in work product (if necessary)

☐ Moderator/ Facilitator:

- Ensure effective running of the review meeting
- Mediate among various points of view

☐ Review leader

- Takes overall responsibility for the review
- Decides who will be involved and organizes when and where it will take place

☐ Reviewer:

- Identify and describe found defects
- Should be represent different perspectives and roles

☐ Manager

- Is responsible for review planning
- Decides on the execution of reviews
- Assigns staff, budget, and time
- Monitors ongoing cost-effectiveness
- Executes control decisions in the event of inadequate outcomes

☐ Scriber (or Recorder)

- Record defect and suggestion
- Document all the issues

3.2.3 Review Types

☐ Informal

- A review not based on a formal (documented) procedure.

☐ Peer

- A review of a software work product by colleagues of the producer of the product for the purpose of identifying defects and improvements. Examples are inspection, technical review and walkthrough.

☐ Walkthroughs

- A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.

☐ Inspection

- A type of peer review that relies on visual examination of documents to detect defects. The most formal review technique and therefore always based on a documented procedure

☐ Technical Review

- A peer group discussion activity that focuses on achieving consensus on the technical approach to be taken.

3.2.3.1 Informal Review

❑ Key characteristics:

- no formal process;
- there may be pair programming or a technical lead reviewing designs and code;
- optionally may be documented;
- may vary in usefulness depending on the reviewer,
- main purpose: inexpensive way to get some benefit.

3.2.3.2 Walkthrough

❑ Key characteristics:

- meeting led by author;
- scenarios, dry runs, peer group;
- open-ended sessions;
 - optionally a pre-meeting preparation of reviewers
 - optionally preparation of a review report including list of finding, review report
- Optional scribe (who is not the author)
- may vary in practice from quite informal to very formal;
- main purposes: learning, gaining understanding, defect finding.

3.2.3.2 Walkthrough

- ❑ The walkthrough team consists of three to five people
 - author
 - the moderator in the inspection process
 - a secretary (a person who records all errors found),
 - a tester

- ❑ Suggestions for the other participants include
 - (1) a highly experienced programmer,
 - (2) a programming-language expert,
 - (3) a new programmer (to give a fresh, unbiased outlook),
 - (4) the person who will eventually maintain the program,
 - (5) someone from a different project, and
 - (6) someone from the same programming team as the programmer.

3.2.3.2 Walkthrough

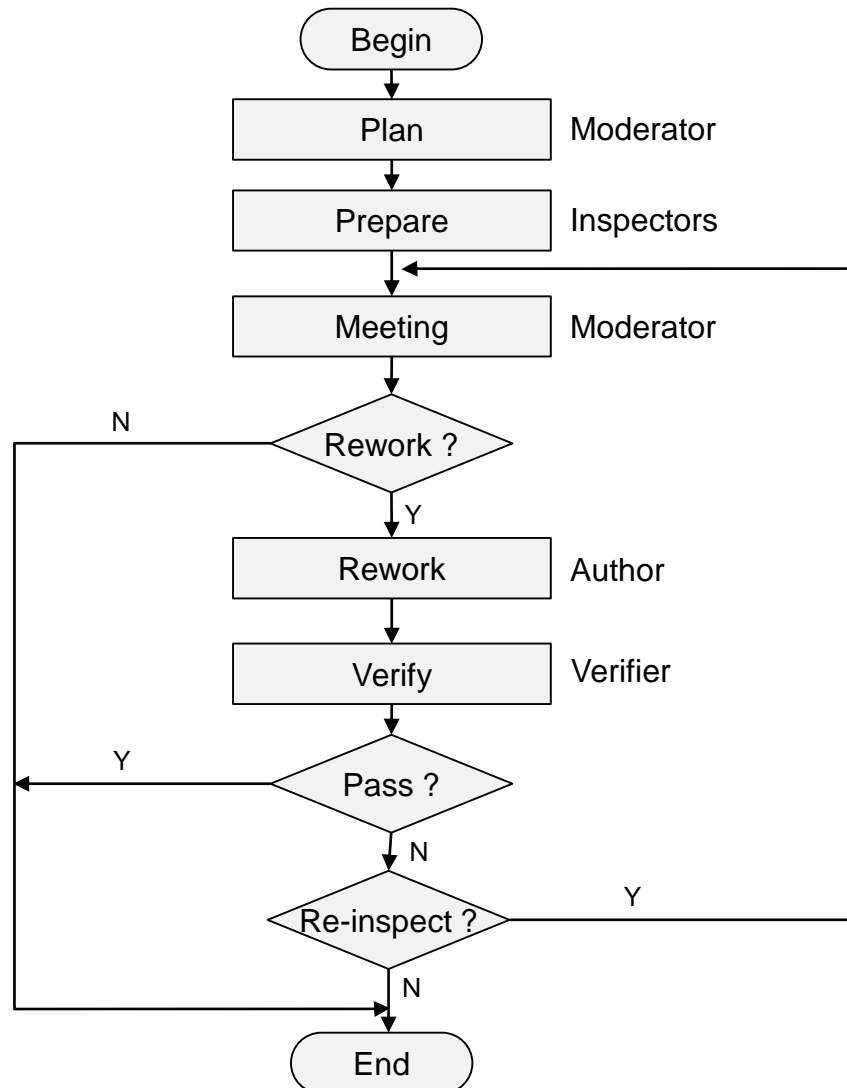
- ❑ The participants are given the materials several days in advance.
- ❑ Rather than simply reading the program or using error checklists, the participants “play computer.”
- ❑ The tester comes to the meeting armed with a small set of paper test cases—representative sets of inputs (and expected out-puts) for the program or module.
- ❑ During the meeting, each test case is mentally executed

3.2.3.3 Inspection

- ❑ Inspection Process is the most formal review process
- ❑ Key characteristics:
 - led by trained moderator (not the author);
 - usually peer examination;
 - defined roles;
 - includes metrics;
 - formal process based on rules and checklists with entry and exit criteria;
 - pre-meeting preparation thoroughly
 - inspection report, list of findings;
 - formal follow-up process;
 - optionally, process improvement and reader;
 - main purpose: find defects.
 - Usually consist 4 participants: Moderator, programmer or author, program's designer and a test specialist.

3.2.3.3 Inspection

- Inspection Process is the most formal review process



3.2.3.3 Inspection

❑ Defect Removal Efficiency

	Low	Median	High		Low	Median	High
No design inspections No code inspections No quality assurance No formal testing	30%	40%	50%	Formal design inspections Formal code inspections Formal quality assurance Formal testing	95%	99%	99.99%
Formal design inspections and nothing else	43%	60%	68%	Formal code inspections and nothing else	43%	57%	66%
Formal quality assurance and nothing else	32%	45%	55%	Formal testing and nothing else	37%	53%	60%
Formal design inspections Formal code inspections	70%	85%	90%	Formal quality assurance Formal testing	50%	65%	75%
Formal design inspections Formal code inspections Formal testing	85%	97%	99%	Formal code inspections Formal quality assurance Formal testing	75%	87%	93%

3.2.3.4 Technical review

- ❑ Technical review is a discussion meeting that focuses on achieving consensus about the technical content of a document
- ❑ Key characteristics:
 - documented, defined defect-detection process that includes peers and technical experts;
 - often performed as a peer review without management participation;
 - ideally led by trained moderator (not the author);
 - pre-meeting preparation;
 - optionally the use of checklists, review report, list of findings and management participation;
 - may vary in practice from quite informal to very formal;
 - main purposes: discuss, make decisions, evaluate alternatives, find defects, solve technical problems and check conformance to specifications and standards.

3.2.3.4 Technical review

- ❑ Technical review:
 - includes peer and technical experts, maybe no management participation. Normally documented, fault-finding. Can be rather subjective.
- ❑ Leader/ Moderator:
 - plans the review / Inspection
 - chooses participants
 - helps & encourages
 - conducts the meeting, media between people
 - performs follow-up
 - manages metrics
- ❑ Author: owner of the document being reviewed / Inspected
- ❑ Reviewers: specialised fault-finding roles, can be colleagues of author or technical expert (for technical review)
- ❑ Managers: excluded from some types of review, need to plan project time for review / Inspection

3.2.3.5 Comparisons (1)

	Walk-through	Inspection	Technical Review	Informal Review
Type	Peer Review	Peer Review	Peer Review (by default)	Peer Review
Formality	Formal or Fairly informal	The most formal	Formal	Informal
Objectives / goals	<ul style="list-style-type: none"> • learn • gain understanding • find defects 	<ul style="list-style-type: none"> • Fault detection • Process improvement (opt) 	<ul style="list-style-type: none"> • Fault detection 	<ul style="list-style-type: none"> • inexpensive way to get some benefit • find defects
Led by	Author	Trained Moderator	Trained Moderator (but also by a technical expert)	Individual
Participant/ Involves	Peer group	Defined roles	Peer group, Technical experts	Anyone
Team Size	team 5-7 person	team 3-7 person	team 3-5 person	
Cost/ Effort	~ inspection	High (10-15% dev budget) Lower maintenance cost	~ inspection	
Others		Objective, not subjective Not last for over 2 hours typically dozens of pages to review	rather subjective Not last for over 2 hours	

3.2.3.5 Comparisons (2) - Characteristics

Characteristics	Walk-through	Inspection	Technical Review	Informal Review
Common Characteristics	<ul style="list-style-type: none"> • Documented • Review Report • List of Finding 	<ul style="list-style-type: none"> • Documented, highly focused • Define Rules, Roles • Using checklist • Metrics kept • Define entry and exit criteria • Inspection report • list of finding 	<ul style="list-style-type: none"> • Documented • Define Roles • Using checklist • No metrics kept • Review Report • List of Finding 	<ul style="list-style-type: none"> • Undocumented • Usefulness depending on the reviewer
Record used	Maybe	Yes	Yes	No
Follow a documented procedure	Maybe	Yes	Maybe	No
Defined participant roles	No	Yes	Yes	No
Defect checklist used	No	Yes	Yes	No
Data collected and analyzed	No	Yes	Maybe	No
Product appraisal determined	No	Yes	Yes	No

3.2.3.5 Comparisons (3) - Activities

Activity	Walk-through	Inspection	Technical Review	Informal Review
Planning	Yes	Yes	Yes	Maybe
Overview / kick-off meeting	Maybe	Yes	Maybe	Maybe
Preparation/ individual checking	No/ Low	Yes (High)	Yes (Extremely High)	No
Review meeting	Yes	Yes	Yes	Continuous
Fixing and reporting	Rework based on reviewer comments	Yes	Yes	Yes
Metrics recording & analysis	No	Yes	Maybe	No

3.2.4 Applying Review Techniques

- ☐ Adhoc
- ☐ Check-list based
- ☐ Scenarios and dry runs
- ☐ Role-based
- ☐ Perspective-based

3.2.5 Success factors for reviews

❑ Organization Success factors:

- Each review has clear objectives defined during review planning, and used as measurable exit criteria
- Suitable review types are applied to achieve the objectives and are appropriate to the type and level of software work products and participants
- Any review techniques used, such as checklist-based or role-based reviewing, are suitable for effective defect identification in the work product to be reviewed
- Any checklists used address the main risks and are up to date
- Large documents are written and reviewed in small chunks, so that quality control is exercised by providing authors early and frequent feedback on defects
- Participants have adequate time to prepare
- Reviews are scheduled with adequate notice
- Management supports the review process (e.g., by incorporating adequate time for review activities in project schedules)

3.2.5 Success factors for reviews

❑ People-related success factors:

- The right people for the review objectives are involved.
- Testers are seen as valued reviewers
- Participants dedicate adequate time and attention to detail
- Reviews are conducted on small chunks, reviewers do not lose concentration
- Defects found are acknowledged, appreciated, and handled objectively
- The meeting is well-managed, so that participants consider it a valuable use of their time
- The review is conducted in an atmosphere of trust; the outcome will not be used for the evaluation of the participants
- Participants avoid body language and behaviors that might indicate boredom, exasperation, or hostility to other participants
- Adequate training is provided
- A culture of learning and process improvement is promoted

3.2.5 Success factors for reviews

❑ Possible difficulties:

- Required persons are not available or not have required qualification or technical aptitude
- Inaccurate estimates during resource planning by management
- Wrong reviewers were chosen
- Missing or insufficient documentation
- Management support is lacking, because the necessary resources will not be provided and results will not be used for process improvement

CHAPTER 4

Test Design Techniques

4. Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.1 Test development process

□ Test development process consists of a number of steps:

- Identify test conditions that are

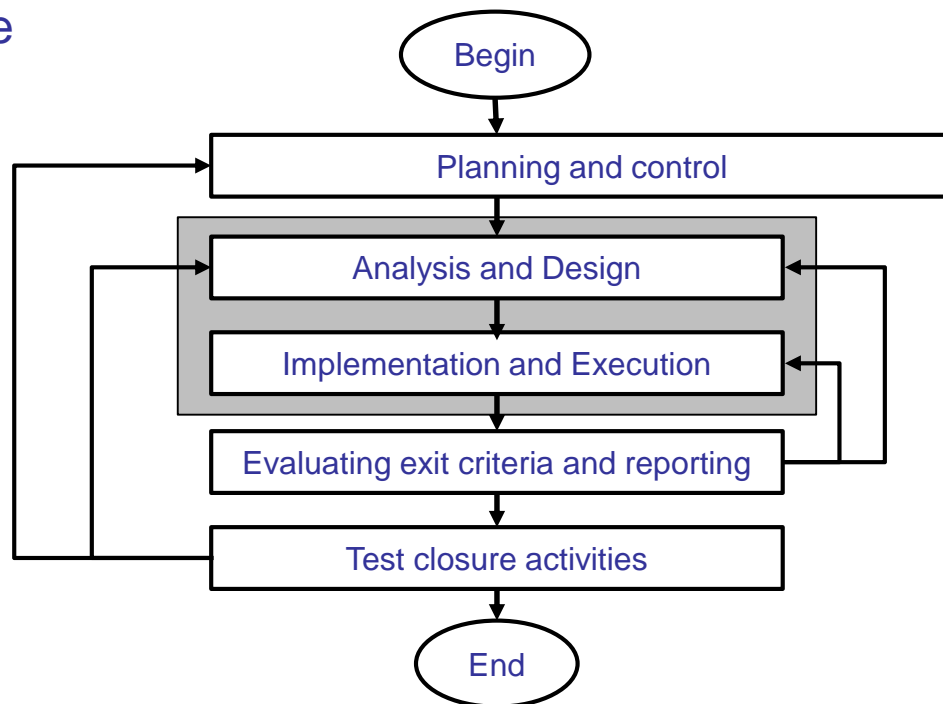
- A function
- Transaction
- Quality characteristic
- Structural element

- Design test cases by specify

- Test input
- Expected result

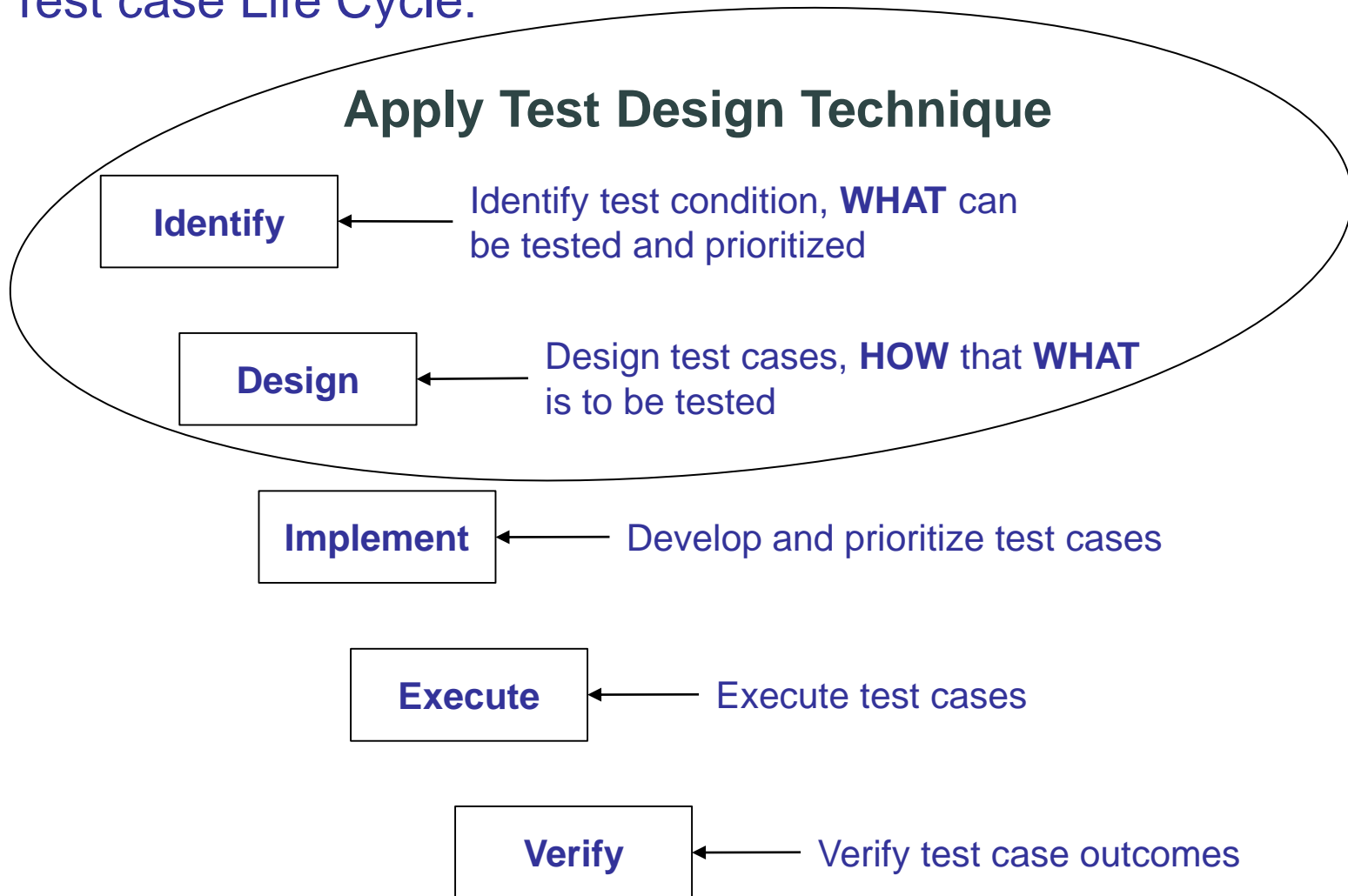
- Develop test procedures/ test cases/ test scenario

- Test steps



4.1 Test development process

❑ Test case Life Cycle:



4.1 Test development process

- ❑ During test design the detailed test approach is implemented based on, among other considerations, the risks identified.

- ❑ The level of formality depends on
 - the organization
 - the maturity of testing
 - development processes
 - time constraints
 - the people involved

4.1 Test development process

❑ Test Bed

- Also called as Test environment
- Platform or environment where testing is being done
- Testers must often setup test bed

❑ Traceability

- Exists between specifications and test cases
- Allows impact analysis of the effects of other specifications on the test process
- Helps to identify where many of test cases overlap
- Identify test cases affected by changes in specifications

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.2 Test development process

- ❑ A test design technique is a systematic approach to **identify test conditions and test cases** that give a high probability of finding defects.

- ❑ Test Design techniques
 - Specification-based
 - Structural-based
 - Experience-based

4.2.1 Specification-Based Techniques

- ❑ **Name:** Specification-based, Behavioral or Black-box technique.
- ❑ **Common features**
 - derive and select test conditions or test cases based on an analysis of the **b**, whether functional or non-functional, for a component or system without reference to its internal structure.
 - Can apply at all levels of testing where a specification exists.

4.2.2 Structure-Based Techniques

- ❑ **Name:** Structure-based, glass-box or white-box

- ❑ **Common features**
 - derive and select test conditions or test cases based on based on an analysis of the **internal structure** of the component or system such as code and design.
 - The extent of coverage of the software can be measured for existing test cases, and further test cases can be derived systematically to increase coverage.

4.2.3 Experienced-Based Techniques

❑ Common features

- The knowledge and experience of people are used to derive the test cases
- Knowledge of testers, developers, users and other stakeholders about the software, its usage and its environment
- Knowledge about likely defects and their distribution

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

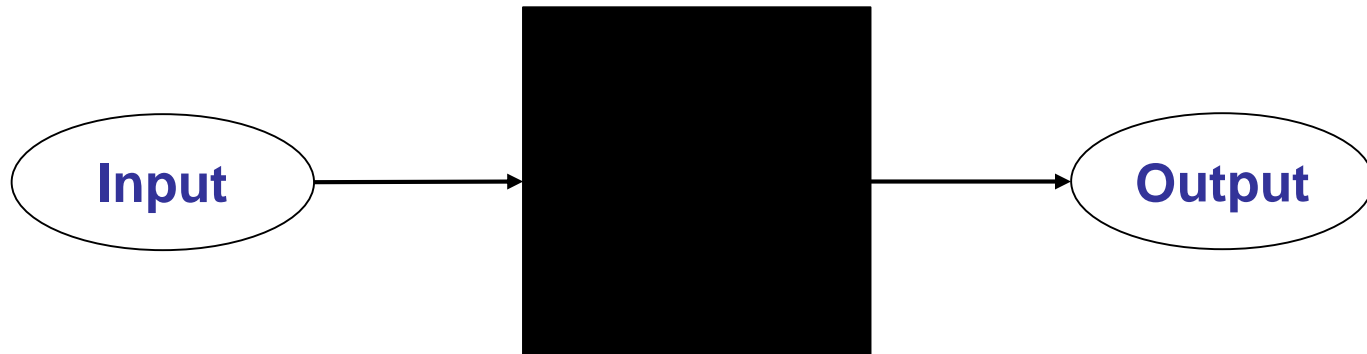
4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

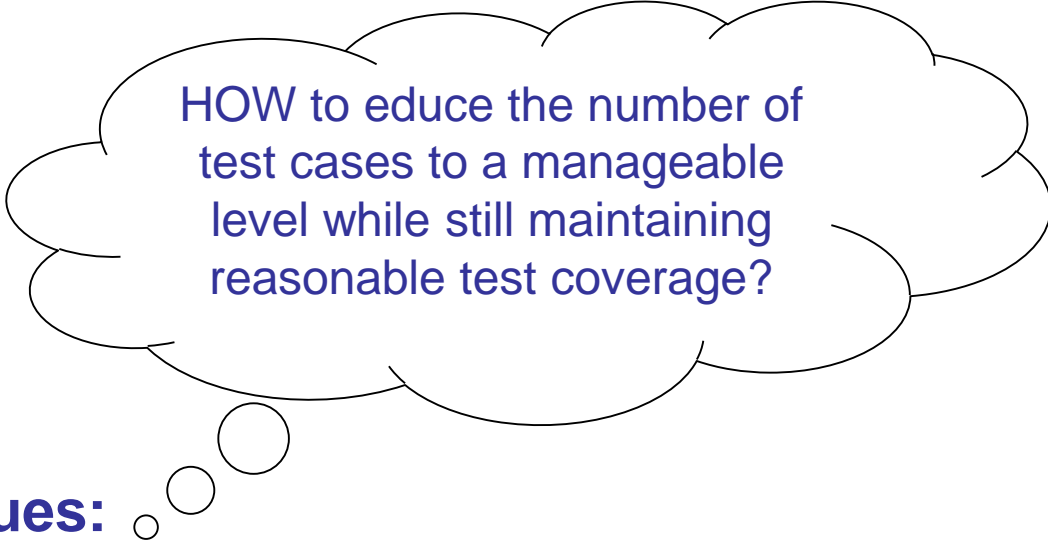
4.8 Writing test cases

4.3 Black-box techniques



- ❑ Targeted at the apparent simplicity of the software
 - Makes assumptions about implementation
 - Good for testing component interactions
- ❑ Tests the interfaces and behavior

4.3 Black-box techniques



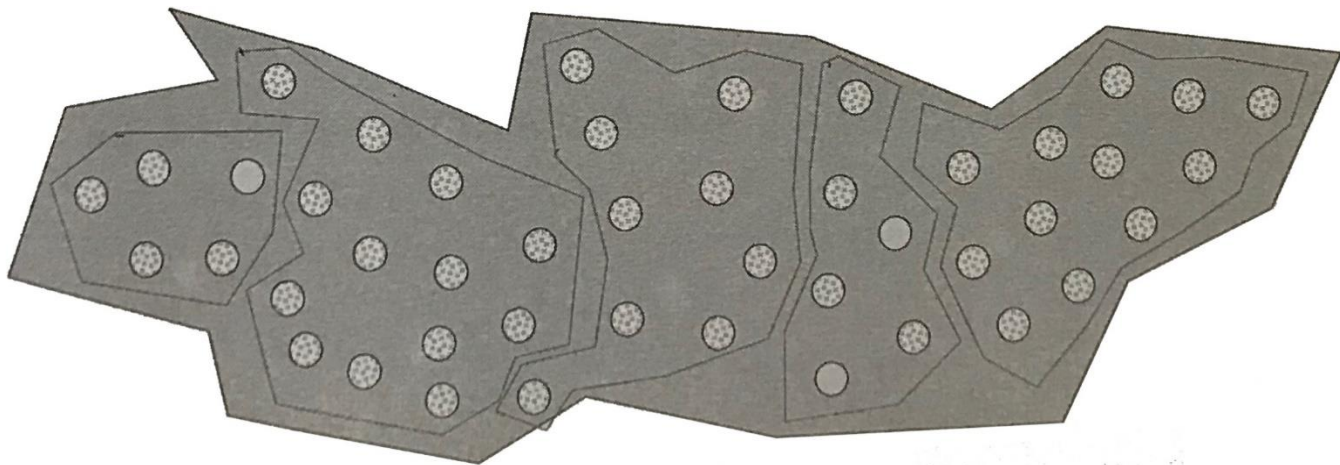
HOW to educe the number of test cases to a manageable level while still maintaining reasonable test coverage?

❑ Black-box techniques:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table
- State Transition
- All pairs
- Use Case

4.3.1 Equivalence Class Testing (partitioning)

- ❑ Input domains are divided into equivalence classes:
 - Equivalence class is a group of data values where tester assumes that test object processes them in the same way.
 - The test of one representative of the equivalence class is seen as sufficient.
- ❑ Besides equivalence classes for correct input, those for incorrect input values must be tested as well.



4.3.1 Equivalence Class Testing (partitioning)

- ❑ Example 1: One of organization's employment applications based on a person's age as follows:
 - 0–15 Don't hire
 - 16–20 Can hire on a part-time basis only
 - 21–55 Can hire as a full-time employee
 - 56–99 Don't hire

- ❑ Identify TCs:
 1. Using EP
 2. Using BVA

4.3.1 Equivalence Class Testing (partitioning)

❑ Example 1:

- Should we test the module for the following ages: 0, 1, 2, 3, 4, 5, 6, 7, 8,...,90, 91, 92, 93, 94, 95, 96, 97, 98, 99? If we had lots of time we certainly could.
- What if you test 4, 17, 33, 59 ?
- Using the equivalence class approach, we have reduce number of test cases from 100 (testing each age) to four (testing one age in each equivalence class)-a significant savings.

4.3.1 Equivalence Class Testing (partitioning)

□ Example 1: Using EC

Parameter	Equivalence Class	Representative value
Age (x)	<u>Valid EC:</u>	
	vEC1: $0 \leq x \leq 15$	10
	vEC2: $16 \leq x \leq 20$	18
	vEC3: $21 \leq x \leq 55$	30
	vEC4: $56 \leq x \leq 99$	60
	<u>Invalid EC:</u>	
	iEC1: $x < 0$	-1
	iEC2: $x > 99$	101

→ We have 6 test cases: 4 valid test cases and 2 invalid test cases

4.3.1 Equivalence Class Testing (partitioning)

- ❑ **Example 2:** Identify a point (x,y) on ViewPort screen with following information:

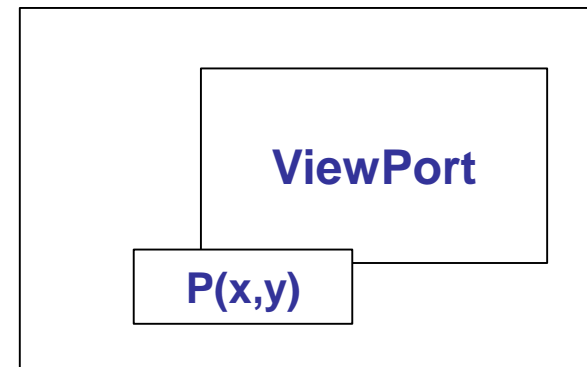
x: 230..1000

y: 300..650

→ Identify Test cases

1. Using EP

2. Using BVA



4.3.1 Equivalence Class Testing (partitioning)

- ❑ Example 2: Using EC - combine representative values of x and y

Parameter	Equivalence Class	Representative value
X	<u>Valid EC:</u> vEC1: $230 \leq x \leq 1000$	300
	<u>Invalid EC:</u> iEC1: $x < 230$	200
	iEC2: $x > 1000$	1100
Y	<u>Valid EC:</u> vEC1: $300 \leq y \leq 650$	400
	<u>Invalid EC:</u> iEC1: $y < 300$	250
	iEC2: $y > 650$	700

→ We have 1 valid test case and ? invalid test cases

4.3.1 Equivalence Class Testing (partitioning)

Steps to do:

- ❑ First, identify the equivalence classes.
- ❑ Second, create a test case for each equivalence class with representative data

4.3.1 Equivalence Class Testing (partitioning)

Key notes:

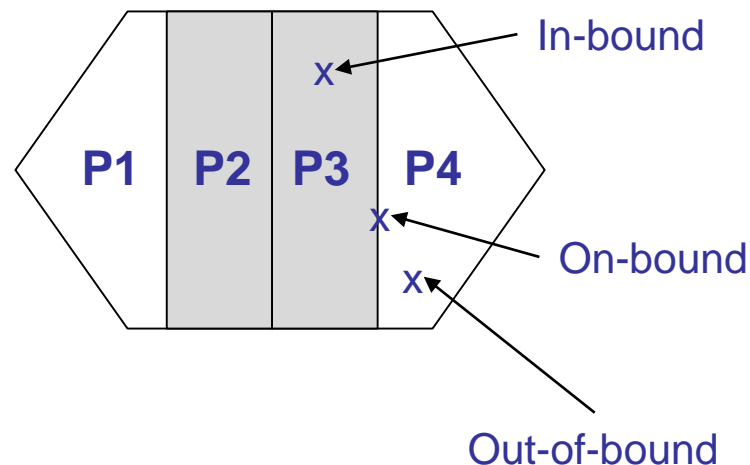
- ❑ Equivalence Class Technique is most suited to systems in which much of the input data takes on values within ranges or within sets.
- ❑ **Input or output data can be partitioned** based on the system's requirements.
- ❑ It can be applicable at unit, integration, system and acceptance test levels.
- ❑ Dual combinations (refer to Pairwise technique) instead of complete combinations (between the representative of one equivalence class with the one of other equivalence class).
- ❑ **Invalid Value should be tested separately:** Representatives of "invalid equivalence classes" should not be combined with representatives of other "invalid equivalence classes". This will cause defect-masking.

4.3.2 Boundary Value Analysis

- ❑ A boundary value is an element close to the edges, both the upper and lower edges of an equivalence class.

- ❑ Boundaries define 3 sets of data:

- **Good (In-Bound)**
- **Bad (Out-Of-Bound)**
- **On-the-border (On-Bound)**



- ❑ Boundaries define 2 sets of data:

$$N = \text{number of EC} * 2 - 2$$

- ❑ It is easy to apply and its defect finding capability is high.

4.3.2 Boundary Value Analysis

❑ Example 1:

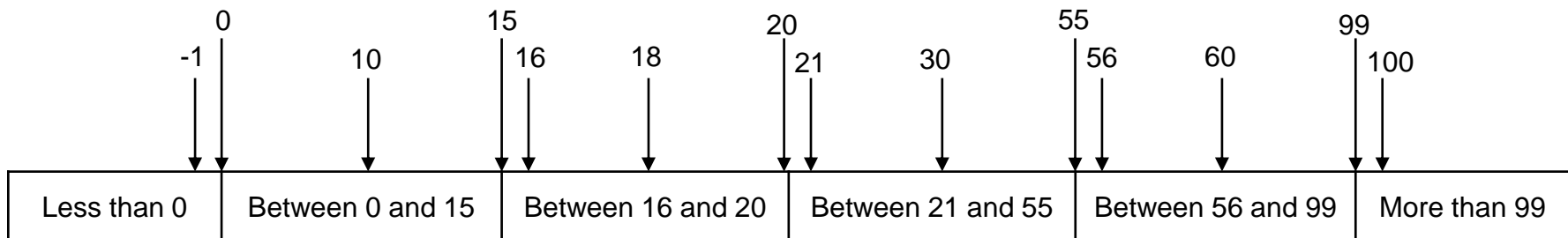
One of organization's employment applications based on a person's age as follows:

- 0–15 Don't hire
- 16-20 Can hire on a part-time basis only
- 21–55 Can hire as a full-time employee
- 56–99 Don't hire

4.3.2 Boundary Value Analysis

□ Example 1:

- Using EC: 6 EC – 4 valid and 2 invalid
- Using BVA: Number of boundary values = $6 * 2 - 2 = 10$



Age – input value

4.3.2 Boundary Value Analysis

□ Example 2:

Age(x)	iEC1	vEC1	vEC2	vEC3	vEC4	iEC2
EC	$x < 0$	$0 \leq x \leq 15$	$16 \leq x \leq 20$	$21 \leq x \leq 55$	$56 \leq x \leq 99$	$x > 99$
EC representative Data	-1	10	18	30	60	100
BVA	-1	0, 15	16, 20	21, 55	56, 99	100

→ Using BVA, 10 boundary values: -1, 0, 15, 16, 20, 21, 55, 56, 99, 100

→ Using EC and BVA, we have 14 test cases to cover following values:
-1, 0, 10, 15, 16, 18, 20, 21, 30, 55, 56, 60, 99, 100.

4.3.2 Boundary Value Analysis

Steps to do:

- ❑ First, identify the equivalence classes.
- ❑ Second, identify the boundaries of each equivalence class.
- ❑ Third, create test cases for each boundary value.

4.3.2 Boundary Value Analysis

Key notes:

- ❑ Boundary Value Technique requires both the upper and lower boundaries of an equivalence class are covered by test cases.
- ❑ Boundary Value Technique is most suited to systems in which much of the input data takes on values within ranges or within sets.
- ❑ All it requires are inputs that can be partitioned and boundaries that can be identified based on the system's requirements.
- ❑ It can be applicable at unit, system, integration and acceptance test levels.

4.3.3 Decision table testing

1. What is a Decision Table?

- ❑ A decision table includes rows listing Cause and Effect and columns listing possible combinations. Each combination of conditions in decision table is referred to as a rule and have a corresponding outcome.
 - Cause also considered as Condition.
 - Effect also considered as Action or Expected Results.

2. What is Decision Table Testing Technique?

- ❑ Decision Table is one of the useful testing design techniques used to record complex rules/flows. It also can serve as a guide to create test cases.

4.3.3 Decision table testing

- ❑ Test cases are designed to have at least one test per rule in the table
- ❑ The strength of this technique is discover omissions and ambiguities in specification.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
<i>Condition 1</i>	T	T	F	F
<i>Condition 2</i>	T	F	T	F
Action/Outcomes				
<i>Outcome 1</i>	Y		Y	

4.3.3 Decision table testing

Example 1

❑ We are going to test the function to withdraw money from an ATM. Here are the conditions for withdrawing money:

1. ATM Card is valid
2. PIN entered is correct (within 3 times)
3. Balance in the account & the machine is enough

❑ Here are the Actions:

- Reject card (if condition 1 failed)
- Eat card (if condition 2 failed)
- Ask new withdrawal amount (if condition 3 failed)
- Pay money (if conditions 1,2 and 3 passed)



4.3.3 Decision table testing

Example 1 - Step 1 - Calculate the number of combinations

- ❑ To calculate the number of combinations, base on the Condition/ Cause. Here are the Conditions:
 1. ATM Card is valid
 2. PIN entered is correct (within 3 times)
 3. Balance in the account & the machine is enough

- ❑ In this example, we have 3 causes, each cause has 2 values (Yes/No). So the combinations should be:
 $2^3 = 8$ combinations
Or: $2 \times 2 \times 2 = 8$ combinations

4.3.3 Decision table testing

Example 1 - Step 2 - List all causes in Decision Table

- ❑ The form of tables can be altered. But the following form is recommended.
 - On the half-top of the table, list the condition or cause.
 - On the half-bottom of the table, list the effect or expected results.

Conditions/Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No								
PIN entered is correct (within 3 times)	Yes/No								
Balance in the account & in machine is enough to withdraw	Yes/No								
Actions/Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

- In the column Value, list the values of each cause & effect. In this example, 'Yes' for the case of conditions are met and reverse to 'No'.
- Next columns (1-8) are the combinations or test cases.

4.3.3 Decision table testing

Example 1 - Step 3 – Fill columns with all possible combinations

Conditions/Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	Y	Y	N	N
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	Y	N	Y	N	Y	N
Actions/Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

4.3.3 Decision table testing

Example 1 - Step 4 – Edit & reduce test combinations

Conditions/Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	-	-	-	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-	-	-	-	-
Actions/Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								

Replace invalid causes with '-'

Duplicated combinations

4.3.3 Decision table testing

Example 1 - Step 5 – Add the checksum row

Conditions/Causes	Value	1	2	3	4	5	6	7	8
Card is valid	Yes/No	Y	Y	Y	Y	N	N	N	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	N	-	-	-	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-	-	-	-	-
Actions/Effects									
Reject card	Yes/No								
Eat card	Yes/No								
Ask new withdrawal amount	Yes/No								
Pay money	Yes/No								
Checksum		1	1	2		4			

Duplicated combinations will be deleted

Checksum number indicates the number of combinations covered.

Total of checksum number must be equal to the total number of combinations.

$1 + 1 + 2 + 4 = 8 \rightarrow$ This means 8 combinations are still covered.

4.3.3 Decision table testing

Example 1 - Step 6 – Add effects to the table

Conditions/Causes	Value	1	2	3	4
Card is valid	Yes/No	Y	Y	Y	N
PIN entered is correct (within 3 times)	Yes/No	Y	Y	N	-
Balance in the account & in machine is enough to withdraw	Yes/No	Y	N	-	-
Actions/Effects					
Reject card	Yes/No	N	N	N	Y
Eat card	Yes/No	N	N	Y	N
Ask new withdrawal amount	Yes/No	N	Y	N	N
Pay money	Yes/No	Y	N	N	N

This is the Decision Table for the example. Each combination should be executed as a test case.

For example: test case 1 (or combination 1) is when ATM card + PIN are valid + enough money in the account and in the ATM → then the ATM pays money.

→ We have 4 test cases

4.3.3 Decision table testing

Example 2: Cellphone: Make a decision table

<i>Condition/Check sum</i>	8	4	2	1	1
Valid Battery	N	Y	Y	Y	Y
Valid Signal	-	N	Y	Y	Y
Valid Credit	-		N	Y	Y
Valid phone number	-	-	-	N	Y
<i>Action</i>					
Recharging	Y	N	N	N	N
Get the Signal	N	Y	N	N	N
Buy Credit	N	N	Y	N	N
Get the correct number	N	N	N	Y	Y
Make call successfully	N	N	N	N	Y

4.3.3 Decision table testing

Step to do:

1. Calculate the number of possible combinations.
 2. List all causes in the decision table.
 3. Fill columns with all possible combinations.
 4. Edit and reduce test combinations.
 5. (Add the checksum row)
 6. Add effects to the table.
-
- ☐ The risk is that mistakes easily occur when filling information in Decision Table. So carefulness is a must to have trustable complete test cases.

4.3.4 Pair-wise Testing

- ❑ Allpairs or Pairwise Testing is a methodology that enables us to generate as few test cases as possible by pairing values of different variables.
- ❑ Allpairs Technique works on the rule of testing every unique pair of combination. Any duplicated ones will be removed.

4.3.4 Pair-wise Testing

Example 1: Test these simple combinations:

☐ Operating System (Parameter A)

- WinXP (Value A1)
- VISTA (Value A2)

☐ Computer (Parameter B)

- HP (Value B1)
- SAMSUNG (Value B2)

☐ CPU (Parameter C)

- 1G (Value C1)
- 2G (Value C2)

→ **All the combinations: $2 \times 2 \times 2 = 8$ test cases**

4.3.4 Pair-wise Testing

Example 1: Test case with all combinations

Test cases	Parameter A	Parameter B	Parameter C
1	A1	B1	C1
2	A1	B1	C2
3	A1	B2	C1
4	A1	B2	C2
5	A2	B1	C1
6	A2	B1	C2
7	A2	B2	C1
8	A2	B2	C2

1. We don't need case 2 because: [A1;B1] is duplicated in case 1, [A1;C2] is duplicated in case 4, [B1;C2] is duplicated in case 6;
2. The same to cases 3, 5, 8

4.3.4 Pair-wise Testing

- ❑ But what will you do when you need to generate test cases for the **large scale of combinations**? Using tool for these cases is smarter than trying to design test cases by hand.
- ❑ Tools recommended:
 1. Allpairs or McDowell
 2. Jenny
 3. Pict/PictMaster
 4. ComTest

4.3.4 Pair-wise Testing

Example 2: Test these another simple combinations:

Parameter A has three values A1, A2 and A3

Parameter B has three values B1, B2 and B3

Parameter C has three values C1, C2 and C3

→ All the combinations? $3 \times 3 \times 3 = 27$ test cases

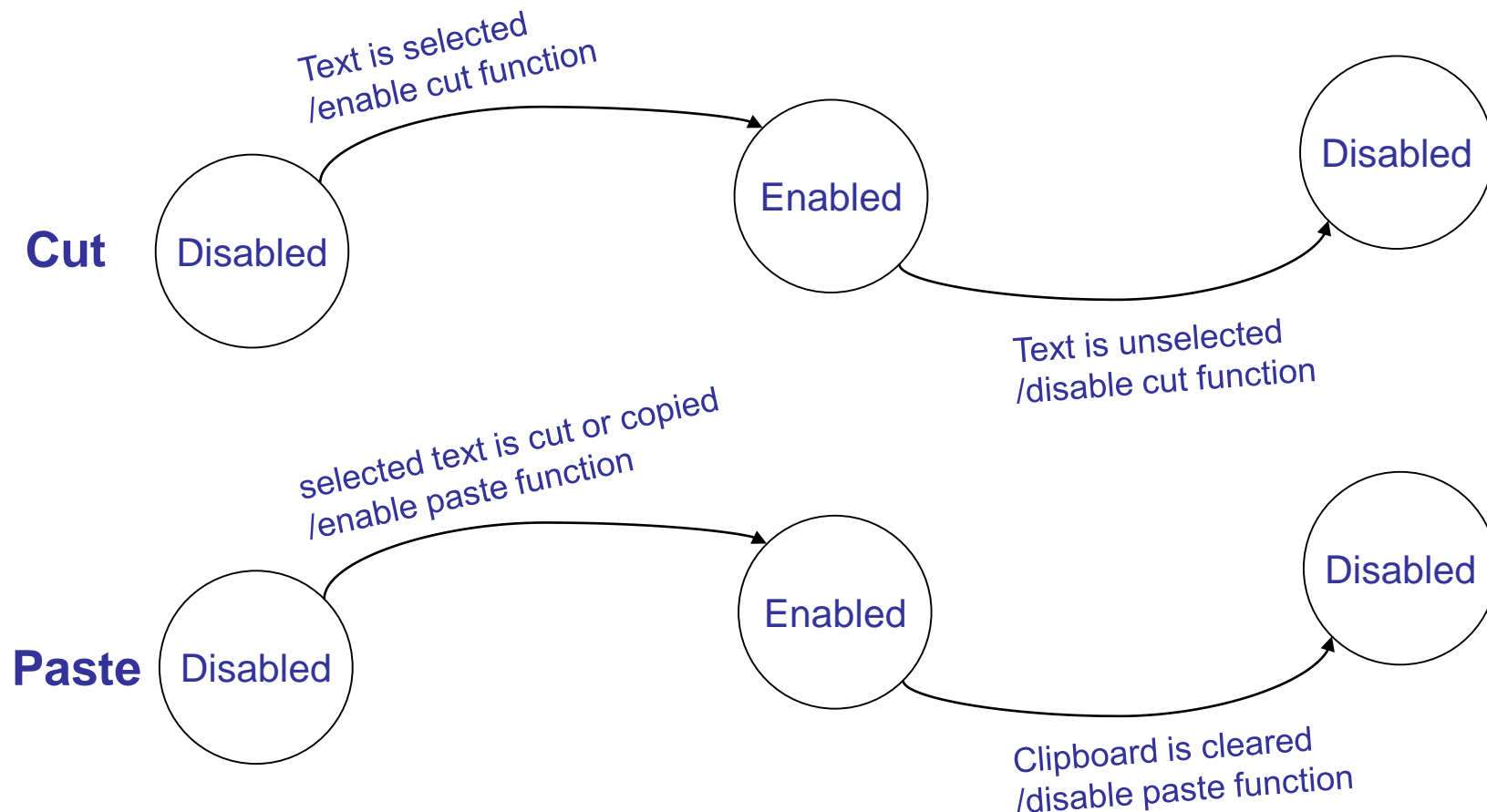
❑ With Allpairs Technique, we just need to execute 9 test

4.3.5 State Transition Testing

- ❑ A **state** is a condition (attribute) in which a system or a component is waiting for one or more events.
- ❑ A **state transition** changes in the attributes (from a state to another) of an object caused by an event.
- ❑ A **event** is something that causes the system to change state.
- ❑ An **action** is an operation initiated because of a state change. Note that action only occurs on transitions between states.

4.3.5 State Transition Testing

❑ Test functions Cut/Paste of notepad – State-transition Diagram



4.3.5 State Transition Testing

❑ Test functions Cut/Paste of notepad – State-transition Diagram

Function	Current state	Event	Action	Next state
Cut	Disabled	Text selected	Enable function	Enabled
	Enabled	Text un-selected	Disabled function	Disabled
Paste	Disabled	Selected text is cut or copied	Enable function	Enabled
	Enabled	Clipboard is cleared	Disabled function	Disabled

4.3.5 State Transition Testing

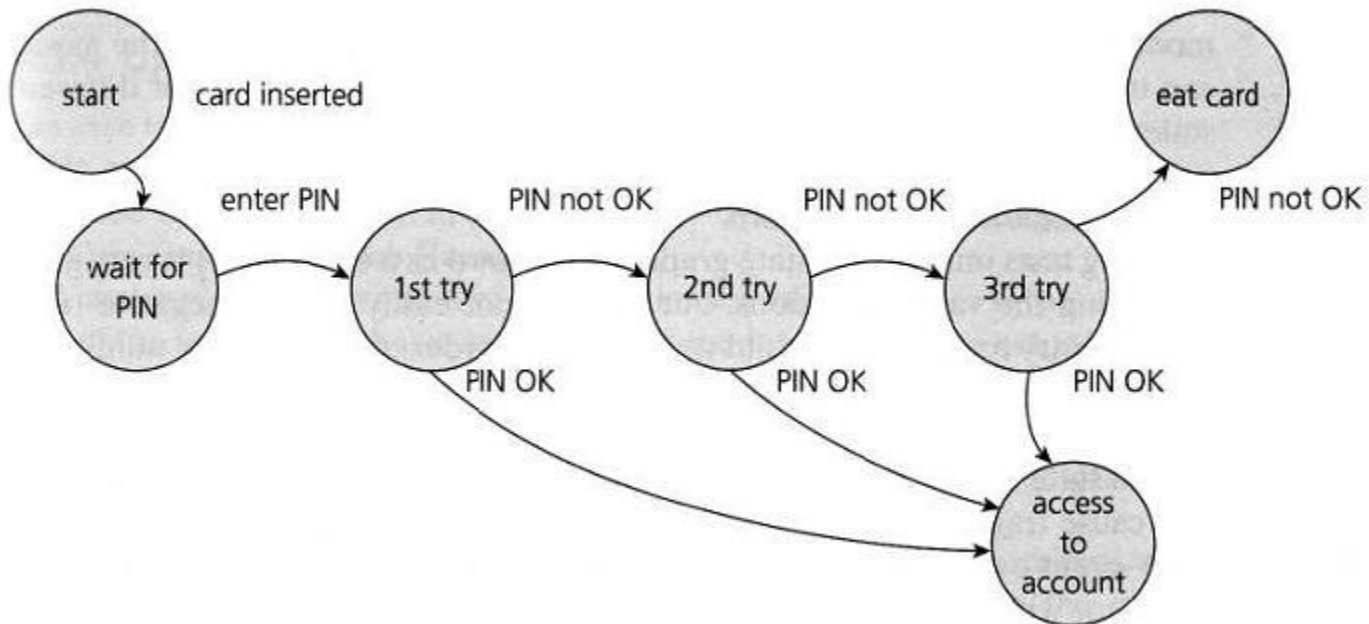
- ❑ State Transition Testing is a test case design technique in which test cases are designed to execute state transitions. State Transition testing uses State Diagram or State Table.

- ❑ **Tests can be designed**
 - to cover a typical sequence of states,
 - to cover every state,
 - to exercise every transition,
 - to exercise specific sequences of transitions
 - to test invalid transitions.

4.3.5 State Transition Testing

□ Example:

State diagram for PIN entry



4.3.5 State Transition Testing

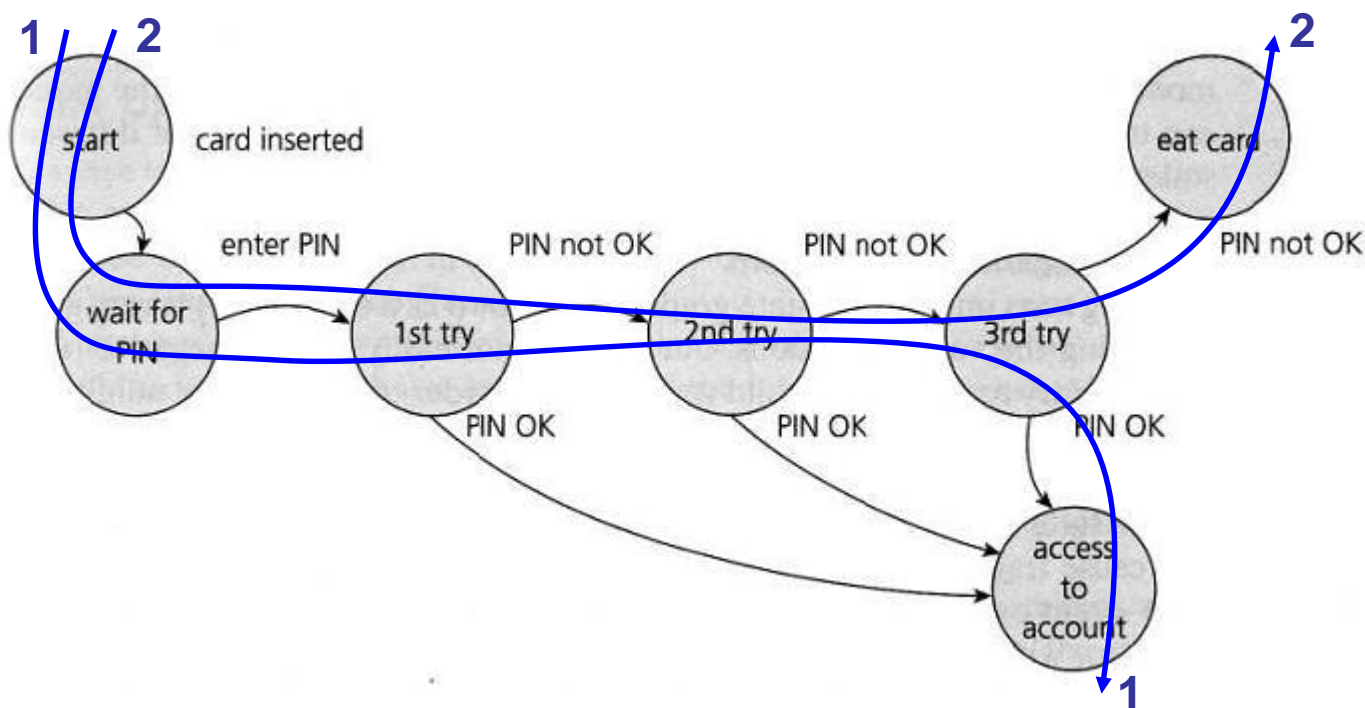
- ❑ Information in the state-transition diagrams can easily be used to create test cases.

- ❑ Four different levels of coverage can be defined:
 1. All States Covered Test Case
 2. All Transitions Covered Test Case
 3. All Events Covered Test Case
 4. All Paths Covered Test Case

4.3.5.1 All States Covered Test Case

❑ All States Covered Test Case

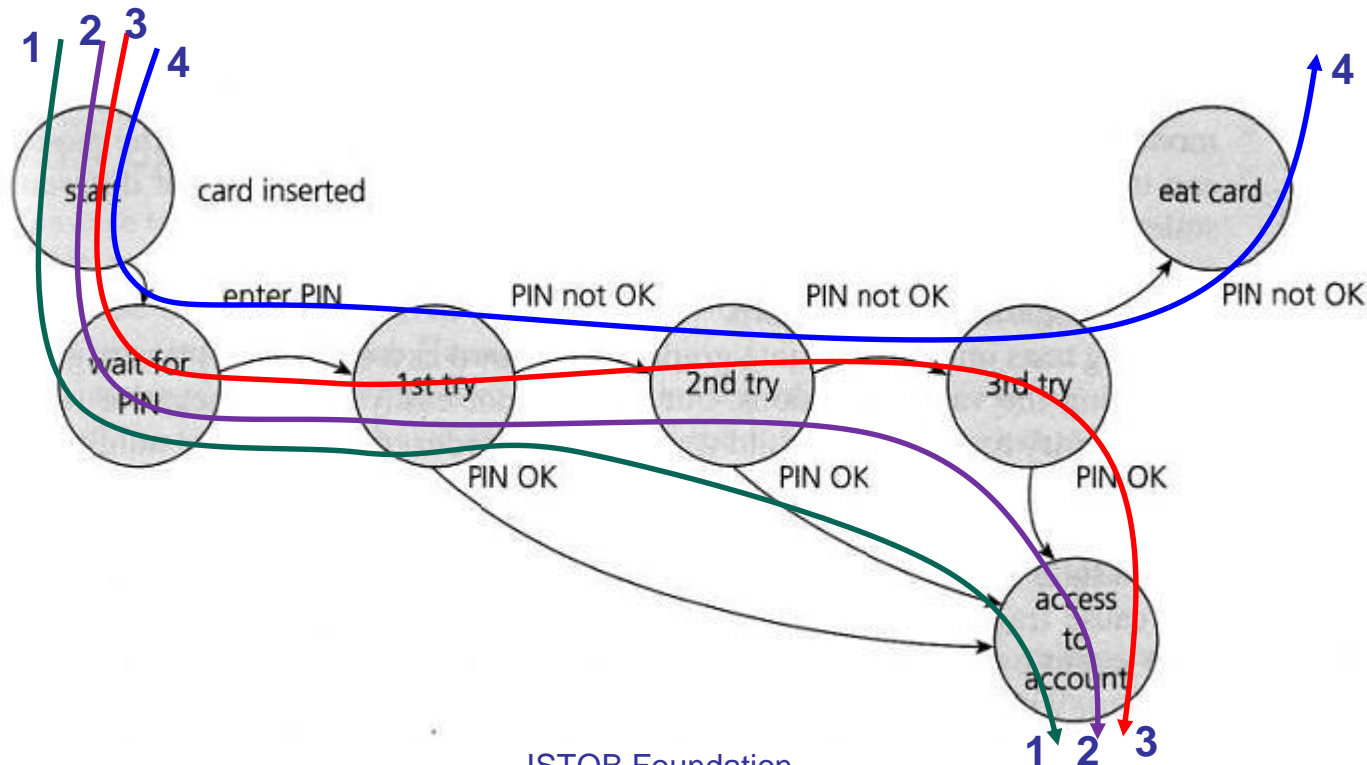
- Create a set of test cases such that **all states** are "visited" at least once under test.
- The set of two test cases shown below meets this requirement.
- Generally this is a weak level of test coverage.



4.3.5.2 All Transitions Covered Test Case

□ All Transitions Covered Test Case

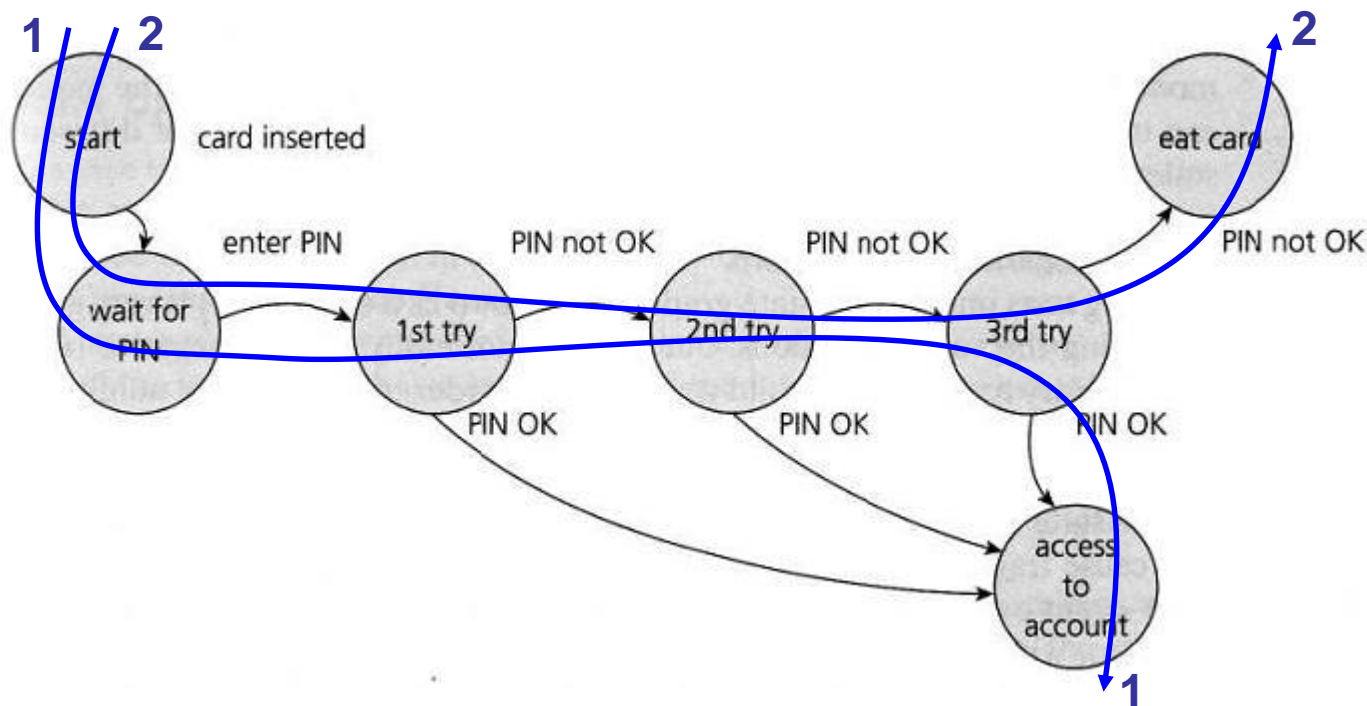
- Create a set of test cases that **all transitions** are exercised at least once under test
- This level of testing provides a good level of coverage without generation large numbers of tests
- This level is generally the one recommended.



4.3.5.3 All Events Covered Test Case

□ All Events Covered Test Case

- Create a set of test cases such that **all events** are triggered at least once under test.
- Note that the test cases that cover each event can be the same as those that cover each state.
- Again, this is a weak level of coverage.



4.3.5.4 All Paths Covered Test Case

□ All Paths Covered Test Case

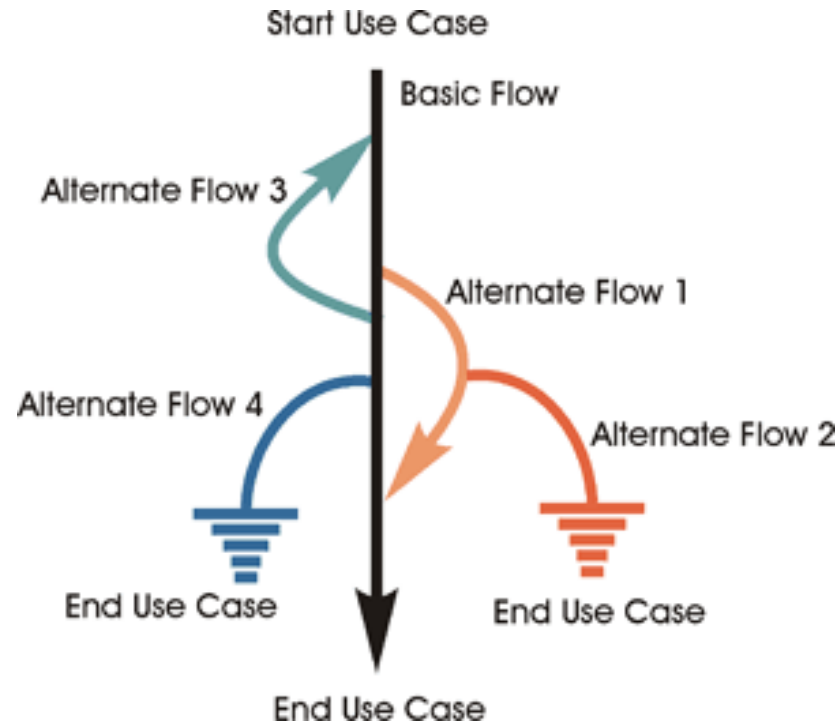
- Create a set of test cases such that all paths are executed at least once under test.
- While this level is the most preferred because of its level of coverage, it may not be feasible.
- If the state-transition diagram has loops, then the number of possible paths may be infinite.
- For example, given a system with two states, A and B, where A transitions to B and B transitions to A.
- A few of the possible paths are:
 - A→B
 - A→B→A
 - A→B→A→B→A→B
 - A→B→A→B→A→B→A
 - A→B→A→B→A→B→A→B→A→B

4.3.6 Use Case testing

- ❑ A "**use case**" is a scenario that describes the use of a system by an actor to accomplish a specific goal.
- ❑ An **actor** is something or someone which interacts with the system under test. Actors may be end users, other systems, or hardware devices.
- ❑ A "**scenario**" is a sequence of steps that describe the interactions between the actor and the system. Note that the use case is defined from the perspective of the user, not the system.
- ❑ **Use-case based testing (UCBT)**: A testing technique in which test cases are designed to execute user scenarios.

4.3.6 Use Case testing

- A Use Case is based on flows of event (basic flows and alternative flows)



4.3.6 Use Case testing

Example: A simple mail application – Normal cases (Basic flows):

Use Case	Actor	Steps	Expected Result
UC_Login	User	1. Type user name 2. Type password	Go to Inbox folder
UC_ReadMails	User	1. Press check mail button 2. Click the new mail	1. List inbox mails 2. New mail is opened
UC_ReplayMail	User	1. Press reply button 2. Enter send address 3. Enter mail title 4. Type the content 5. Press send button	1. A new empty mail opens 5. “The mail has been sent” appears on the screen

UC_Login: Abnormal cases (Alternative flows):

Use Case	Actor	Steps	Expected Result
Login	User	1. Type the wrong user name 2. Leave password blank	“User invalid” appears on the screen
Login	User	1. Type correct user name 2. Type the wrong password	“Password is not correct” appears on the screen

And go on with normal and abnormal cases till the requirements are covered.

4.3.6 Use Case testing

Example:

UC_Login: Basic flow → 1 Test Case

Test Case ID	Summary	Steps	Expected Result
UC_Login_01	Login successfully	1. Type user name 2. Type password	Go to Inbox folder

UC_Login: Abnormal cases (Alternative flows): → 2 Test cases

Test Case ID	Summary	Steps	Expected Result
UC_Login_02	Login with invalid user name	1. Type the wrong user name 2. Leave password blank	“User invalid” appears on the screen
UC_Login_03	Login with invalid password	1. Type correct user name 2. Type the wrong password	“Password is not correct” appears on the screen

4.3.6 Use Case testing

Key notes:

- ❑ Use Cases are used to specify the required functionality of an system.

- ❑ UCBT is best used in the phase of System test or wherever the tester is interested in exploring behavior that flows through multiple use case (scenarios)

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

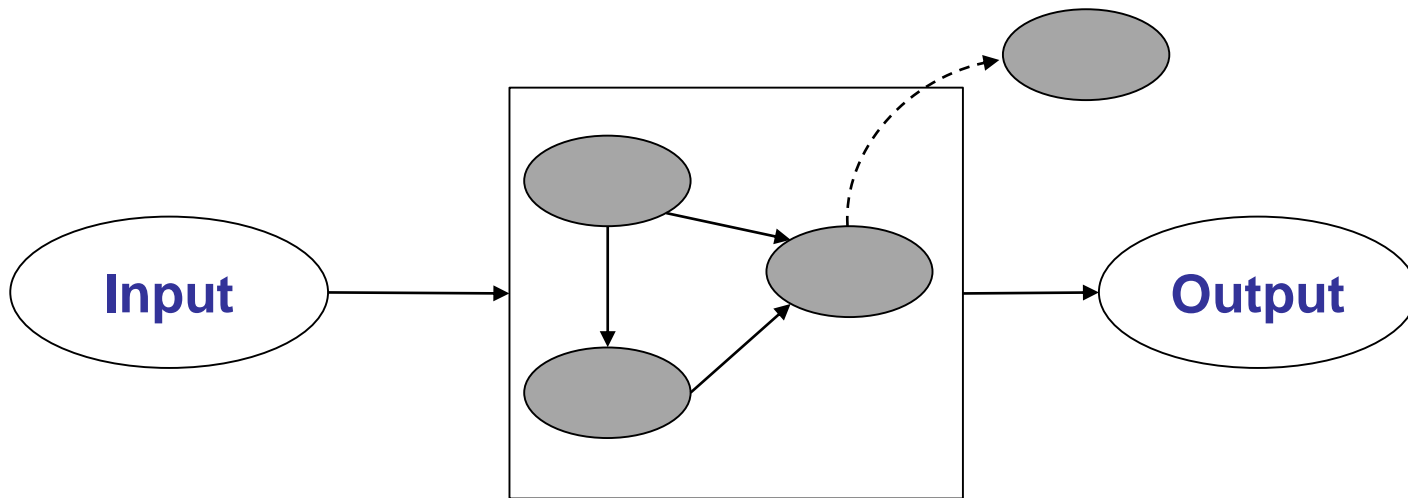
4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.4 White Box Tests



- ❑ Targeted at the underlying complexity of the software
 - Intimate knowledge of implementation
 - Good for testing individual functions
- ❑ Test the implementation and design

4.4.1 Control Flow Testing

- ❑ Control flow testing approach identifies the execution paths through a module of program code and then creates and executes test cases to cover those paths
- ❑ Control flow graphs are the foundation of control flow testing.
- ❑ These graphs document the module's control structure.
- ❑ Modules of code are converted to graphs, the paths through the graphs are analyzed, and test cases are created from that analysis.

4.4.1 Control Flow Testing

❑ Level 1: “100% statement coverage”

- The lowest coverage level
- This means that every statement within the module is executed, under test, at least once.

$$\text{Statement Coverage} = \frac{\text{Number of statement exercised}}{\text{Total number of statement}} \times 100\%$$

- ❑ While this may seem like a reasonable goal, many defects may be missed with this level of coverage.

4.4.1 Control Flow Testing

❑ Level 1 – Example:

Let's look at a code sample below:

```
1.READ A
2.READ B
3.C = A + 2*B
4.IF C > 50 THEN
5. PRINT "Large C"
6.END
```

❑ We have 3 test cases:

Test 1: A=2, B=3

Test 2: A=0, B=25 → Statement Coverage: → 85% (5/6 statements)

Test 3: A=47, B=1

❑ Statement Coverage: → increase to 100% (6/6 statements)

Test 4: A=15, B=20

4.4.1 Control Flow Testing

❑ Level 2: “100% decision coverage”

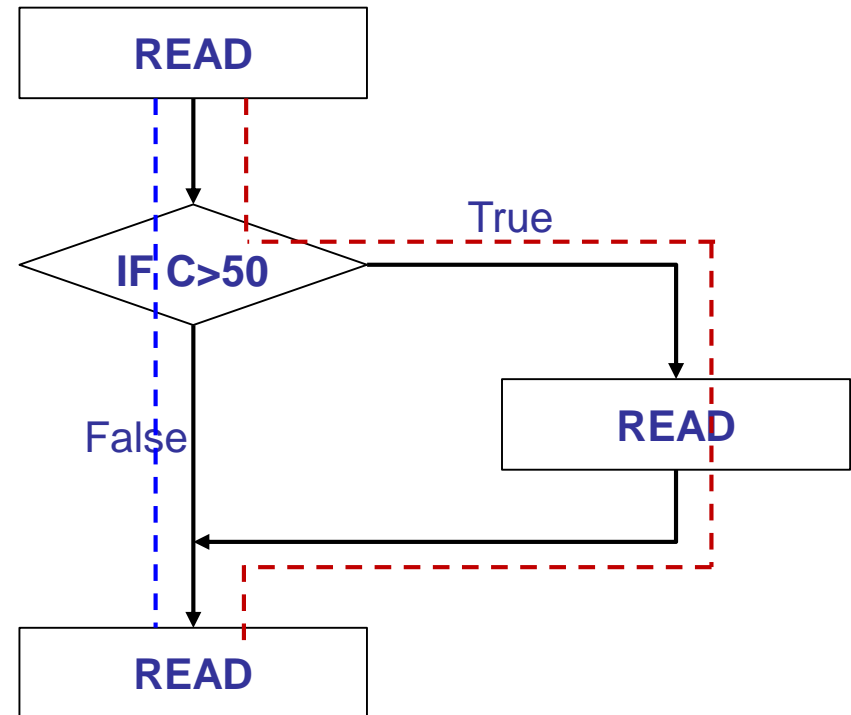
- This is also called "branch coverage."
- Each decision that has a **TRUE** and **FALSE** outcome is evaluated at least once.

$$\text{Decision Coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\%$$

4.4.1 Control Flow Testing

❑ Level 2 – Example:

- 1.READ A
- 2.READ B
- 3.C = A + 2*B
- 4.IF C > 50 THEN
5. PRINT "Large C"
- 6.END



❑ We have 1 test cases to cover 100% statement coverage:

Test 1: A=15, B=20 → Decision Coverage: → 50% (1/2 decisions)

❑ Decision Coverage: → increase to 100% (2/2 decisions)

Test 2: A=10, B=15

4.4.1 Control Flow Testing

- ❑ **Level 3: “100% condition coverage”**
 - This is also called “Atomic condition coverage.”
- ❑ Consider these more complicated statements:
if (a>0 && c==1)
 {x=x+1;}
if (b==3 || d<0)
 {y=0;}

❑ To be **TRUE**, the first statement requires **a** greater than 0 **and** **c** equal 1. The second requires **b** equal 3 **or** **d** less than 0.
- ❑ In the first statement if the value of **a** were set to 0 for testing purposes then the **c==1** part of the condition would not be tested.
- ❑ At this level enough test cases are written so that each condition that has a **TRUE** and **FALSE** outcome that makes up a decision is evaluated at least once.

4.4.1 Control Flow Testing

❑ Level 4: “100% decision/condition coverage”

- Given the possible combination of conditions such as these, to be more complete "100% decision/condition" coverage can be selected.
- At this level test cases are created for every condition and every decision.

❑ Consider this situation:

```
if(x&&y) // note: && indicates logical AND  
    { conditionedStatement; }
```

- ### ❑ We can achieve condition coverage with two test cases (**x=TRUE, y=FALSE** and **x=FALSE, y=TRUE**) but note that with these choices of data values the conditioned Statement will never be executed

4.4.1 Control Flow Testing

☐ Level 5: “100% multiple condition coverage”

- All combinations of atomic conditions

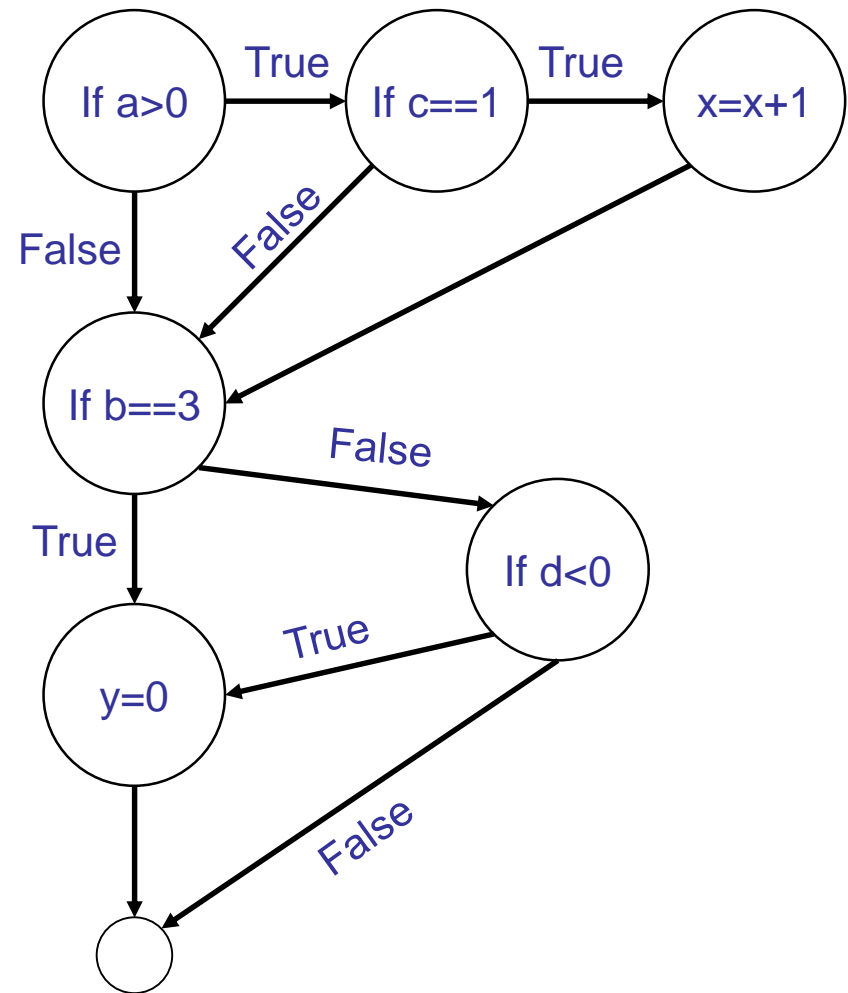
☐ if (a>0 && c==1) {x=x+1;} if (b==3 || d<0) {y=0;}

☐ // note: || means logical OR

☐ will be evaluated as:

4.4.1 Control Flow Testing

- ❑ This level of coverage can be achieved with four test cases: .
 - $a > 0$, $c = 1$, $b = 3$, $d < 0$
 - $a \sim 0$, $c = 1$, $b = 3$, $d \sim 0$
 - $a > 0$, $C1$, $b \ 3$, $d < 0$
 - $a \sim 0$, $c = 1$, $b = 3$, $d \sim 0$
- ❑ Achieving 100% multiple condition coverage also achieves decision coverage, condition coverage, and decision/condition coverage.
- ❑ Note that multiple condition coverage does not guarantee path coverage



4.4.1 Control Flow Testing

☐ Level 6

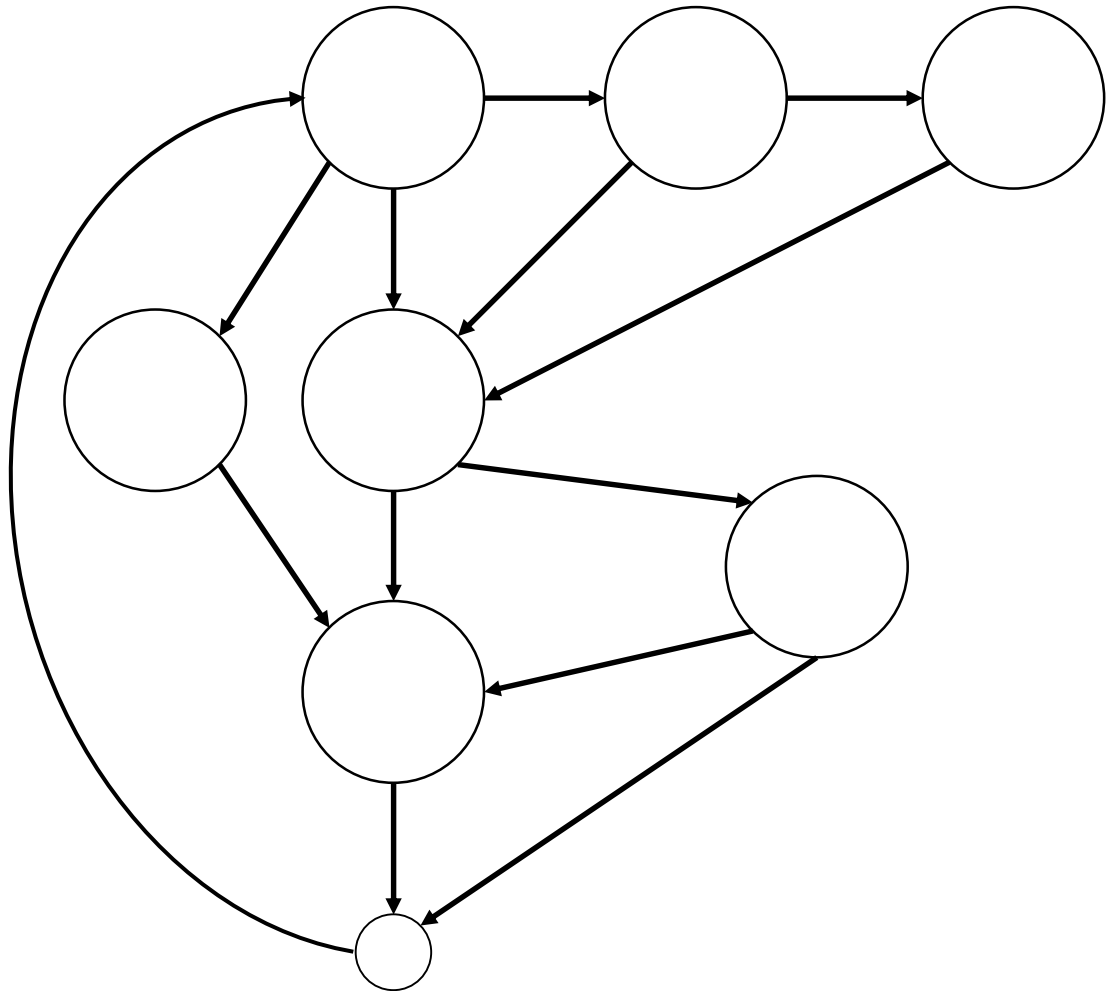
- ☐ When a module has loops in the code paths such that the number of paths is infinite, a significant but meaningful reduction can be made by limiting loop execution to a small number of cases.

- ☐ The first case is to execute the loop n times where n is a small number representing a typical loop value, the fourth is to execute the loop its maximum number of times m . in addition you might try $m-1$ and $m+1$.

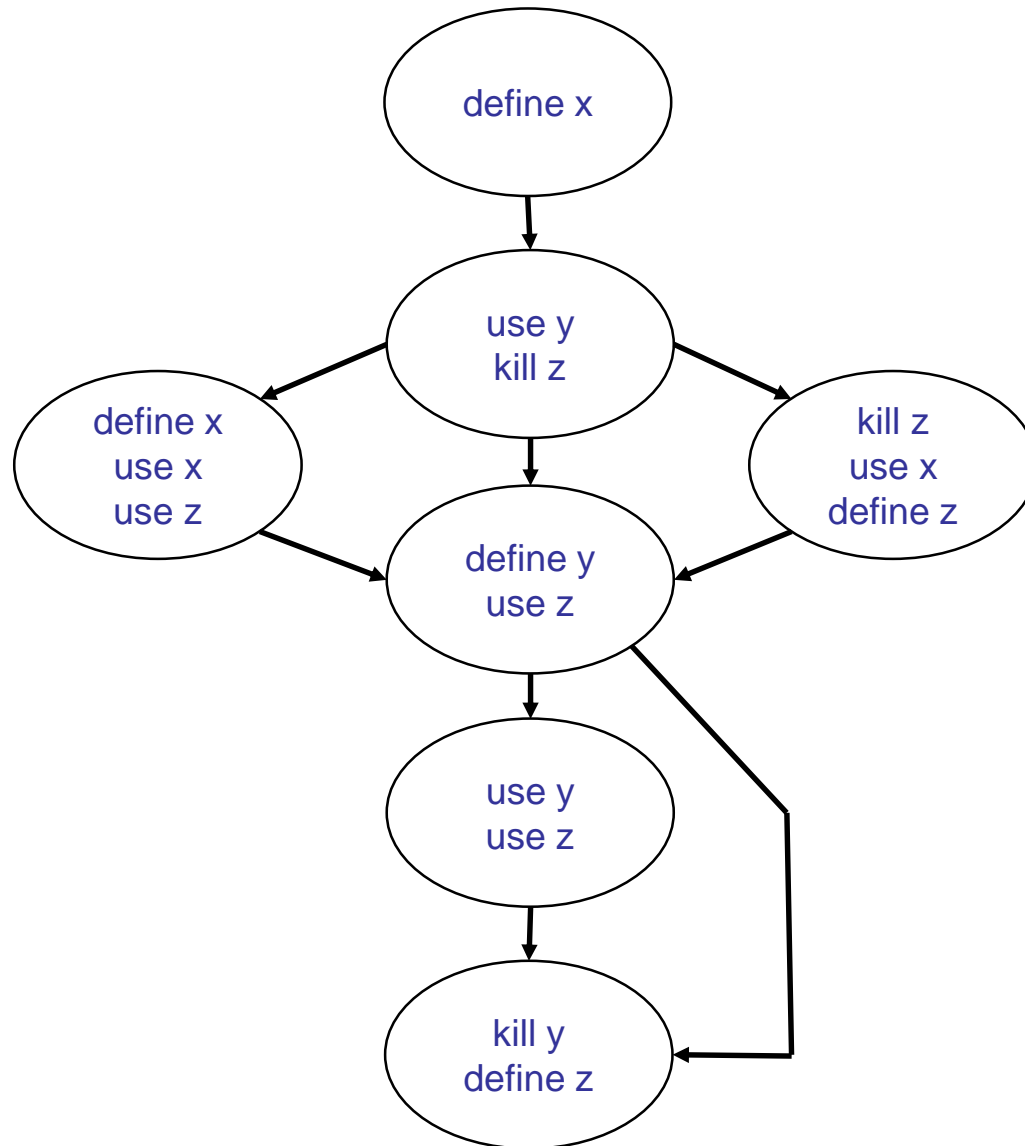
4.4.1 Control Flow Testing

❑ Level 7: - “100% path coverage”

- ❑ For code modules without loops the number of paths is generally small enough that a test case can actually be constructed for each path.



4.4.1 Control Flow Testing



4.4.2 Control Flow Testing

- ❑ There are stronger levels of structural coverage beyond decision coverage, for example, condition coverage and multiple condition coverage.
- ❑ The concept of coverage can also be applied at other test levels (e.g. at integration level) where the percentage of modules, components or classes that have been exercised by a test case suite could be expressed as module, component or class coverage.
- ❑ Tool support is useful for the structural testing of code.

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.5 Experience-based techniques

- ❑ Tests are derived from the tester's skill and intuition and their experience with similar applications and technologies.
- ❑ When used to augment systematic techniques, intuitive testing can be useful to identify special tests not easily captured by formal techniques, especially when applied after more formal approaches.
- ❑ However, this technique may yield widely varying degrees of effectiveness, depending on the tester's experience.

4.5.1 Exploratory Testing

❑ Exploratory testing is

- Concurrent (simultaneous)
 - test design,
 - test execution,
 - test logging
 - learning,
- Based on a test charter containing test objectives,
- And carried out within time-boxes.

❑ This is an approach that is most useful where:

- There are few or inadequate specifications and severe time pressure
- Or in order to complement other, more formal testing.

4.5.1 Exploratory Testing

Chartered Exploratory Testing

- ❑ When performing "chartered exploratory testing," a charter is first created to guide the tester within the timebox.
- ❑ This charter defines
 - What to test
 - What documents (requirements, design, user manual, etc.) are available to the tester
 - What tactics to use
 - What kinds of defects to look for
 - What risks are involved
- ❑ Ex: "Find out how many concurrent instances of PowerPoint can be run at the same time. Use large and small presentations. See if you can get Windows to crash"

4.5.1 Exploratory Testing

Chartered Exploratory Testing

❑ Charters include:

- Thoroughly investigate a specific system function
- Define and then examine the system's workflows
- Identify and verify all the claims made in the user manual
- Understand the performance characteristics of the software
- Ensure that all input fields are properly validated
- Force all error conditions to verify each error message
- Check the design against user interface standards

4.5.1 Exploratory Testing

Exploratory Testing Tasks

☐ Explore

- The Elements of the product
- How the product should work

☐ Design Tests

- Which elements
- Speculate on possible quality problems

☐ Execute Tests

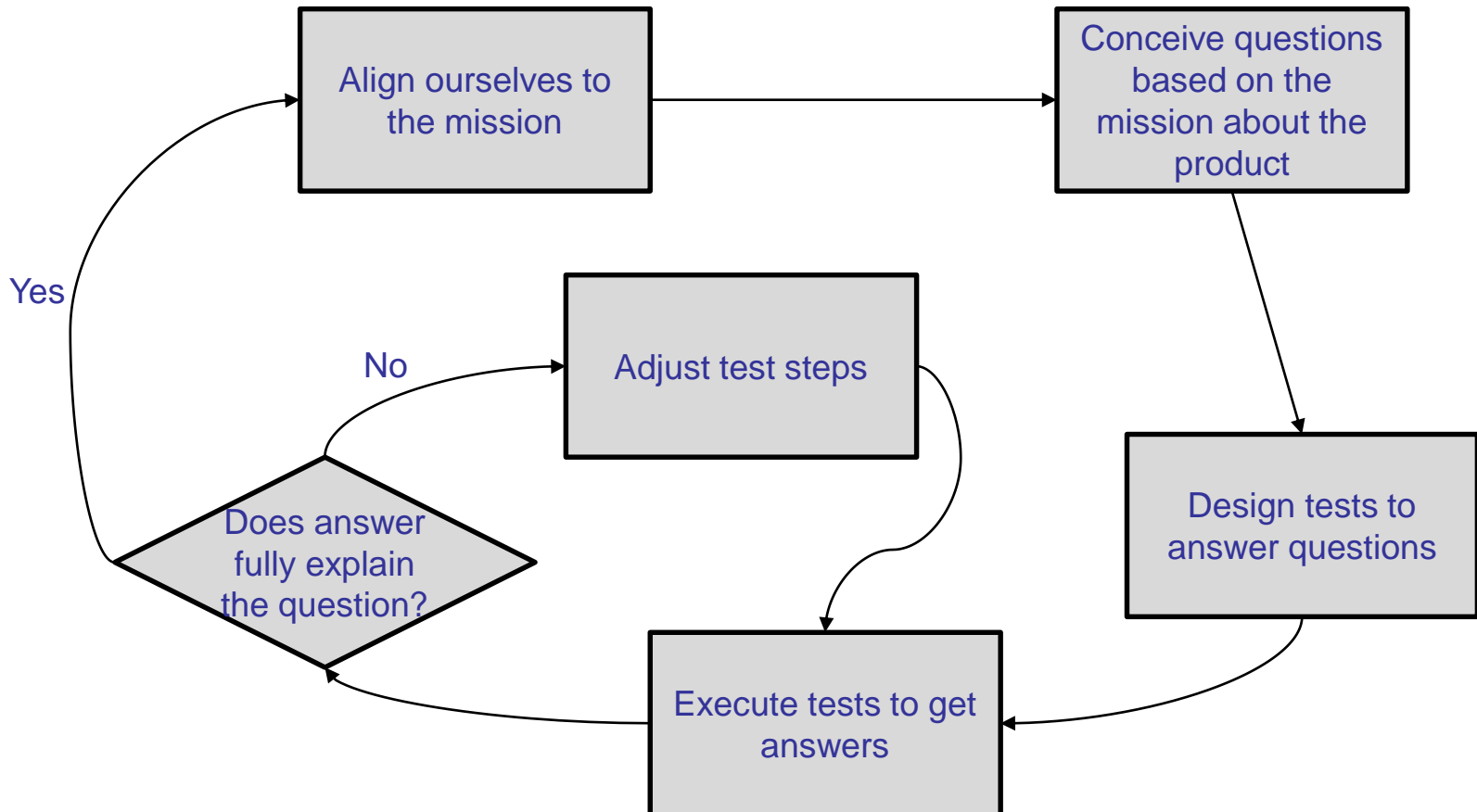
- Observe behavior
- Evaluate against expectations

☐ All with test design techniques best suited for the product

4.5.1 Exploratory Testing

Exploratory Testing Process

□ ET Process (cont.)



4.5.1 Exploratory Testing

Exploratory Testing Process

□ Exploratory Testing Process

- Creating a conjecture (a mental model) of the proper functioning of the system
- Designing one or more tests that would disprove the conjecture
- Executing these tests and observing the outcomes
- Evaluating the outcomes against the conjecture
- Repeating this process until the conjecture is proved or disproved

4.5.1 Exploratory Testing

Exploratory Testing – Tester

❑ Good exploratory Testers are:

- Good modelers, able to create mental models of the system and its proper behavior.
- Careful observers, able to see, hear, read, and comprehend.
- Skilled test designers, able to choose appropriate test design techniques in each situation. Bach emphasizes, "An exploratory tester is first and foremost a test designer."

4.5.1 Exploratory Testing

Advantages of Exploratory Testing

❑ Good exploratory Testers are:

- Useful when you are asked to provide rapid feedback on a product's quality on short notice, with little time, off the top of your head,
 - when requirements are vague or even nonexistent, or early in the development process when the system may be unstable.
- Useful when, once a defect is detected, we want to explore the size, scope, and variations of that defect to provide better feedback to our developers.
- Useful addition to scripted testing when the scripted tests become "tired," that is, they are not detecting many errors

4.5.1 Exploratory Testing

Disadvantages of Exploratory Testing

❑ Good exploratory Testers are:

- No ability to prevent defects.
 - Because the design of scripted test cases begins during the requirements gathering and design phases, defects can be identified and corrected earlier.
- If you are already sure exactly which tests must be executed, and in which order, there is no need to explore. Write and then execute scripted tests.
- If you are required by contract, rule, or regulation to use Scripted testing then do so. Consider adding exploratory tests as a complementary technique

4.5.2 Error Guessing

- ❑ Error guessing is a test technique where the experience of the tester is used to:
 - "guess" what/where defects might be present in the system
 - Design test cases specifically to expose these defects
- ❑ The defect and failure lists can be built based on:
 - Tester's experience and intuition
 - Available defects and failure data
 - Common knowledge about why software fails
 - Test for errors that you know about in similar programs.
 - Once an error has been detected and identified as connected with one or more input items, test for similar errors connected to other input items
- ❑ Ad hoc, not really a technique, But can be quite effective

4.5.2 Error Guessing

- ❑ Some samples of error situations:
 - Initialization of data
 - Wrong kind of data
 - Handling of real data
 - Error management
 - Restart/Recovery
 - Proper handling of concurrent procedure

4.5.3 Checklist-based Testing

- ❑ Checklist is high-level list of test cases to cover test
- ❑ Checklist can be built based on experience, knowledge about
 - what is important for the user, or
 - an understanding of why and how software fails.
- ❑ Checklists can be created to support various test types, including functional and non-functional testing.
- ❑ Checklist for Checklist-based testing
 - can provide guidelines and a degree of consistency
 - some variability in the actual testing is likely to occur, resulting in potentially greater coverage but less repeatability

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.6 Choosing Test techniques

- ❑ The choice of which test techniques to use depends on a number of factors:
 - The Type of component or system
 - Component or system complexity
 - Regulatory Standards
 - Customer or Contractual Requirements
 - Level of risk, Type of Risk
 - Test Objective
 - Documentation Available
 - Tester knowledge and skills
 - Available tools
 - Time and Budget
 - Development Life Cycle models
 - Expected use of the software
 - Previous experience with using the test techniques
 - The types of defects expected in component or system

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.7.1 Test Design Specification

- ❑ The purpose of the test design specification
 - To identify a set of features to be tested and to describe a group of test cases that will adequately test those features.
 - Refinements to the approach listed in the test plan may be specified.

- ❑ Test Design Specification

- Test design specification identifier
- Features to be tested
- Approach refinements
- Test identification
- Feature pass/fail criteria

4.7.1 Test Design Specification

- ❑ Test design specification identifier
 - A unique identifier so that this document can be distinguished from all other documents.
- ❑ Features to be tested
 - Identifies the test items and the features that are the object of this test design specification.
- ❑ Approach refinements
 - Specifies the test techniques to be used for this test design.
- ❑ Test identification
 - Lists the test cases associated with this test design. Provides a unique identifier and a short description for each test case.
- ❑ Feature pass/fail criteria
 - The criteria used to determine whether each feature has passed or failed testing

4.7.2 Test Case Specification

- ❑ The purpose of the test case specification is to specify in detail each test case listed in the test design specification

- ❑ Test Case Specification

- Test case specification identifier
- Test items
- Input specifications
- Output specifications
- Environmental needs
- Special procedural requirements
- Intercase dependencies

4.7.2 Test Case Specification

☐ Test case specification identifier

- A unique identifier so that this document can be distinguished from all other documents.

☐ Test items

- Identifies the items and features to be tested by this test case.

☐ Input specifications

- Specifies each input required by this test case.

4.7.2 Test Case Specification

☐ Output specifications

- Specifies each output expected after executing this test case.

☐ Environmental needs

- Any special hardware, software, facilities, etc. required for the execution of this test case that were not listed in its associated test design specification.

☐ Special procedural requirements

- Defines any special setup, execution, or cleanup procedures unique to this test case.

☐ Intercase dependencies

- Lists any test cases that must be executed prior to this test case.

4.7.2 Test Procedure Specification

- ❑ The purpose of the test procedure specification
 - To specify the steps for executing a test case and the process for determining whether the software passed or failed the test.

- ❑ Test Procedure Specification

- Test procedure specification identifier
- Purpose
- Special requirements
- Procedure steps

4.7.2 Test Procedure Specification

☐ Test procedure specification identifier

- A unique identifier so that this document can be distinguished from all other documents.

☐ Purpose

- Describes the purpose of the test procedure and its corresponding test cases.

☐ Special requirements

- Lists any special requirements for the execution of this test procedure.

☐ Procedure steps

- Lists the steps of the procedure. Possible steps include: Set up, Start, Proceed, Measure, Shut Down, Restart, Stop, and Wrap Up.

Test Design Techniques

4.1 Test development process (K2)

4.2 Categories of test design techniques (K2)

4.3 Specification-based or black-box techniques (K3)

4.4 Structure-based or white-box techniques (K3)

4.5 Experience-based techniques (K2)

4.6 Choosing test techniques (K2)

4.7 Test specification documentation

4.8 Writing test cases

4.8 Writing test cases



Exercises

CHAPTER 5

Test Management

5. Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

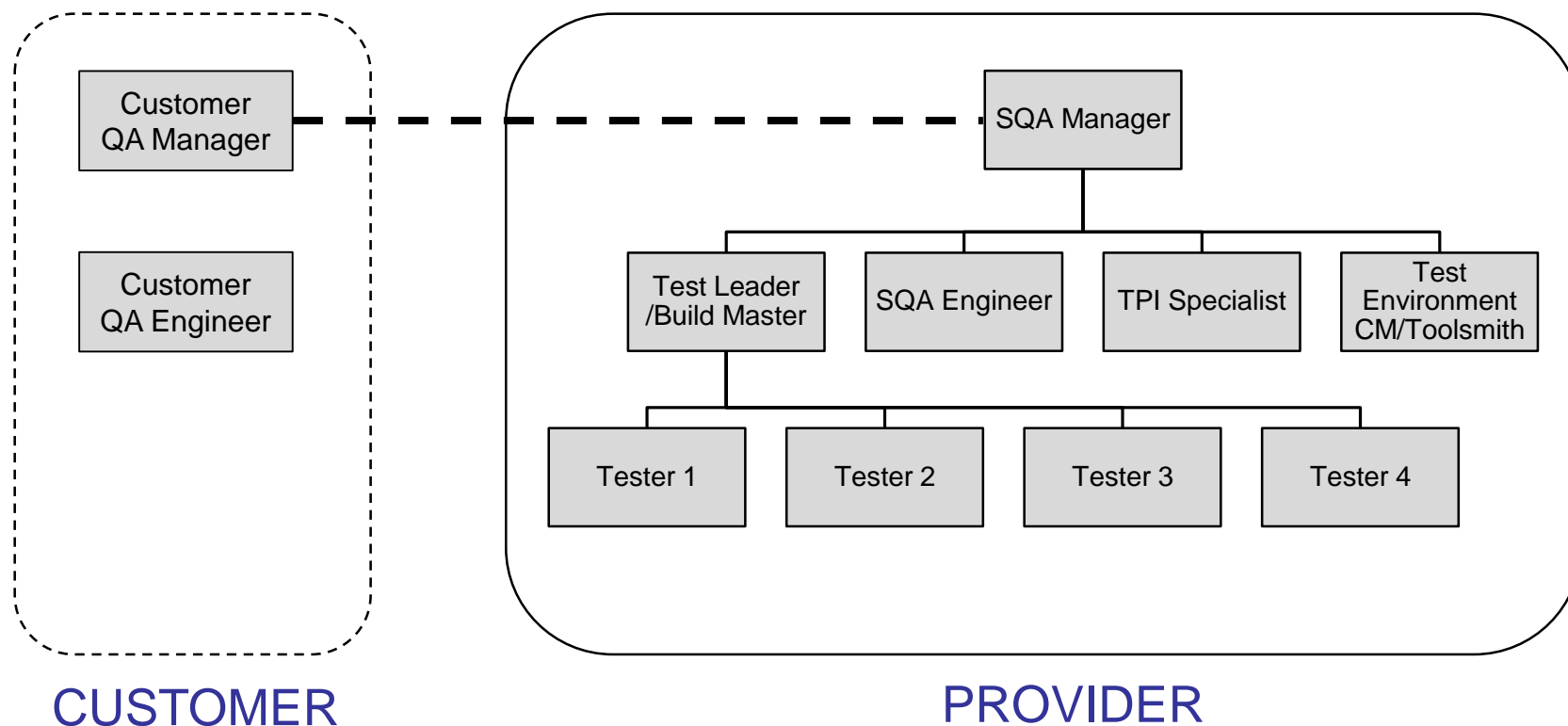
5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.1.1 Independence testing



5.1.1 Independence testing

- ❑ Degrees of independence in testing:
 - No independent testers. Developers test their own code.
 - Independent testers within the development teams.
 - Independent test team or group within the organization
 - Independent test specialists for specific test types such as usability testers, security testers or certification testers
 - Independent testers outsourced or external to the organization.

5.1.1 Independence testing

❑ Potential benefits of independence in testing:

- Independent testers are likely to recognize different kinds of failures compared to developers because of their different backgrounds, technical perspectives, and biases
- An independent tester can verify, challenge, or disprove assumptions made by stakeholders during specification and implementation of the system.

❑ Potential drawbacks of independence in testing:

- Isolation from the development team leading to a lack of collaboration, delays in providing feedback to the development team, or an adversarial relationship with the development team
- Developers may lose a sense of responsibility for quality
- Independent testers may be seen as a bottleneck or blamed for delays in release
- Independent testers may lack some important information

5.1.2 Tasks and Qualification

❑ Test manager tasks

- Write a test policy for the organization.
- Plan the test activities
 - Selecting test approaches
 - estimating the time
 - effort and cost of testing
 - acquiring resources
 - defining test levels, cycles, approach, and objectives
 - planning defect management;
- Write and update the test plan(s)/ test strategy for the project
- Coordinate the test plan(s) with project managers, product owners...
- Contribute the testing perspective to other project activities such as integration planning

5.1.2 Tasks and Qualification

❑ Test manager tasks

- Initiate the specification, preparation, implementation and execution of tests, and monitor and control the execution
- Ensure configuration management
- Report test progress, summary report
- Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product
- support the selection and implementation of tools to support the test process
- Decide about the implementation of test environment(s)
- Promote and advocate the testers, the test team, and the test profession within the organization
- Develop the skills and careers of testers (e.g., through training plans, performance evaluations, coaching, etc.)

5.1.2 Tasks and Qualification

❑ Tester tasks

- Review and contribute to test plans
- Analyze, review, and assess requirements, user stories and acceptance criteria, specifications, and models for testability (i.e., the test basis)
- Identify and document test conditions, and capture traceability between test cases, test conditions, and the test basis
- Design, set up, and verify test environment(s), often coordinating with system administration and network management
- Design and implement test cases and test procedures
- Prepare and acquire test data
- Create the detailed test execution schedule

5.1.2 Tasks and Qualification

❑ Tester tasks

- Execute tests, evaluate the results, and document deviations from expected results
- Use appropriate tools to facilitate the test process
- Automate tests as needed (may be supported by a developer or a test automation expert)
- Evaluate non-functional characteristics such as performance efficiency, reliability, usability, security, compatibility, and portability
- Review tests developed by others.

Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.2.1 Test Plan

☐ What is project success?

- It is completed within the scheduled time.
- It is completed within the budgeted cost.
- It results in a good-quality product that fulfills customer requirements.

☐ Planning is required to achieve success

5.2.1 Test Plan

☐ Test planning

- Defining overall approach and strategy for testing
- Deciding about the test environment
- Definition of the test levels
- Deciding how to evaluate the test results
- Selecting metrics for monitoring and controlling test work, defining test exit criteria
- Determining how much test documentation shall be prepared and deciding about templates
- Writing the test plan
- Estimating test effort and test costs

- ### ☐ Test Plan is a document describing the scope, approach resources and schedule of intended test activities.

5.2.1 Test Plan

❑ Test Plan Template

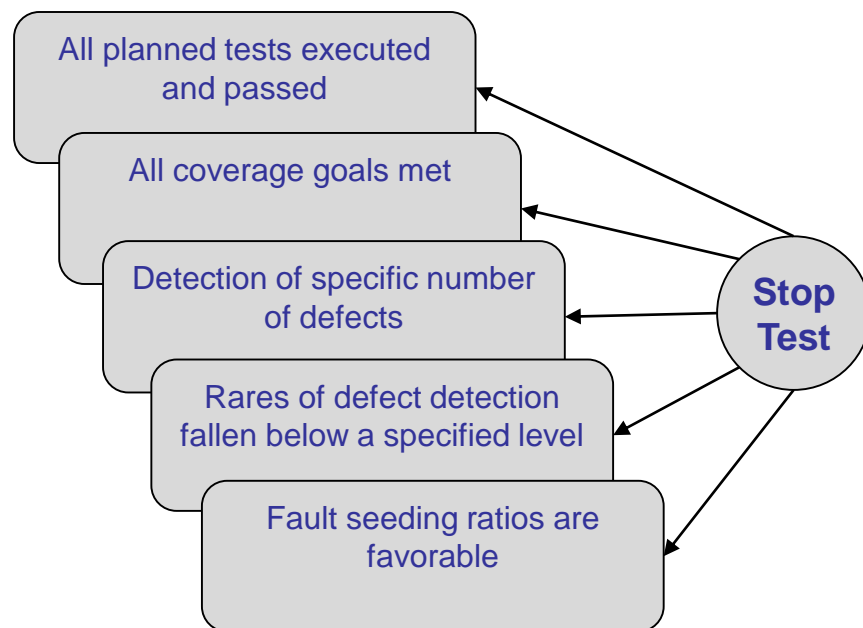
- | | |
|---|--------------------------------------|
| ➤ Test Plan Identifier | ➤ Test Deliverables |
| ➤ Introduction | ➤ Test Tasks |
| ➤ Test Items | ➤ Environmental needs |
| ➤ Features to be tested | ➤ Responsibilities |
| ➤ Features not to be tested | ➤ Staffing and training needs |
| ➤ Approach | ➤ Schedule |
| ➤ Item pass/fail criteria | ➤ Risks and contingencies |
| ➤ Suspension and resumption criteria | ➤ Approvals |

5.2.2 Prioritizing Tests

- ☐ The usage frequency of a function or the probability of failure in software use
- ☐ Risk of failure
- ☐ The visibility of a failures for the end user is a further criterion for prioritization of test cases.
- ☐ Priority of the requirements
- ☐ Quality characteristics
- ☐ The complexity of the individual component and system
- ☐ Project risk

5.2.3 Exit criteria

- ❑ The purpose of exit criteria is to define when to stop testing.
- ❑ Typically exit criteria may consist of:
 - Test coverage: Thoroughness measures, such as coverage of code, functionality or risk
 - Product quality: Estimates of defect density or reliability measures
 - Residual risks, such as defects not fixed or lack of test coverage in certain areas.
 - Economics constraints (cost, time, quality)



5.2.4 Cost and Economic Aspects

❑ Cost of quality:

1. Cost of prevention: Training, documents processes, Equipment..
2. Cost of detection (Appraisal costs): testing, inspections...
3. Cost of Defect/ failure:
 - Direct failure cost
 - Indirect failure cost (External Cost)
 - Cost for defect correction (Internal Cost)

❑ **The factors are important to estimate the costs of testing:**

- Maturity of the development process
- Quality and testability of the software
- Test infrastructure
- Qualification of employees
- Quality requirement
- Test strategy

5.2.5 Test estimation

- ❑ General test effort estimation approaches:
 - The metrics-based approach: based on effort data/metrics of former or similar projects or based on typical values.
 - The expert-based approach: based on estimates made by the owner of the tasks or by experts.
 - Work Breakdown Structure (WBS): list down all testing tasks/ features (Decomposition).

- ❑ Note: if no data at hand, rule of thumb can be helpful.

- ❑ Once the test effort is estimated, resources can be identified and a schedule can be drawn up.

5.2.5.1 Test estimation – Testing Effort

- ❑ The testing effort depends on
 - Characteristics of the product:
 - the quality of the specification
 - other information used for test models (i.e. the test basis)
 - the size of the product
 - the complexity of the problem domain
 - the requirements for reliability
 - security
 - the requirements for documentation.
 - Characteristics of the development process (the stability of the organization, tool used, resource skill, time pressure)
 - The outcome of testing: the number of defects and the amount of rework required.

5.2.5.2 Test estimation - Test Case Estimation

- ❑ Number of test cases required is based on:
 - Testing all functions and features in the SRS
 - Including an appropriate number of ALAC (Act Like A Customer) tests including:
 - Do it wrong
 - Use wrong or illegal combination of inputs
 - Don't do enough
 - Do nothing
 - Do too much
 - Achieving some test coverage goal
 - Achieving a software reliability goal

Considerations in Test Estimation

- ❑ Test Complexity – It is better to have many small tests that a few large ones.
- ❑ Different Platforms - Does testing need to be modified for different platforms, operating systems, etc.
- ❑ Automated or Manual Tests – Will automated tests be developed? Automated tests take more time to create but do not require human intervention to run.

Estimating Tests Required

SRS Reference	Estimated Number of Tests Required	Notes
4.1.1	3	2 positive and 1 negative test
4.1.2	2	2 automated tests
4.1.3	4	4 manual tests
4.1.4	5	1 boundary condition, 2 error conditions, 2 usability tests
...		
Total	165	

Estimation Test Development Time

- ❑ **Estimated Number of Tests: 165**
- ❑ **Average Test Development Time: 0.5 (man-hours/test)**
- ❑ **Estimated Test Development Time: 82.5 (man-hour)**

Estimation Test Execution Time

- ❑ **Estimated Number of Tests: 165**
- ❑ **Average Test Execution Time: 0.5 (man-hours/test)**
- ❑ **Estimated Test Execution Time: 82.5 (man-hour)**
- ❑ **Estimated Regression Testing (50%): 41.25 (man-hour)**
- ❑ **Total Estimated Test Execution Time: 123.75 (man-hour)**

5.2.6 Test approaches (strategies)

- ❑ Classify test approaches or strategies
- ❑ Defines the project's testing objectives and means to achieve them
- ❑ Determines testing effort and costs
- ❑ Selecting appropriate test strategy is one of the most important planning task decisions of test manager
- ❑ Goal is to choose test approach that optimizes the relation between costs of testing and costs of defects as well as minimizes the risk
- ❑ Test costs should be less than the costs of defects and deficiencies in the final product.

5.2.6 Test approaches (strategies)

- ❑ It's based on the point in time at which the bulk of the test design work is begun:
 - Preventative approaches, where tests are designed as early as possible.
 - Testers are involved from the beginning
 - Test manager can optimize testing and reduce testing costs
 - The use of the general V-model with emphasis on design reviews, will contribute a lot to prevent defects
 - Reduced defect density during test execution
 - Especially, in safety critical software, a preventive approach may be mandatory.
 - Reactive approaches, where test design comes after the software or system has been produced.
 - Testers are involved (too) late
 - When a preventive approach cannot be chosen
 - Test manager must react appropriately
 - "Exploratory testing strategy can be used:
 - Tester "explores" the test object and the test design
 - Execution and evaluation occurs nearly concurrently

5.2.6 Test approaches (strategies)

- ❑ It's based on different sources of information during test planning and test design:
 - Analytical approach (e.g: Risk-based testing)
 - Test planning is founded on data and (mathematical) analysis of these data
 - Amount and intensity of testing are chosen based on individual or multiple parameters (costs, time, coverage, etc.)
 - Heuristic approach:
 - Test planning is founded on experience of experts and/or on rules of thumb.
 - When no data is available

5.2.6 Test approaches (strategies)

❑ The selection of a test approach

- Risk of failure of the project, hazards to the product and risks of product failure to humans, the environment and the company.
- Skills and experience of the people in the proposed techniques, tools and methods.
- The objective of the testing Endeavour and the mission of the testing team.
- Regulatory aspects, such as external and internal regulations for the development process.
- The nature of the product and the business

Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.3.1 Test progress monitoring

❑ Test monitoring:

- Is a task of test management
- Periodically checking the status of a test project
- Report to compare the actual and planned

❑ The purpose of test monitoring

- The feedback on how testing is going
- Test results
- The status of testing
- Data for future test estimation

5.3.1 Test progress monitoring

- ❑ To measure your monitoring, test metrics may be useful:
 - Percentage of work done in test case preparation.
 - Percentage of work done in test environment preparation.
 - Test case execution
 - Defect information
 - Test coverage.
 - Subjective confidence of testers in the product
 - Dates of test milestones. >Testing costs.

5.3.2 Test control

☐ Test monitoring:

- Is a task of test management
- Periodically checking the status of a test project
- Report to compare the actual and planned

☐ The purpose of test monitoring

- The feedback on how testing is going
- Test results
- The status of testing
- Data for future test estimation

5.3.2 Test control

- ❑ Any guiding or corrective actions taken as a result of information and metrics gathered and reported.

- ❑ Examples
 - Making decisions based on information from test monitoring
 - Re-prioritize tests when an identified risk occurs (e.g. software delivered late).
 - Change the test schedule due to availability of a test environment.
 - Set an entry criterion requiring fixes to have been retested by a developer before accepting them into a build

- ❑ Changes to test plan must be communicated clearly

5.3.3 Metrics used in Testing

- ❑ Test case execution (e.g., number of test cases run/not run, test cases passed/failed, and/or test conditions passed/failed)
- ❑ Defect information (e.g., defect density, defects found and fixed, failure rate, and confirmation test results)
- ❑ Test coverage of requirements, user stories, acceptance criteria, risks, or code
- ❑ Task completion, resource allocation and usage

5.3.4 Purposes, Contents, and Audiences for Test Reports

- ❑ The purpose of the test report is to summarize and communicate test activity information, both during and at the end of a test activity.

- ❑ Two types of test reports (Refer to ISO standard (ISO/IEC/IEEE 29119-3))
 - Typical test progress report
 - Test summary report

5.3.4.1 Test Summary Report

❑ The purpose: to summarize the results of the testing activities and to provide an evaluation based on these results.

❑ Test summary report

- **Test summary report identifier**
- **Summary**
- **Variances**
- **Comprehensive assessment**
- **Summary results**
- **Evaluation**
- **Summary of activities**
- **Approval**

5.3.4.1 Test Summary Report

- ☐ Test summary report identifier:
 - A unique label so you can refer to that document.
- ☐ Summary
 - Summarize what was tested and what happened. Point to all relevant documents.
- ☐ Variances:
 - If any test items differed from their specifications, the testing process didn't go as planned, why things were different.
- ☐ Comprehensiveness assessment
 - How thorough was testing, in the light of how thorough the test plan said it should be? What wasn't tested well enough? Why not?
- ☐ Summary of results
 - Which problems have been dealt with? What problems remain?
- ☐ Evaluation
 - How good are the test items? What's the risk that they might fail?
- ☐ Summary of activities
 - In outline, what were the main things that happened? What did they cost (people, resource use, time, money)?
- ☐ Approvals
 - Who has to approve this report? Get their signatures.

Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.4 Configuration management

❑ The purpose

- To establish and maintain the integrity of the products (components, data and documentation) of the software or system through the project and product life cycle.

❑ Topics covered

- Configuration management planning
- Change management
- Version and release management
- System building

❑ Tools:

- SVN, Visual Source Safe, Perforce, GIT

5.4 Configuration management - WHY

- ❑ New versions of software systems are created as they change
 - For different machines/OS
 - Offering different functionality
 - Tailored for particular user requirements

- ❑ Configuration management is concerned with managing evolving software systems
 - System change is a team activity
 - CM aims to control the costs and effort involved in making changes to a system

5.4 Configuration management - WHY

- ❑ New versions of software systems are created as they change
 - For different machines/OS
 - Offering different functionality
 - Tailored for particular user requirements

- ❑ Configuration management is concerned with managing evolving software systems
 - System change is a team activity
 - CM aims to control the costs and effort involved in making changes to a system

Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.5.1 Definition of risks

- ❑ Risk involves the possibility of an event in the future which has negative consequences.

- ❑ The level of risk is determined by
 - the likelihood of the event and
 - the impact (the harm) from that event

5.5.2 Project Risks

- ❑ Project risks are the risks that surround the project's capability to deliver its objectives, such as:
 - Logistics or product quality problem that block the tests
 - Excessive change to the product that invalidates test results
 - Insufficient or unrealistic test environments that yield misleading results

5.5.2 Project Risks

Other Project risks:

☐ Organizational factors:

- Skill and staff shortages;
- Personal and training issues;
- Political issues
- Improper attitude toward or expectations of testing

☐ Technical issues:

- Problems in defining the right requirements;
- The quality of the design, code and tests.

☐ Supplier issues

- failure of a third party;
- contractual issues.

5.5.3 Product Risks

- ❑ Product risks: Potential failure areas in the software or system such as:
 - Failure-prone software delivered.
 - The potential that the software/hardware could cause harm.
 - Poor software characteristics (e.g. functionality, reliability, usability and performance).
 - Poor data integrity and quality (e.g., data migration issues, data conversion problems, data transport problems, violation of data standards)
 - Software that does not perform its intended functions.
- ❑ To ensure that the chance of a product failure is minimized, risk management activities provide a disciplined approach to:
 - Assess (and reassess on a regular basis) what can go wrong (risks).
 - Determine what risks are important to deal with.
 - Implement actions to deal with those risks.
- ❑ In addition, testing may support the identification of new risks, may help to determine what risks should be reduced, and may lower uncertainty about risks.

5.5.4 Risk-based Testing and Product Quality

- ❑ A risk-based approach to testing provides proactive opportunities to reduce the levels of product risk.
- ❑ In a risk-based approach, the results of product risk analysis are used to:
 - Determine the test techniques to be employed
 - Determine the particular levels and types of testing to be performed (e.g., security testing, accessibility testing)
 - Determine the extent of testing to be carried out
 - Prioritize testing in an attempt to find the critical defects as early as possible
 - Determine whether any activities in addition to testing could be employed to reduce risk (e.g., providing training to inexperienced designers)

Risk Analysis – example

Krav nr.	URS requirements	Priority	Relevance	DISTURBANCE	IMPACT	PREVENTIVE ACTION	RISK LEVEL	CONCLUSION
2.8	Approved data must not be changed, when master data are updated	4	GxP	Already approved and published records will be altered	Wrongfull data in the system	Must be assured in system design	High	Q
2.9	It shall be possible to reuse data entered once	3	Business	Poor efficiency Risk for data error	Reduced competitive advantage Adverse publicity	Must be assured in system design Peer review	Low	C
3	Electronic signature, electronic record and audit trail requirements							
3.1	It shall be possible to use electronic signature to support the electronic workflow for non conformity handling between auditors and auditees	4	GxP	Non compliance to GxP	Not approvable by regulatory authorities	Must be assured in system design	High	Q
3.3	The use of electronic signature must be fully 21 CFR part 11 compliant	4	GxP	Failure to comply with regulatory requirements regarding electronics	Expensive delay/system down time while correcting non-compliance	Requirements must be defined in the URS that ensures that the system will comply with 21 CFR Part11, and System Specification and Qualification phase must ensure that these requirements	High	Q

Test Management

5.1 Test Organization (K2)

5.2 Test planning and estimation (K2)

5.3 Test progress monitoring and control (K2)

5.4 Configuration management (K2)

5.5 Risk and testing (K2)

5.6 Defect Management (K3)

5.6 Defect Management

❑ Test Log:

- Test log analysis needs to be done after test run
- Document defects if problem found caused by the test object, an defect report is initiated
- Cause-analysis (debugging) is developers' task

❑ Defect Reporting:

- Central database should be established for each project
- Personnel involved in development, as well as customers and users can report defects
- Reports can refer to problems in the tested programs, errors in specifications, user manuals, or other documents
- Document all information

5.6.1 Test Log

- ❑ The purpose of the test log is to provide a chronological record about relevant details observed during the test execution.

❑ Test Log Template

- | | |
|--|---|
| ➤ Test log identifier | ➤ Activity and event entries |
| ➤ Description <ul style="list-style-type: none">▪ items being tested▪ environment in which the testing is conducted | <ul style="list-style-type: none">▪ execution description▪ procedure results▪ environment info▪ anomalous events▪ incident report identifiers |

5.6.2 Defect Report

- ❑ The purpose of the test incident report is to document any event observed during testing that requires further investigation

❑ Test Defect Report Template

- | | |
|----------------------------|------------------------|
| ➤ Test incident identifier | ▪ Date and time |
| ➤ Summary | ▪ Procedure step |
| ➤ Incident description | ▪ Environment |
| ▪ Input | ▪ Attempts to repeat |
| ▪ Expected results | ▪ Testers and observed |
| ▪ Actual results | ➤ Impact |
| ▪ Anomalies | |

5.6.2 Defect Report

Table 6-1. Incident report template

	Attribute	Meaning
Identification	Id/Number	Unique identifier/number for each report
	Test object	Identifier or name of the test object
	Version	Identification of the exact version of the test object
	Platform	Identification of the HW/SW platform or the test environment where the problem occurs
	Reporting person	Identification of the reporting tester (possibly with test level)
	Responsible developer	Name of the developer or the team responsible for the test object
	Reporting date	Date and possibly time when the problem was observed

5.6.2 Defect Report

Table 6-1. Incident report template

	Attribute	Meaning
Classification	Status	The current state (and complete history) of processing for the report
	Severity	Classification of the severity of the problem
	Priority	Classification of the priority of correction
	Requirement	Pointer to the (customer-) requirements which are not fulfilled due to the problem
	Problem source	The project phase, where the defect was introduced (analysis, design, programming); useful for planning process improvement measures
Problem description	Test case	Description of the test case (name, number) or the steps necessary to reproduce the problem
	Problem description	Description of the problem or failure occurred; expected vs. actual observed results or behavior
	Comments	List of comments on the report from developers and other staff involved
	Defect correction	Description of the changes made to correct the defect
	References	Reference to other related reports

5.6.3 Defect Classification

Example:

❑ Severity Classes

- Critical, High, Medium, Low, Cosmetic
- 1,2,3,4,5

❑ Urgency Classes

- Critical (Business, Test)
- High (Business, Test)
- Medium (Business, Test)
- Low (Business, Test)

		Severity			
		Critical	High	Medium	Low
Urgency	Critical	1	1	2	3
	High	1	2	3	3
	Medium	2	3	3	4
	Low	3	3	4	4

❑ Priority Classes

- 1,2,3,4,5

CHAPTER 6

Tool Support for Testing

6. Tool Support for Testing

6.1 Types of test tools (K2)

6.2 Effective use of tools: potential benefits and risks (K2)

6.3 Introducing a tool into an organization (K1)

6.1.1 Test tool classification

- ❑ Purpose of tool support for testing:
 - the efficiency of testing activities by automating repetitive tasks.
 - improve the reliability of testing by automating unreliable manual tasks, for example, automating large data comparisons or simulating behavior.
 - Making it possible to execute testing tasks that are impossible manually. For example, performance and load testing
- ❑ An important area for tools is monitoring anything interesting to the tester, like memory use, network traffic, etc.
- ❑ CAST tools: tools exists for supporting or automating test activities (Computer Aided Software Testing)

6.1.1 Test tool classification

- ❑ Test tools are grouped by the testing activities or areas that supported by a set of tools.
- ❑ E.g.
 - Tools that support management activities,
 - Tools to support static testing.
 - Tools to support configuration management of testware
 - Tools to support defect management,
 - Tool to support requirements management and traceability
 - Tool to support coverage measurement and test design support.
- ❑ Probe effect: the effect on a component or system while it is being measured/analyzed

6.1.2.1 Test management tools

❑ Test management tools include:

- Support for the management of tests and the testing activities carried out.
- Interfaces to test execution tools, defect tracking tools and requirement management tools.
- Independent version control or interface with an external configuration management tool.
- Support for traceability of tests, test results and defects to source documents, such as requirement specifications.
- Logging of test results and generation of progress reports.
- Quantitative analysis (metrics) related to the tests (e.g. tests run and tests passed) and the test object (e.g. defects raised), in order to give information about the test object, and to control and improve the test process.

6.1.2.2 Requirement management tools

❑ Requirements management tools

- Storing requirement statements
- Storing information about requirement attributes
- Checking consistency of requirements
- Identifying undefined, missing or 'to be defined later' requirements;
- Prioritizing requirements for testing purposes;
- Traceability of requirements to tests and tests to requirements, functions or features;
- Traceability through levels of requirements;
- Interfacing to test management tools;
- Coverage of requirements by a set of tests (sometimes).

6.1.2.3 Defect management tools

- ❑ These tools enable
 - The progress of defects to be monitored over time
 - Provide support for statistical analysis
 - Provide reports about defects.

- ❑ They are also known as defect tracking tools

- ❑ Example
 - Jira
 - Bugzilla
 - Rally

6.1.2.4 Configuration management tools

❑ Configuration management tools

- Storing information about versions and builds of software and testware
- Traceability between software and testware and different versions or variants;
- Keeping track of which versions belong with which configurations (e.g. operating systems, libraries, browsers);
- Build and release management;
- Base lining (e.g. all the configuration items that make up a specific release);
- Access control (checking in and out).

❑ Tools: SVN, Visual Source Safe, Perforce, GIT

6.1.3 Tools support for static testing

Review process tools

- ❑ Review process support tools may
 - store information about review processes,
 - store and communicate review comments,
 - report on defects and effort,
 - manage references to review rules and/or checklists
 - keep track of traceability between documents and source code.
 - provide aid for online reviews, which is useful if the team is geographically dispersed.
 - Collecting metrics and reporting on key factors

6.1.4 Tools support for test specification

Test design tools

- ❑ Test design tools generate
 - test inputs or
 - expected result (may use a test oracle)
 - from
 - requirements
 - a graphical user interface
 - design models (state, data or object)
 - code.
- ❑ The generated tests from a state or object model are useful for verifying the implementation of the model in the software, but are seldom sufficient for Verifying all aspects of the software or system.
- ❑ They can save valuable time and provide increased thoroughness of testing because of the completeness of the tests that the tool can generate.

6.1.5 Tools support for test specification

Test data preparation tools

- ❑ Test data preparation tools manipulate databases, files or data transmissions to set up test data to be used during the execution of tests.
 - Extract selected data records
 - Enable records to be sorted or arranged
 - Generate new records populated with pseudo-random data
 - Construct a large number of similar records

6.1.6 Tools support for execution and logging

Test execution tools

- ❑ Test execution tools enable tests to be executed automatically, or semi-automatically, using stored inputs and expected outcomes, through the use of a scripting language.
- ❑ Generally these tools include
 - Capture/playback
 - Data driven
 - Keyword driven
 - dynamic comparison features
 - provide a test log for each test run.
- ❑ The scripting language makes it possible to manipulate the tests with limited effort, for example, to repeat the test with different data or to test a different part of the system with similar steps

6.1.6 Tools support for execution and logging

Test harness/unit test framework tools

- ❑ A test harness may facilitate the testing of components or part of a system by simulating the environment in which that test object will run.
- ❑ A framework may be created where part of the code, object, method or function, unit or component can be executed, by calling the object to be tested and/or giving feedback to that object.
- ❑ It can do this by providing artificial means of supplying input to the test object, and/or by supplying stubs to take output from the object, in place of the real output targets.
- ❑ Test harness tools can also be used to provide an execution framework in middleware, where languages, operating systems or hardware must be tested together.
- ❑ They may be called unit test framework tools when they have a particular focus on the component test level. This type of tool aids in executing the component tests in parallel with building the code.

6.1.6 Tools support for execution and logging

Test comparators

- ❑ Test comparators determine differences between files, databases or test results.
- ❑ Test execution tools typically include dynamic comparators, but post-execution comparison may be done by a separate comparison tool.
- ❑ A test comparator may use a test oracle, especially if it is automated

6.1.6 Tools support for execution and logging

Coverage measurement tools

- ❑ Coverage measurement tools can be either intrusive or non-intrusive depending on the measurement techniques used, what is measured and the coding language.

- ❑ Code coverage tools measure the percentage of specific types of code structure that have been exercised
 - statements
 - branches
 - decisions
 - module or function calls

- ❑ These tools show how thoroughly the measured type of structure has been exercised by a set of tests

6.1.6 Tools support for execution and logging

Security tools

- ❑ Security tools check for
 - computer viruses
 - denial of service attacks.

- ❑ Example
 - a Firewall is not strictly a testing tool, but may be used in security testing.

- ❑ Other security tools stress the system by searching specific vulnerabilities of the system.

6.1.6 Tools support for execution and logging

Dynamic analysis

- ☐ Acquire additional information on the internal state of the software (e.g., information on allocation, usage, and release of memory)
- ☐ Memory leaks, pointer allocation, or pointer arithmetic problems can be detected.

6.1.7 Tools support for performance and monitoring

Performance Tools

- ❑ Performance-testing, load-testing and stress-testing tools
 - Generating a load on the system to be tested;
 - Measuring the timing of specific transactions as the load on the system varies;
 - Measuring average response times;
 - Producing graphs or charts of responses over time.

6.1.7 Tools support for performance and monitoring

Monitoring Tools

- ❑ Identifying problems and sending an alert message to the administrator (e.g. network administrator);
- ❑ Logging real-time and historical information;
- ❑ Finding optimal settings;
- ❑ Monitoring the number of users on a network;
- ❑ Monitoring network traffic (either in real time or covering a given length of time of operation with the analysis performed afterwards)

6.1.8 Why Test Automation?

☐ Why?

- It worked fine on Previous Project
- Save MONEY
- Raise Quality

☐ ROI (Return on Invest)

- Manual testing alone (e.g.: 158%)
- Manual + Automation Tool Testing (e.g.: 208%)

☐ Save Time

☐ Manual Testing (e.g.: 50 M/D save)

☐ Manual + Automation Tool (e.g.: 85M/D save)

- Save Resource
- Raise Quality
- More Coverage
- >...

Tool Support for Testing

6.1 Types of test tools (K2)

6.2 Effective use of tools: potential benefits and risks (K2)

6.3 Introducing a tool into an organization (K1)

6.2.1 Potential benefits of using tools

- ❑ Potential benefits of using tools include:
 - Repetitive work is reduced.
 - Greater consistency and repeatability.
 - Objective assessment (e.g. static measures, coverage).
 - Ease of access to information about tests or testing (e.g. statistics and graphs about test progress, defect rates and performance).

6.2.2 Risks of using tools

❑ Risks of using tools include:

- Unrealistic expectations for the tool (including functionality and ease of use).
- Underestimating the time, cost and effort for the initial introduction of a tool (including training and external expertise).
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool (including the need for changes in the testing process and continuous improvement of the way the tool is used).
- Underestimating the effort required to maintain the test assets generated by the tool.
- Over-reliance on the tool (replacement for test design or where manual testing would be better).

Tool Support for Testing

6.1 Types of test tools (K2)

6.2 Effective use of tools: potential benefits and risks (K2)

6.3 Introducing a tool into an organization (K1)

6.3.1 Tool selection

- ❑ Selection process consists of the following 5 steps:
 1. Requirement specification for the tool application
 2. Market research (creating an overview of possible candidates)
 3. Tool demonstrative and creation of a short list
 4. Evaluating the tools on the short list
 5. Reviewing of the results and selection of the tool

6.3.1 Tool selection

- ❑ The following factors are important during tool selection:
 - Assessment of the organization's maturity (e.g. readiness for change);
 - Identification of the areas within the organization where tool support will help to improve testing processes;
 - Evaluation of tools against clear requirements and objective criteria;
 - Proof-of-concept to see whether the product works as desired and meets the requirements and objectives defined for it;
 - Evaluation of the vendor (training, support and other commercial aspects) or open-source network of support;
 - Identifying and planning internal implementation (including coaching and mentoring for those new to the use of the tool).

6.3.2 Introducing a tool into an organization

- ❑ The main principles of introducing a tool into an organization include:
 - Assessment of organizational maturity, strengths and weaknesses and identification of opportunities for an improved test process supported by tools.
 - Evaluation against clear requirements and objective criteria.
 - A proof-of-concept to test the required functionality and determine whether the product meets its objectives.
 - Evaluation of the vendor (including training, support and commercial aspects).
 - Identification of internal requirements for coaching and mentoring in the use of the tool.

6.3.2 Introducing a tool into an organization

- ❑ Six steps to introduce a tool to an organization:
 1. Execute a pilot project
 2. Evaluate the pilot project experiences
 3. Adapt the processes and implementation rules for usage
 4. Train the users
 5. Introduce the tool in a stepwise fashion
 6. Offer accompany coaching

6.3.2 Introducing a tool into an organization

- ❑ Success factors for the deployment of the tool within an organization:
 - Rolling out the tool to the rest of the organization incrementally.
 - Adapting and improving processes to fit with the use of the tool.
 - Providing training and coaching/mentoring for new users.
 - Defining usage guidelines.
 - Implementing a way to learn lessons from tool use.
 - Monitoring tool use and benefits.

- ❑ Objectives of starting a pilot project
 - Learn more details about the tool.
 - Evaluate how the tool fits with existing processes and practices, and determine what would need to change.
 - Decide on standard ways of using, managing, storing and maintaining the tool and the test assets.
 - Assess whether the benefits will be achieved at reasonable cost.

CHAPTER 7

Reading Technical Document & Creating Test Requirement

7. Tool Support for Testing

7.1 Some Common Reading Techniques

7.2 Reading Technical Documents – Do's and Don'ts Tips

7.3 Creating Test Requirement

7.1 Some Common Reading Techniques

- ❑ **SQ3R** – Increasing your retention of written information
- ❑ **Speed Reading** – Radically increasing your reading speed
- ❑ **Reading Strategies**

→ Find necessary information quickly and easily

7.1.1 SQ3R

- ❑ S: Survey
- ❑ Q: Question
- ❑ R: Read
- ❑ R: Recall
- ❑ R: Review

SQ3R Objectives:

- Extract the maximum amount of benefit from reading time
- Organize the structure of a subject in mind
- Set study goals
- Separate important information from irrelevant data

→ Using SQ3R to actively read a document, you can get the maximum benefit from your reading time

How to Use this Tool

❑ Survey:

- Scan the contents to pick up a shallow overview of the text
- Discard helpless information

❑ Question:

- Make note of any questions on the subject that come to mind
 - Scan the document again to see if any stands out
- These questions are studying goals - understanding the answers to structure the information

❑ Read:

- Read through useful sections in detail, make sure to understand all the points that are relevant
- Read slowly dense and complicated information
- Take notes when reading

How to Use this Tool

□ Recall:

- Run through useful sessions several times
- Isolate the core facts or the essential processes behind the subject
- See how other information fits around these core facts

□ Review:

- Rereading the document by:
 - Expanding notes
 - Discussing the material with colleagues
 - Teaching it to someone else!

7.1.2 Speed Reading

☐ A skilled reader:

- Reads many words in each block, dwell on each block for an instant, and then move on
- Reduces the amount of work that the reader's eyes have to do by skipping back to a previous block of words

☐ A poor reader:

- Spends a lot of time reading small blocks of words
- Skips back often
- Loses the flow and structure of the text
- Confuses their overall understanding of the subject
- Frequent eye movement makes reading tiring

7.1.2 Speed Reading

❑ Speed Reading's Purpose:

- Read and understand written information much more quickly
- Master large volumes of information quickly
- Reduce suffering from "information overload"

❑ The Key Insight:

- The most important trick: know what information you want from a document before reading it
 - Want an outline of the issue? → skim the document quickly and extract only the essential facts
 - Need to understand the real detail? → read it slowly enough to gain the full understanding

How to Use this Tool

- ❑ Increasing the number of words in each block
 - Expand the number of words you read at a time
 - Tip:** Holding the text a little further from your eyes 😊

- ❑ Reducing fixation time
 - The minimum length of time needed to read each block is a quarter of a second

- ❑ Reducing skip-back
 - Run a pointer along the line as you read
 - Your eyes will follow the tip of your pointer, smoothing the flow of your reading
 - The speed will largely depend on the speed the pointer moves

7.1.3 Reading Strategies

6 different strategies and techniques to read more effectively:

1. Knowing what you need to know, and reading appropriately
2. Knowing how deeply to read the document: skimming, scanning or studying
3. Using active reading techniques to pick out key points and keep your mind focused on the material
4. Understanding how to extract information from different article types
5. Creating your own table of contents for reviewing material
6. Using indexes, tables of contents, and glossaries to help you assimilate technical information

Tool Support for Testing

7.1 Some Common Reading Techniques

7.2 Reading Technical Documents – Do's and Don'ts Tips

7.3 Creating Test Requirement

7.2 Reading Technical Documents

– Do's Tips

- ☐ Make an overview of documents (read the headlines and index)
- ☐ Focus on the structure of the documents
- ☐ Find out technical terms
- ☐ Focus on diagram to acknowledge the ways application goes (combine with doing on application)
- ☐ Focus on activities (functionalities) and UI mockups
- ☐ Verify the same level of flow/ transaction
- ☐ Read the Help file
- ☐ Read the documents many times

7.2 Reading Technical Documents

– Don'ts Tips

- ☐ Read all the documents
- ☐ Ignore some supplement documents
- ☐ Forget the Help file
- ☐ Not pay attention when reading

Tool Support for Testing

7.1 Some Common Reading Techniques

7.2 Reading Technical Documents – Do's and Don'ts Tips

7.3 Creating Test Requirement

7.3 Creating Test Requirement

- ☐ What is User Stories?
- ☐ What is Test Requirement (TR)?
- ☐ How to write TR basing on user story
- ☐ How to write TR basing on other information
- ☐ Structure of TR
- ☐ Things to pay attention
- ☐ Things to avoid

7.3.1 What is User Stories?

- ❑ User Story is a story about how the system is supposed to solve a problem or support a business process
- ❑ A user story is a very high-level definition of a requirement, containing just enough information collected from client so that the developers can implement it.
- ❑ User stories are simple enough that people can understand easily
- ❑ Each User Story is written on a User Card (by collecting from client directly), is stored in a website or in a full document
- ❑ A User Story can be an only statement or contains more than one step

7.3.2 What is Test Requirement (TR)?

- ❑ A statement of what should be tested in the AUT
- ⇒ Functional Requirement: the requirement for the functions that the application should do
- ⇒ Non-functional requirement: the requirement for the properties that the functions should have or should look like. There are 3 types of non-functional TR:
 - ❑ Look-n-feel
 - ❑ Boundary
 - ❑ Negative

7.3.3 How to write TR Basing on User Stories

- ☐ User stories examples
- ☐ Classify user stories in specified function
- ☐ How to derive TR based on user stories

7.3.3.1 User stories examples

User story 1:

1. *User can open webpage and log in with valid account as child*

User story 2:

1. *User opens webpage, logs in as child. From landing page, clicks on Downloads tab or link*
2. *Downloads home opens.*

User story 3:

1. *User opens webpage, logs in as child. From landing page, clicks on Downloads tab or link*
2. *Downloads home opens. User clicks on a title's Download Audio button*
3. *Downloading message appears in a pop-up. User clicks OK to download or close to cancel downloading*

User story 4:

1. *User opens webpage, logs in as child. From landing page, clicks on Downloads tab or link*
2. *Downloads home opens. User clicks on a title's Download Audio button*
3. *Downloading status message appears in a pop-up. User clicks OK to download*
4. *If errors in downloading occur – e.g. internet connection drops - failure message will pop up telling user to try again by clicking the Download Audio button once more*

7.3.3.2 Classify user stories in specified function

- ❑ Client usually sends us a classified user stories set. In case they don't:
 - Go through all user stories for the overview of the AUT and understand its functions/structure
 - Refer to other documents such as Help file to “see” functions visually
 - Write down core-functionalities of AUT
 - Group user stories which have the same context and/or same function into corresponding functionality
 - Consider to split big functionality group into smaller group

7.3.3.2 Classify user stories in specified function

Example:

❑ *The first way*

- Group Login: UserStory 1
- Group Download: UserStory 2, UserStory 3, UserStory 4

❑ The second way

- Group Login: UserStory 1
- Group Basic Download: UserStory 2, UserStory 3
- Group Download Errors: UserStory 4

7.3.3.3 How to derive TR based on user stories

- ❑ User story usually contains duplicated steps -> Find out the main purpose of the user story

Example:

- ❑ User story 1: User can login with valid account (not can open webpage)
- ❑ User story 2: User can click on Downloads tab or link to open Downloads Homepage (not can login)
- ❑ User story 3: Downloading message appears . User clicks OK to download or close to cancel downloading
- ❑ User story 3: Failure message pops up telling user to try again by clicking the Download Audio button again

7.3.3.3 How to derive TR based on user stories

- ❑ Consider how many main TRs included in the user story (main purpose)
- ❑ Each test requirement should contain only one verification point to check

Example:

- ❑ User story 1: 1 test requirement
- ❑ User story 2: 2 test requirements
- ❑ User story 3: 3 test requirements
- ❑ User story 4: 1 test requirement (in case there is another story requires user click on Download Audio button to resume downloading)

7.3.3.3 How to derive TR based on user stories

- ❑ Re-write the sentences of user story to create clear and standard TRs

Example:

- ❑ User story 1:
 - Verify that user can log in with valid account
- ❑ User story 2:
 - Verify that Downloads home can be opened by clicking Download tab
 - Verify that Downloads home can be opened by clicking Download link

7.3.3.3 How to derive TR based on user stories

Example (cont):

❑ User story 3:

- Verify that Downloading message appears in a pop-up
- Verify that user can click OK to download
- Verify that user can click Close to cancel downloading

❑ User story 4:

- Verify that failure message will pop up when error appears in downloading

7.3.4 How to Write TR Basing on Other Information

- ☐ Based on the Supplemental Requirement
- ☐ Based on the Enhancement Request
- ☐ Based on the Stakeholder Request
- ☐ Based on the Defect
- ☐ Based on the Issues

7.3.4.1 Based on the Supplemental Requirement

- ❑ User stories document can contain supplemental requirements to describe the expected behavior of story more detailed

Requirement Number: SUPP12.11.10

Status: Approved

Priority: 2 - High

Target Release: WFM 1.1

Name: Project display order in workspace

Text: All projects associated with the workspace are displayed in alphabetical order.

- ❑ The properties of the Supplemental requirement should be considered to re-write TR
- ❑ The content should be rewritten as a comprehensive and relevant test requirement
- ❑ Complex supplemental requirements should be divided into more than one TR

7.3.4.2 Based on Enhancement Request

- ❑ User stories document can contain enhancement request
- ❑ Enhancement request is future features which are recommended to improve the application

Enhancement Request Id: APEDB00004828

State: Assigned

Priority: 2 - Medium

Target Release: WFM 1.1

Headline: Project Explorer - Project Display Order is apparently random

Description: The order in which 'My Projects' are displayed under the Project Explorer is neither alphabetical nor chronological (creation or accessed). Ideally it should be possible to control these, but failing that we need consistency on where to find a project - my preference for a default is alphabetical.

- ❑ The content of the enhancement request should be considered to re-write TR
- ❑ The content should be rewritten as a comprehensive and relevant test requirement
- ❑ Complex enhancement request should be divided into more than one TR

7.3.4.3 Based on Stakeholder Request

- ❑ User stories document can contain stakeholder request
- ❑ Stakeholder request is added by client or investor of the application

Requirement Number: STRQ107.53

Status: Approved

Priority: 1 - Imperative

Target Release: WFM 2.0

Planned Iteration:

Name: On/Off option toggling automatic refresh of project information

Text: Continual checks for updates may slow the performance of the system. This implies the need to turn off the automatic checking and perform the 'refresh' of the project information manually.

- ❑ The content of the stakeholder request should be considered to re-write TR
- ❑ The content should be rewritten as a comprehensive and relevant test requirement
- ❑ Complex stakeholder request should be divided into more than one TR

7.3.4.4 Based on the Defect

- ❑ User stories document can include known defects in detail

Defect Id: APEDB00003889

Headline: User is given impression additional objects exist in the Project Explorer tree

Description: Steps to Reproduce:

1) Add a project to the Project Explorer (either by creating a new project or by loading an existing project)

2) Expand the project in the Project Explorer

Expected and Actual Results:

Expected: Only the "Workflows" folder should appear with a "+" in front of it since it contains detailed workflow categories.

Actual: The "Data" folder appears with a "+" in front of it even though there are no data files loaded.

- ❑ The defect should be considered to re-write TR if there is no created TR which is related to this defect before
- ❑ The expected result should be rewritten as a comprehensive and relevant test requirement

7.3.4.5 Based on the Issue

- ❑ User stories document can include known issues in detail
- ❑ An issue is not a defect. It might be an un-logical point, not friendly interfaces or latencies of the AUT

Issue Id: APEDB00000808

State: Assigned

Priority: 3- Low

Headline: UI - Need to review status bar, console window, and project explorer view tab for Iteration 2 user interface.

Description: UI - Need to review status bar, console window, and project explorer view tab for Iteration 2 user interface.

- ❑ The issue should be considered to re-write TR. However, this is not an official TR approved by client
- ❑ The content should be rewritten as a comprehensive and relevant test requirement

7.3.4.6 Structure of Test Requirement

- ❑ TR ID: Follow the defined rule to name the ID
- ❑ TR description: Content of TR
- ❑ Type: Functionality, Boundary, Negative and Look n Feel
- ❑ Traceability: The user story that TR belongs to. The ID of the specified Supplemental requirement, Enhancement request, Defects... which TR is based on
- ❑ Notes: Necessary notes to explain more about the TR

7.3.4.7 Things to Pay Attention to

- ❑ Use terminology of application and document. (Refer application terminologies in the *Term Glossary* section of UC document)
- ❑ Use standard words of testing skill: should be, check if, verify that, validate that, acceptable, allowed...
- ❑ Refer Help section of application for more information and explanation
- ❑ Come up with questions for unclear issues
- ❑ Spelling check while writing TR
- ❑ Read UI mockup documents (This document is very useful for describing the features of application visually)
- ❑ Keep the consistence of TRs: they should not contradict each other

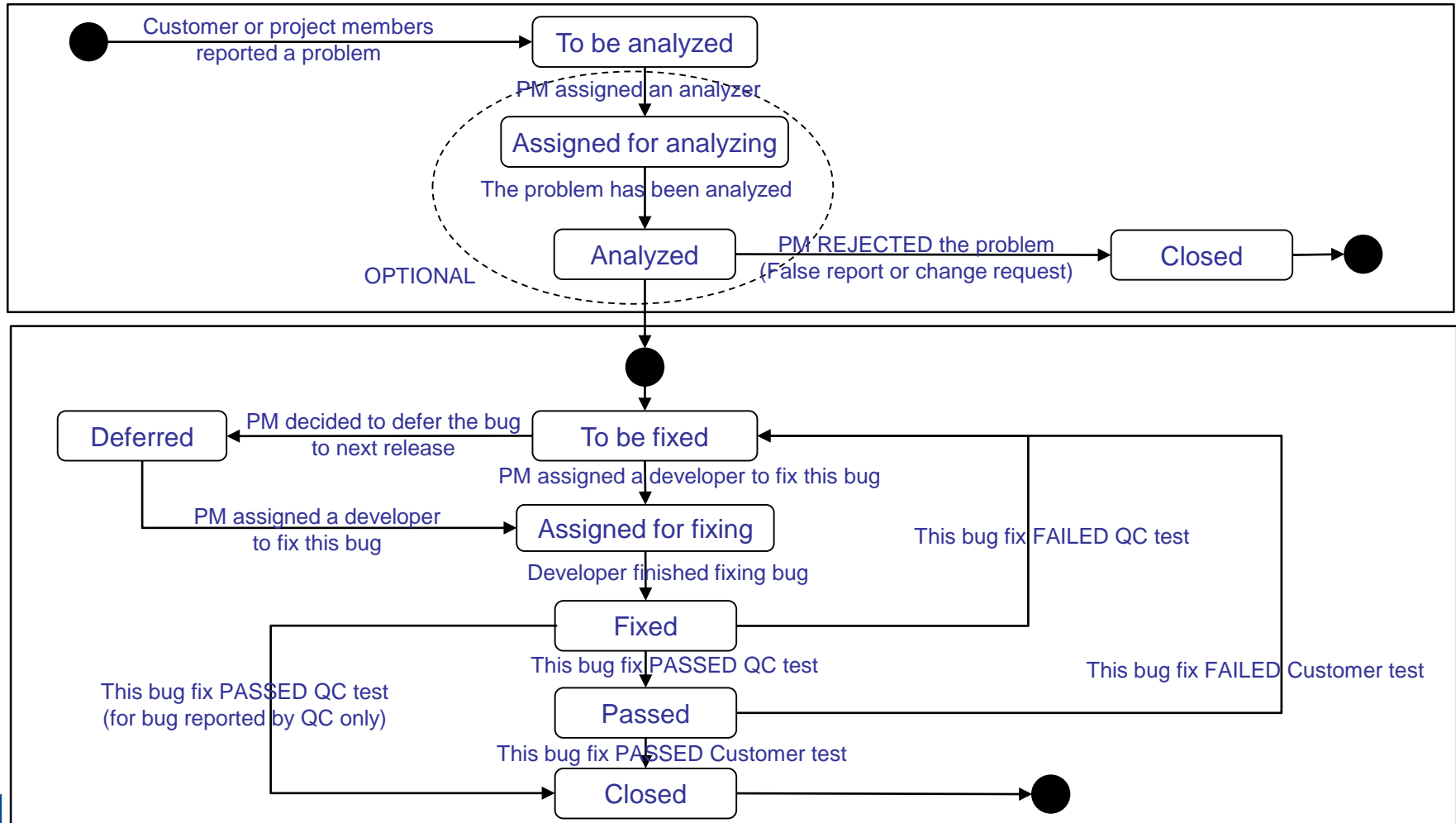
7.3.4.8 Things to Avoid

- ❑ Copy all the user story, supplemental requirement, business rule, enhancement request... and paste as a requirement.
- ❑ Write too long and too obvious TR
- ❑ Base too much on the current version of application when analyzing TR for the next version.
- ❑ Include steps into TR that makes the TR complicated.
- ❑ Assume the expected result instead of basing on the documents.
- ❑ Use speaking language to write TR.
- ❑ Duplicated TRs.
- ❑ Consider when writing TR that will not be able to implement due to TA limitation.

CHAPTER 8

Issue Report

Activity flow (To be tailored to cover Issue's Life Cycle)



8. Common activities of issue report

8.1 Open new issue

8.2 Reopen an existing issue

8.3 Clarification/ Comment on existing issue

8.4 Verifying issue

8.1 Open new issue

Identification:

- Issue ID
- Issue type (*)
- Summary (*)
- Test Object
- Components (*)
- Affected version
- Environment
- Reporter
- Report date
- Assignee

(*) Important field

Classification:

- Status
- Priority (*)
- Severity (*)

Problem Description

- Description (***)
- Attachment (*)

Others:

- Due date
- Resolution
- Fix version
- ...

8.1.1 Open new issue – Issue Type

☐ Bug

- A flaw in a component or system that can cause the component or system to fail to perform its required function.

☐ Enhancement

- An improvement or enhancement to an existing feature.

☐ Documentation

- Bug in the documentation (Requirement, Design...)

8.1.2 Open new issue – Summary

❑ Basic syntax:

- Bug + Action+ [Condition]

❑ Hints

- Concise (should be one-line summary)
- Clear (avoid generic description)
- Grammar should be correct
- Words must be correct as used in documentation
- Should report one bug per report
- Report exactly the bug, not a symptom of the bug
- Should have Prefix/Suffix

8.1.2 Open new issue – Summary

❑ Hints/ Best practices: (cont)

- Report the issue, not a symptom of the issue

Scenario: We are testing an Phone input field. We find that the Phone field allows “1234567890123456” and it also allows “!@#\$%^&*()_+” ==> These are two different symptoms of the same bug. Closer analyst would detected that the real issue is the Phone field isn't being validated at all. The problem may be more serious than the first symptom you find. So focus and report the real bug instead.

❑ Specific regulations in projects:

- Prefix. E.g: Specific environment, feature name etc...
- Suffix

8.1.3 Open new issue – Priority

- ❑ The level of (business) importance assigned to an issue.
 - P1: Stopper (Must be fixed in order to continue testing): No further processing is possible.
 - P2: High (Must be fixed in order to launch): A major, serious and high impact functionality of the application is not working.
 - P3: Medium (Need to be fixed but product may launch with exception): A small and low impact function of the application is not working and user is able to continue using the product.
 - P4: Low (May be fixed, but not critical for launch): Very low priority, a rare or random defect that occurs at a very low re-productivity rate.

8.1.4 Open new issue – Severity

- ❑ The degree of impact that an defect has on the development or operation of a component or system
 - **Critical:** Loss of functionality or data, crashes, high impact performance issues
 - **Major:** Non-critical defects in the core functionality, but no workaround
 - **Minor:** Minor functional defects
 - **Trivial:** Minor cosmetic issues like misspelling or misalignment of texts

8.1.5 Open new issue – Description

❏ Sections:

- [Brief]
- [Preconditions]
- Steps to reproduce
- Actual result
- Expected result
- [Notes]
- Workaround: Workaround solution to come over the issue

8.1.5 Open new issue – Description (cont.)

- ❑ [Brief]: Brief about the issue if you cannot express fully in the summary
- ❑ [Preconditions]: pre-steps/environment settings to get ready to perform the first step.
- ❑ **Steps to reproduce (*)**: How to reproduce the bug
 - Steps by number. Combine simple steps by ">"
 - Keep steps short, easy to read and direct path to the issue
 - Include input data in the step (especially for the complex case)
 - Emphasize/highlight on the important part/words

8.1.5 Open new issue – Description (cont.)

- ❑ Actual result: Describe what did happen, not what didn't happen.
 - Always include the error message or error code if available.
 - Question: Should symptom/ impact if doing further step be put here!
- ❑ Expected result: Explain what should happen, not what shouldn't happen
 - Use as many same words/sentence as possible on both Expected & Actual results
 - State clearly why the actual result is not expected.
 - Mention clearly if it is not the same as stated in a specific requirement or if the behavior is not consistent on all users/platform...
 - Put directly related spec into the issue so as to provide more information for the expected result. Attachments are required if they help to clarify the result.
- ❑ [Notes]: An optional section, but it is helpful for Developer to investigate an issue. It also represent maturity, seniority of the reporter
 - Ability/ Frequency to reproduce an intermittent issue
 - Specific situation that the actual result is different

8.1.6 Open new issue – Attachment

- ☐ Allowed/ recommended format
- ☐ Screen shots
 - Highlight issue area(s) in the bug.
 - Use images to illustrate static issues (try to combine them into one)
 - Use JPG format instead of PNG
- ☐ Video
 - Actions in the video should match the steps listed in the bug report and at the time the issue was created.
 - Use popular video format: Windows Media Video (.wmv), etc..
- ☐ Data input:
 - simple scenario: put in each step to reproduce
 - Complex scenario: attach data input file/ screenshot with data inputted
- ☐ Log file:
 - Highlight events that you want the reader to pay attention
 - Just focus log file (not a heavy file that have tracked log for several days with no meaning)

8.1.7 Open new issue – Important Notes

- ❑ Must follow requirement, standards from client or committed with client
- ❑ Pitfalls:
 - Reporter's assumption, mind set => provide lack of info
 - Wrong foresee readers
 - Is not on end-user's shoes => wrong definition of priority, severity
 - Too rely on attachment
 - Afraid of change
 - Changing QA lead...
- ❑ Avoid duplication
 - Search in bug tracking system
 - Follow common standards
 - Double check with teammates
 - Follow definition from client for same or different bug situations

8.2 Reopen existing issue

- ❑ Should we reopen the closed issue?
- ❑ Consider
 - Tested version
 - Test scenario
 - Environment
- ❑ How to
 - Add comment
 - Re-open
 - Add other info: affected version, environment...

8.3 Clarifying/ Adding comment on issue

- ❑ When do you need to clarify/add comment on an existing issue?
 - Provide more information
 - Encounter this issue on another environment
 - ...

- ❑ Action:
 - Comment on issue
 - Reopen issue
 - Add effect version
 - Add more info...

8.4 Verifying issue

☐ When

- The issue has been fixed and released for testing.
- New version has been launched, the opening issue was found in previous versions.

☐ Action:

- Recheck the main scenario on affected environments and non-affected environment.
- Analyze the impact => do regression test.
- Add comment what you have done, what result you get.

Certified Tester in Foundation Level



Thank You.