



ISTQB Foundation level Basics

21-June-2016

Girish Goutam

Employee Id 681117

gk.goutam@tcs.com

Athene Annuity and Life Insurance

Confidentiality Statement

Include the confidentiality statement within the box provided. This has to be legally approved

Confidentiality and Non-Disclosure Notice

The information contained in this document is confidential and proprietary to TATA Consultancy Services. This information may not be disclosed, duplicated or used for any other purposes. The information contained in this document may not be released in whole or in part outside TCS for any purpose without the express written permission of TATA Consultancy Services.

Tata Code of Conduct

We, in our dealings, are self-regulated by a Code of Conduct as enshrined in the Tata Code of Conduct. We request your support in helping us adhere to the Code in letter and spirit. We request that any violation or potential violation of the Code by any person be promptly brought to the notice of the Local Ethics Counsellor or the Principal Ethics Counsellor or the CEO of TCS. All communication received in this regard will be treated and kept as confidential.

Table of contents

Page #	Title
3	Fundamentals of testing (K1,K2)
20	Software development models (K2)
46	Static techniques (K2)
66	Test design techniques (K3)
98	Test organization (K3, K4)
121	Tool support for testing (K4)

CHAPTER 1

Fundamentals of testing

Agenda

- Why is testing necessary (K2)
- What is testing (K2)
- General testing principles (K2)
- Fundamental test process (K1)
- The psychology of testing (K2)

Why is Testing necessary ?

- Software is not defect free
- Defects cause failures
- Unreliable software can cause failures
- Failures have associated costs like loss of business
- Testing helps to find defects and learn about reliability of software

Defect

- A flaw in a component or system that can cause the component or system to fail to perform its required functions
- Can be defined in simpler words as:
 - Variance/ difference between the expected and actual result is called as “Defect/Bug”.

Causes of software defects

- Miscommunication
- Software complexity
- Programming errors
- Changing requirements
- Time pressures
- Egos
- Poorly documented code
- Software development tools
- Environmental conditions

What is Testing

Software Testing is :

- Process of finding out gaps between customer's requirements and system to be delivered.
- Process of finding defects (variance between expected result and actual result) is also called as Software testing.

Quality

Quality is meeting customer's **needs** and **expectations**.

“Testing improves the quality” How ?

Finding defects and measuring quality in terms of defects

Building confidence

Preventing defects

Reducing risk

Quality is further classified into:

Quality assurance

Quality Control

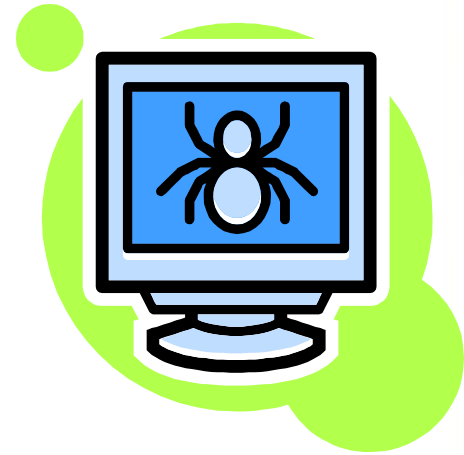
General testing principles

- Testing shows presence of defects
- Exhaustive testing is impossible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is context dependent
- Absence-of-errors fallacy

Principle 1

Testing shows presence of defects

- Testing can show that defects are present, but cannot prove that there are no defects
- Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness



Exhaustive testing is impossible

- Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, we use risk and priorities to focus testing efforts



Principle 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



Principle 4

Defect clustering

- A small number of modules contain most of the defects discovered during pre-release testing, or show the most operational failures



Principle 5

Pesticide paradox

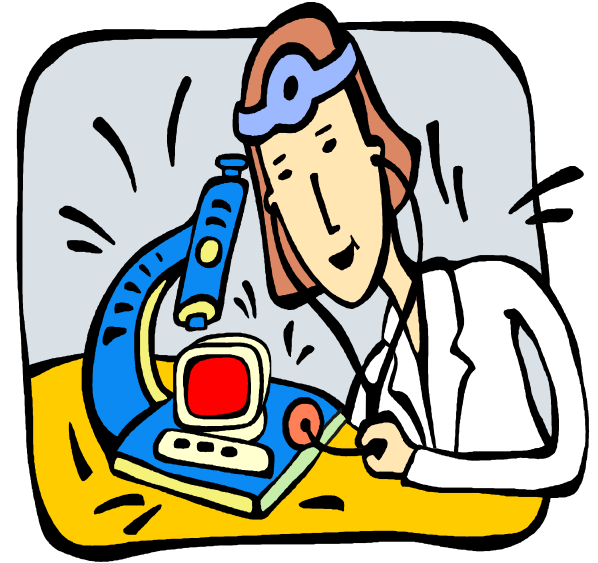
- If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs
- To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects



Principle 6

Testing is context dependent

- Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site



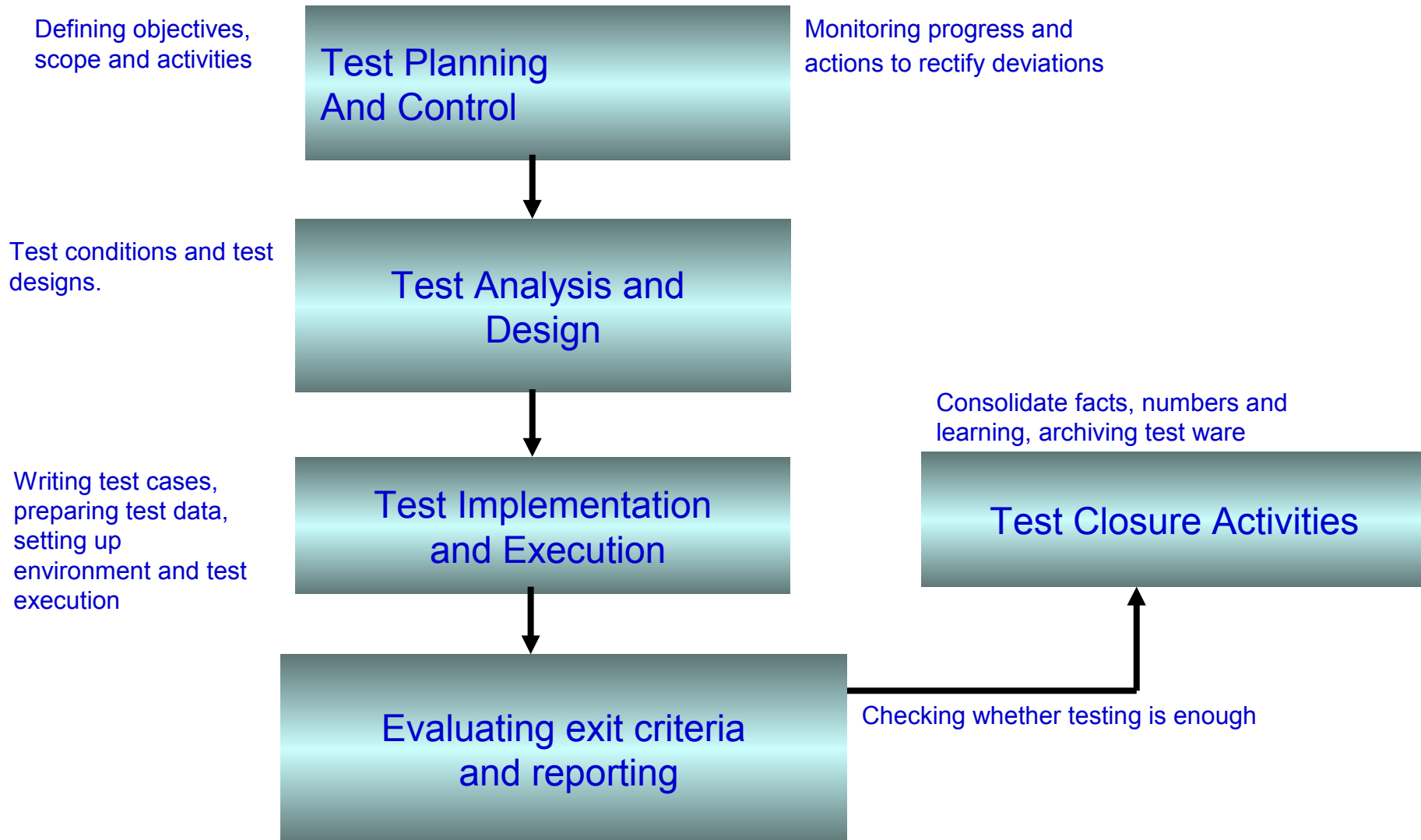
Principle 7

Absence-of-errors fallacy

- Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations



Fundamental test process/STLC



Psychology of testing

Why test ?

- Build confidence in software under test
- To find defects
- Prove that the software conforms to user requirements/ functional specifications
- Reduce failure cost

Can testing prove software is correct?

- Not possible to prove system has no defects
- Only possible to prove system has defects

CHAPTER 2

Software development models

Agenda

- Software development models (K2)
- Test levels (K2)
- Maintenance testing (K2)

Software Development models

- Waterfall model
- Iterative development model
- Prototyping
- Rapid application development (RAD)
- Agile Development methodology

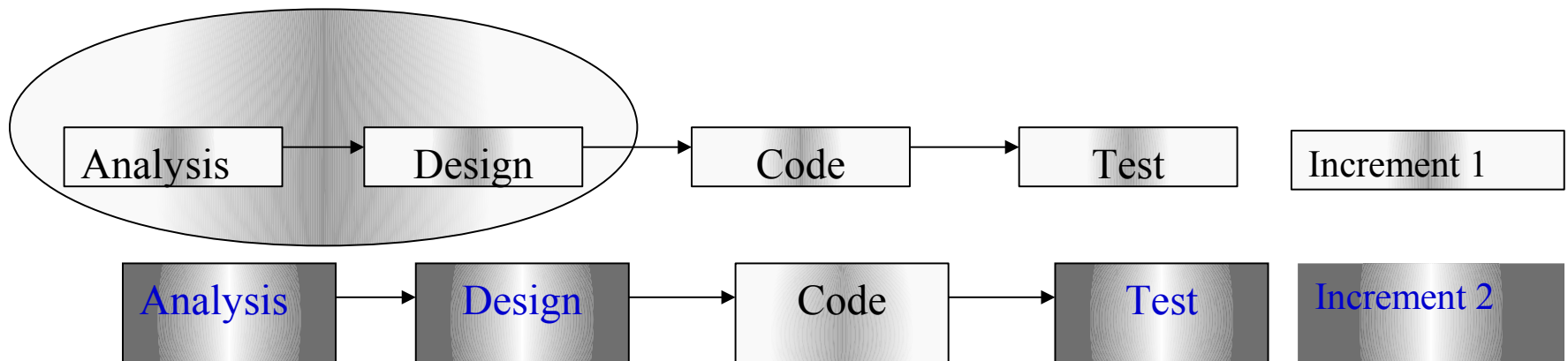
Waterfall model

- Development broken into series of sequential stages
- One of the earliest models developed for small projects
- Also called, linear sequential model
- Testing also follows the same sequence



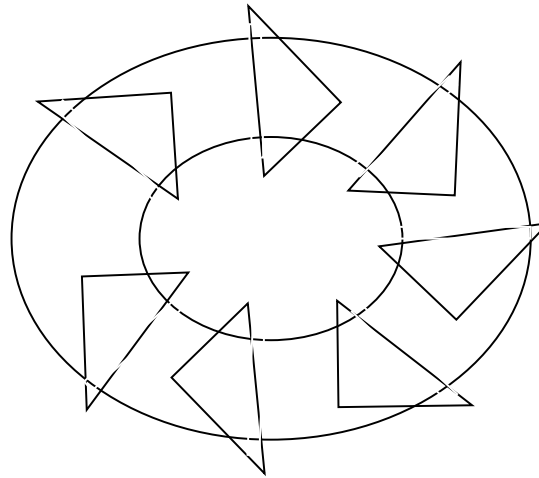
Iterative development model

- Iterative development is the process of establishing requirements, designing, building and testing a system, done as a series of smaller developments
- More realistic approach to development of large scale systems and software
- Each increment is fully tested before beginning the next
- Total planned testing effort would be more than that for waterfall model



Prototyping

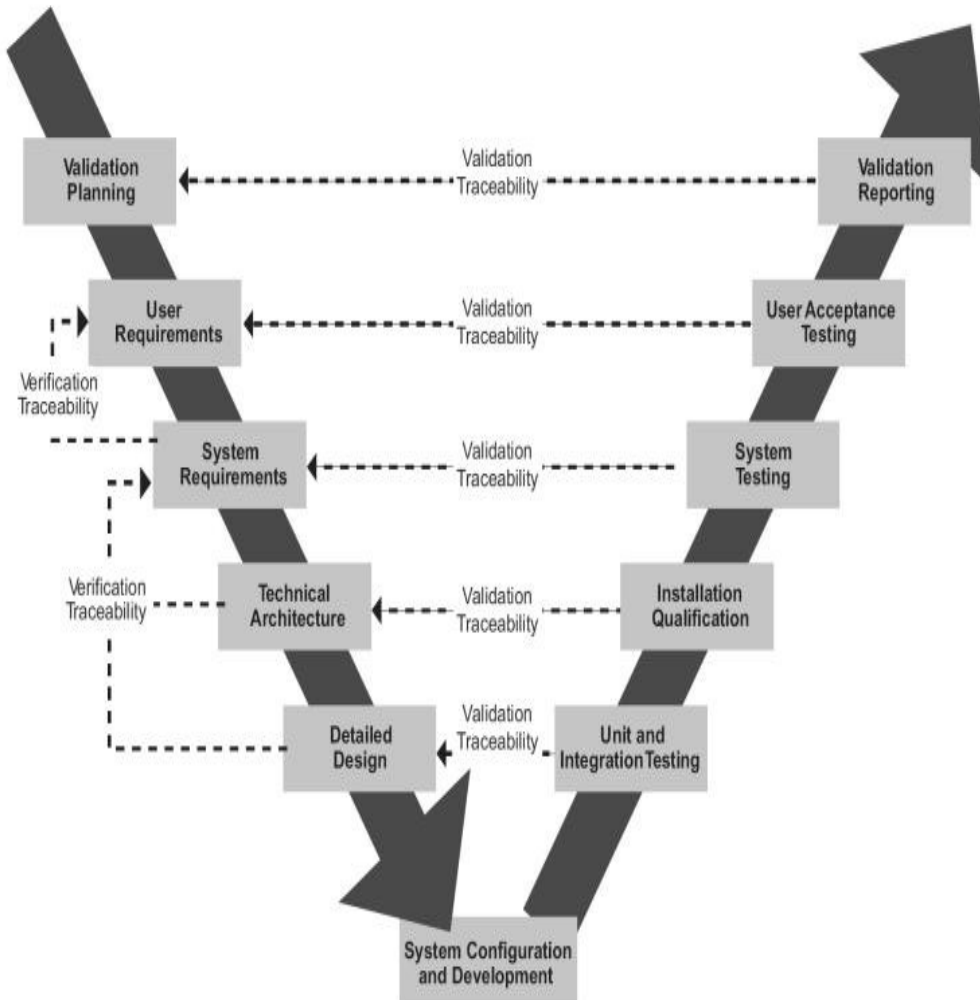
- Prototyping model is a systems development method (SDM) in which a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed
- Each level in prototype is fully tested before building the next level. Total planned testing effort would be more than that required for simple waterfall model



Rapid application development (RAD)

- Emphasizes on extremely short development cycle
- Requirements are well understood
- Create a fully functional system in very short time period
- Achieved by component based construction
- Since well tested components are used , they will be only integration tested when they are used
- The total planned testing effort will be reduced as compared that to other models

V-model



- Every development activity has a corresponding testing activity
- Each test level has test objectives specific to that level

Benefits of early testing

- Testers should be involved in reviewing documents as soon as drafts are available in the development life cycle
- Early test design leads to finding defects early and hence its cheaper to fix defects
- Prevents defect multiplication
- Prevents defect to leak into next stage
- Better quality, less time in running tests because fewer defects found, reduced costs

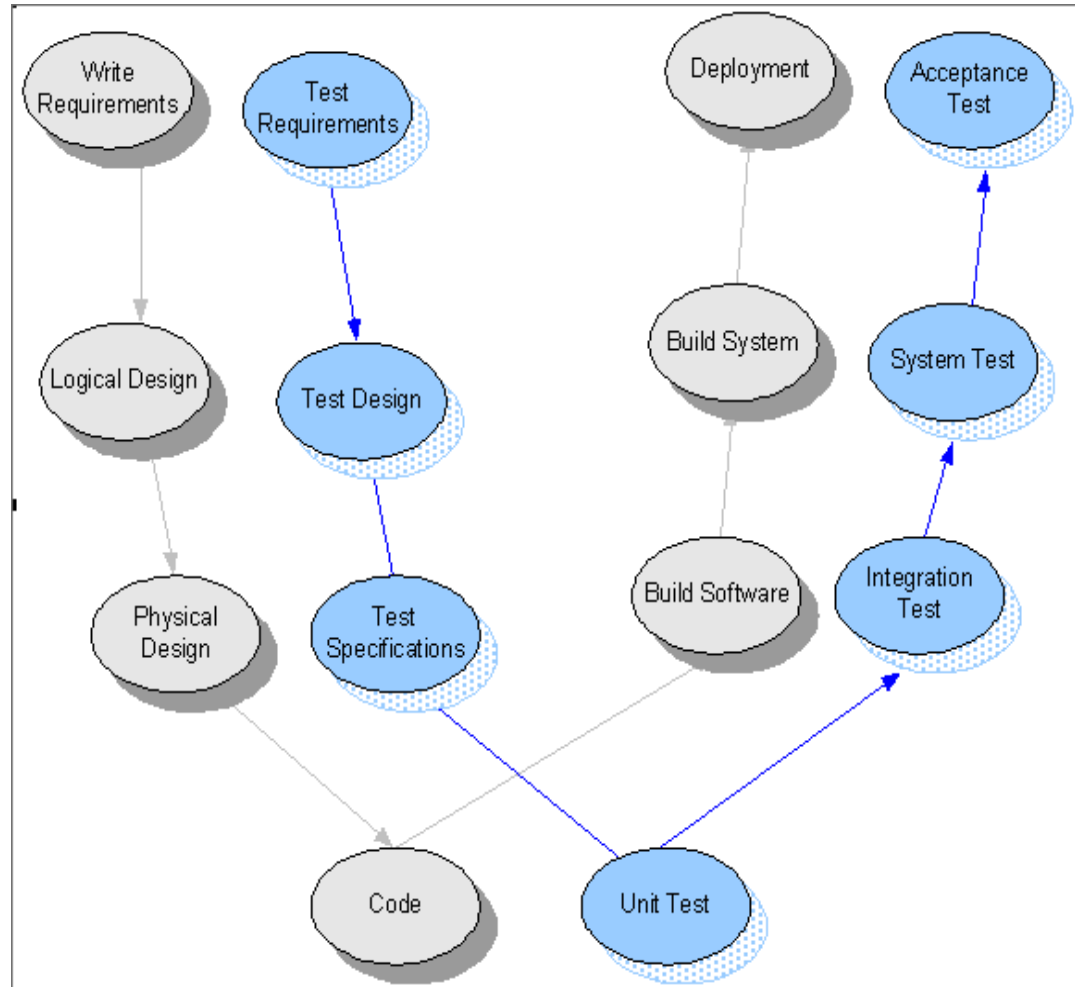
Verification

- Process of evaluating a system or component to determine whether the product satisfies the conditions imposed on them as per users requirements.
- It is also known as Static testing or Quality assurance.
- Ensures that software conforms to its specification by systematically reading the content of software product with the intention of finding defect.
- Ensure whether we are building the software right.

Validation

- Is the determination of the correctness of the products of software development with respect to the user needs and requirements
- Validation ensures that the system has implemented each of the requirements and satisfies the specific intended use.
- It is also known as Dynamic testing or Quality Control.
- Validation checks whether we are building the right software.

Verification and Validation



Validation asks “Did we build the right system?”

Verification asks “Did we build the system right?”

Component testing/unit testing

Component is defined as “ *a minimal software item / unit module for which a separate specification is available*”

- Also known as unit / module / program testing
- Searches for defects in software that is separately testable (E.g. objects, modules, etc.)
- Done in isolation from the rest of the systems
- Tests the functionality of Units
- Uses both white box and black box testing techniques
- Stubs and drivers may be used.
- Mostly done by Developers.

Integration testing

- Also called “link testing” or “component integration testing”
- Tests interfaces between components and interactions to different parts of a system E.g. O/S, H/W, files systems etc.
- Group of modules tested together for dependencies and interfaces
- Major focus is on design and construction of software architecture.

Types Of Integration testing

Levels of integration testing

- Component integration testing tests the interactions between software components and is done after component testing
- System integration testing tests the interactions between different systems and may be done after system testing

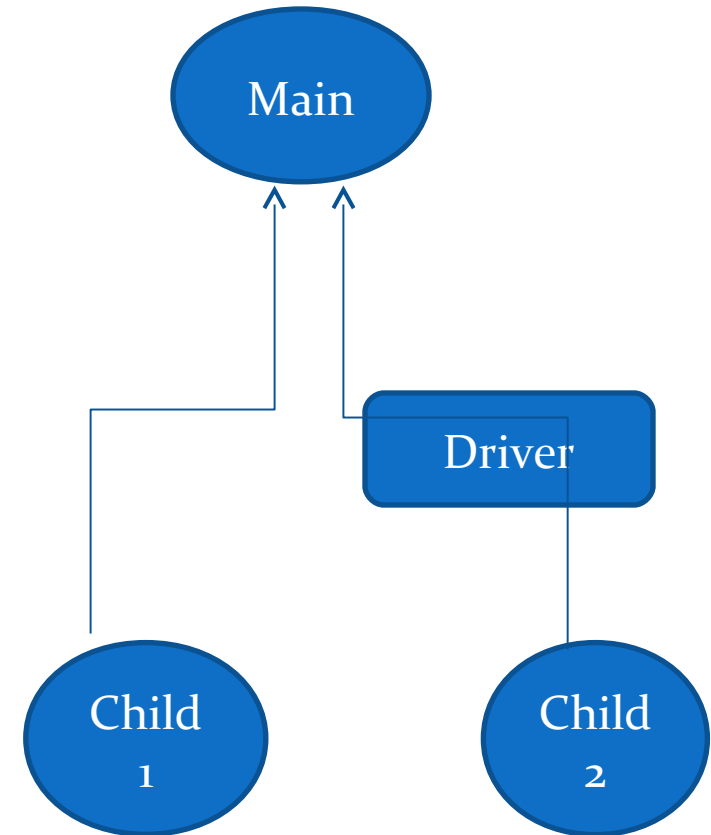
Integration strategies/Approaches:

- ➔ Bottom up
- ➔ Top down
- ➔ Big Bang

Integration testing

Bottom-up Integration

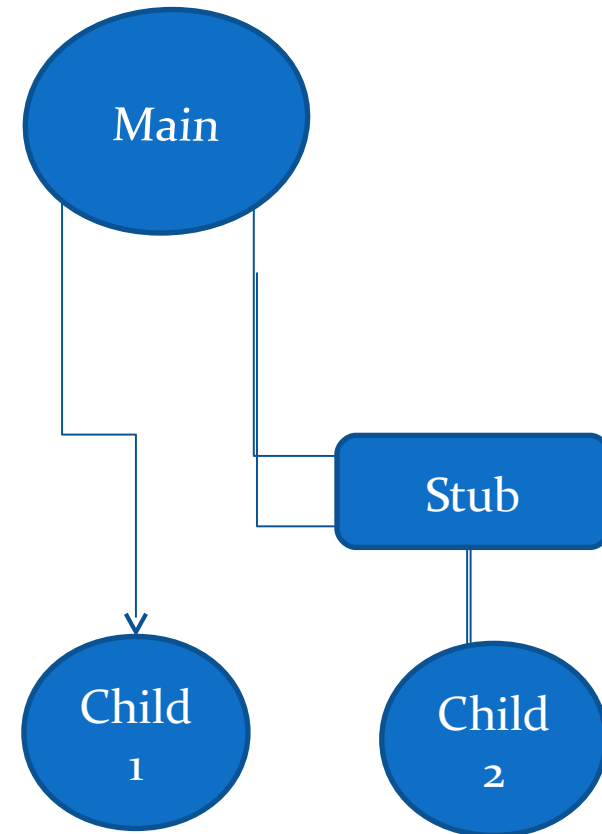
- Combine components starting with the lowest level in the hierarchy
- Since all calling components are not available, drivers can be used.
- *What is a driver?*
 - Driver is a temporary program used for integration level testing when the main module/calling program is not ready for testing
 - Also known as a test harness or scaffolding



Integration testing

Top-down integration

- Combine components starting with the highest level in the hierarchy.
- Since all components that will be called are not available, stubs can be used.
- *What is a stub?*



Stub is a temporary program used for integration level testing when the called program is not ready for testing.

Big Bang Integration

- Put all components together at once and tested.
- Assume that since all components have already been tested at the unit level and they have no defects, they can now be put together at once
- Can save time, but if there is a defect then there is a risk of late defect discovery
- Ultimately, one spends more time !!!

System testing

- Test the behavior of a whole system/product as defined by the user requirements.
- Test environment should correspond to the final target or production environment as much as possible
- Minimizes the risk of environment-specific failures not being found in testing
- An independent test team often carries out system testing
- Is divided into
 - Functional system testing
 - Non-functional system testing

Functional system testing

- End to end functionality testing/ thread testing
- Testing is based on functional requirements specification document
- Use cases are often used to create test cases from a business perspective
- Smoke testing and Sanity testing is usually performed in this test.

Non-functional system testing

Includes:

- Performance, load & stress testing
- Usability testing
- Configuration & installation testing
- Reliability testing
- Back-up & recovery testing
- Compatibility testing

User Acceptance Testing

- After completion of system testing acceptance testing is followed
- Acceptance testing is responsibility of the customers or end users or stakeholders of a system
- Acceptance testing is followed to gain confidence in the system, parts of the system
- To assess the system's readiness for deployment and use

Types:

➤ Alpha Testing

➤ Beta Testing

Alpha & beta (or field) testing

- **Alpha testing**

- Done by users at developing organization's site.
- Also Known as "factory acceptance testing"
- Defects are noted down by the developers and fixed before release

- **Beta testing**

- Or field testing, is performed by users at their own locations
- Also known as "site acceptance testing"
- Defects found by users clubbed and reported the development organization

Retesting and Regression testing

Retesting / Confirmation testing

Retesting the software to confirm that the original defect has been successfully removed after it is fixed.

Regression testing / Confidence testing

Repeated testing of an already tested program, after modification, to discover any defects introduced.

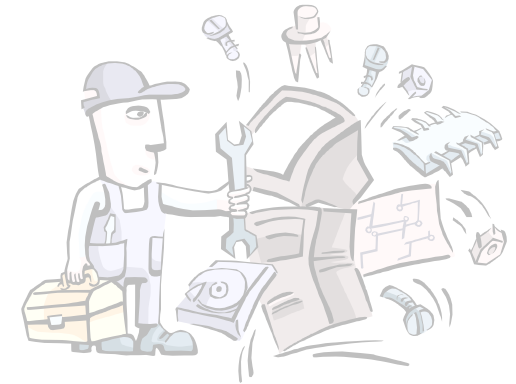
Done when the software is changed or environment is change changed

It is side effect testing.

Maintenance testing

Maintenance testing is done on

- ◆ An existing operational system
- ◆ Is triggered by modifications
- ◆ Migration (one-platform to another)
- ◆ Retirement of the software system
- ◆ For all test levels and all test types



Determining how the existing system may be affected by changes is called impact analysis, which is used to help decide how much regression testing to do

Maintenance testing

- Maintenance testing can be difficult because:
 - ◆ Lack of specifications
 - ◆ Out-of-date documentation
 - ◆ Regression scripts not available
 - ◆ Limited knowledge of the system
- Best possible solution to above:
 - ◆ Talk to users and understand the system's behavior
 - ◆ Document everything, including your views
 - ◆ Explore the current system to identify expected behavior
 - ◆ Refer to any available documentation, including user guides and manuals

CHAPTER 3

Static Testing Techniques

Agenda

- Static techniques and the test process (K2)
- Review process (K2)
- Static analysis by tools (K2)

Static techniques and testing process

- During the Testing process we use
 - ◆ Static testing techniques
 - ◆ Dynamic Testing Techniques
- Static Testing techniques do not execute the software that is being tested
- Static testing techniques are
 - ◆ Manual (reviews)
 - ◆ Automated (static analysis).
- Defects detected during reviews early in the life cycle are often much cheaper to remove than those detected while running tests (e.g. defects found in requirements).

Static testing techniques

In Static Testing documents under review are

- Requirement specifications
- Design specifications
- Code
- Test plans and test specifications, test cases, test scripts
- User guides or web pages

Static techniques - benefits

- Testing work products early in the life cycle
- Find and correct defects early
- Prevent defect multiplication
- Cost of correcting defects is less
- Increased quality
- Improved customer relations
- Higher software development productivity

Static and Dynamic testing

- **Comparison:**

- ◆ Both have the **same objective – identifying defects**
- ◆ They are **complementary**; Both can find different types of defect effectively and efficiently
- ◆ In contrast to dynamic testing, reviews **find defects** rather **than failures**

- **Typical defects found**

- ◆ Deviations from standards
- ◆ Requirement defects
- ◆ Design defects
- ◆ Insufficient maintainability
- ◆ Incorrect interface specifications

Review process

- Reviews vary from very **informal** to **very formal**
- The **formality** of a review process is related to factors
 - 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
- Reviews include verification and validation of documents against specifications and standards
- Generally reviews aim at coming to a common agreement amongst all participants
- Reviews should ideally have process improvement as a goal

Types of reviews

- A single work product may be subjected to more than one review.
- Order of reviews may vary
- Common review types:
 - ◆ Informal review
 - ◆ Walk through
 - ◆ Technical review
 - ◆ Inspection/Formal Review

Informal review

- One-on-one meeting between author and peer
- No formal process – results not formally reported
- Initiated as a request
- Usefulness depends on reviewer
- Need based
- Applied to written documents and unwritten topics
- Better used for discussions/decision making
- Main purpose: inexpensive way to get some benefit

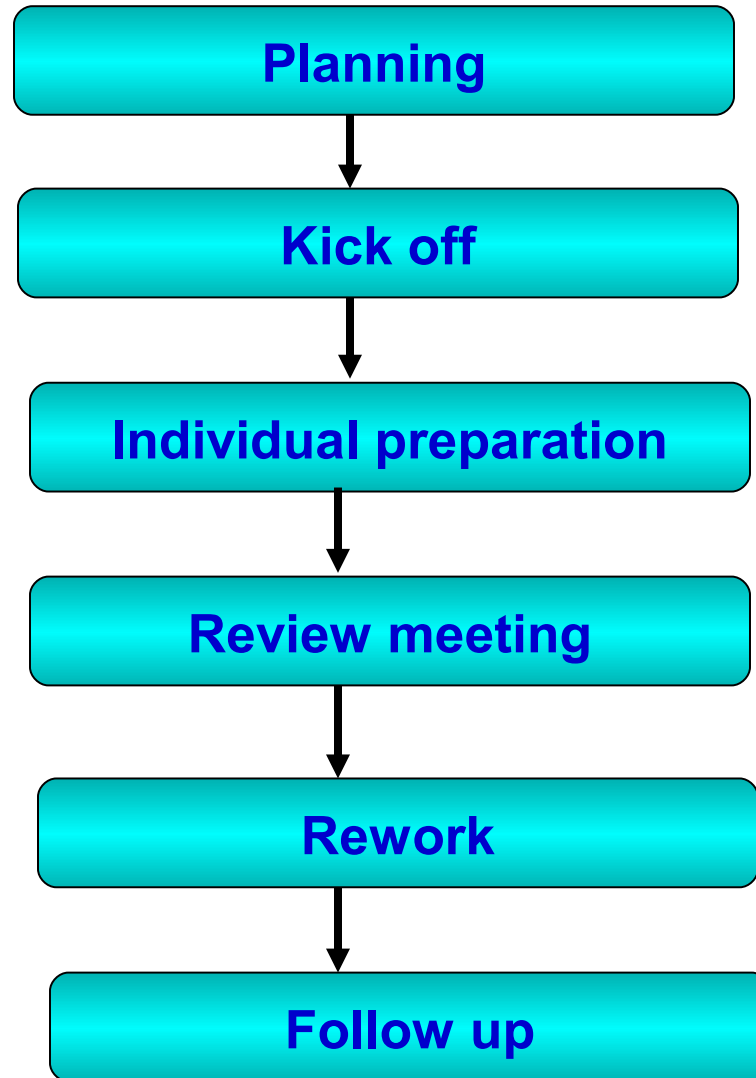
Technical review / peer review

- Led by author
- Participants are educated
- Can include dry runs, scenarios, peer group
- Open-ended sessions
- Optionally a pre-meeting preparation
- May vary in practice from quite informal to very formal
- Main purposes: learning, gaining understanding, defect finding

Walkthrough

- Documented, defined defect-detection process that includes peers and technical experts
- Peer review without management participation
- Led by trained moderator (not the author)
- Pre-meeting preparation
- Use of checklists
- Varies 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

Inspection/Formal review - phases



Formal review roles & responsibilities



Formal review roles & responsibilities

- **Manager**

- ◆ Decides on the execution of reviews
- ◆ Allocates time in project schedules
- ◆ Determines if the review objectives have been met

- **Moderator**

- ◆ Leads the review of the document or set of documents
- ◆ Plans the review
- ◆ Runs the meeting
- ◆ Follows-up after the meeting

- **Author/writer**

- ◆ The writer or person with chief responsibility for the document(s) to be reviewed

Formal review roles & responsibilities

- **Reviewers / checkers / inspectors**

- Individuals with a specific technical or business background
- After the necessary preparation, **identify** and **describe** findings (E.g. Defects) in the product under review

- **Scribe / recorder**

- Documents all the issues, problems and open points that were identified during the meeting
- Looks at documents from different perspectives
- Uses checklists which makes reviews more effective and efficient

Formal review or Inspection

- Most formal of all review techniques
- Strictly follows entry and exit criteria
- Led by moderator (not the author)
- Roles are pre-defined
- Defect searching based on defined rules and checklists
- Metrics are kept
- Inspection report, list of findings are maintained
- Formal follow-up process
- process improvement is the goal
- Main purpose: to find defects

Success factors for reviews

- Each review has a clear predefined objective
- The right people for the review objectives are involved
- Defects found are welcomed, and expressed objectively
- Checklists or roles are used if appropriate to increase effectiveness of defect identification
- Training is given in review techniques, especially the more formal techniques, such as inspection
- Management supports a good review process
- There is an emphasis on learning and process improvement

Static analysis by tools

Objective

- To find defects in software source code and software models without actually executing the software
- Analyze program code as well as generated output such as HTML and XML
- Typically used by developers before and during component and integration testing, and by designers during software modeling

Typical defects discovered by static analysis tools include

- Referencing a variable with an undefined value
- Inconsistent interface between modules and components
- Variables that are never used
- Unreachable (dead) code
- Programming standards violations
- Security vulnerabilities
- Syntax violations of code and software models
- Parameter type mismatches
- Uncalled functions and procedures
- Possible array bound violations

Data flow analysis

- Study of program variables
- Represents a program by a flow graph annotated with information about variable definition, references and indefiniteness
- Identifies: Data flow errors, undefined variable used

E.g.

```
Int a,b,c;
```

```
A=10;b=15;
```

```
If (a>b)
```

```
A=c; //c has not been defined, hence its value may
```

be

```
//different each time the program executes
```

```
Display a;
```

CHAPTER 4

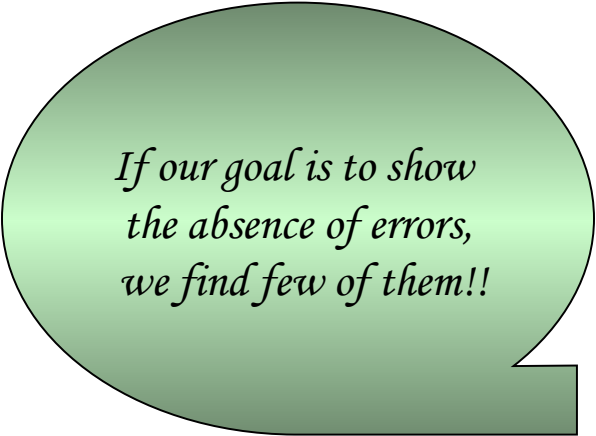
Test design techniques

Agenda


- Identifying test conditions and designing test cases (K3)
- Categories of Test Design Techniques (K2)
- Specification based or Black Box testing Techniques (K3)
- Structure based or White Box Techniques (K3)
- Experience based techniques (K2)

What is Test technique

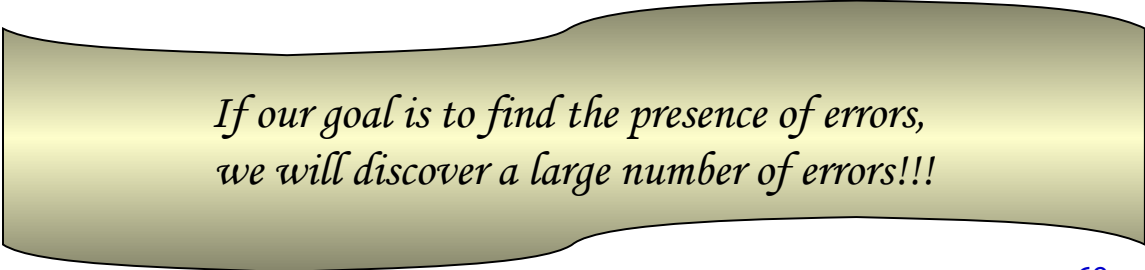
Test techniques are used to select the 'best' tests from the infinite number of all possible tests for a given system



*If our goal is to show
the absence of errors,
we find few of them!!*



*Testing is the state
of mind!!*



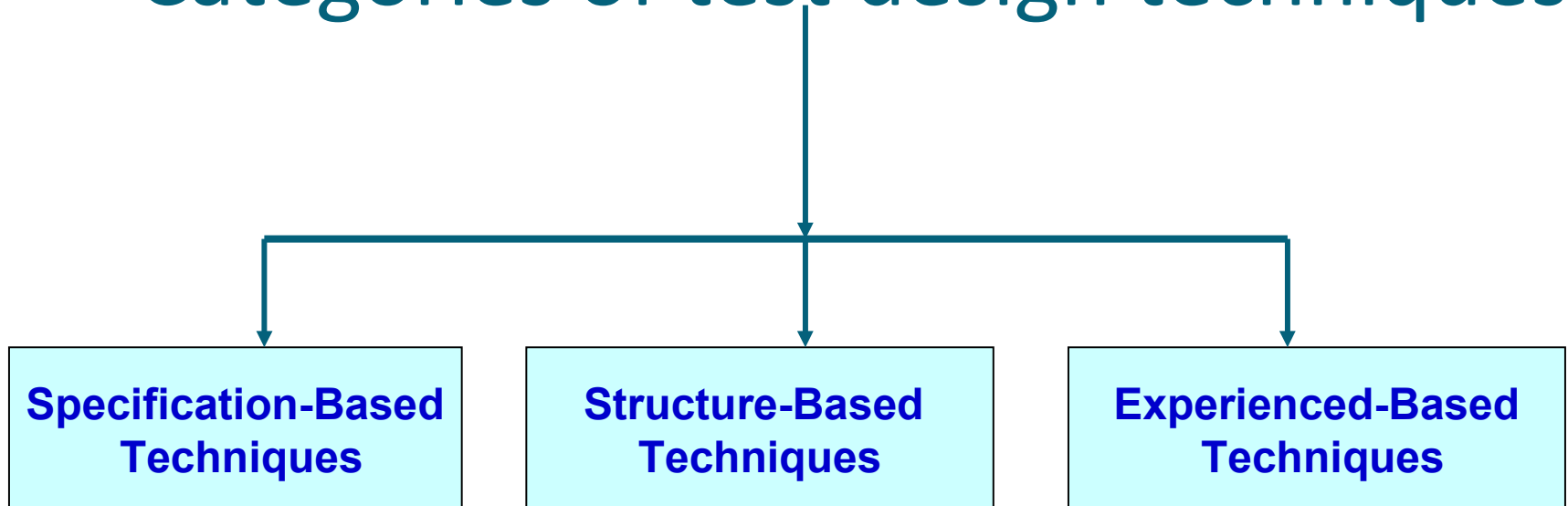
*If our goal is to find the presence of errors,
we will discover a large number of errors!!!*

Identifying test conditions and design test cases

- Test condition
 - ◆ An item or event that could be checked by one or more test cases (E.g. a function, transaction, quality characteristic or structural element)
- Test cases and test data
 - ◆ Developed and described in detail by using test design techniques
- Expected results
 - ◆ Produced as part of the specification of a test case and include outputs, changes to data and states, and any other consequences of the test
- Test procedure (or manual test script)
 - ◆ Specifies the sequence of action for the execution of a test

Test Design Technique

Categories of test design techniques



Specification-based Techniques

- Also called Black-box techniques
- Based on functional/non functional requirements designs test case

Structure-based techniques

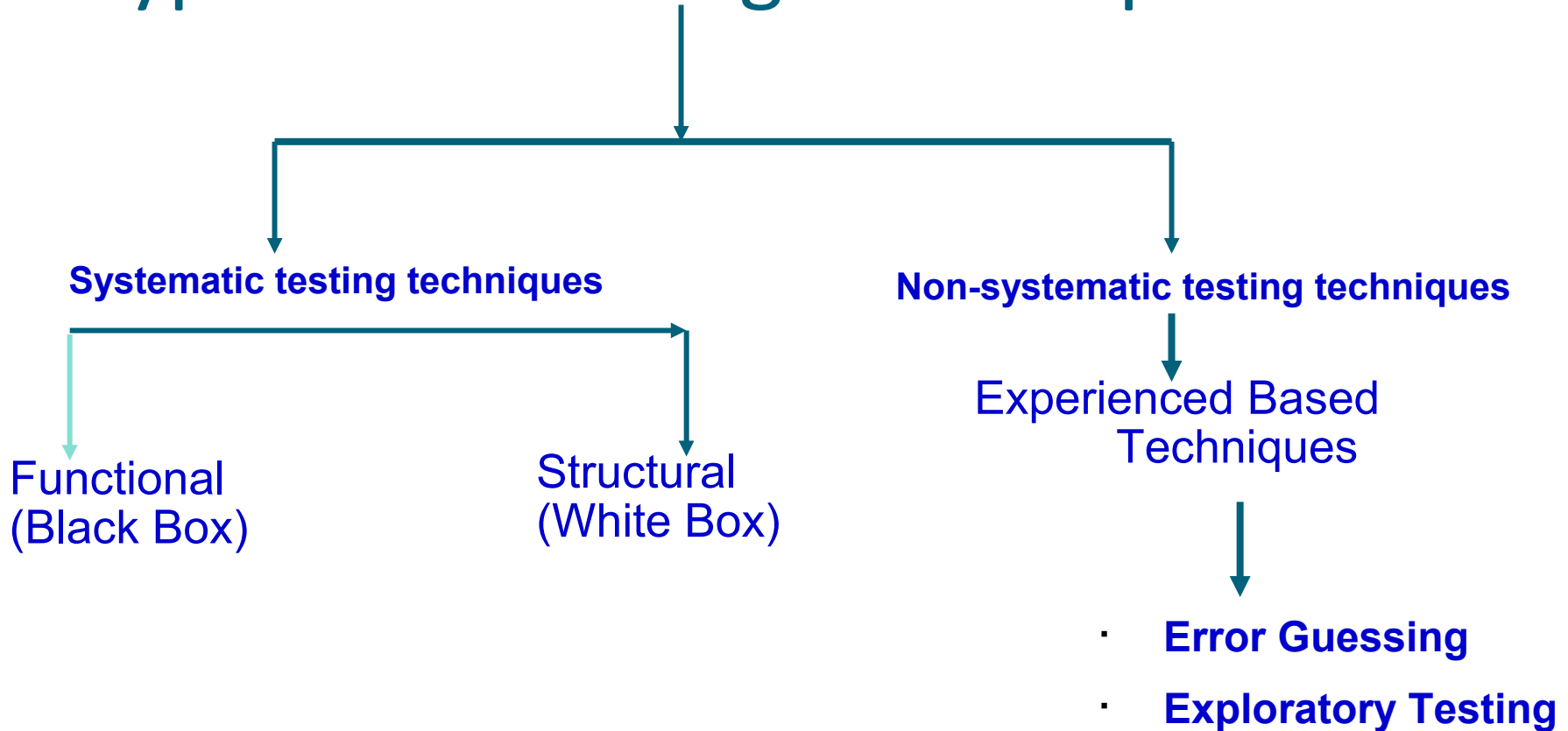
- Also called White-box or Glass box techniques
- Information about how the software is constructed is used to derive the test cases, for example, code and design
- Coverage of the software can be measured for existing test cases, and further test cases can be derived systematically to increase coverage

Experience-based techniques

Experience-based techniques

- Knowledge and experience of people like testers, developers, users is used to derive the test cases
- Knowledge of software, its usage, its environment, likely defects and its distribution is required for deriving test cases

Types of test design techniques



Specification based or Black-box techniques

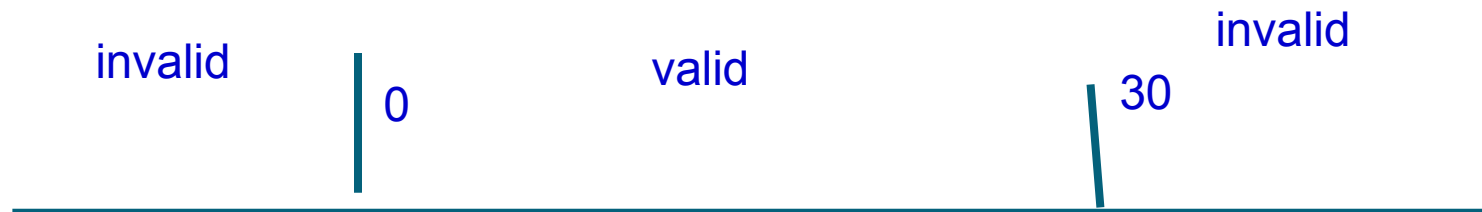
- Also called functional , behavioral or data-driven, input/output driven, or requirements-based testing
- Focus is on evaluating the function
- Determines whether the combination of inputs and operations produce expected results
- Test conditions are derived from the specifications

Black-box Testing techniques

- Equivalence partitioning
- Boundary value analysis
- Decision table testing
- State transition testing
- Use Case testing

Equivalence partitioning or equivalence classes

- Equivalence Partitioning or Equivalence Class defines set of valid and invalid states for input condition
- Divide the inputs and outputs into equivalent classes
- Equivalence partitioning (EP) is applicable at all levels of testing



Example:

Employee Details

Employee
Code

Employee Name

Date of Birth

Date of Joining

Probation End date

Current Gross Salary

Net Salary

Department

Designatin

→ Valid date (dd/mm/yy)

→ Valid date (dd/mm/yy)

→ Valid date (dd/mm/yy)

→ 1000 to 999999

Equivalence partitioning

- Date of birth, Date of joining, Probation end date

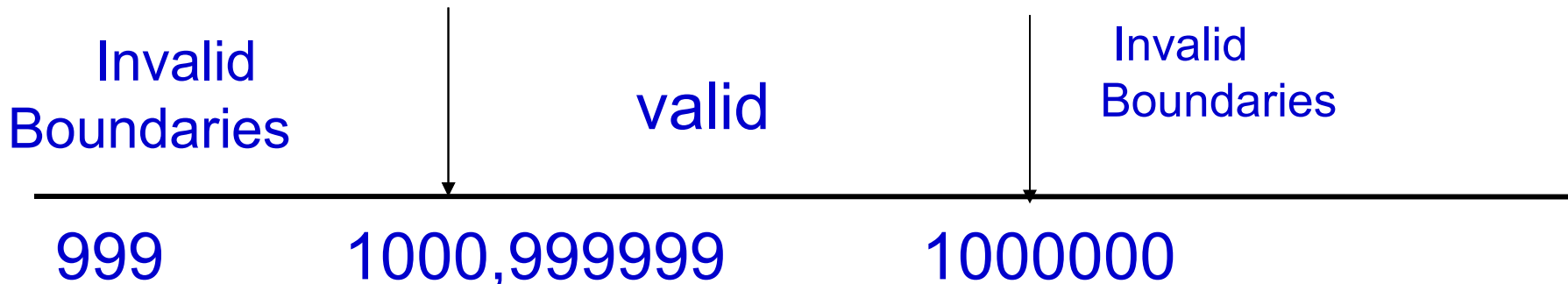
invalid	valid
32/12/70,30/14/70	01/01/1980, 02/10/87,31/01/67

- Current Gross Salary

invalid	valid	invalid
<1000	1001 to 999999	999999<

Boundary value analysis

- Defects tend to lurk near boundaries
- Good place to look for defects
- Test Values on and either side of each valid boundary
- BVA enhances Equivalence Partitioning by selecting elements just on, and just beyond the borders of each Equivalence Class
- Applies at all levels of testing



Decision table testing

- A table containing combinations of true and false for all input conditions and the resulting actions for each
- Each column corresponds to a business rule that defines a unique combination of conditions that result in the execution of the actions associated with that rule
- Requirements that contain logical conditions can be captured
- At least one test case is written per column

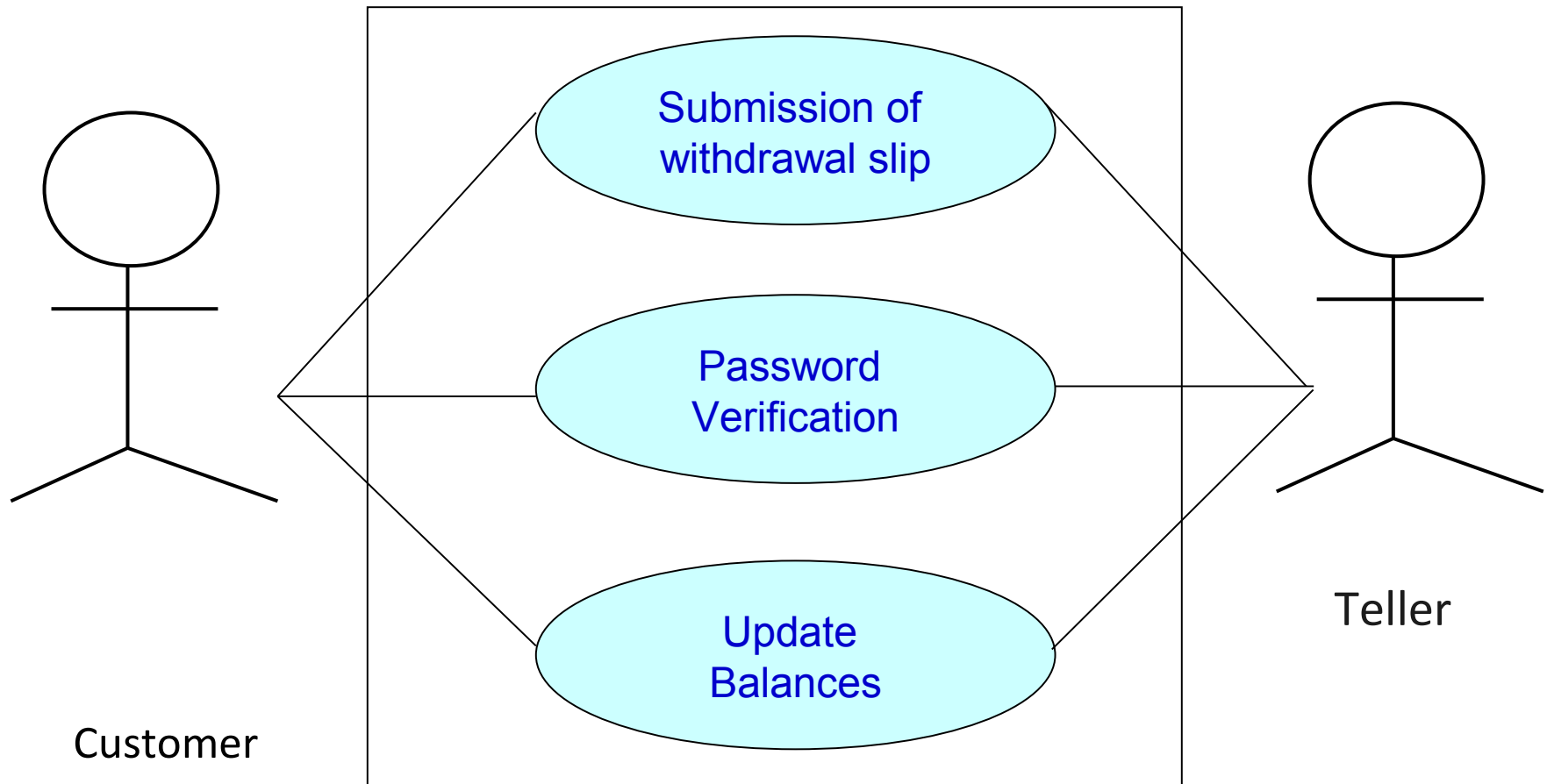
State transition testing (analysis)

- State transition diagram indicates how the system behaves as a consequence of external events by analyzing the conditions under which a state change occurs
- A state table shows the relationship between the states and inputs
- Tests can be designed to cover a typical sequence of states
- Useful for testing the behavior of individual objects over the full set of use cases that affect those objects

Use case testing

- A use case defines a goal-oriented set of interactions between external actors and the system under consideration
- Use cases capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals
- Anything (user or another system) that interacts with the system is referred to as an Actor
- It describes the “process flows” through a system based on its actual likely use
- Example : Cash withdrawal from Savings account system

Use case diagram for cash withdrawal from a saving's account system



Use Case testing

- Use cases are very useful for designing acceptance tests with customer/user participation
- They also help uncover integration defects caused by the interaction and interference of different components, which individual component testing would not see

Structure-based or white-box testing techniques

- Also called 'Logic Driven Technique'/'Glass Box Technique'
- Uses specific knowledge of programming code to examine outputs and assumes that the tester knows the path of logic in a unit or program
- Carried out during the coding and unit-testing phase of the software development life cycle
- Evaluates that all aspects of the structure have been tested and that the structure is sound
- Structural tests predominantly use verification

Decision/Branch/Edge coverage

- In branch/decision testing we test every possible branch, regardless of the outcome
- Decision/branch coverage is the assessment of the percentage of decision outcomes (E.g. the True and False options of an IF statement) **at least once with minimum runs**
- Decision coverage is stronger than statement coverage: 100% decision coverage guarantees 100% statement coverage, but not vice versa
- Also known as Edge Coverage requires that
- Branch Coverage=
$$\frac{\text{Number of branches exercised}}{\text{Total Number of Decisions or Branches}} \times 100$$

Typical adhoc testing achieves 40-60 %
of branch/decision coverage

- Example :

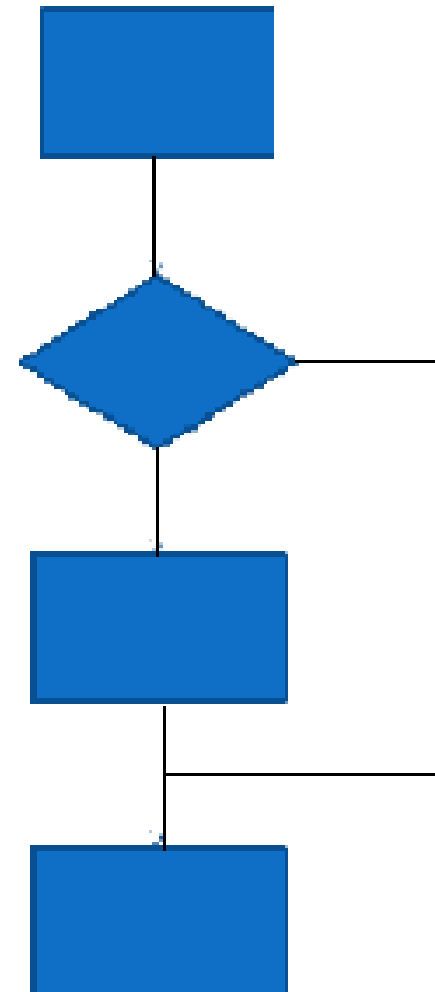
Read A

IF A>50

Print "Apple"

ENDIF

Branch coverage= 2



Statement testing and coverage

Statement coverage measures the percentage of all executable statements executed by software under test **at least once with**

$$\text{minimum runs.} \quad = \frac{\text{no of statements exercised}}{\text{Total number of statements}} \times 100$$

Statement coverage = Total number of statements

Total number of Statements in the program is always = 100

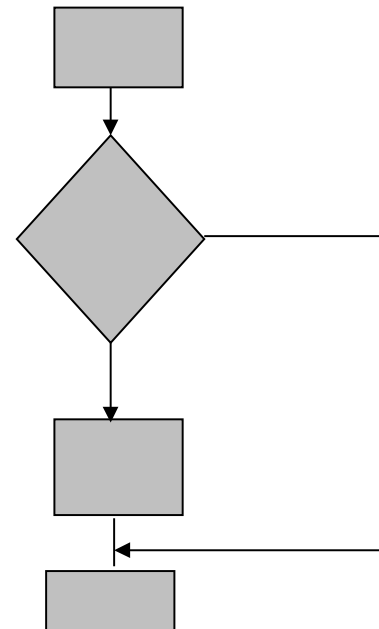
Example: Read A

IF A>50

Print "Apple"

ENDIF

Statement Coverage = 1



Example 1: BC and SC

To check the biggest of two numbers.

Read A

Read B

If $A > B$ then

Print "A is greater"

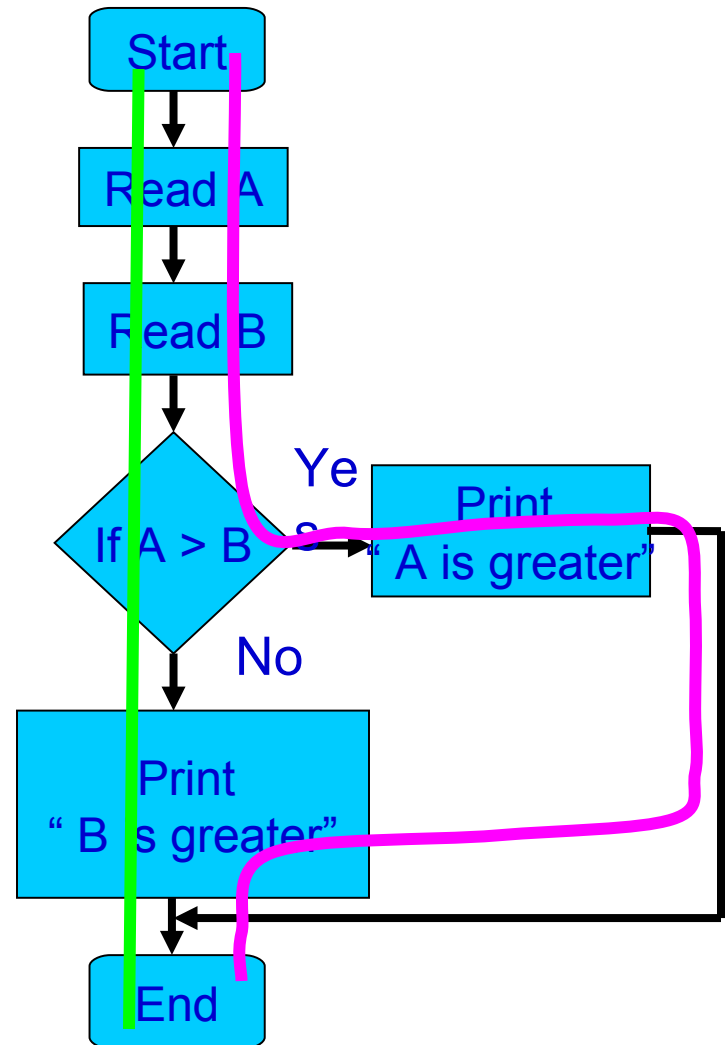
Else

Print "B is greater"

End if

BC= 2

SC= 2

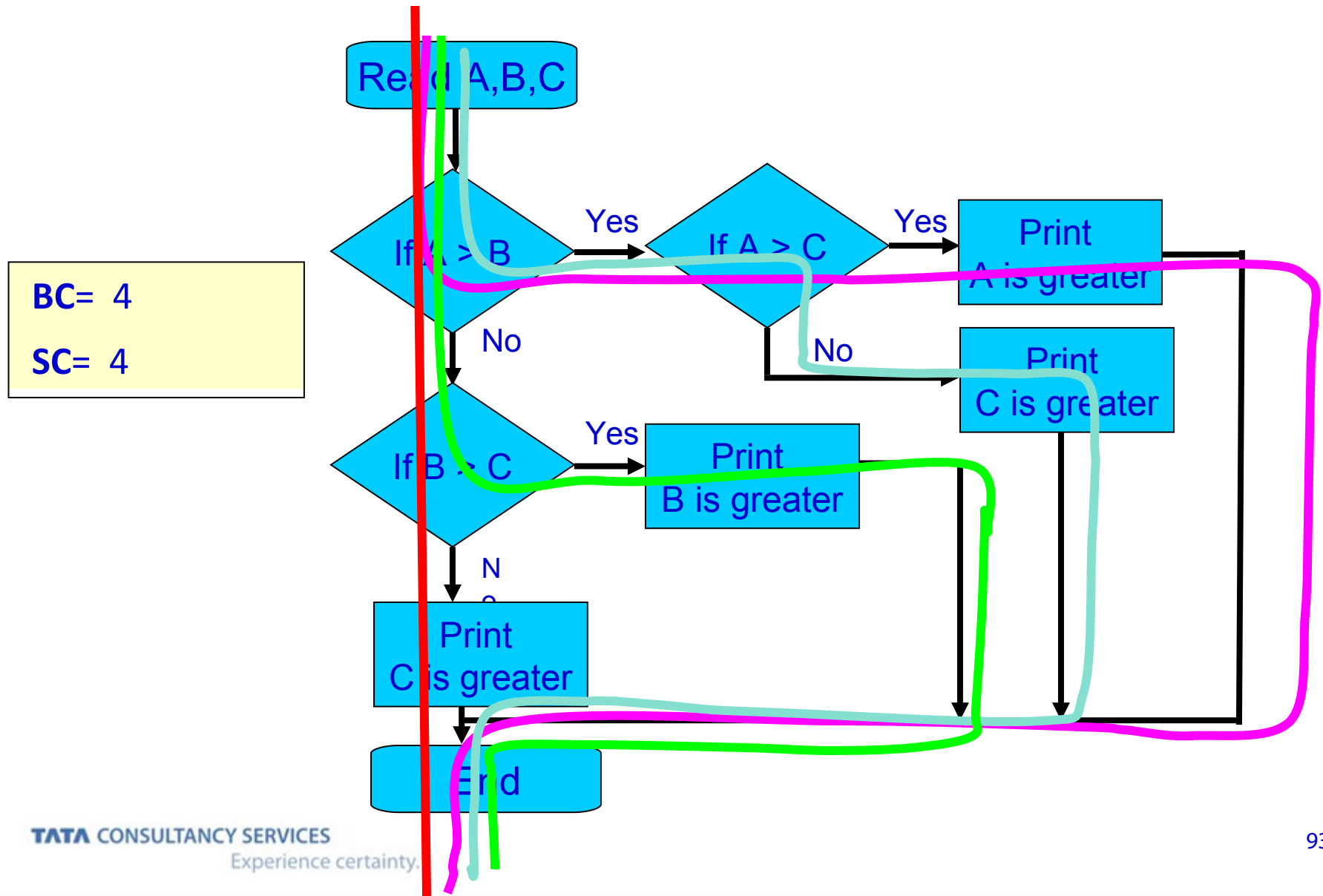


Example 2:-BC,SC

To check the largest of three numbers

```
Read A,B,C
If A > B
  If A > C
    print "A is greater"
  else
    print "C is greater"
  end if
else
  If B > C
    print "B is greater"
  else
    print "C is greater"
  end if
end if
```

Example 2 Solution:-



Example 3:-BC, SC

To check whether numbers are negative

Read A

Read B

If $a < 0$ then

 Print "A negative"

Else

 Print "A positive"

End if

If $b < 0$ then

 Print "B negative"

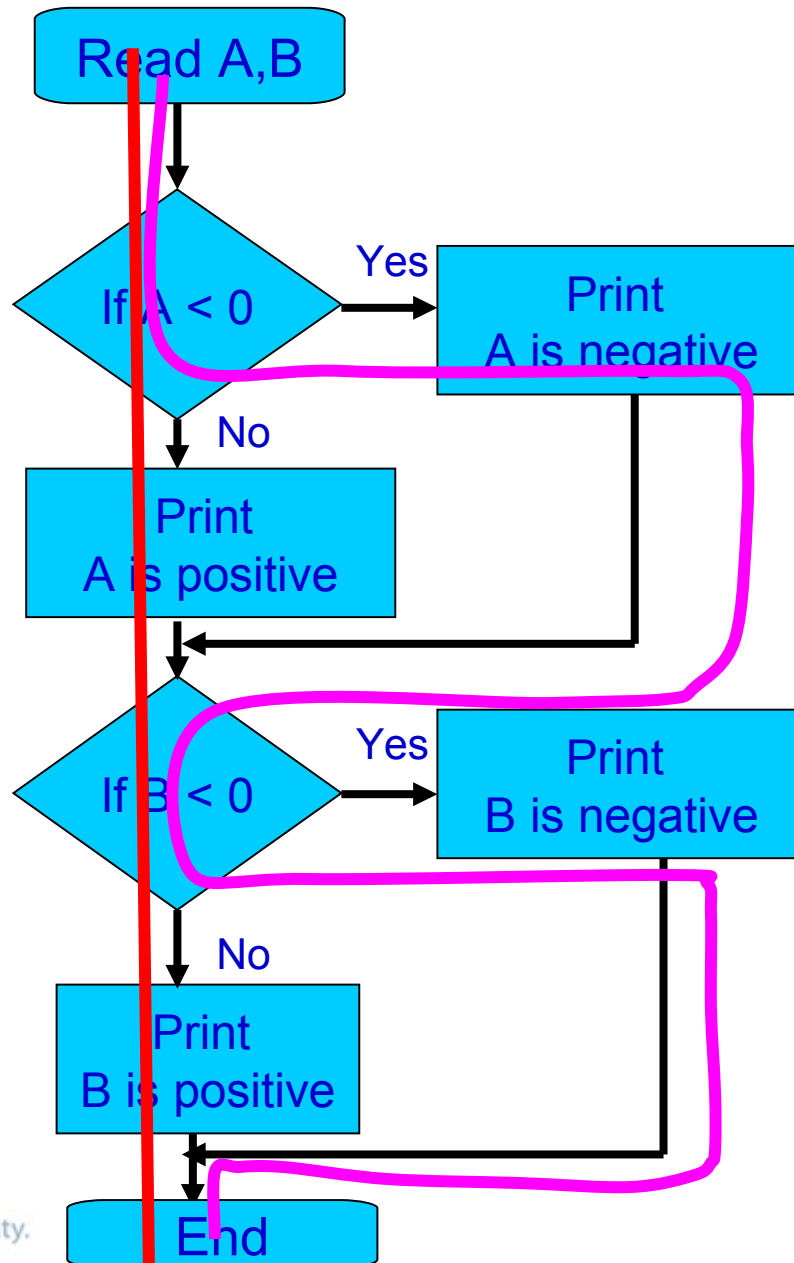
Else

 Print "B positive"

End if

Example 3 Solution:-

BC= 2
SC= 2



Experience based techniques

- Tests are derived from the tester's skill and intuition and their experience with similar applications and technologies
- Used to augment systematic techniques, to identify special tests not easily captured by formal techniques
- May yield widely varying degrees of effectiveness, depending on the testers' experience

Exploratory testing

- Concurrent test design, test execution, test logging and learning, based on a test charter containing test objectives, and carried out within time-boxes
- An approach that is most useful where there are few or inadequate specifications and severe time pressure, or in order to augment or complement other, more formal testing
- Can serve as a check on the test process, to help ensure that the most serious defects are found

Error guessing

- Most widely practiced experience-based technique
- A structured approach to the error guessing technique is to enumerate a list of possible errors and to design tests that attack these errors
- Error guessing by an experienced engineer is probably the single most effective method of designing tests that uncover bugs
- A well-placed error guess can show a bug that one could easily miss by many of the other test case design techniques. Conversely, in the wrong hands error guessing can be a waste of time

CHAPTER 5

Test Organization

Agenda

- Test organization (K2)
- Test planning and estimation (K2)
- Test progress monitoring and control (K2)
- Configuration management (K2)
- Risk and testing (K2)
- Incident Management (K3)

Test organization

- Complex or safety critical projects should have multiple levels of testing, with some or all of the levels done by independent testers
- Development staff can also participate in testing
- Independent testers may have the authority to require and define test processes and rules
- Testers can take on process-related roles only in the presence of a clear management mandate to do so

Common practices

Testing Level	Tester
Unit/Component	Programmers(or buddy)
Integration	Independent testers from same organization
System Testing	Independent testing team
Acceptance Testing	Business experts and users
Operational Acceptance Testing	Operators

Tasks of a Test leader

- Test strategy and test policy creation/review
- Test planning and test estimation
- Deciding the test approach
- Coordination
- Scheduling
- Acquiring resources
- **Who can be a test leader ?**
 - The Project Manager or QA Manager or Development Manager can be a Test Leader for small and simple projects
 - Large / complex projects will have a separate Test Leader helped by a team

Tasks of a Test leader

- Test strategy and test policy creation/review
- Test planning and test estimation
- Deciding the test approach
- Coordination
- Scheduling
- Acquiring resources
- **Who can be a test leader ?**
 - The Project Manager or QA Manager or Development Manager can be a Test Leader for small and simple projects
 - Large / complex projects will have a separate Test Leader helped by a team

Tasks of the tester

- Contribute to test plans
- Create and/or review test specifications
- Set up the test environment and test data
- Check for testability
- Implement tests
- Evaluate the results and document the deviations
- **Who is a Tester?**
 - ◆ Person who specializes in:
Test analysis, test design, specific test types ,test automation
,keeps in view the test level and the risks related to the product
and the project

Test Planning- IEEE 829 STANDARD TEST PLAN TEMPLATE

1. Test plan identifier
2. Test deliverables
3. Introduction
4. Test tasks
5. Test items
6. Environmental needs
7. Features to be tested and Features not to be tested
8. Responsibilities
9. Staffing and training needs
10. Approach

Test Planning- IEEE 829 STANDARD TEST PLAN TEMPLATE

11 .Schedule

12.Item pass/fail criteria

13. Risks and contingencies

14. Test Estimation

15.Suspension and resumption criteria

16. Approvals

Test estimation approaches

Two approaches are used for the estimation of test effort :

- ◆ Based on **past projects data**
- ◆ Based on **owners of the tasks** or experts in the



Test approaches (Test strategies)

- **Based on Time :**

- ◆ **Preventive** approaches, where tests are designed as early as possible
- ◆ **Reactive** approaches, where test design comes after the software or system has been produced

- **Other Approaches :**

- ◆ Analytical approaches
- ◆ Model-based approaches
- ◆ Methodical approaches
- ◆ Process-or standard-compliant approaches
- ◆ Dynamic and heuristic approaches
- ◆ Regression-averse approaches

Test progress monitoring

- Measures the actual progress against the milestones
- Gives feedback and visibility about test activities
- Metrics may be used for measuring progress
- Manual or Automated data collection

Test Summary Reporting

- **Test Summary Reporting means reporting on what happened during the period of testing**
- **Sample contents of a test report :**
 - Analyzed information and metrics collected
 - Assessment of defects remaining
 - Economic benefit of continued testing
 - Outstanding risks
 - Level of confidence in tested software
 - Recommendations and decisions about future actions

Test control

- Test control means any guiding or corrective actions taken as a result of information and metrics gathered and reported
- Actions may cover any test activity and may affect any other software life cycle activity or task

Examples of test control actions

- ◆ 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

Configuration management

- Identifying and defining Configuration Items (CI)
- Controlling the release and changes to CI
- Recording and reporting the status of CI
- Verifying the completeness and correctness of CI
- **Purpose:**
 - To establish and maintain the integrity of the products
 - (components, data and documentation) of the software
 - or system through the project and product lifecycle
 - Release

Configuration management in testing

- Configuration Items of testing include :
 - Test plans
 - Test specifications
 - Test data and test harness
 - Test results
- Is maintaining integrity and traceability of the test ware
- Is the responsibility of the system development team
- The system release documentation includes the details on the version number of release and changes made to the release when the system is released for testing

Risks and testing

- **Risk** is defined as
 - ◆ Chance of an adverse event occurring
 - ◆ Threat or situation occurring and its undesirable consequences
 - ◆ A potential problem
- The **level of risk** will be determined by
 - ◆ likelihood of an adverse event happening
 - ◆ impact (the harm resulting from that event)



Risk management

- **Risks types**

- ◆ **Project risks**

The risks that surround the project's capability to deliver its objectives

- ◆ **Product risks**

Potential failure areas (adverse future events or hazards) in the software or system, as they are a risk to the quality of the product

Defect

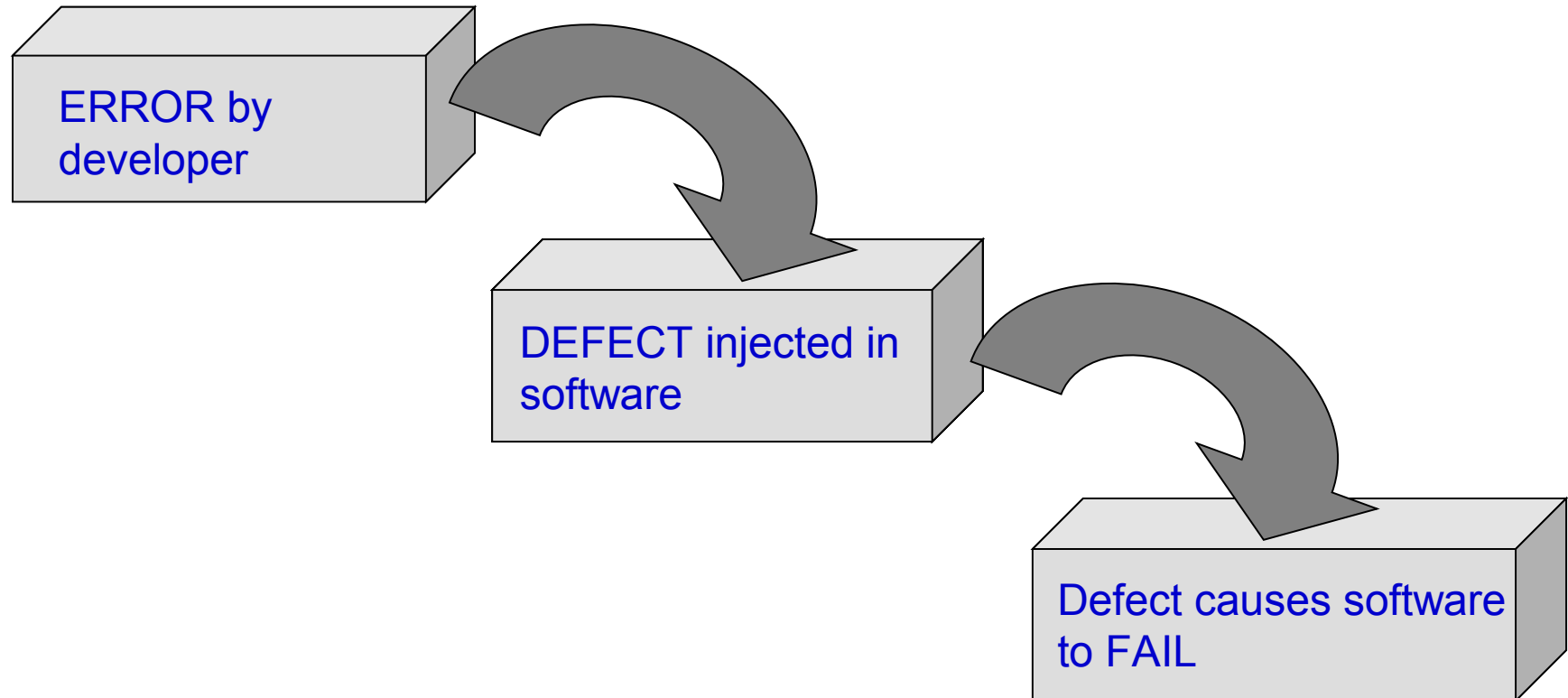
- Variance/ difference between the expected and actual result is called as “Defect/Bug”.

Incident

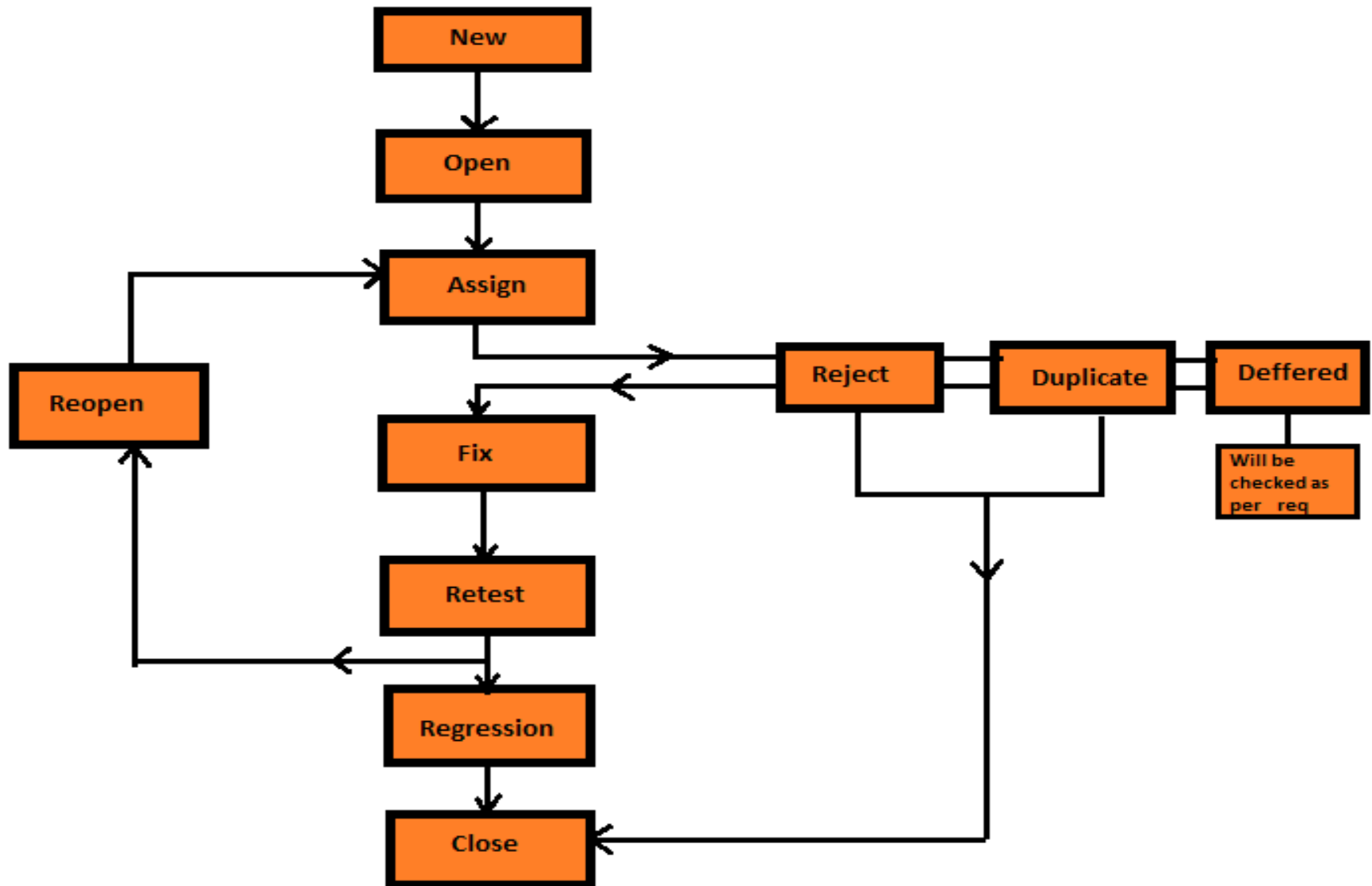
- The discrepancies between expected and actual outcomes which need to be logged and **investigated** is called as **Incidents**.

Note: Incident after acceptance by the developer as valid is called as defect as per process.

Error → Fault → Failure



Defect/Bug life cycle



Incident report/Defect report

Open defects - Defect Priority per System Part

status (Multiple Items) ▼

Count of Id	Column Labels				
Row Labels	1. Urgent	2. High	2. Medium	4. Low	Grand Total
Billing	4	6	5	4	19
Contract	2	5	5	6	18
Intake	1	2	4	4	11
Interfaces - System B	1			4	5
Interfaces - System A			1	3	4
Login		1		2	3
Grand Total	8	14	15	23	60

Detailed progress

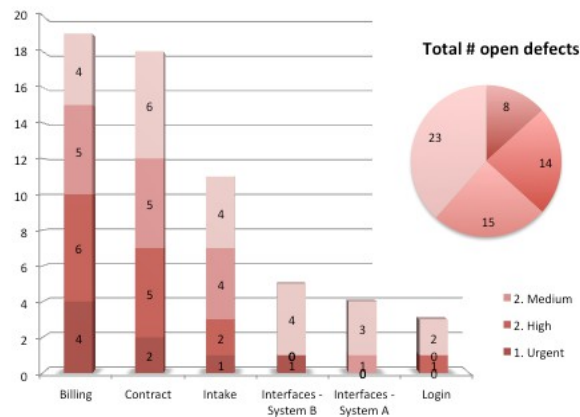
	1. Urgent	2. High	2. Medium	4. Low	Grand Total
week 1	2	5	1	4	12
week 2	1	4	8	10	23
week 3	7	10	15	13	45
week 4	5	11	19	20	55
week 5	8	14	15	23	60
week 6					
week 7					
week 8					
week 9					
week 10					
week 11					
week 12					

Open defects - Defect Causation

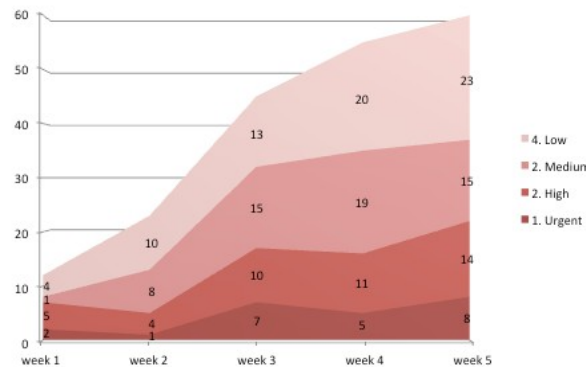
status (Multiple Items) ▼

Count of Id	Column Labels				
Row Labels	1. Urgent	2. High	2. Medium	4. Low	Grand Total
code back end		2	3	1	6
code front end		2	4	10	17
Configuration	6	2	3	5	16
design	1	2	2	1	6
migration		4	1	6	11
test		2	2		4
Grand Total	8	14	15	23	60

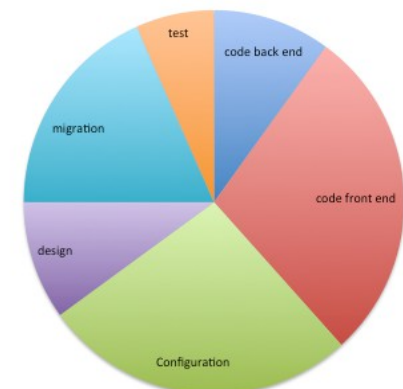
Open defects - Defect priority per System Part



Open defects - Defect priority progress



Open defects - Defect causation



Comment:

Login and interfaces - System A have been tested once completely. All other system parts have still unresolved blocking issues. Billing and Contract are delivered with a lower quality than expected. The modules could not be tested completely due to the blocking and critical issues found that need urgent and high fixing priorities. The gap between open and closed defects keeps growing. More attention is required on defect fixing. 6 out of 8 open blocking issues that need urgent fixing are found to be due to the same system configuration. An deployment is scheduled tomorrow morning to resolve these. Configuration needs more attention in general. Too many defects are found due to configuration.

Priority of defect

Priority tells us how soon is desired problem to fixed.

E.g.

- **High** – within a day or two

Level 1

- **Medium** – within a week or two

Level 2

- **Low** - within 3 to 4 weeks

Level 3

Defects which are high on severity need not be high on priority and the vice versa

Severity of Defect

Severity tells us how bad the defect is

E.g.

Critical: The software crashes, hangs, or causes you to lose data.

Level 1

High: A major feature does not exist/not working.

Level 2

Medium: It's a bug that should be fixed.

Level 3

Low: Minor loss of function, and there's an easy work around.

Level 4

Enhancement: Request for new feature or enhancement.

Level 5

CHAPTER 6

Tool support For testing

Tool support for testing

- Types of testing tools (K2)
- Effective use of tools: Potential benefits and risks (K2)
- Introducing a tool into an organization (K1)

Test Tool Classification

- **Types of Test tools**

- Management of testing and tests activities
- Static testing
- Test specification
- Test execution and logging
- Performance and monitoring
- Specific application areas
- Other tools

Test management tools

- For managing testing activities
- Interface with
 - ◆ Test execution tools
 - ◆ Defect logging and tracking tools
 - ◆ Requirement management
 - ◆ Allow version control
- Support traceability
- Generate progress reports
- Provide quantitative analysis (metrics)
- Improve test process
- E.g. Test Director/Quality Control, Star Team(SCM tool basically)

Requirement management tools

- Store requirement statements
- Check for consistency and undefined (missing) requirements
- Prioritize requirements
- Traceability between individual tests and requirements
- Provides information on test coverage of requirements
- E.g. Jira, WinA&D, AnalystPro, CaliberRM

Incident management Tools

- Incident logging/ Defect tracking tool.
- Prioritize and assign incidents to the development team for closure
- Tracks closure of incidents
- Provides incident or anomaly report and other reports for incident management

E.g. Bugzilla, Test Director/Quality Control, Star Team(SCM tool basically)

Configuration management tools

- Maintains information about versions and builds,
- Traceability between complete test ware and software work products and product variants.
- E.g. Star Team, Visual Source Safe, Clear Case, PVCS, VSS-SOS

Tool support for static testing

- **Review process tools**

- ◆ Store information about review processes
- ◆ Store and communicate review comments
- ◆ Provide reference to review rules and
- ◆ checklists
- ◆ Traceability between review documents and source code
- ◆ Facilitate online communication between reviewers in case of geographically dispersed teams

Static analysis tools (D)

- Support in finding defects before dynamic testing
- Used by developers, testers and quality assurance persons
- Support in enforcing coding standards
- Help in analyzing structures and dependencies between software components
- Help in understanding code
- Calculate metrics which help in risk analysis e.g., complexity
- Suite for UML, ERWIN for data modeling
- E.g. Code Sonar for C/C++, STI for C/C++, jtest for Java, Klocwork Suite for Java

Modeling tools (D)

- Validate software development models (data models, static model, object model) and point out inconsistencies in them
- Help in designing test cases
- E.g. Rational Suite for UML, ERWIN for data modeling

Tool support for Test Specification

- **Test design**

- ◆ Generates test inputs or the actual tests from requirements, from a graphical user interface, from design models (state, data or object) or from code
- ◆ May generate expected outcomes also
- ◆ May generate test frames to create tests or test stubs to speed up the testing process

- **Test data preparation – (D)**

- ◆ Create test data using scripts
- ◆ Create test data by manipulating existing data

Tool support for test execution and logging

- **Test execution**

- ◆ Enables tests to be executed automatically, or semi-automatically
- ◆ Uses stored inputs and expected outcomes for comparing outputs
- ◆ Use of a scripting language to automate tests
- ◆ Record and replay facility
- ◆ Automatic logging of defects
- ◆ E.g. Quick Test Professional (QTP/UFT), Selenium, Rational functional tester (RFT), Silk Test, Win Runner

Tool support for test execution and logging

- **Test comparator**

- Comparing files, data, test results
- Built in most of test execution tools but a separate comparator tool is required for post-execution comparison
- Uses a test oracle, especially if it is automated
- E.g. Change, Winrunner also has image and file comparison facility

Tool support for test execution and logging

Coverage measurement tools –:

- Measure the percentage of specific types of code structure that have been exercised (e.g. statements, branches or decisions, and module or function calls) by the test cases
- Can be intrusive or non-intrusive
- E.g. Adatest measures coverage of Ada code (statement ,branch, condition)

Tool support for test execution and logging

- **Security tools**

- Check for computer viruses
- Check for denial of service attacks
- Check for unauthorized access

E.g. Antivirus, E-security, Apshield.

- **Dynamic analysis tools (D)**

- Find defects that are evident only when software is executing, such as time dependencies or memory leaks
- Used in component or component integration testing
- E.g. Bounds Checker checks for memory leaks

Tool support for performance and monitoring

- **Performance testing/load testing/stress testing tools**

- ◆ Performance testing tools monitor and report on how a system behaves under a variety of simulated usage conditions. Eg: data volume, environment stress, users
- ◆ E.g. Load Runner, Silk Performer, Junit

- **Monitoring tools**

- ◆ Monitoring tools continuously analyze, verify and report on usage of specific system resources, and give warnings of possible service problems
- ◆ E.g. Astra Site manager

Other tools

- Spreadsheets
- Word
- SQL,
- Resource or debugging tools (D)
- E.g. of debugging tools are objdump, strace, ltrace, Debug tools from IBM
for z/OS

Effective use of tools: Potential benefits

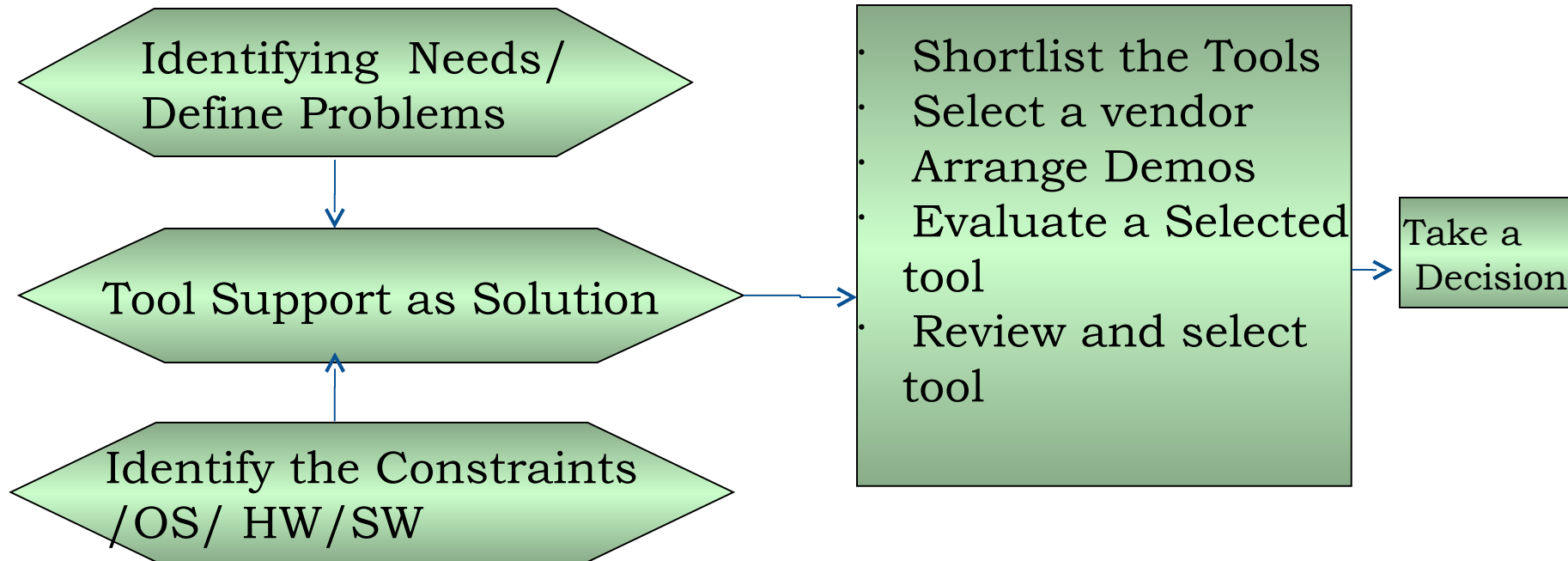
- **Benefits of using tools**

- ◆ Repetitive work is reduced
- ◆ Greater consistency and repeatability
- ◆ Objective assessment over-reliance on the tool
- ◆ Ease of access to information about tests or testing

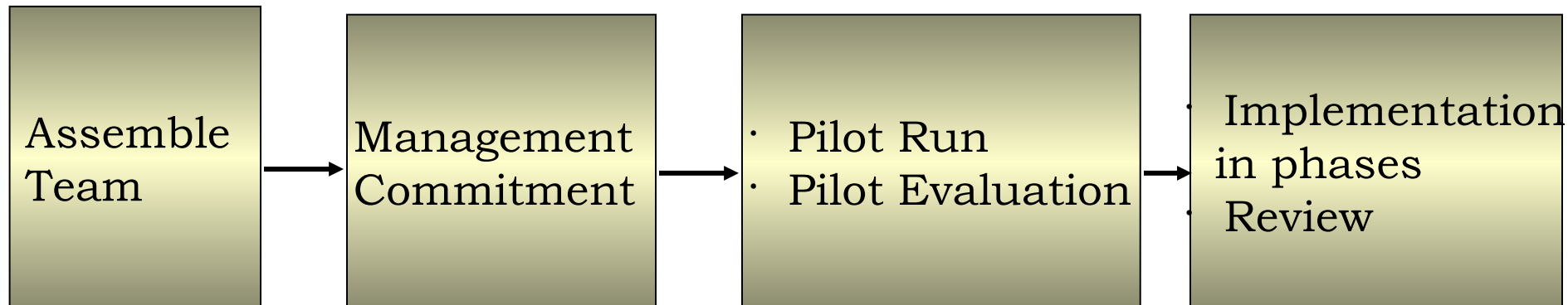
Risks of using tools

- ◆ Unrealistic expectations for the tool
- ◆ Underestimating the time, cost and effort for the initial introduction of a tool
- ◆ Underestimating the time and effort needed to achieve significant and continuing benefits from the tool
- ◆ Underestimating the effort required to maintain the test assets generated by the tool
- ◆ Over-reliance on the tool

Tool Selection Process



Tool implementation process



- **Objectives Of Pilot Process:**

- Understand the tool, its features and limitations in greater detail
- Check if the tool meets the evaluation criteria
- Check if the existing process need to change for the tool
- Creating process for using and maintaining the tool



THANK YOU