Software Testing Background

Mr .Sachin J Pukale

Lecture in Information Technology

Government Polytechnic Nagpur

Infamous Software Error Case

- The Y2K (Year 2000) Bug, circa 1974

- For nearly 50 years, the people who wrote computer code used a standard two-digit shorthand to indicate the year — substituting, say, “99” for “1999.”
- Early on, when computer memory was always in short supply, that practice made a lot of sense. But programmers continued using this shorthand even as memory capacity expanded.
- As a result, when some computers see “00,” they can’t tell whether that figure refers to 1900 or 2000.
- And if your PC gets the date wrong, your system could lose or delete email that it thinks is 100 years old; your contact-management and scheduling software could fail to list crucial appointments.
- Applications, such as Microsoft Word and Excel and Netscape Navigator, must also be Y2K-compliant

Infamous Software Error Case

- Bug in Space Code

➢ Project Mercury’s FORTRAN code had the following fault:

    DO I=1.10 instead of ... DO I=1,10

➢ The fault was discovered in an analysis of why the software did not seem to generate results that were sufficiently accurate.

➢ The erroneous 1.10 would cause the loop to be executed exactly once!

Software Bug

- Software Bug :

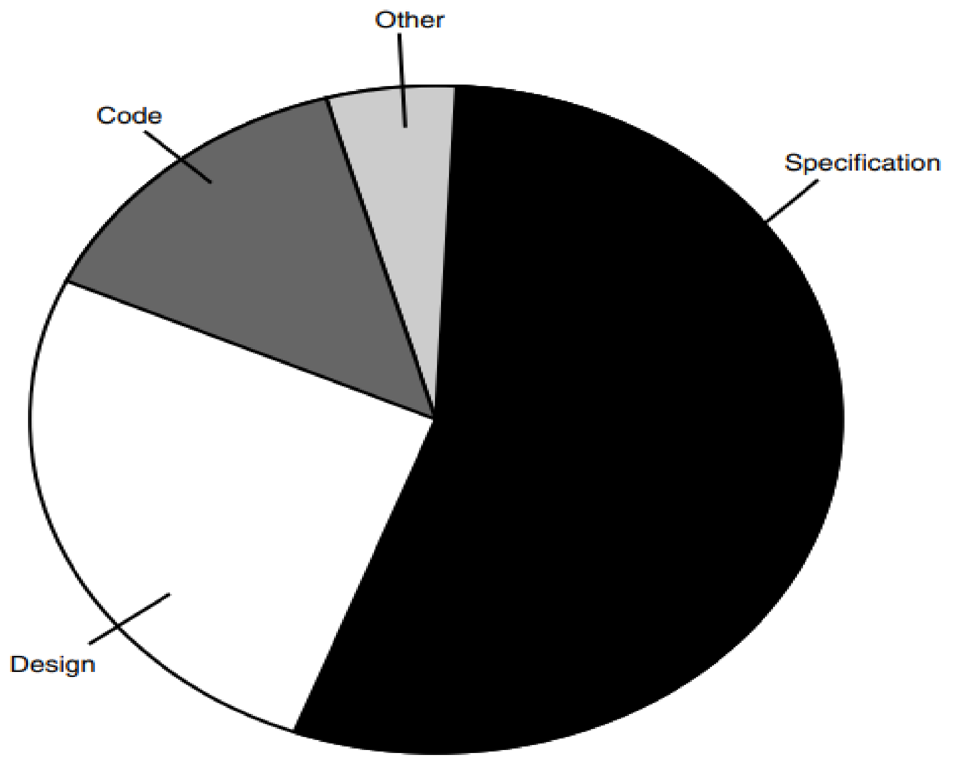
➢ A software bug is an error, flaw or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

➢ a software bug occurs when: one or more of the following five rules is true:

1. The software doesn’t do something that the product specification says it should do.
2. The software does something that the product specification says it shouldn’t do.
3. The software does something that the product specification doesn’t mention.
4. The software doesn’t do something that the product specification doesn’t mention but should.
5. The software is difficult to understand, hard to use, slow,

Why Do Bugs Occur

- Bugs are caused for numerous reasons, but the main cause can be traced to the specification.

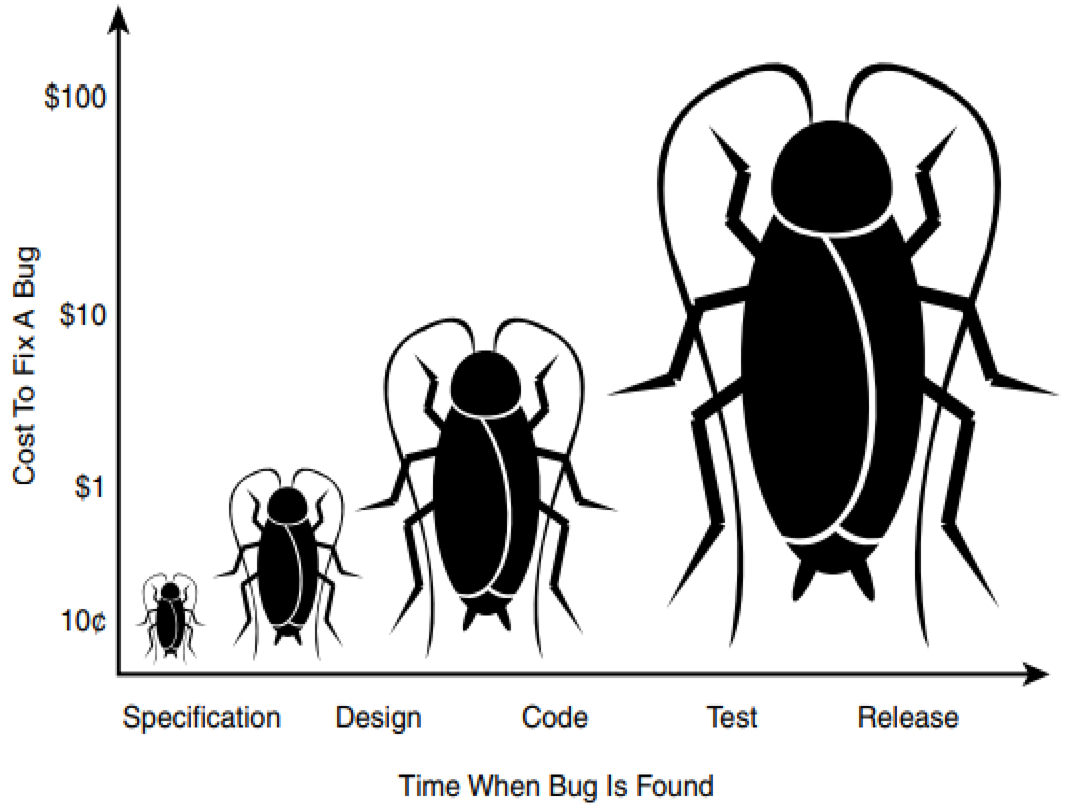


Specification (~= 55%)
Design (~= 25%)
Code (~= 15%)
Other (~= 5%)

There’s an old saying, “If you can’t say it, you can’t do it.” This applies perfectly to software development and testing.

Cost of Bugs

- The cost of fixing these bugs grows over time.



1. The cost to fix bugs increases dramatically over time.

2. The costs are logarithmic—that is, they increase tenfold as time increases.

3. A bug found and fixed during the early stages when the specification is being written might cost next to nothing, or 10 cents in our example.

4.The same bug, if not found until the software is coded and tested, might cost $1 to $10. If a customer finds it, the cost could easily top $100.

Relation between No of Bugs & Cost of fixing Bugs

The optimal amount of testing

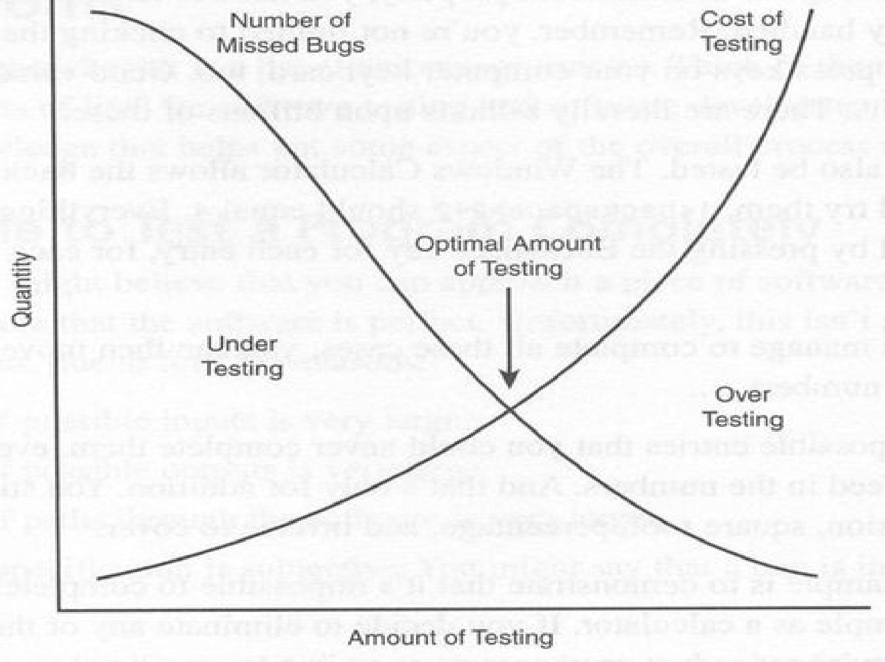


FIGURE 3.2
Every software project has an optimal test effort.

(source: "Software Testing", Ron Patton)

Exactly Does a Software Tester Do?

1. The goal of a software tester is to find bugs.
2. The goal of a software tester is to find bugs, and find them as early as possible.
3. The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

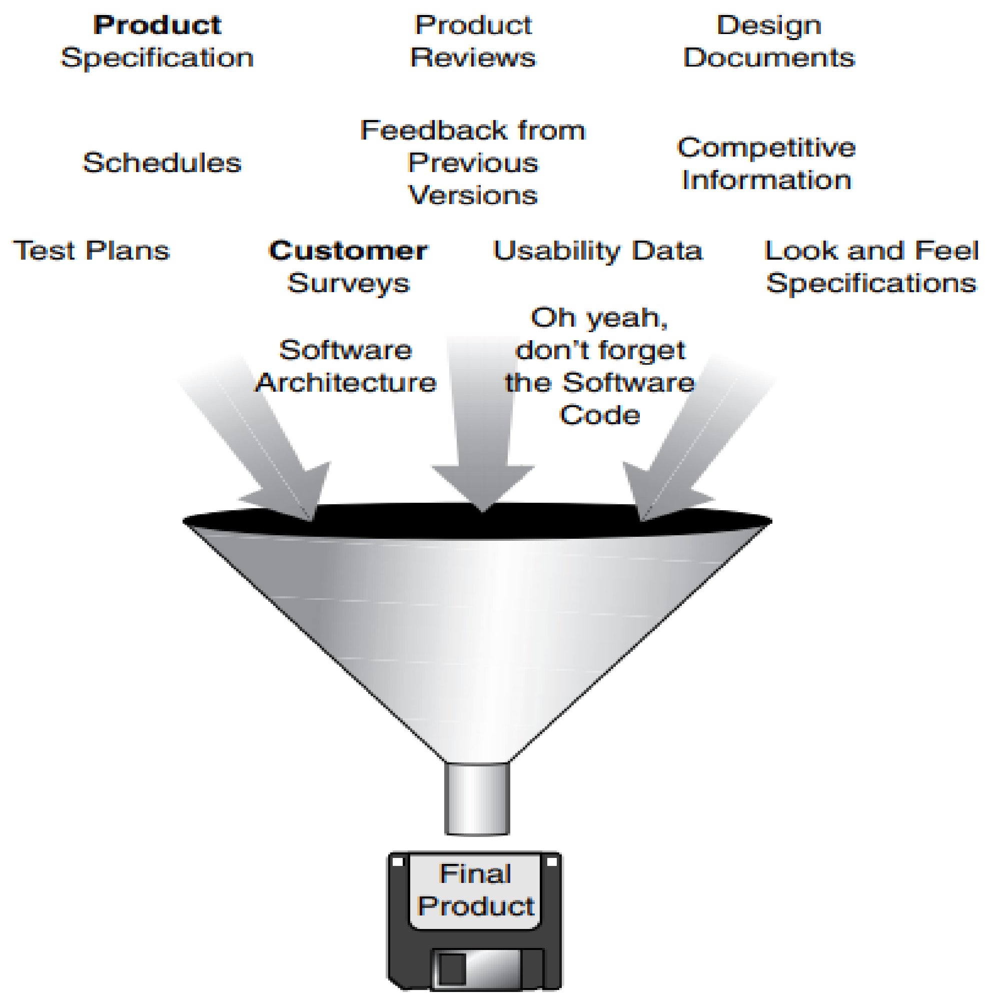
What Makes a Good Software Tester?
A fundamental trait of software testers is that they simply like to break things.

4. They are explorers.
5. They are trouble-shooters.
6. They are relentless. Software testers keep trying.
7. They are creative.
8. They are (mellowed) perfectionists.
9. They exercise good judgment.
10. They are tactful and diplomatic
11. They are persuasive
12. Programming Skills.
13. Good Communication Skills.
14. Knowing What to Prioritize.
15. Continuous Learning
16. Knowledge of Testing Techniques
17. Don’t Compromise On Quality
18. Be Open to Suggestions

Types of Bug

1. Logical Bugs/Functional Bugs
2. API Bug:
3. Browser Compatibility:
4. Application Crash
5. GUI Related Bugs:
6. Database Bugs:-
7. System Related bugs
8. Software Service Pack
9. Functionality Errors
10. Communication Errors
11. Missing command errors
12. Syntactic Error

What Effort Goes Into a Software Product



“A lot of hidden effort goes into a software product”

Components of software products

1. Customer Requirements: Information from customers is then studied, condensed, and interpreted to decide exactly what features the software product should have.
2. Specifications: The result of the customer requirements studies is really just raw data. It doesn't describe the proposed product, it just confirms whether it should (or shouldn't) be created and what features the customers want. The specifications take all this information plus any unstated but mandatory requirements and truly define what the product will be, what it will do, and how it will look.
3. Schedules: A key part of a software product is its schedule. As a project grows in size and complexity, with many pieces and many people contributing to the product, it becomes necessary to have some mechanism to track its progress.
4. Software Design Documents:

a. Architecture: A document that describes the overall design of the software

b. Data Flow Diagram: A formalized diagram that shows how data moves through a program.

c. State Transition Diagram

d. Flowchart

e. Commented Code

Verification and Validation

Verification

1. Verification is the process confirming that something—software—meets its specification.
2. Validation is the process confirming that it meets the user’s requirements.
3. Verification is the process of checking that a software achieves its goal without any bugs.
4. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.
5. Verification is static testing.
6. Verification means Are we building the product right?

Verification and Validation

Validation

1. Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements.
2. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.
3. Validation is the dynamic testing.
4. Validation means Are we building the right product?

Software quality & Reliability

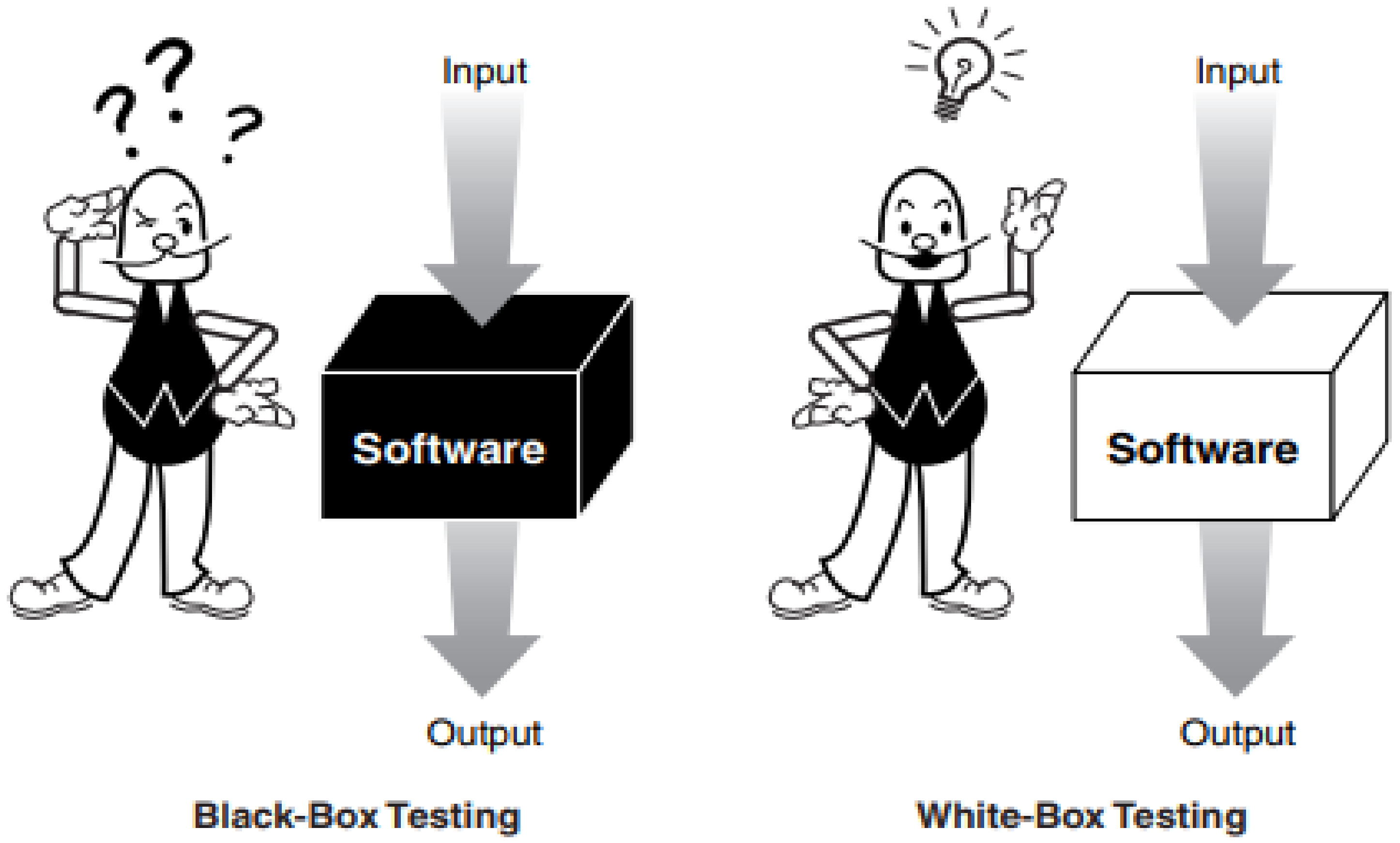
1. Software quality: Software quality is defined as a field of study and practice that describes the desirable attributes of software products.
2. There are two main approaches to software quality: defect management and quality attributes.

- ➢ Functional suitability
- ➢ Reliability: Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment
- ➢ Operability
- ➢ Performance efficiency
- ➢ Security
- ➢ Compatibility
- ➢ Maintainability
- ➢ Transferability

Quality assurance and Testing

1. The difference between quality assurance and testing is that quality assurance is about the activities designed to make sure the project is conforming to the expectations of the stakeholders, while test is a process to explore a system to find defects.
2. Testing is focused on system inspection and finding bugs, with a product orientation and corrective activity.
3. Testing's aim is to control the quality, while quality assurance is to assure the quality.
4. Quality Assurance (QA) is defined as an activity to ensure that an organization is providing the best possible product or service to customers.

Black box & White Box Testing



With black-box testing, the software tester doesn’t know the details of how the software works

Differences between Black Box Testing vs White Box Testing

- Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester.
- Ex: search something on google by using keywords

- White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
- Ex: by input to check and verify loops

Sr No	BLACK BOX TESTING	WHITE BOX TESTING
1	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2	It is mostly done by software testers.	It is mostly done by software developers.
3	No knowledge of implementation is needed.	Knowledge of implementation is required.
4	It can be referred as outer or external software testing.	It is the inner or the internal software testing.
5	It is functional test of the software.	It is structural test of the software.
6	This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
7	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
8	It is the behavior testing of the software.	It is the logic testing of the software.
9	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.

Static and Dynamic Testing

- Two other terms used to describe how software is tested are static testing and dynamic testing.
- Static testing refers to testing something that’s not running—examining and reviewing it.
- Dynamic testing is what you would normally think of as testing—running and using the software.

Sr .No	Static Testing	Dynamic Testing
1	Testing was done without executing the program	Testing is done by executing the program
2	This testing does the verification process	Dynamic testing does the validation process
3	Static testing is about prevention of defects	Dynamic testing is about finding and fixing the defects
4	Static testing gives an assessment of code and documentation	Dynamic testing gives bugs/bottlenecks in the software system.
5	Static testing involves a checklist and process to be followed	Dynamic testing involves test cases for execution
6	This testing can be performed before compilation	Dynamic testing is performed after compilation
7	Static testing covers the structural and statement coverage testing	Dynamic testing techniques are Boundary Value Analysis & Equivalence Partitioning.
8	Cost of finding defects and fixing is less	Cost of finding and fixing defects is high
9	Return on investment will be high as this process involved at an early stage	Return on investment will be high as this process involved at an early stage
10	More reviews comments are highly recommended for good quality	More defects are highly recommended for good quality.
11	Requires loads of meetings	Comparatively requires lesser meetings

Static Black-Box Testing: Testing the Specification

- Testing the specification is static black-box testing.
- The specification is a document, not an executing program, so it’s considered static.
- It’s also something that was created using data from many sources—usability studies, focus groups, marketing input, and so on.
- You don’t necessarily need to know how or why that information was obtained or the details of the process used to obtain it.
- You can test a specification with static black-box techniques no matter what the format of the specification. It can be a written or graphical document or a combination of both. You can even test an unwritten specification by questioning the people who are designing and writing the software

Performing a High-Level Review of the Specification

- Pretend to Be the Customer: The easiest thing for a tester to do when he first receives a specification for review is to pretend to be the customer. Do some research about who the customers will be.
- Research Existing Standards and Guidelines: The difference between standards and guidelines is a matter of degree. A standard is much more firm than a guideline. Standards should be strictly adhered to. Guidelines are optional but should be followed.
    1. **Corporate Terminology and Conventions**
    2. **Industry Requirements**
    3. **Government Standards.**
    4. **Graphical User Interface (GUI).**
    5. **Hardware and Networking Standards**

- **Review and Test Similar Software:** Some things to look for when reviewing competitive products include

1. **Scale.** Will your software be smaller or larger?
2. **Complexity.** Will your software be more or less complex?
3. **Testability.** Will you have the resources, time, and expertise to test software such as this?
4. **Quality/Reliability.** Is this software representative of the overall quality planned for your software? Will your software be more or less reliable?

Low-Level Specification Test Techniques

- **Specification Attributes Checklist:** A good, well-thought-out product specification, has eight important attributes

1. **Complete.** Is anything missing or forgotten? Is it thorough? Does it include everything necessary to make it stand alone?
2. **Accurate.** Is the proposed solution correct? Does it properly define the goal? Are there any errors?
3. **Precise, Unambiguous, and Clear.** Is the description exact and not vague? Is there a single interpretation? Is it easy to read and understandable?
4. **Consistent.** Is the description of the feature written so that it doesn’t conflict with itself or other items in the specification?
5. **Relevant.** Is the statement necessary to specify the feature? Is it extra information that should be left out? Is the feature traceable to an original customer need?
6. **Feasible.** Can the feature be implemented with the available personnel, tools, and resources within the specified budget and schedule?

7. Code-free. Does the specification stick with defining the product and not the underlying software design, architecture, and code?

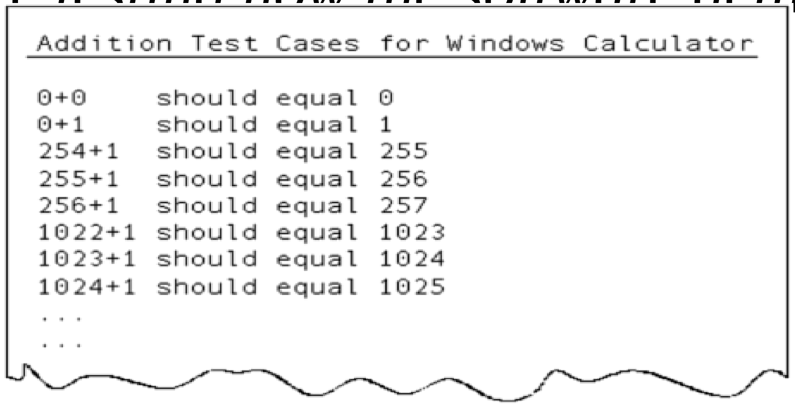
8. **Testable.** Can the feature be tested? Is enough information provided that a tester could create tests to verify its operation?

Specification Terminology Checklist

- **Always, Every, All, None, Never.** If you see words such as these that denote something as certain or absolute, make sure that it is, indeed, certain. Put on your tester’s hat and think of cases that violate them.
- **Certainly, Therefore, Clearly, Obviously, Evidently.** These words tend to persuade you into accepting something as a given. Don’t fall into the trap.
- **Some, Sometimes, Often, Usually, Ordinarily, Customarily, Most, Mostly.** These words are too vague. It’s impossible to test a feature that operates “sometimes.”
- **Etc., And So Forth, And So On, Such As.** Lists that finish with words such as these aren’t testable.
- **Good, Fast, Cheap, Efficient, Small, Stable.** These are unquantifiable terms. They aren’t testable.
- **Handled, Processed, Rejected, Skipped, Eliminated.** These terms can hide large amounts of functionality that need to be specified.
- **If...Then...(but missing Else).** Look for statements that have “If…Then” clauses but don’t have a matching “Else.” Ask yourself what will happen if the “if” doesn’t happen.

Dynamic Black-Box Testing: Testing the Software While Blindfolded

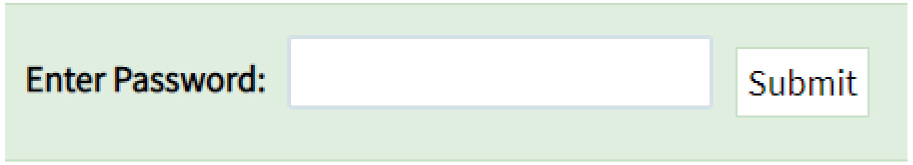
- *Testing software without having an insight into the details of underlying code is dynamic black-box testing. It's dynamic because the program is running—you're using it as a customer would.*
- *black-box because you're testing it without knowing exactly how it works—*
- *You're entering inputs, receiving outputs, and checking the results.*
- *Another name commonly used for dynamic black-box testing is behavioural testing because you're testing how the software actually behaves when it's used.*



```
Addition Test Cases for Windows Calculator

0+0     should equal 0
0+1     should equal 1
254+1   should equal 255
255+1   should equal 256
256+1   should equal 257
1022+1  should equal 1023
1023+1  should equal 1024
1024+1  should equal 1025
...
...
```

SAMPLE TEST CASES



Following password field accepts minimum 6 characters and maximum 10 characters

Test Scenario #	Test Scenario Description	Expected Outcome
1	Enter 0 to 5 characters in password field	System should not accept
2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Test-to-Pass and Test-to-Fail

- Test-to-pass-you really assure only that the software minimally works. You don’t push its capabilities.

1. You don’t see what you can do to break it.
2. Applying the simplest and most straightforward test cases.
3. *Use test-to-pass to reveal bugs before you test-to-fail.*

- Designing and running test cases with the sole purpose of breaking the software is called testing-to-fail or *error-forcing*.

1. Test Cases are Chosen to probe for common weaknesses in the software

Equivalence Partitioning

- Is type of black box testing technique
- Selecting test cases is the single most important task that software testers do and *equivalence partitioning*, sometimes called *equivalence classing*, is the means by which they do it.
- Equivalence partitioning is the process of methodically reducing the huge (infinite) set of possible test cases into a much smaller, but still equally effective, set.

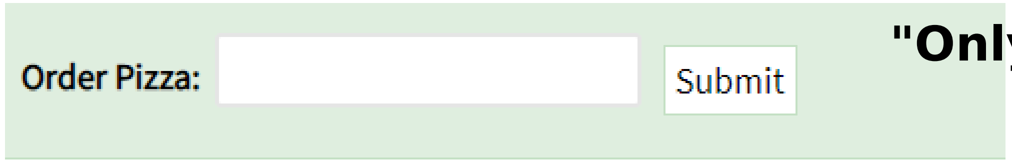
- Assume that the application accepts an integer in the range 100 to 999.

1. Valid Equivalence Class partition: 100 to 999 inclusive.
2. Non-valid Equivalence Class partitions: less than 100, more than 999, decimal numbers and alphabets/non-numeric characters.

Cont..

When looking for equivalence partitions, think about ways to group similar inputs,
similar outputs, and similar operation of the software. These groups are your equivalence
partitions.

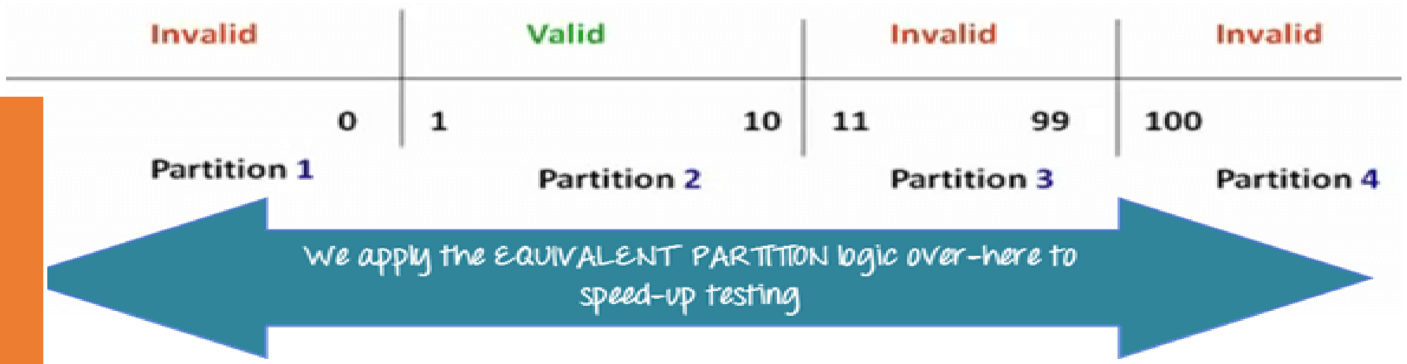
- Let's consider the behaviour of Order Pizza Text Box Below



"Only 10 Pizza can be ordered

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Nu



An equivalence class or equivalence partition is a set of test cases that tests the same thing or reveals the same bug.

Data Testing

- The simplest view of software is to divide its world into two parts:

1. the data (or its domain) and the program. The data is the keyboard input, mouse clicks, disk files, printouts, and so on.
2. The program is the executable flow, transitions, logic, and computations.

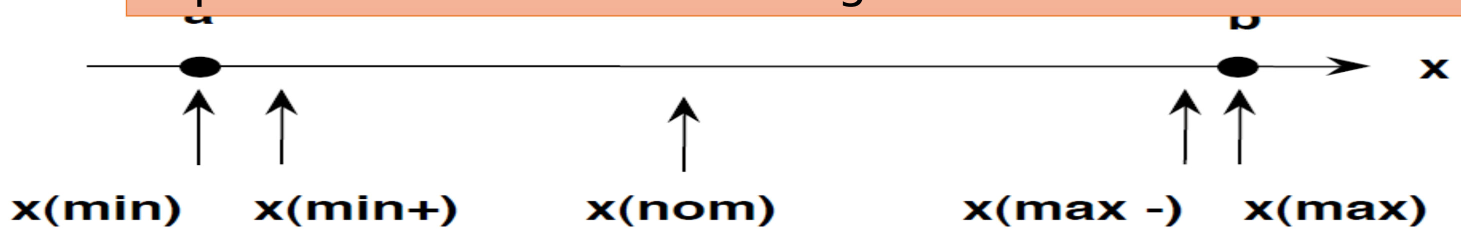
Examples of data would be

- The words you type into a word processor
- The numbers entered into a spread sheet
- The picture printed by your photo software

Boundary Value Analysis

- Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.
- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:
- Minimum
- Just above the minimum
- A nominal value
- Just below the maximum
- Maximum

In Boundary Testing, Equivalence Class Partitioning plays a good role Boundary Testing comes after the Equivalence Class Partitioning.

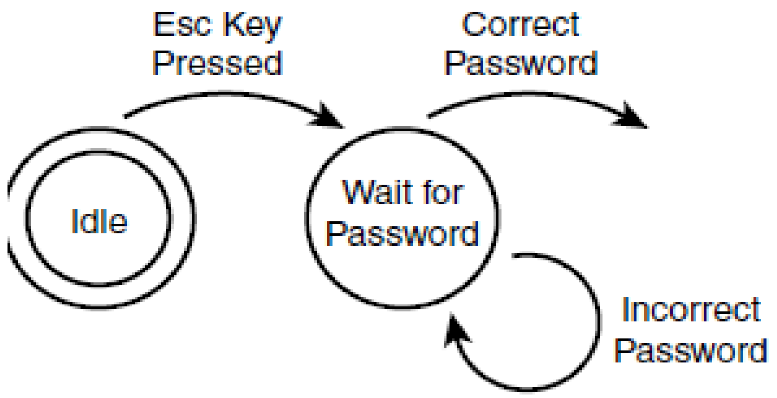


Why Equivalence & Boundary Analysis Testing

- This testing is used to reduce a very large number of test cases to manageable chunks.
- Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
- Appropriate for calculation-intensive applications with a large number of variables/inputs
- Boundary Analysis testing is used when practically it is impossible to test a large pool of test cases individually
- In Equivalence Partitioning, first, you divide a set of test condition into a partition that can be considered.
- In Boundary Value Analysis you then test boundaries between equivalence partitions
- Appropriate for calculation-intensive applications with variables that represent physical quantities

State Testing

- Software testing is to verify the program’s logic flow through its various states.
- A *software state* is a condition or mode that the software is currently in.
- A software tester must test a program’s states and the transitions between them.



1. **Each unique state that the software can be in.**

2.The input or condition that takes it from one state to the next

3. Set conditions and produced output when a state is entered or exited

Repetition, Stress, and Load

- *Repetition testing* involves doing the same operation over and over. This could be as simple as starting up and shutting down the program over and over.
- The main reason for doing repetition testing is to look for *memory leaks*.
- used a program that works fine when you first start it up, but then becomes slower and slower or starts to behave erratically over time, it's likely due to a memory leak bug.
- *Stress testing* is running the software under less-than-ideal conditions—low memory, low disk space, slow CPUs, slow modem etc. Your goal is to starve the software modems, and so on.
- *Load testing* is the opposite of stress testing. With stress testing, you starve the software; with

load testing, you feed it all that it can handle. Operate the software with the largest possible data files.

Other Black-Box Test Techniques

- **Behave Like a Dumb User (***inexperienced user* or *new user***)**

1. person who's unfamiliar with the software in front of your program and they'll do and try things that you never imagined.

- **Look for Bugs Where You've Already Found Them**

1. more bugs you find, the more bugs there are.

- **Follow Experience**

1. *Experience is the name every one gives to their mistakes.*
2. Experience and intuition can't be taught. They must be gained over time.

Static White-Box Testing

- *Static white-box testing* is the process of carefully and methodically reviewing the software design, architecture, or code for bugs without executing it. It's sometimes referred to as *structural analysis.*
- Perform static white-box testing is to find bugs early and to find bugs that would be difficult to uncover or isolate with dynamic black-box testing.
- A side benefit of performing static white-box testing is that it gives the team's black-box testers ideas for test cases to apply when they receive the software for testing.
- Too time-consuming, too costly, or not productive. (unfortunate thing about static white-box testing)

Formal Reviews

- A *formal review* is the process under which static white-box testing is performed. A formal review can range from a simple meeting between two programmers to a detailed, rigorous inspection of the code.
- It involves

1. **Identify Problems:** The goal of the review is to find problems with the software—not just items that are wrong, but missing items as well.
2. **Follow Rules.** A fixed set of rules should be followed. They may set the amount of code to be reviewed (usually a couple hundred lines), how much time will be spent (a couple hours), what can be commented on, and so on.
3. **Prepare.** Each participant is expected to prepare for and contribute to the review. Depending on the type of review, participants may have different roles.
4. **Write a Report.** The review group must produce a written report summarizing the results of the review and make that report available to the rest of the product development team.

Advantages:

1. great way to find **bugs early.**
2. **Communications.** Information not contained in the formal report is communicated.
3. **Quality.** A programmer’s code that is being gone over in detail, function by function, line by line, often results in the programmer being more careful.
4. **Team Building .** If a review is run properly, it can be a good place for testers and programmers to build respect for each other’s skills and to better understand each other’s jobs and job needs.
5. **Solutions.** Solutions may be found for tough problems

Peer Reviews

- The easiest way to get team members together and doing their first formal reviews of the software is through *peer reviews.*
- The least formal method. Sometimes called *buddy reviews.*
- Peer reviews are often held with just the programmer who wrote the code and one or two other programmers or testers acting as reviewers.
- That small group simply reviews the code together and looks for problems and oversights.
- Peer reviews are informal

Walkthroughs

- Walkthroughs are the next step up in formality from peer reviews.
- In a walkthrough, the programmer who wrote the code formally presents (walks through) it to a small group of five or so other programmers and testers.
- The reviewers should receive copies of the software in advance of the review so they can examine it and write comments and questions that they want to ask at the review.
- Having at least one senior programmer as a reviewer is very important.

Cont

- Walkthrough are informal meetings but with purpose.
- it’s much more important for them to prepare for the review and to follow the rules.
- It’s also very important that after the review the presenter write a report telling what was found and how he plans to address any bugs discovered.

Inspections

- Inspections are the most formal type of reviews.
- They are highly structured and require training for each participant. Inspections are different from peer reviews and walkthroughs in that the person who presents the code, the presenter or reader, isn’t the original programmer.
- The other participants are called inspectors.
- Some inspectors are also assigned tasks such as moderator and recorder to assure that the rules are followed and that the review is run effectively

Cont

- Inspections have proven to be very effective in finding

1. bugs in any software deliverable, especially design documents and code, and are gaining popularity as companies and product development teams discover their benefits.

- An inspection team consists of three to eight members who plays roles of moderator, author, reader, recorder and inspector.
- Inspection activity follows a specified process and participants play well-defined roles.

Stages in the inspections process :

- **Planning :** Inspection is planned by moderator.
- **Overview meeting :** Author describes background of work product.
- **Preparation :** Each inspector examines work product to identify possible defects.
- **Inspection meeting :** During this meeting, reader reads through work product, part by part and inspectors points out the defects for every part.
- **Rework :** Author makes changes to work product according to action plans from the inspection meeting.
- **Follow-up :** Changes made by author are checked to make sure that everything is correct.

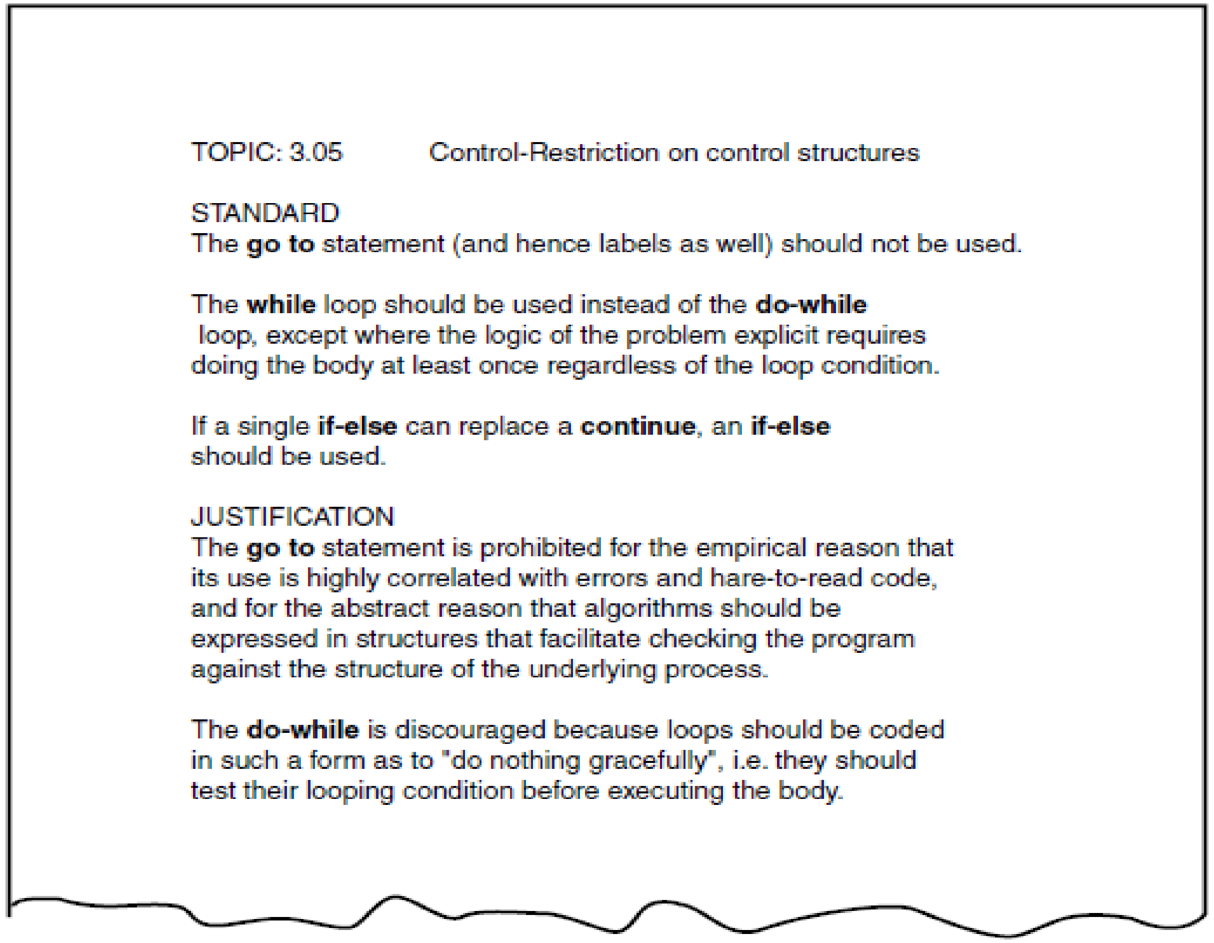
S.NO.	INSPECTION	WALKTHROUGH
1.	It is formal.	It is informal.
2.	Initiated by project team.	Initiated by author.
3.	A group of relevant persons from different departments participate in the inspection.	Usually team members of the same project take participation in the walkthrough. Author himself acts walkthrough leader.
4.	Checklist is used to find faults.	No checklist is used in the walkthrough.
5.	Inspection processes includes overview, preparation, inspection, and rework and follow up.	Walkthrough process includes overview, little or no preparation, little or no preparation examination (actual walkthrough meeting), and rework and follow up.
6.	Formalized procedure in each step.	No formalized procedure in the steps.
7.	Inspection takes longer time as list of items in checklist is tracked to completion.	Shorter time is spent on walkthrough as there is no formal checklist used to evaluate program.
8.	Planned meeting with the fixed roles assigned to all the members involved.	Unplanned
9.	Reader reads product code. Everyone inspects it and comes up with detects.	Author reads product code and his teammate comes up with the defects or suggestions.
10.	Recorder records the defects.	Author make a note of defects and suggestions offered by teammate.
11.	Moderator has a role that moderator making sure that the discussions proceed on the productive lines.	Informal, so there is no moderator.

Coding Standards and Guidelines

- There are also problems where the code may operate properly but may not be written to meet a specific *standard* or *guideline*.

1. **Reliability.** It’s been shown that code written to a specific standard or guideline is more reliable and bug-free than code that isn’t.
2. **Readability/Maintainability.** Code that follows set standards and guidelines is easier to read, understand, and maintain.
3. **Portability.** Code often has to run on different hardware or be compiled with different compilers.

Cont….Example



TOPIC: 3.05 Control-Restriction on control structures

STANDARD
The **go to** statement (and hence labels as well) should not be used.

The **while** loop should be used instead of the **do-while** loop, except where the logic of the problem explicit requires doing the body at least once regardless of the loop condition.

If a single **if-else** can replace a **continue**, an **if-else** should be used.

JUSTIFICATION
The **go to** statement is prohibited for the empirical reason that its use is highly correlated with errors and hare-to-read code, and for the abstract reason that algorithms should be expressed in structures that facilitate checking the program against the structure of the underlying process.

The **do-while** is discouraged because loops should be coded in such a form as to "do nothing gracefully", i.e. they should test their looping condition before executing the body.

main parts of standard

- The standard has four main parts:
- *Title* describes what topic the standard covers.
- *Standard (or guideline)* describes the standard or guideline explaining exactly what’s allowed and not allowed.
- *Justification* gives the reasoning behind the standard so that the programmer understands why it’s good programming practice.
- *Example* shows simple programming samples of how to use the standard. This isn’t always necessary.

Obtaining Standards

- National and international standards for most computer languages and information technology can be obtained from:

1. American National Standards Institute (ANSI), www.ansi.org
2. International Engineering Consortium (IEC), www.iec.org
3. International Organization for Standardization (ISO), www.iso.ch
4. National Committee for Information Technology Standards (NCITS), www.ncits.org

Guidelines

- Guidelines and best practices available from professional organizations such as

1. Association for Computing Machinery (ACM), www.acm.org
2. Institute of Electrical and Electronics Engineers, Inc (IEEE), www.ieee.org

Generic Code Review Checklist

- **Data Reference Errors:** bugs caused by using a variable, constant, array, string, or record that hasn't been properly initialized for how it's being used and referenced.

1 Example: Is an uninitialized variable referenced? Looking for omissions is just as important as looking for errors.

2. Is a variable used where a constant would actually work better—for example, when checking the boundary of an array?

3. Are array and string subscripts integer values and are they always within the bounds of the array's or string's dimension?

Cont.

- **Data Declaration Errors:** Data declaration bugs are caused by improperly declaring or using variables or constants.
- Examples:

1. Are all the variables assigned the correct length, type, and storage class.
2. If a variable is initialized at the same time as it’s declared, is it properly initialized and consistent with its type?
3. Are any variables declared that are never referenced or are referenced only once?

Cont..

- **Computation Errors:** Computational or calculation errors are essentially bad math. The calculations don’t result in the expected result.
- Examples:

1. Do any calculations that use variables have different data types, such as adding an integer to a floating-point number.
2. Do any calculations that use variables have the same data type but are different lengths—adding a byte to a word, for example?

Cont..

- **Comparison Errors:** Less than, greater than, equal, not equal, true, false. Comparison and decision errors are very susceptible to boundary condition problems.
- Examples:

1. Are there comparisons between fractional or floating-point values?
2. Does each Boolean expression state what it should state?
3. Are the operands of a Boolean operator Boolean?

Cont..

- **Control Flow Errors:** Control flow errors are the result of loops and other control constructs in the language not behaving as expected. They are usually caused, directly or indirectly, by computational or comparison errors.
- Examples:

1. If the language contains statement groups such as begin...end and do...while, are the ends explicit and do they match their appropriate groups?
2. Will the program, module, subroutine, or loop eventually terminate?

Cont..

- **Subroutine Parameter Errors:** Subroutine parameter errors are due to incorrect passing of data to and from software subroutines.
- Examples:
- 1. Do the types and sizes of parameters received by a subroutine match those sent by the calling code? Is the order correct?
- 2. If constants are ever passed as arguments, are they accidentally changed in the subroutine?

Cont..

- **Input / Output Errors:** These errors include anything related to reading from a file, accepting input from a keyboard or mouse, and writing to an output device such as a printer or screen.
- Examples:
- 1. Does the software strictly adhere to the specified format of the data being read or written by the external device?
- 2. Have all error messages been checked for correctness, appropriateness, grammar, and spelling?

Cont..

- **Other Checks:** list defines a few items that didn’t fit well in the other categories.

1. Will the software work with languages other than English(ASCII)
2. If the software is intended to be portable to other compilers and CPUs, have allowances been made for this?
3. Has compatibility been considered so that the software will operate with different amounts of available memory, hardware ,peripherals such as printers and modems?
4. Does compilation of the program produce any “warning” or “informational” messages?