Software Testing & Audit

Unit-1

Jayash Kumar Sharma
Department of Computer Science & Engineering
Anand Engineering College
Mob: +91-9639325975
Email: Jayash.sharma@gmail.com

Overview

- What is software testing?
- Why do we need to test software?
- Can we live without testing?
- How do we handle software bugs reported by the customers?
- Who should work hard to avoid frustrations of software failures?

- Software organizations are regularly getting failure reports of their products and this number is increasing day by day.

Overview

- The developers normally ask: how did these bugs escape unnoticed?
- Fact: software developers are experiencing failures in testing their coded programs and such situations are becoming threats to modern automated civilization.

- We cannot imagine a day without using cell phones, logging on to the internet, sending e-mails, watching television and so on.

- All these activities are dependent on software, and software is not reliable. The world has seen many software failures and some were even fatal to human life.

Overview of Software Evolution

- Software Crisis

Software Development Life Cycle

1. Waterfall Model
2. Prototype Model
3. Spiral Model
4. Incremental Model
5. Rapid Application Development
6. V-Model
7. Agile Methodology

Some Software Failures

The Explosion of the Ariane 5 Rocket

- Unmanned Rocket launched on June 4, 1996 by European Space Agency.
- Exploded only after 40 seconds of its take off.
- The design and development took ten long years with a cost of $7 billion.
- The failure of the Ariane 5 was caused by the complete loss of guidance and altitude information, 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off).

Some Software Failures

The Explosion of the Ariane 5 Rocket

- A software fault in the inertial reference system was identified as a reason for the explosion
- Ariane 5 development programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system
- This loss of information was due to specification and design errors in the software.

Some Software Failures

The Explosion of the Ariane 5 Rocket

- The inertial reference system of the rocket had tried to convert 64 bit floating point number of horizontal velocity to a 16 bit signed integer. However, the number was greater than 32,767 (beyond the permissible limit of 16 bit machine) and the conversion process failed.

- The navigation system of Ariane 4 was used in Ariane 5 without proper testing and analysis.

Some Software Failures

Y2K Problem

- The Y2K problem was the most critical problem of the last century.
- Significant sums of money were spent by software companies to get rid of this problem.
- It was simply the case of using two digits for the year instead of four digits. For instance, 1965 was considered as 65.

Some Software Failures

Y2K Problem

- The developers could not imagine the problem of year 2000. What would happen on January 1, 2000? The last two digits i.e. 00 may belong to any century like 1800, 1900, 2000, 2100, etc.

- Most of the software was re-tested and modified or discarded, depending on the situation.

Some Software Failures

The USA Star-Wars Program

- 'Patriot missile' was the result of the USA 'Star Wars' program.
- This missile was used for the first time in the Gulf war against the Scud missile of Iraq.
- Surprisingly, 'Patriot missiles' failed many times to hit the targeted Scud missile.

Some Software Failures

The USA Star-Wars Program

- One of the failures killed 28 American soldiers in Dhahran, Saudi Arabia.

- The cause of the failure was a slight timing error in the system's clock after 14 hours of its operation. Hence, the tracking system was not accurate after 14 hours of operations and at the time of the Dhahran attack, the system had already operated for more than 100 hours.

Some Software Failures

Failure of London Ambulance System

- The software controlling the ambulance dispatch system of London collapsed on October 26-27, 1992 and also on November 4, 1992 due to software failures. The system was introduced on October 26, 1992.

- The London Ambulance Service was a challenging task that used to cover an area of 600 square miles and handled 1500 emergency calls per day.

- Due to such a failure, there was a partial or no ambulance cover for many hours.

Some Software Failures

Failure of London Ambulance System

- The position of the vehicles was incorrectly recorded and multiple vehicles were sent to the same location. Everywhere people were searching for an ambulance and nobody knew the reason for non-arrival of ambulances at the desired sites.

- The repair cost was estimated to be £9m, but it is believed that twenty lives could have been saved if this failure had not occurred.

Some Software Failures

Failure of London Ambulance System

- The enquiry committee clearly pointed out the administrative negligence and over-reliance on ‘cozy assurances’ of the software company.

- The administration was allowed to use this system without proper alternative systems in case of any failure.

- When the system went live, it could not cope with the volume of calls and broke under the strain. The transition to a back-up computer system had not been properly rehearsed and also failed.”

Some Software Failures

USS Yorktown Incident

- A guided missile cruiser was in the water for several hours due to the software failure in 1998.
- A user wrongly gave a zero value as an input which caused a division by zero error. (Input validation was not done)
- This fault further failed the propulsion system of the ship and it did not move in the water for many hours.

Some Software Failures

Experience of Windows XP

- Microsoft released Windows XP on October 25, 2001.
- Same day, the company posted 18 megabyte of patches on its website for bug fixes, compatibility updates, and enhancements. Two patches fixed important security holes. Or rather, one of them did; the other patch did not work.
- Microsoft advised (still advises) users to back up critical files before installing patches."

Some Software Failures

Conclusion

- Automobile engineers give their views about cars, they do not say that the quality of today's cars is not better than the cars produced in the last decade. Similarly aeronautical engineers do not say that Boeing or Airbus makes poor quality planes as compared to the planes manufactured in the previous decade.

- Everyone feels that things are improving day by day. But software seems different.

Some Software Failures

Conclusion

- Most of the software developers are confused about the quality of their software products.

- If they are asked about the quality of software being produced by companies, they generally say, “It is getting worse day by day.”
-

Some Software Failures

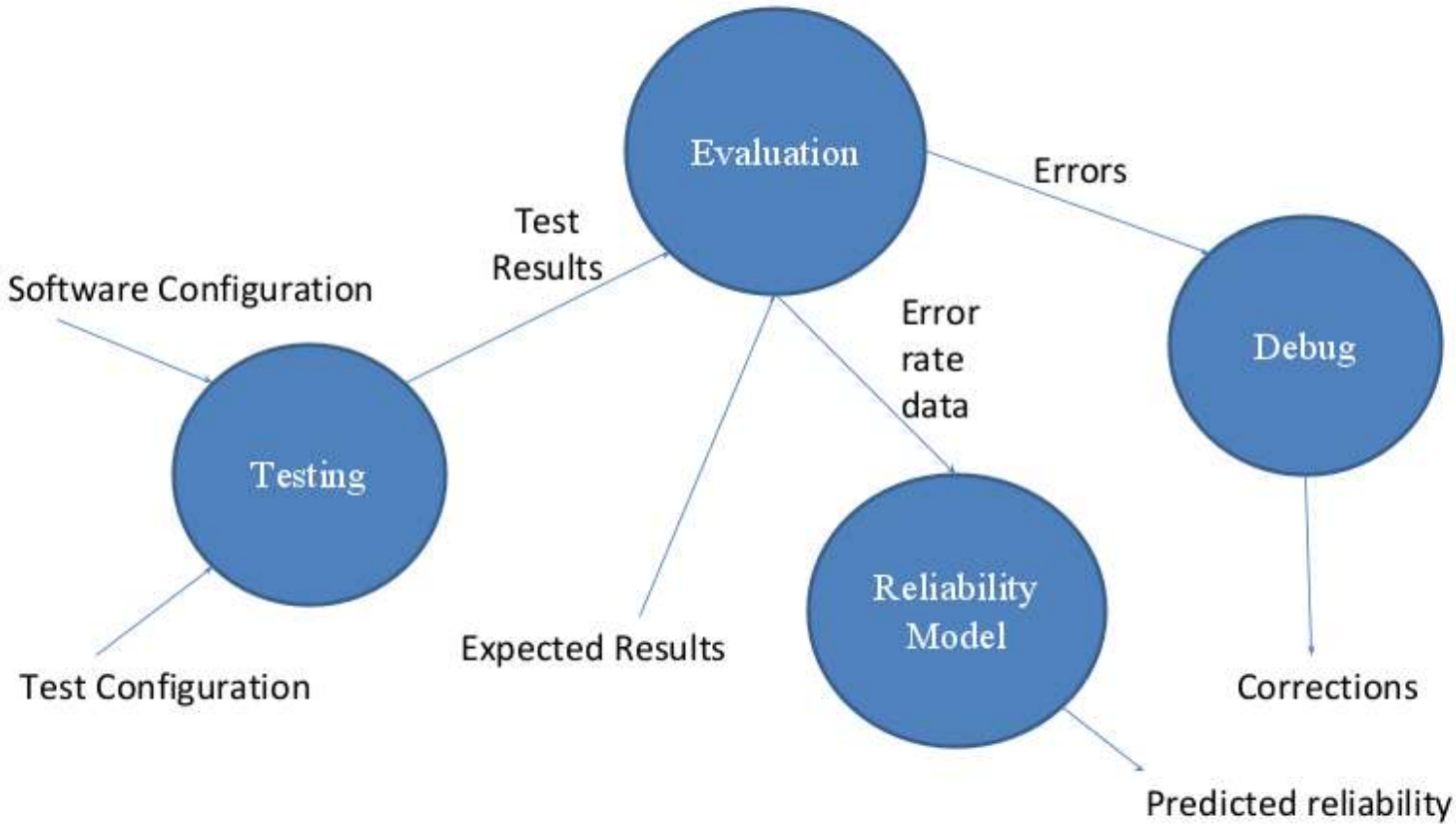
Who is Responsible?

1. Software Companies that rush products to market without adequately testing them.
2. Software Developers who do not understand the importance of detecting and removing faults before customers experience them as failures.
3. Legal System that has given the software developers a free pass on error-related damages.
4. Universities that stress more on software development than testing.

Testing Process

- Testing is an important aspect of the software development life cycle.
- It is the process of testing / checking the newly developed software, prior to its actual use.
- The program is executed with desired input(s) and the output(s) is/are observed accordingly.
- The observed output(s) is/are compared with expected output(s). If both are same, then the program is said to be correct as per specifications, otherwise there is something wrong somewhere in the program.

Testing Process



Testing Process

- Testing is a very expensive process.
- It may consume one-third to one-half of the cost of a typical development project.
- It is largely a systematic process but partly intuitive too.
- Hence, good testing process entails much more than just executing a program a few times to see its correctness.

Testing Process - Example

A good testing process entails much more than just executing a program a few times to see its correctness.

Let's consider a program termed as 'Minimum' that reads a set of integers and prints the smallest integer.

```
LINE NUMBER    /*SOURCE CODE*/
               #include<stdio.h>
               #include<limits.h>
               #include<conio.h>
1.             void Minimum();
2.             void main()
3.             {
4.                     Minimum();
5.             }
```

Testing Process - Example

```
6.  void Minimum()
7.  {
8.          int array[100];
9.          int Number;
10.         int i;
11.         int tmpData;
12.         int Minimum=INT_MAX;
13.         clrscr();
14.         "printf("Enter the size of the array:");
15.         scanf("%d",&Number);
16.         for(i=0;i<Number;i++) {
17.                 printf("Enter A[%d]=",i+1);
18.                 scanf("%d",&tmpData);
19.                 tmpData=(tmpData<0)?-tmpData:tmpData;
20.                 array[i]=tmpData;
21.         }
22.         i=1;
```

Testing Process - Example

```
23.                     while(i<Number-1) {
24.                             if(Minimum>array[i])
25.                             {
26.                                     Minimum=array[i];
27.                             }
28.                             i++;
29.                     }
30.                     printf("Minimum = %d\n", Minimum);
31.                     getch();
32.             }
```

Testing Process - Example

We may execute this program with a number of inputs and compare the expected output with the observed output.

Test Case	Inputs		Expected Output	Observed Output	Match?
	Size	Set of Integers			
1.	5	6, 9, 2, 16, 19	2	2	Yes
2.	7	96, 11, 32, 9, 39, 99, 91	9	9	Yes
3.	7	31, 36, 42, 16, 65, 76, 81	16	16	Yes
4.	6	28, 21, 36, 31, 30, 38	21	21	Yes
5.	6	106, 109, 88, 111, 114, 116	88	88	Yes
6.	6	61, 69, 99, 31, 21, 69	21	21	Yes
7.	4	6, 2, 9, 5	2	2	Yes
8.	4	99, 21, 7, 49	7	7	Yes

Testing Process - Example

- Given 8 sets of inputs are sufficient for this program. In all these test cases, the observed output is the same as the expected output.

- We may also design similar test cases to show that the observed output is matched with the expected output.

- Observation / Possible definitions of testing :

    1. Testing is the process of demonstrating that errors are not present.
    2. The purpose of testing is to show that a program performs its intended functions correctly.
    3. Testing is the process of establishing confidence that a program does what it is supposed to do.

Testing Process - Example

- All three definitions / observations are not correct.
- When testing a program, we want to establish confidence about the correctness of the program.
- Hence, our objective should not be to show that the program works as per specifications. But, we should do testing with the assumption that there are faults.
- Therefore; "Testing is the process of executing a program with the intent of finding faults."

Testing Process - Example

- Objective-1 : To show that a program has no errors > Existing test cases will work.
- Objective-2 : To show that a program has errors, we must select those test cases which have a *higher probability of finding errors*.
- Focus must be on weak and critical portions of the program to find more errors.
- This type of testing will be more useful and meaningful.

Testing Process - Example

- Some typical and critical situations in the program:

1. A very short list (of inputs) with the size of 1, 2, or 3 elements.
2. An empty list i.e. of size 0.
3. A list where the minimum element is the first or last element.
4. A list where the minimum element is negative.
5. A list where all elements are negative.
6. A list where some elements are real numbers.
7. A list where some elements are alphabetic characters.
8. A list with duplicate elements.
9. A list where one element has a value greater than the maximum permissible limit of an integer.

Testing Process - Example

S. No.		Inputs: Size	Inputs: Set of Integers	Expected Output	Observed Output	Match?
Case 1						
A very short list with size 1, 2 or 3	A	1	90	90	2147483647	**No**
	B	2	12, 10	10	2147483647	**No**
	C	2	10, 12	10	2147483647	**No**
	D	3	12, 14, 36	12	14	**No**
	E	3	36, 14, 12	12	14	**No**
	F	3	14, 12, 36	12	12	**Yes**
Case 2						
An empty list, i.e. of size 0	A	0	–	Error message	2147483647	**No**
Case 3						
A list where the minimum element is the first or last element	A	5	10, 23, 34, 81, 97	10	23	**No**
	B	5	97, 81, 34, 23, 10	10	23	**No**

Testing Process - Example

Case 4						
A list where the minimum element is negative	A	4	10, –2, 5, 23	–2	2	**No**
	B	4	5, –25, 20, 36	–25	20	**No**
Case 5						
A list where all elements are negative	A	5	–23, –31, –45, –56, –78	–78	31	**No**
	B	5	–6, –203, –56, –78, –2	–203	56	**No**
Case 6						
A list where some elements are real numbers	A	5	12, 34.56, 6.9, 62.14, 19	6.9	34 (The program does not take values for index 3,4 and 5)	**No**
	B	5.4	2, 3, 5, 6, 9	2	858993460 (The program does not take any array value)	**No**

Testing Process - Example

Case 7						
A list where some elements are characters	A	5	23, 2l, 26, 6, 9	6	2 (The program does not take any other index value for 3, 4 and 5)	**No**
	B	1l	2, 3, 4, 9, 6, 5, 11, 12, 14, 21, 22	2	2147483647 (Program does not take any other index value)	**No**
Case 8						
A list with dupli-	A	5	3, 4, 6, 9, 6	3	4	**No**
cate elements	B	5	13, 6, 6, 9, 15	6	6	**Yes**
Case 9						
A list where one element has a value greater than the maximum permissible limit of an integer	A	5	530, 4294967297, 23, 46, 59	23	1	**No**

Testing Process - Example

Conclusion:

- Goal is to find critical situations of any program.
- Test cases shall be designed for every critical situation in order to make the program fail in such situations.
- So, the most appropriate definition is “Testing is the process of executing a program with the intent of finding faults.”
- Testing never shows the absence of faults, but it shows that the faults are present in the program.

Assignment-1

Identity cause of each mismatch between expected output and actual output.

Testing Process – Example - Reasons

S. No.	Possible Reasons
Case 1	
A very short list with size 1, 2 or 3	While finding the minimum, the base value of the index and/or end value of the index of the usable array has not been handled properly (see line numbers 22 and 23).
Case 2	
An empty list i.e. of size 0	The program proceeds without checking the size of the array (see line numbers 15 and 16).
Case 3	
A list where the minimum element is the first or last element	Same as for Case 1.
Case 4	
A list where the minimum element is negative	The program converts all negative integers into positive integers (see line number 19).
Case 5	
A list where all elements are negative	Same as for Case 4.

Testing Process – Example - Reasons

S. No.	Possible Reasons
Case 6 A list where some elements are real numbers	The program uses scanf() function to read the values. The scanf() has unpredictable behaviour for inputs not according to the specified format. (See line numbers 15 and 18).
Case 7 A list where some elements are alphabetic characters	Same as for Case 6.
Case 8 A list with duplicate elements	(a) Same as for Case 1. (b) We are getting the correct result because the minimum value is in the middle of the list and all values are positive.
Case 9 A list with one value greater than the maximum permissible limit of an integer	This is a hardware dependent problem. This is the case of the overflow of maximum permissible value of the integer. In this example, 32 bits integers are used.

Question

(Write test cases for both objectives and identify causes of output mismatch)

```c
/* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

for (c = 0 ; c < ( n - 1 ); c++)
{
  for (d = 0 ; d < n - c - 1; d++)
  {
    if (array[d] > array[d+1]) /* For decreasing order use < */
    {
      swap       = array[d];
      array[d]   = array[d+1];
      array[d+1] = swap;
    }
  }
}

printf("Sorted list in ascending order:\n");

for ( c = 0 ; c < n ; c++ )
   printf("%d\n", array[c]);

return 0;
```

Why Should We Test?

- Software testing > Expensive & critical activity > Essential Activity

    1. How much testing is required?
    2. Do we have methods to measure it?
    3. Do we have techniques to quantify it?

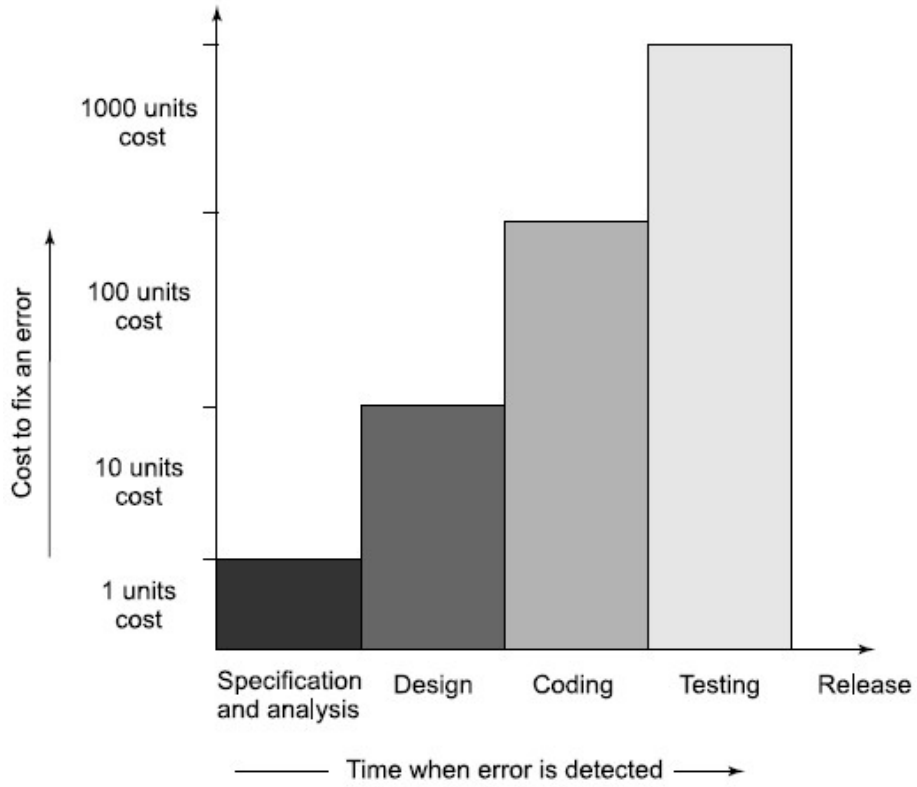
- The answer is not easy. All projects are different in nature and functionalities.

- It is a unique area with altogether different problems.

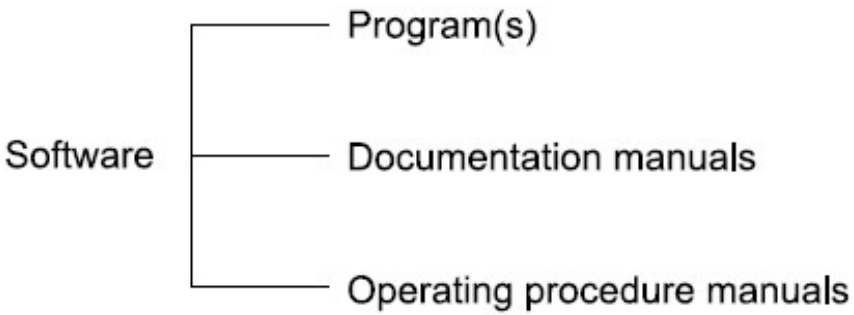
When Should We Test?

- When to release the software is a very important decision.
- Economics plays an important role.
- We shall try to find more errors in the early phases of software development.
- The cost of removal of such errors will be very reasonable as compared to those errors which we may find in the later phases of software development.

Why Should We Test?



Terminologies – Program & Software



<u>According to Software Engineering</u>

- Source Code
- Data Structure
- Documentation

Software = Program(s) + Documentation manuals + Operations procedure manuals

Terminologies – Program & Software

- The software is the superset of the program(s).
- Software consists of one or many program(s), documentation manuals and operating procedure manuals.
- The program is a combination of source code and object code.

Terminologies – Verification & Validation

- Verification is related to static testing which is performed manually. We only inspect and review the document.

- Validation is dynamic in nature and requires the execution of the program.

Terminologies – Verification & Validation

Verification:

- The process of evaluating the system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

- Verification activities are applied from the early phases of the software development and check / review the documents generated after the completion of each phase.

Are we building the product right?

Terminologies – Verification & Validation

Verification:

- It is the process of reviewing the requirement document, design document, source code and other related documents of the project.

- This is manual testing and involves only looking at the documents in order to ensure what comes out is what we expected to get.

Terminologies – Verification & Validation

Validation:

- The process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements.

- It is dynamic testing and requires the actual execution of the program.

- Here, we experience failures and identify the causes of such failures.

Are we building the right product?

Terminologies – Verification & Validation

- Testing includes both verification and validation.

Testing = Verification + Validation

- Both are essential and **complementary** activities of software testing.

- If effective verification is carried out, it may minimize the need of validation and more number of errors may be detected in the early phases of the software development.

- Unfortunately, testing is primarily validation oriented.

Terminologies – Fault, Error, Bug

- All terms are used interchangeably.
- Error, mistake and defect are synonyms in software testing terminology.
- When we make an error during coding, we call this a '**bug**'.
- Hence, error / mistake / defect in coding is called a **bug**.

Terminologies – Fault, Error, Bug

- A fault is the representation of an error where representation is the mode of expression such as data flow diagrams, ER diagrams, source code, use cases, etc.

- If fault is in the source code, we call it a **bug**.

- A **failure** is the result of execution of a fault and is dynamic in nature.

- When the expected output does not match with the observed output, we experience a failure.

Terminologies – Fault, Error, Bug

- The program must be executed for a failure to occur.

- A fault may lead to many failures depending on the inputs to the program.

Terminologies – Test, Test Case & Test Suite

- Test and test case terms are synonyms.
- A test case consists of inputs given to the program and its expected outputs.
- Inputs may also contain precondition(s) (circumstances that hold prior to test case execution), if any, and actual inputs identified by some testing methods.
- Expected output may contain post-condition(s)(circumstances after the execution of a test case), if any, and outputs which may come as a result when selected inputs are given to the software.

Terminologies – Test, Test Case & Test Suite

- Every test case will have a unique identification number.
- When we do testing, we set desire pre-condition(s), if any, given selected inputs to the program and note the observed output(s).
- We compare the observed output(s) with the expected output(s) and if they are the same, the test case is successful. If they are different, that is the failure condition with selected input(s) and this should be recorded properly in order to find the cause of failure.
- A good test case has a high probability of showing a failure condition. Test case designers should identify weak areas of the program and design test cases accordingly.

Terminologies – Test Case Template

Test Case Identification Number :		
Part I (Before Execution)		
1.	Purpose of test case:	
2.	Pre-condition: (Optional)	
3.	Input:	
4.	Expected Output:	
5.	Post-condition:	
6.	Written by :	
7.	Date of Design :	

Terminologies – Test Case Template

Test Case Identification Number :		
Part II (After Execution)		
1.	Output:	
2.	Post-condition: (Optional)	
3.	Pass / Fail :	
4.	If fails, any possible reason of failure (optional)	
5.	Suggestions (optional)	
6.	Run by :	
7.	Date of Suggestion :	

Terminologies – Test, Test Case & Test Suite

- The set of test cases is called a **Test Suite**.

- We may have a test suite of all test cases, test suite of all successful test cases and test suite of all unsuccessful test cases. Any combination of test cases will generate a test suite.

- All test suites should be preserved as they are equally valuable and useful for the purpose of maintenance of the software.

- Sometimes test suite of unsuccessful test cases gives very important information because these are the test cases which have made the program fail in the past.

Terminologies – Deliverables & Milestones

- Different deliverables are generated during various phases of the software development.

- Examples:

  - Source code
  - Software Requirements and Specification document (SRS)
  - Software Design Document (SDD)
  - Installation guide
  - User reference manual, etc.

Terminologies – Deliverables & Milestones

- The milestones are the events that are used to ascertain / determine the status of the project.

- Example:
  - Finalization of SRS is a milestone;
  - Completion of SDD is another milestone.

- The milestones are essential for monitoring, planning and tracking the progress of the software development.

Terminologies – Alpha, Beta and Acceptance Testing

Acceptance Testing

- This term is used when the software is developed for a specific customer.
- The customer is involved during acceptance testing.
- He/she may design ad-hoc test cases or well-planned test cases and execute them to see the correctness of the software.
- The discovered errors are fixed and modified and then the software is delivered to the customer.

Terminologies – Alpha, Beta and Acceptance Testing

Alpha & Beta Testing

- These terms are used when the software is developed as a product for anonymous customers where acceptance testing is not possible.
- Some potential customers are identified to test the product.
- The alpha tests are conducted at the developer's site by the customer.
- These tests are conducted in a controlled environment and may start when the formal testing process is near completion.

Terminologies – Alpha, Beta and Acceptance Testing

Alpha & Beta Testing

- The beta tests are conducted by potential customers at their sites.
- Unlike alpha testing, the developer is not present here.
- It is carried out in an uncontrolled real life environment by many potential customers.
- Customers are expected to report failures, if any, to the company.

Terminologies – Alpha, Beta and Acceptance Testing

Alpha & Beta Testing

- These failure reports are studied by the developers and appropriate changes are made in the software.

- Beta tests have shown their advantages in the past and releasing a beta version of the software to the potential customer has become a common practice.

- The company gets the feedback of many potential customers without making any payment and reputation of the company is not at stake even if many failures are encountered.

Terminologies – Quality and Reliability

- Software reliability is defined as “the probability of failure free operation for a specified time in a specified environment”.

- Although software reliability is defined as a probabilistic function and comes with the notion of time, it is not a direct function of time.

- The software does not wear out like hardware during the software development life cycle.

- There is no aging concept in software and it will change only when we intentionally change or upgrade the software.

Terminologies – Quality and Reliability

- Software quality determines how well the software is designed (quality of design), and how well the software conforms to that design (quality of conformance).
- Some software practitioners also feel that quality and reliability is the same thing.
- That is not necessarily true.
- Reliability is just one part of quality.

Terminologies – Static & Dynamic Testing

Static testing :

- Testing activities without executing the source code.

- All verification activities like inspections, walkthroughs, reviews, etc. come under this category of testing.

Terminologies – Static & Dynamic Testing

Dynamic testing :

- Refers to executing the source code and seeing how it performs with specific inputs.

- All validation activities come in this category where execution of the program is essential.

Terminologies – Testing vs Debugging

- The purpose of testing is to find faults and find them as early as possible.
- When any fault is found, the process used to determine the cause of this fault and to remove it is known as debugging.
- These are related activities and are carried out sequentially.

Limitations of Testing

- Expectation : Test everything before giving the software to the customers.

What do you mean by everything?

- Execute every statement of the program
- Execute every true and false condition
- Execute every condition of a decision node
- Execute every possible path
- Execute the program with all valid inputs
- Execute the program with all invalid inputs

All above objectives are impossible to achieve.

Limitations of Testing

- Reason:
  - Time constraint
  - Resource constraints
  - Large input domain
  - Too many paths

- Hence **'Everything' / Exhaustive Testing is impossible** and we have to settle for 'less than everything' in real life situations.

Verification Methods

- The most important aspect of software verification is its implementation in the early phases of the software development life cycle.

- If we manually examine / review any document for the purpose of finding faults, it is called verification.

- Verification is also called as static testing because the execution of the program is not required.

- As per IEEE, “verification is the process of evaluating the system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase”

Verification Methods

1. Peer Reviews
2. Walkthroughs
3. Inspections

Verification Methods – Peer Review

- Any type of testing (verification or validation), even ad-hoc and undisciplined, is better than no testing if it is carried out by person(s) other than the developers / writers of the document with the purpose of finding faults.

- Simplest way of reviewing the documents / programs to find out **faults** and observe **quality** during verification.

- We give the document(s) / program(s) to someone else and ask to review the document(s) / program(s).

Verification Methods – Peer Review

- Informal activity – Gives good results without spending any significant resources.
- Thrust should be to find faults in the document(s) / program(s)
- The reviewer may prepare a report of observations and findings or may inform verbally during discussions.
- Activity to be carried out by peers and may be very effective if reviewers have domain knowledge, good programming skills and proper involvement.

Assignment-2

Apply Peer Review Technique on given source code of Binary Search program.

Verification Methods – Walkthrough

- Formal and systematic approach.
- In a walkthrough, the author of the document presents the document to a small group of two to seven persons.
- Participants are not expected to prepare anything. Only the presenter, who is the author, prepares for the meeting.
- The document(s) is / are distributed to all participants. During the meeting, the author introduces the material in order to make them familiar with it.

Verification Methods – Walkthrough

- All participants are free to ask questions.
- All participants may write their observations on any display mechanism like boards, sheets, projection systems, etc. so that every one may see and give views.
- After the review, the author writes a report about findings and any faults pointed out in the meeting.
- Walkthroughs may help us to find potential faults and may also be used for sharing the documents with others.

Verification Methods – Walkthrough

Disadvantages:

- Non-preparation of participants and incompleteness of the document(s) presented by the author(s).
- The author may hide some critical areas and unnecessarily emphasize on some specific areas of his / her interest.
- The participants may not be able to ask many penetrating questions.

Verification Methods – Inspection

- Many names are used for this verification method like:
  - Formal reviews
  - Technical reviews
  - Inspections
  - Formal technical reviews, etc.
- Most structured and most formal type of verification method.
- The presenter is not the author but some other person who prepares and understands the document being presented.

Verification Methods – Inspection

- This forces that person to learn and review that document prior to the meeting.
- The document(s) is / are distributed to all participants in advance in order to give them sufficient time for preparation.
- Rules for such meetings are fixed and communicated to all participants.
- A team of three to six participants are constituted which is led by an impartial moderator.

Verification Methods – Inspection

- A presenter and a recorder are also added to this team to assure that the rules are followed and views are documented properly.

- Every person in the group participates openly, actively and follows the rules about how such a review is to be conducted.

- Everyone may get time to express their views, potential faults and critical areas.

- Important points are displayed by some display mechanism so that everyone can see them.

Verification Methods – Inspection

- The moderator, preferably a senior person, conducts such meetings and respects everyone's views.
- The idea is not to criticize anyone but to understand their views in order to improve the quality of the document being presented.
- Sometimes a checklist is also used to review the document.
- After the meeting, a report is prepared by the moderator and circulated to all participants.

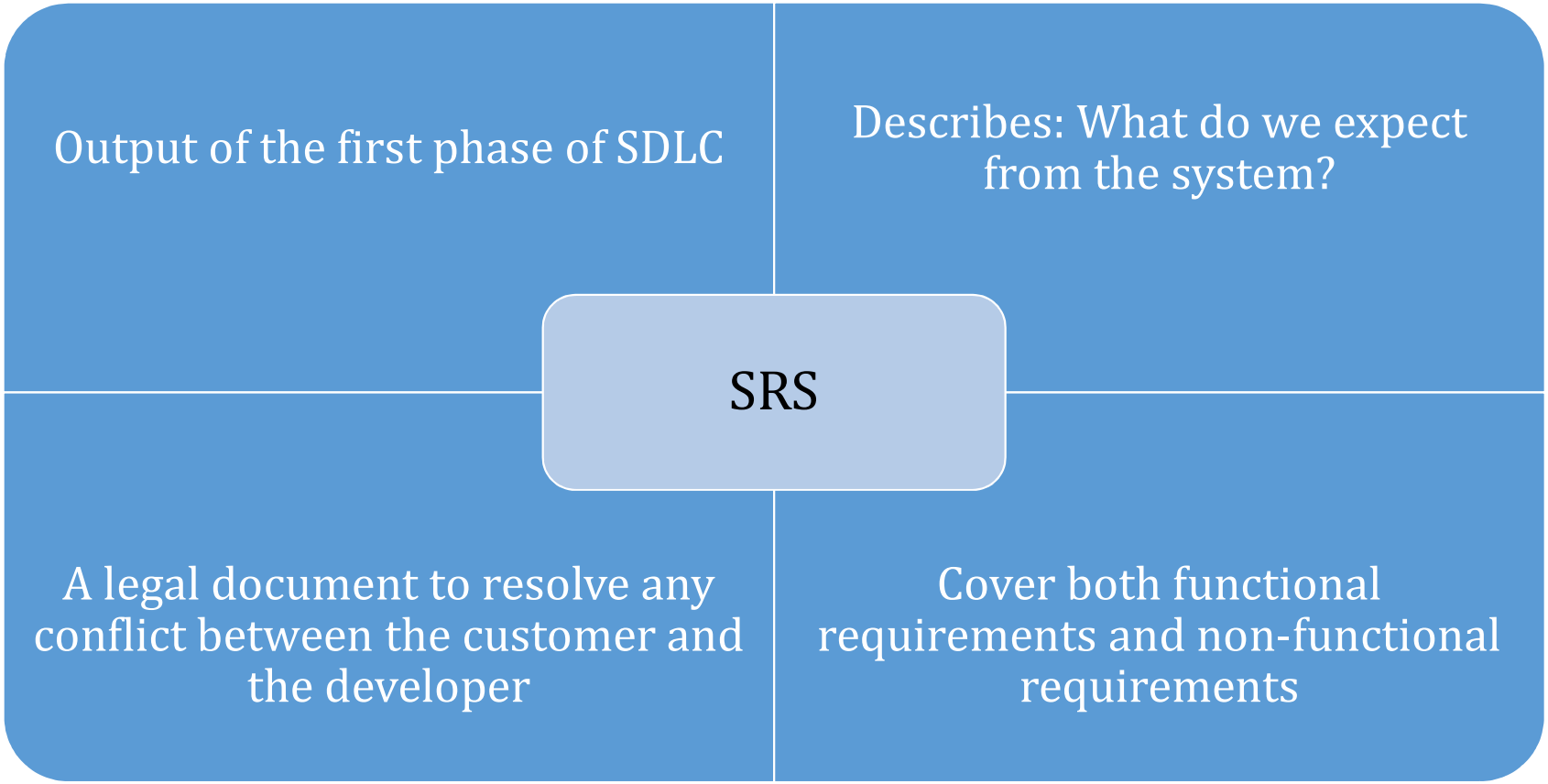
Verification Methods – Inspection

- They may give their views again, if any, or discuss with the moderator.
- A final report is prepared after incorporating necessary suggestions by the moderator.
- Inspections are very effective to find potential faults and problems in the document like SRS, SDD, source code, etc.
- Critical inspections always help find many faults and improve these documents, and prevent the propagation of a fault from one phase to another phase of the software development life cycle.

Verification Methods – Comparison

S. No.	Method	Presenter	Number of Participants	Prior preparation	Report	Strengths	Weaknesses
1.	Peer reviews	No one	1 or 2	Not required	Optional	Inexpensive but find some faults	Output is dependent on the ability of the reviewer
2.	Walkthrough	Author	2 to 7 participants	Only presenter is required to be prepared	Prepared by presenter	Knowledge sharing	Find few faults and not very expensive
3.	Inspections	Someone other than author	3 to 6 participants	All participants are required to be prepared	Prepared by moderator	Effective and find many faults	Expensive and requires very skilled participants

SRS Verification



SRS Verification – Nature of SRS Document

The SRS should include the following:

1. **Expectations from the software**: The SRS document should clearly specify 'what do we expect from the software?' and broadly describe functions of the software.

2. **Interfaces of the software**: The software will interact with many persons, hardware, other devices and external software. These interfaces should be written and 'forms' for interaction may also be provided.

SRS Verification – Nature of SRS Document

3. **Non-functional requirements:** Non-functional requirements (response time, speed, availability, recovery time of various software functions, portability, correctness, maintainability, reliability, security) should also be properly placed in the SRS document.

4. **Implementation difficulties and limitations:** All constraints of project implementation including resource limitations and operating environment should also be specified.

- Design and implementation details should not be included.

- SRS should be written in simple, clear and unambiguous language which may be understandable to all developers and customers.

SRS Verification – Organization of SRS Document

1. Introduction
   - 1.1 Purpose0
   - 1.2 Scope
   - 1.3 Definitions, Acronyms and Abbreviations
   - 1.4 References
   - 1.5 Overview
2. The Overall Description
   - 2.1 Product Perspective
     - 2.1.1 System Interfaces
     - 2.1.2 Interfaces
     - 2.1.3 Hardware Interfaces
     - 2.1.4 Software Interfaces
     - 2.1.5 Communications interfaces
     - 2.1.6 Memory Constraints
     - 2.1.7 Operations
     - 2.1.8 Site Adaptation Requirements
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and Dependencies
   - 2.6 Apportioning of Requirements
3. Specific Requirements
   - 3.1 External interfaces
   - 3.2 Functions
   - 3.3 Performance Requirements
   - 3.4 Logical Database Requirements
   - 3.5 Design Constraints
     - 3.5.1 Standards Compliance
   - 3.6 Software System Attributes
     - 3.6.1 Reliability
     - 3.6.2 Availability
     - 3.6.3 Security
     - 3.6.4 Maintainability
     - 3.6.5 Portability
   - 3.7 Organizing the Specific Requirements
     - 3.7.1 System Mode
     - 3.7.2 User Class
     - 3.7.3 Objects
     - 3.7.4 Feature
     - 3.7.5 Stimulus
     - 3.7.6 Response
     - 3.7.7 Functional Hierarchy
   - 3.8 Additional Comments

SRS Verification – Organization of SRS Document

4. Change Management Process
5. Document Approvals
6. Supporting Information

SRS Verification – Characteristics of SRS Document

• Correctness	• Feasibility
• Ambiguity	• Verifiability
• Completeness	• Modifiability
• Consistency	• Traceability

SRS Verification – SRS Document Checklist

- The SRS document is reviewed by the testing people by using any verification method (peer reviews, walkthroughs, inspections, etc.).
- Code inspections is recommended due to its effectiveness and capability to produce good results.
- Reviews can be conducted twice or even more often.
- Every review will improve the quality of the document but may consume resources and increase the cost of the software development.

SRS Verification – SRS Document Checklist

- A **checklist** is a popular verification tool which consists of a list of critical information content that a deliverable should contain.

- A checklist may also look for :
  - Duplicate information, Missing information, Unclear information, Wrong information, etc.

- Checklists are used during reviewing and may make reviews more structured and effective.

- A good checklist must address the above-mentioned characteristics of SRS document.

SRS Verification – Generic Checklist

Section - I	
Name of the reviewer	
Organization	
Group Number	
Date of Review	
Project Title	

SRS Verification – Generic Checklist

Section – II (Introduction)			
Sr. No.	Description	Yes/No/NA	Remarks
1.	Is the purpose of the project clearly defined?		
2.	Is the scope clearly defined?		
3.	Is document format as per standard (IEEE)?		
4.	Is the project formally approved by the customer?		
5.	Are all requirements, interfaces, constraints, definitions listed in appropriate sections?		
6.	Is the expected response time from user's point of view specified for all operations?		

SRS Verification – Generic Checklist

Section – II (Introduction)			
Sr. No.	Description	Yes/No/NA	Remarks
7.	Do all stated requirements express the expectations of the customer?		
8.	Are there areas not addressed in the SRS document that need to be?		
9.	Are non-functional requirements stated?		
10.	Are validity checks properly defined for every input condition?		

SRS Verification – Generic Checklist

Section – II (Ambiguity)			
Sr. No.	Description	Yes/No/NA	Remarks
11.	Are functional requirements separated from non-functional requirement?		
12.	Is any requirement conveying more than one interpretation?		
13.	Are all requirements clearly understandable?		
14.	Does any requirement conflict with or duplicate with other requirement?		
15.	Are there ambiguous requirements?		

SRS Verification – Generic Checklist

Section – II (Completeness)			
Sr. No.	Description	Yes/No/NA	Remarks
16.	Are all functional & non-functional requirements stated?		
17.	Are forms available with validity check?		
18.	Are all reports available in the specified format?		
19.	Are all references, constraints, assumptions, terms, units of measure clearly stated?		
20.	Has analysis been performed to identify missing requirements?		

SRS Verification – Generic Checklist

Section – II (Consistency)			
Sr. No.	Description	Yes/No/NA	Remarks
21.	Are the requirements specified at a consistent level of details?		
22.	Should any requirement be specified in more details?		
23.	Should any requirement be specified in less details?		
24.	Are the requirements consistent with other documents of the project?		
25.	Is there any difference in the stated requirements at two places?		

SRS Verification – Generic Checklist

Section – II (Verifiability)			
Sr. No.	Description	Yes/No/NA	Remarks
26.	Are all stated requirements verifiable?		
27.	Are requirements written in a language and vocabulary that the stakeholders can understand?		
28.	Are there any non-verifiable words?		
29.	Are all paths of a use-case verifiable?		
30.	Is each requirement testable?		

SRS Verification – Generic Checklist

Section – II (Modifiability)			
Sr. No.	Description	Yes/No/NA	Remarks
31.	Are all stated requirements modifiable?		
32.	Have redundant requirements been consolidated?		
33.	Has the document been designed to incorporate change?		
34.	Is the format structure and style of the document standard?		
35.	Is there any procedure to document a change?		

SRS Verification – Generic Checklist

Section – II (Traceability)			
Sr. No.	Description	Yes/No/NA	Remarks
36.	Can any requirement be traced back to its origin / source?		
37.	Is every requirement uniquely identifiable?		
38.	Are all requirement clearly understandable for implementation?		
39.	Has each requirement been cross referenced to requirements in previous project documents that are relevant?		
40.	Is each requirement identified such that it facilitates referencing of each requirement in future development and enhancement efforts?		

SRS Verification – Generic Checklist

Section – II (Feasibility)			
Sr. No.	Description	Yes/No/NA	Remarks
41.	Is every requirement feasible?		
42.	Is any requirement non-feasible due to technical reason?		
43.	Is any requirement non-feasible due to lack of resources?		
44.	Is any requirement feasible but very difficult to implement?		
45.	Is any requirement very complex?		

SRS Verification – Generic Checklist

Section – II (General)			
Sr. No.	Description	Yes/No/NA	Remarks
46.	Is the document concise and easy to follow?		
47.	Are requirements stated clearly and consistently without contradicting themselves or other requirements?		
48.	Are all forms, figures, tables uniquely numbered?		
49.	Are hardware and other communication requirements stated clearly?		
50.	Are all stated requirements necessary?		

SDD Document Verification

SDD - Software Design Description

- SDD document is prepared from SRS document.
- Every requirement is translated into design information required for planning and implementation of a software system.
- It represents the system as a combination of design entities and describes the important properties and relationship among those entities.

SDD Document Verification

- Design Entity: an element (unit) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.

- The design entities may have different nature, but may also have common characteristics.

- Each design entity shall have a purpose, function and a name.

- There is a common relationship among entities such as interfaces or shared data.

SDD Document Verification

- The common characteristics are described by attributes of entities.
- Attributes are the questions about entities.
- The answer to these questions is the values of the attributes.
- The collection of answers provides a complete description of an entity.
- The SDD should address all design entities along with their attributes.

SDD Document Verification – SDD Organization

- There may be a number of ways to view the design.
- Each design view gives a separate concern about the system.
- These views provide a comprehensive description of the design in a concise and usable form.
- Two popular design techniques are function oriented design and object oriented design.

SDD Document Verification – SDD Organization

- Any approach can be used depending on the nature and complexity of the project.
- Purpose is to prepare a quality document that translates all requirements into design entities along with its attributes.
- The verification process may be carried out many times in order to improve the quality of the SDD.
- The SDD provides a bridge between software requirements and implementation. Hence, strength of the bridge is the strength of the final software system.

SDD Document Verification – SDD Checklist

Section - I	
Name of the reviewer	
Organization	
Group Number	
Date of Review	
Project Title	

SDD Verification – Generic Checklist

Section – II (General Issues)			
Sr. No.	Description	Yes/No/NA	Remarks
1.	Is the document easy to read?		
2.	Is the document easy to understand?		
3.	Is the document format as per IEEE standard?		
4.	Does the document look professional?		
5.	Is system architecture specified?		

SDD Verification – Generic Checklist

Section – II (System Architecture)			
Sr. No.	Description	Yes/No/NA	Remarks
6.	Is the architecture understandable?		
7.	Are figures used to show the architecture of the system?		
8.	Are all essentials described clearly and consistently? (Software components, OS, DB etc.)		
9.	Is the software architecture consistent with existing policies, guidelines and standards?		
10.	Is the architecture complete with essential details?		

SDD Verification – Generic Checklist

Section – II (Software Design)			
Sr. No.	Description	Yes/No/NA	Remarks
11.	Is the design as per standards?		
12.	Are all design entities described?		
13.	Are all attributes clearly defined?		
14.	Are all interfaces shown amongst the design entities?		
15.	Are all stated objectives addressed?		
16.	Is the data dictionary specified in tabular format?		

SDD Verification – Generic Checklist

Section – II (Data Design)			
Sr. No.	Description	Yes/No/NA	Remarks
17.	Are all definitions of data elements included in data dictionary?		
18.	Are all appropriate attributes that describe each data element included in the data dictionary?		
19.	Is interface of data design described?		
20.	Is data design consistent with existing policies, procedures, guidelines, standards and technological directives?		

SDD Verification – Generic Checklist

Section – II (Interface Design)			
Sr. No.	Description	Yes/No/NA	Remarks
21.	Is the user interface for every application described?		
22.	Are all fields available on screen?		
23.	Is the quality of the screen acceptable?		
24.	Are all major functions supporting each interface addressed?		
25.	Are all validity checks for every field specified?		

SDD Verification – Generic Checklist

Section – II (Traceability)			
Sr. No.	Description	Yes/No/NA	Remarks
26.	Is every requirement stated in SRS addressed in design?		
27.	Does every design entity addressed at least one requirement?		
28.	Is there any missing requirement?		
29.	Is the requirement traceability matrix (RTM) prepared?		
30.	Does the RTM indicate that every requirement has been addressed clearly?		

Source Code Review

- A source code review involves one or more reviewers examining the source code and providing feedback to the developers, both positive and negative.

- Reviewers should not be from the development team.

- Source code can be reviewed for syntax, standards defined, readability and maintainability.

- Typically, reviews will have a standard checklist as a guide for finding common mistakes and to validate the source code against established coding standards.

Source Code Review - Issues

- Always use meaningful variables.
- Avoid confusing words in names. Do not abbreviate ‘Number’ to ‘No’; ‘Num’ is a better choice.
- Declare local variables and avoid global variables to the extent possible. Thus, minimize the scope of variables.
- Minimize the visibility of variables.
- Do not overload variables with multiple meanings.

Source Code Review - Issues

- Define all variables with meaningful, consistent and clear names.
- Do not unnecessarily declare variables.
- Use comments to increase the readability of the source code.
- Comments should describe what the source code does and not how the source code works.
- Always update comments while changing the source code.

Source Code Review - Issues

- Use spaces and not TABS.
- All divisors should be tested for zero or garbage value.
- Always remove unused lines of the source code.
- Minimize the module coupling and maximize the module strength.
- File names should only contain A-Z, a-z, 0-9, ‘_’ and ‘.’.

Source Code Review - Issues

- The source code file names should be all lower case.
- All loops, branches and logic constructs should be complete, correct and properly nested and also avoid deep nesting.
- Complex algorithms should be thoroughly explained.
- The reasons for declaring static variables should be given.
- Always ensure that loops iterate the correct number of times.

Source Code Review - Issues

- When memory is not required, it is essential to make it free.
- Release all allocated memory and resources after the usage.
- Stack space should be available for running a recursive function. Generally, it is better to write iterative functions.
- Do not reinvent the wheel. Use existing source code as much as possible. However, do not over-rely on this source code during testing. This portion should also be tested thoroughly.

Source Code Review - Checklist

Section - I	
Name of the reviewer	
Organization	
Group Number	
Date of Review	
Project Title	

Source Code Review - Checklist

Section – II (Structure)			
Sr. No.	Description	Yes/No/NA	Remarks
1.	Does the source code correctly and completely implement the design		
2.	Is there any coding standard being followed?		
3.	Has the developer tested the source code?		
4.	Does the source code executed as expected?		
5.	Is the source code clear and easy to understand?		

Source Code Review - Checklist

Section – II (Structure)			
Sr. No.	Description	Yes/No/NA	Remarks
6.	Are all functions in the design coded?		
7.	Is the source code properly structured?		
8.	Are there any blocks of repeated source code that can be combined?		
9.	Is any module very complex and should be decomposed into two or more modules?		
10.	Is the source code fault tolerant?		

Source Code Review - Checklist

Section – II (Variables)			
Sr. No.	Description	Yes/No/NA	Remarks
11.	Are all variables clearly defined with appropriate names?		
12.	Are there any redundant / unused variable?		
13.	Is there unnecessary usage of Global variables?		
14.	Are variable declarations properly commented?		
15.	Are all variables properly initialized?		

Source Code Review - Checklist

Section – II (Variables)			
Sr. No.	Description	Yes/No/NA	Remarks
16.	Is the scope of every variables minimized?		
17.	Is any variable names ambiguous?		
18.	Are all variable names spelt correctly and consistently?		

Source Code Review - Checklist

Section – II (Comments)			
Sr. No.	Description	Yes/No/NA	Remarks
19.	Is readability of the source code acceptable?		
20.	Is the source code well commented and documented properly?		
21.	Are all given comments necessary?		
22.	Is there any requirement of additional comments?		
23.	Are all comments consistent with the source code?		

Source Code Review - Checklist

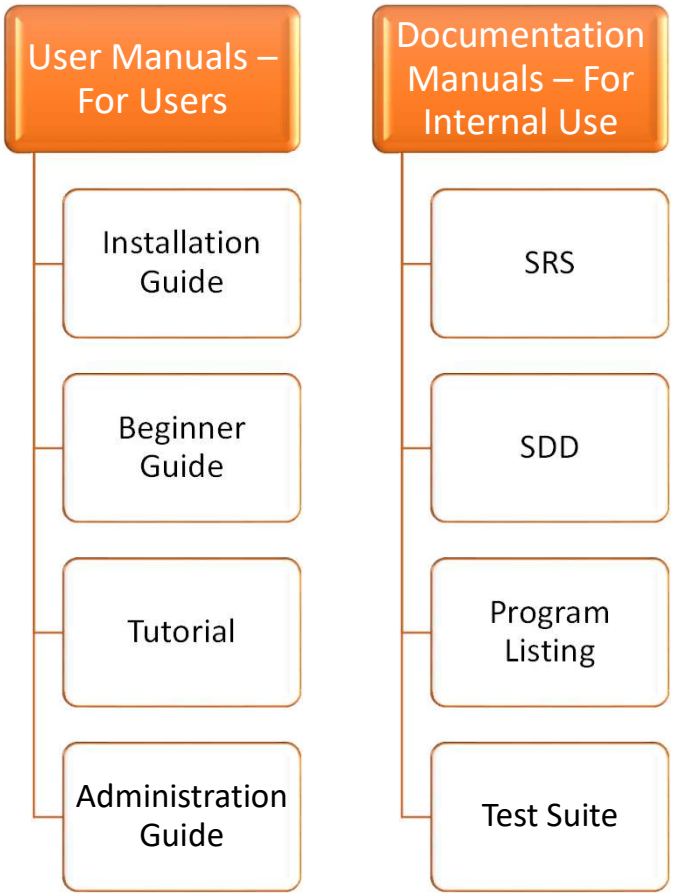
Section – II (Comments)			
Sr. No.	Description	Yes/No/NA	Remarks
24.	Are all loops, constructs and branches correct, complete and properly nested?		
25.	Does the source code make use of an infinite loop?		
26.	Does the loop execute the number of times specified?		
27.	Are loop exit conditions accurate?		
28.	Does every case statement has a *default*?		

Source Code Review - Checklist

Section – II (General)			
Sr. No.	Description	Yes/No/NA	Remarks
29.	Is every allocated memory de-allocated?		
30.	Does the source code make use of exception handling?		
31.	Does the source code appear to pose a security concern?		
32.	Does the source code avoids deadlocks?		
33.	Does the implementation match the documentation?		
34.	Is there any identifier that conflict with the keyword?		
35.	Is the source code maintainable?		

User Documentation Verification

- Many documents are prepared during the software development life cycle.
- Verification of the internal documents is essential for the success of implementation and quality of the final product
- Verification of user manuals is also important for the overall success of the project.



User Documentation Verification – Review Process

- Documents should be reviewed thoroughly.
- Proper consistency should be maintained in all documents.
- The documents should be written in simple, clear and short sentences.
- Installation procedure of the software must be explained step by step with proper justifications.

User Documentation Verification – Review Process

- All tables, figures and graphs should be numbered properly.
- Explanations, if possible, should be supported by suitable examples.
- A checklist may help to structure the review process and must highlight these issues.

User Documentation Verification – Checklist

Section - I	
Name of the reviewer	
Organization	
Group Number	
Date of Review	
Project Title	

User Documentation Verification – Checklist

Section – II (General Issues)			
Sr. No.	Description	Yes/No/NA	Remarks
1.	Is the document easy to read?		
2.	Is the document easy to understand?		
3.	Is the document well organized?		
4.	Are things easy to find?		
5.	Are spellings and grammar correct?		
6.	Are all references properly placed in text		
7.	Is consistency maintained?		
8.	Are all abbreviations and assumptions properly written at proper places?		

User Documentation Verification – Checklist

Section – II (Installation)			
Sr. No.	Description	Yes/No/NA	Remarks
9.	Is everything operated as stated in the document?		
10.	Is there any step omitted?		
11.	Does it specify a minimum system configuration requirement?		
12.	Does it specify reasons for failure of a particular activity?		

User Documentation Verification – Checklist

Section – II (Operational)			
Sr. No.	Description	Yes/No/NA	Remarks
13.	Does it clearly describes all toolbars, menus, commands and options?		
14.	Do toolbars, menu and command options operate as stated?		
15.	Are examples documented properly?		
16.	Are all steps explained properly?		
17.	Does the document specify all steps as accepted to operate a GUI?		
18.	Does it include sample screenshot identical to GUI?		

User Documentation Verification – Checklist

Section – II (Table, Graphs and Figures)			
Sr. No.	Description	Yes/No/NA	Remarks
19.	Are all tables, graph and figures properly numbered?		
20.	Are they identical to actual GUI?		
21.	Are they properly referenced in text?		
22.	Are they properly placed?		
23.	Are they consistent with previous tables, graphs and figures?		
24.	Are all given tables, graph and figures necessary?		
25.	Are there any requirement of new tables, graphs and figures?		

Software Project Audit

- Audit of a software project is an important activity and may be carried out at any time during the software development life cycle.

- Generally, auditors are appointed by the top management to review the progress of the project.

- The auditors are different from the developers and testers and may not have any involvement in the project.

- They examine the progress of the project and quality of the processes with respect to many attributes like project planning, management, quality management, resourcing, users, development approaches, testing etc.

Software Project Audit

- The auditing process is a continuous activity and may be carried out many times during the software development life cycle.
- We may audit the SRS, SDD, source code and other relevant documents.
- The audit process is a verification activity.
- Auditor prepares an audited report after examining the relevant records and documents.
- This report may help the management to initiate timely action.

Software Project Audit - Checklist

Checklist is based on following attributes:

1. Project planning
2. Project Management
   - Project Scheduling and tracking
   - Project Status Reporting
   - Project Estimating
   - Risk Management
3. Quality Management
4. Software Configuration Management

Software Project Audit - Checklist

5. Management Procedures
   - Vendor Management
   - Issues Management
   - Stakeholder Management
6. Resourcing
7. Users
8. Development Approach
   - Methodology
   - CASE
   - Analysis and Design
   - Development
   - Testing

Software Project Audit - Checklist

9. Application Architecture
10. Data Architecture and Standard
11. Technical Architecture

Assignment-3

Case Study (University Registration System):

Verify given SRS and prepare checklist