Unit-iv

Static analysis involves no dynamic execution of the software under test and can detect possible defects in an early stage, before running the program.

Static analysis is done after coding and before executing unit tests.

Static analysis can be done by a machine to automatically “walk through” the source code and detect noncomplying rules. The classic example is a compiler which finds lexical, syntactic and even some semantic mistakes.

Static analysis can also be performed by a person who would review the code to ensure proper coding standards and conventions are used to construct the program. This is often called Code Review and is done by a peer developer, someone other than the developer who wrote the code.

Static analysis is also used to force developers to not use risky or buggy parts of the programming language by setting rules that must not be used.

When developers performs code analysis, they usually look for

- Lines of code
- Comment frequency
- Proper nesting
- Number of function calls
- Cyclomatic complexity
- Can also check for unit tests

Quality attributes that can be the focus of static analysis:

- Reliability
- Maintainability
- Testability
- Re-usability
- Portability
- Efficiency

What are the Advantages of Static Analysis?

The main advantage of static analysis is that it finds issues with the code before it is ready for integration and further testing.

Static code analysis advantages:

It can find weaknesses in the code at the exact location.

It can be conducted by trained software assurance developers who fully understand the code.

Source code can be easily understood by other or future developers

It allows a quicker turn around for fixes

Weaknesses are found earlier in the development life cycle, reducing the cost to fix.

Less defects in later tests

Unique defects are detected that cannot or hardly be detected using dynamic tests

Unreachable code

Variable use (undeclared, unused)

Uncalled functions

Boundary value violations

Static code analysis limitations:

It is time consuming if conducted manually.

Automated tools produce false positives and false negatives.

There are not enough trained personnel to thoroughly conduct static code analysis.

Automated tools can provide a false sense of security that everything is being addressed.

Automated tools only as good as the rules they are using to scan with.

It does not find vulnerabilities introduced in the runtime environment.

What is Dynamic Analysis?

In contrast to Static Analysis, where code is not executed, dynamic analysis is based on the system execution, often using tools.

Dynamic program analysis is the analysis of computer software that is performed with executing programs built from that software on a real or virtual processor (analysis performed without executing programs is known as static code analysis). Dynamic program analysis tools may require loading of special libraries or even recompilation of program code.

The most common dynamic analysis practice is executing Unit Tests against the code to find any errors in code.

Dynamic code analysis advantages:

It identifies vulnerabilities in a runtime environment.

It allows for analysis of applications in which you do not have access to the actual code.

It identifies vulnerabilities that might have been false negatives in the static code analysis.

It permits you to validate static code analysis findings.

It can be conducted against any application.

Dynamic code analysis limitations:

Automated tools provide a false sense of security that everything is being addressed.

Cannot guarantee the full test coverage of the source code

Automated tools produce false positives and false negatives.

Automated tools are only as good as the rules they are using to scan with.

It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

What is Software Testing:

Software testing can be stated as the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by it's design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- **Quality Check** - Software testing helps in determining following set of properties of any software such as
    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - Portability

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy and usability. It mainly aims at measuring specification, functionality and performance of a software program or application.

Software testing can be divided into two steps:
1. **Verification:** it refers to the set of tasks that ensure that software correctly implements a specific function.
2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
Verification: “Are we building the product right?”
Validation: “Are we building the right product?”

Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

TRINITY INSTITUTE OF TECHNOLOGY AND RESEARCH BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Sr.No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

What are different types of software testing?
Software Testing can be broadly classified into two types:

1. **Manual Testing:** Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.
Apart from regression testing, automation testing is also used to test the application from load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

What are different techniques of Software Testing?
Software techniques can be majorly classified into two categories:

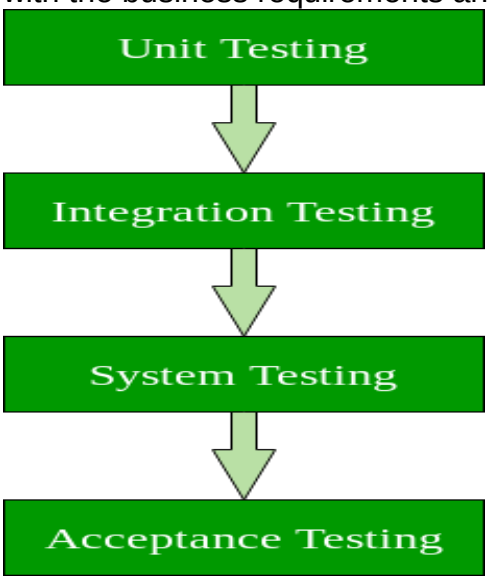
1. **Black Box Testing:** The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without concerning with the internal logical structure of the software is known as black box testing.
2. **White-Box Testing:** The technique of testing in which the tester is aware of the internal workings of the product, have access to it's source code and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.

BLACK BOX TESTING	WHITE BOX TESTING
Internal workings of an application are not required.	Knowledge of the internal workings is must.
Also known as closed box/data driven testing.	Also knwon as clear box/structural testing.
End users, testers and developers.	Normally done by testers and developers.
THis can only be done by trial and error	Data domains and internal boundaries can be

BLACK BOX TESTING	WHITE BOX TESTING
method.	better tested.

What are different levels of software testing?
Software level testing can be majorly classified into 4 levels:

1. **Unit Testing:** A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
2. **Integration Testing:** A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
3. **System Testing:** A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
4. **Acceptance Testing:** A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.



Types of Black Box Testing:

1. Functionality Testing

2. Non-functionality Testing

Functional testing:

In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black box testing and the testers need not concern about the source code of the application.

Non-functional testing:

In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility etc., Main focus is to improve the user experience on how fast the system responds to a request.

Testing Artifacts:

Test Artifacts are the deliverables which are given to the stakeholders of a software project. A software project which follows SDLC undergoes the different phases before delivering to the customer. In this process, there will be some deliverables in every phase. Some of the deliverables are provided before the testing phase commences and some are provided during the testing phase and rest after the testing phase is completed.

Some of the test deliverables are as follows:

- Test plan
- Traceability matrix
- Test case
- Test script
- Test suite
- Test data or Test Fixture
- Test harness

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester

will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.
Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	Disadvantages
Offers combined benefits of black-box and	Since the access to source code is not available, the ability

white-box testing wherever possible.	to go over the code and test coverage is limited.
Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.	The tests can be redundant if the software designer has already run a test case.
Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.
The test is done from the point of view of the user and not the designer.	

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by	Performed by end-users and also by	Normally done by testers and

testers and developers.	testers and developers.	developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure −

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing −

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows −

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

Testing documentation involves the documentation of artifacts that should be developed before or during the testing of Software.

Documentation for software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing, etc. This section describes some of the commonly used documented artifacts related to software testing such as −

- Test Plan
- Test Scenario
- Test Case
- Traceability Matrix

Test Plan

A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, and the limitations of the testing and the schedule of testing activities. Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.

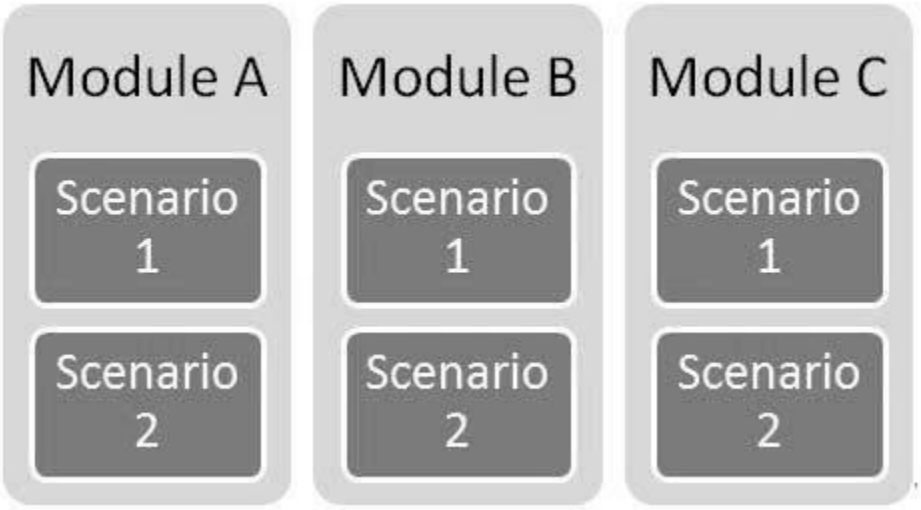
A test plan includes the following −

- Introduction to the Test Plan document
- Assumptions while testing the application
- List of test cases included in testing the application
- List of features to be tested
- What sort of approach to use while testing the software
- List of deliverables that need to be tested
- The resources allocated for testing the application
- Any risks involved during the testing process
- A schedule of tasks and milestones to be achieved

Test Scenario

It is a one line statement that notifies what area in the application will be tested. Test scenarios are used to ensure that all process flows are tested from end to end. A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application.

The terms 'test scenario' and 'test cases' are used interchangeably, however a test scenario has several steps, whereas a test case has a single step. Viewed from this perspective, test scenarios are test cases, but they include several test cases and the sequence that they should be executed. Apart from this, each test is dependent on the output from the previous test.



Test Case

Test cases involve a set of steps, conditions, and inputs that can be used while performing testing tasks. The main intent of this activity is to ensure whether a software passes or fails in terms of its functionality and other aspects. There are many types of test cases such as functional, negative, error, logical test cases, physical test cases, UI test cases, etc.

Furthermore, test cases are written to keep track of the testing coverage of a software. Generally, there are no formal templates that can be used during test case writing. However, the following components are always available and included in every test case −

- Test case ID

- Product module
- Product version
- Revision history
- Purpose
- Assumptions
- Pre-conditions
- Steps
- Expected outcome
- Actual outcome
- Post-conditions

Many test cases can be derived from a single test scenario. In addition, sometimes multiple test cases are written for a single software which are collectively known as test suites.

Traceability Matrix

Traceability Matrix (also known as Requirement Traceability Matrix - RTM) is a table that is used to trace the requirements during the Software Development Life Cycle. It can be used for forward tracing (i.e. from Requirements to Design or Coding) or backward (i.e. from Coding to Requirements). There are many user-defined templates for RTM.

Each requirement in the RTM document is linked with its associated test case so that testing can be done as per the mentioned requirements. Furthermore, Bug ID is also included and linked with its associated requirements and test case. The main goals for this matrix are −

- Make sure the software is developed as per the mentioned requirements.
- Helps in finding the root cause of any bug.
- Helps in tracing the developed documents during different phases of SDLC.

Estimating the efforts required for testing is one of the major and important tasks in SDLC. Correct estimation helps in testing the software with maximum coverage. This section describes some of the techniques that can be useful in estimating the efforts required for testing.

Functional Point Analysis

This method is based on the analysis of functional user requirements of the software with the following categories −

- Outputs
- Inquiries
- Inputs
- Internal files
- External files

Test Point Analysis

This estimation process is used for function point analysis for black-box or acceptance testing. The main elements of this method are: Size, Productivity, Strategy, Interfacing, Complexity, and Uniformity.

Mark-II Method

It is an estimation method used for analyzing and measuring the estimation based on end-user's functional view. The procedure for Mark-II method is as follows −

- Determine the viewpoint
- Purpose and type of count
- Define the boundary of count
- Identify the logical transactions
- Identify and categorize data entity types
- Count the input data element types
- Count the functional size

Testing and Debugging

Testing − It involves identifying bug/error/defect in a software without correcting it. Normally professionals with a quality assurance background are involved in bugs identification. Testing is performed in the testing phase.

Debugging − It involves identifying, isolating, and fixing the problems/bugs. Developers who code the software conduct debugging upon encountering an error in the code. Debugging is a part of White Box Testing or Unit Testing. Debugging can be performed in the development phase while conducting Unit Testing or in phases while fixing the reported bugs.

Testing Tools:

Tools from a software testing context can be defined as a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.

Classification of Tools

Tools can be classified based on several parameters. They include:

- The purpose of the tool
- The Activities that are supported within the tool
- The Type/level of testing it supports
- The Kind of licensing (open source, freeware, commercial)
- The technology used

Types of Tools:

S.No.	Tool Type	Used for	Used by
1.	Test Management Tool	Test Managing, scheduling, defect logging, tracking and analysis.	testers
2.	Configuration management tool	For Implementation, execution, tracking changes	All Team members
3.	Static Analysis Tools	Static Testing	Developers
4.	Test data Preparation Tools	Analysis and Design, Test data generation	Testers
5.	Test Execution Tools	Implementation, Execution	Testers
6.	Test Comparators	Comparing expected and actual results	All Team members

7.	Coverage measurement tools	Provides structural coverage	Developers
8.	Performance Testing tools	Monitoring the performance, response time	Testers
9.	Project planning and Tracking Tools	For Planning	Project Managers
10.	Incident Management Tools	For managing the tests	Testers

Tools Implementation - process

- Analyse the problem carefully to identify strengths, weaknesses and opportunities
- The Constraints such as budgets, time and other requirements are noted.
- Evaluating the options and Shortlisting the ones that are meets the requirement
- Developing the Proof of Concept which captures the pros and cons
- Create a Pilot Project using the selected tool within a specified team
- Rolling out the tool phase wise across the organization

Structured Analysis vs. Object Oriented Analysis

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are −

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

Now, we will look at the relative advantages and disadvantages of structured analysis approach and object-oriented analysis approach.

TRINITY INSTITUTE OF TECHNOLOGY AND RESEARCH BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Advantages/Disadvantages of Object Oriented Analysis

Advantages	Disadvantages
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	It cannot identify which objects would generate an optimal system design.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	The object-oriented models do not easily show the communications between the objects in the system.
It allows effective management of software complexity by the virtue of modularity.	All the interfaces between the objects cannot be represented in a single diagram.
It can be upgraded from small to large systems at a greater ease than in systems following structured analysis.	

Advantages/Disadvantages of Structured Analysis

Advantages	Disadvantages
As it follows a top-down approach in contrast to bottom-up approach of object-oriented analysis, it can be more easily comprehended than OOA.	In traditional structured analysis models, one phase should be completed before the next phase. This poses a problem in design, particularly if errors crop up or requirements change.
It is based upon functionality. The overall purpose is identified and then functional decomposition is	The initial cost of constructing the system is high, since the whole system needs to be designed at once

done for developing the software. The emphasis not only gives a better understanding of the system but also generates more complete systems.	leaving very little option to add functionality later.
The specifications in it are written in simple English language, and hence can be more easily analyzed by non-technical personnel.	It does not support reusability of code. So, the time and cost of development is inherently high.