Global Journal of Computer Science & Technology
Volume 11 Issue 5 Version 1.0 April 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
ISSN: 0975-5861

3W's of Static Software Testing Techniques

By Sheikh Umar Farooq, S. M. K. Quadri
University of Kashmir

Abstract- : No fault detection technique is capable of finding all classes of fault in software, so we have to use variety of techniques that will help us to ensure that a variety of faults are found, resulting in more effective testing. As we know dynamic testing can only be applied to code; using static techniques becomes imperative to ensure V&V process is more efficient and effective. Before using any technique we should have profound knowledge of that technique. In this paper, we present an in-depth review of all static testing techniques. The goal of this paper is to look at the technical aspects of static techniques and to highlight their importance in STLC.

Keywords: *Static Techniques, Reviews, Static Analysis.*

Classification: *GJCST Classification: FOR Code: 080309,080302*

Strictly as per the compliance and regulations of:

3W's of Static Software Testing Techniques

Sheikh Umar Farooq[α], S. M. K. Quadri[Ω]

Abstract- No fault detection technique is capable of finding all classes of fault in software, so we have to use variety of techniques that will help us to ensure that a variety of faults are found, resulting in more effective testing. As we know dynamic testing can only be applied to code; using static techniques becomes imperative to ensure V&V process is more efficient and effective. Before using any technique we should have profound knowledge of that technique. In this paper, we present an in-depth review of all static testing techniques. The goal of this paper is to look at the technical aspects of static techniques and to highlight their importance in STLC.

Keywords - *Static Techniques, Reviews, Static Analysis*

I. Introduction

Software testing is the process of executing a program or system with the intent of finding errors [1][2]. Software testing is commonly known as a test to evaluate and then guarantee the level of quality of software, or to verify if there are no risks. Software testing is also used to test the software for other software quality factors like reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility etc [3]. Effective software testing will contribute to the delivery of reliable and quality oriented product, more satisfied user, lower maintenance and more correct and reliable results [4]. To perform software testing we have wide variety of testing techniques [5][6]. Perhaps at present the single most important thing to understand is that the best testing technique is no single testing technique. Because each testing technique is good at finding one specific class of defect, using just one technique will help ensure that many (perhaps most but not all) defects of that particular class are found. Unfortunately, it may also help to ensure that many defects of other classes are missed! Using a variety of techniques will therefore help ensure that a variety of defects are found, resulting in more effective testing [7].

About [α]- Research Scholar in P. G. Department of Computer Sciences, University of Kashmir, India. He did his Bachelor's degree in Computer Applications from Islamia College of Science & Commerce and Master's degree in Computer Applications from Kashmir University.
E-mail- shiekh.umar.farooq@gmail.com
About[Ω]- Head, P. G. Department of Computer Sciences, University of Kashmir, India. He did his M. Tech. in Computer Applications from Indian School of Mines and Ph. D. in Computer Sciences from Kashmir University.
E-mail- quadrismk@hotmail.com

As we know testing only code is not sufficient enough to guarantee that all faults and flaws in the software system are identified, so using techniques other than dynamic testing techniques becomes essential. This not only guarantees that different classes of faults are identified, but it also makes sure that they are identified before going in the following phases in software development life cycle, thereby saving the overall software cost. In addition to that using such techniques can also ensure that we are not missing any requirement or violating any procedure. We all know prevention is better than cure; Static testing is about prevention, while dynamic testing is about cure. In this paper, besides providing a review of static techniques, we will also look at the technical aspects of static testing techniques. Besides that we will also look at their importance and their use in software testing life cycle.

II. Terminology Used

The Terminology used in this paper is adopted from [8]. Following terms are mainly used in this paper.

Error: The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Fault (Defect): Abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function.

Note: Can also be defined thus: A requirements, design, or implementation flaw or deviation from a desired or intended state [9].

Failure: The inability of a system to perform its required functions within specified performance requirements.

Note: Failures are either random (in hardware) or systematic (in hardware or software)

III. Static Testing Techniques

In each testing type there are many software testing techniques that are used to test a system [10]. Testing techniques refer to different methods or ways of testing particular features of a computer program, system or product. Software testing techniques are diverse methods to do software testing and authenticate them. Based on whether the actual execution of software under evaluation is needed or not, there are two major categories of quality assurance activities: static and dynamic [11, 12]. Static techniques are concerned with the analysis and checking of system representations such as the requirements documents, design diagrams and the program source code, either manually or automatically, without actually executing the

code [13]. Documentation in the form of text, models or code are analyzed, often by hand. In a number of cases, e.g. in the compilation of program code, tools are used [14]. Figure 1 shows the where static techniques are applied. Static technique can include different kinds of activities within the development/testing process and can be applied at different testing levels, for instance reviewing requirements documents, checking designs against certain design flaws, code inspection or even manual verification of test case descriptions. The defects detected are often related to requirements, design or code: either there are parts missing or the documents (e.g. source code) are not consistent within themselves or do not conform to the standard in force. The fundamental objective of static testing technique is to improve the quality of the software products by finding errors in early stages of software development life cycle [15]. Static techniques can explore abstractions of all possible program behaviors.

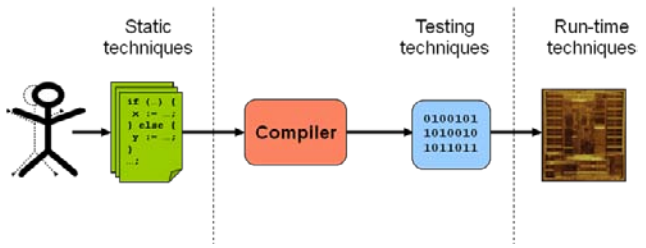

Figure 1: Static Techniques [16]

Static techniques has got following advantages:

- Since static testing can start early in the life cycle, early feedback on quality issues can be established, e.g. an early validation of user requirements and not just late in the life cycle during acceptance testing [14].
- By detecting defects at an early stage, rework costs are most often relatively low and thus a relatively cheap improvement of the quality of software products can be achieved.
- Since rework effort is substantially reduced, development productivity figures are likely to increase [17].
- The evaluation by a team usually involved has the additional advantage that there is an exchange of information between the participants.
- Static tests contribute to an increased awareness of quality issues.
- Static testing can discover dead codes, infinite loops, uninitialized and unused variables, standard violations and is effective in finding 30-70% of errors.

One important thing about static techniques is that in contrast to dynamic testing they help us to find defects rather than failures.Types of defects that are easier to find during static testing are:

- Deviations from standards.
- Missing requirements.
- Design defects.
- Non-maintainable code.
- Inconsistent interface specifications.
- Programming errors (e.g. Infinite Loops)

The main disadvantage of static testing is that

- They do not take the interaction of the program's input parameters into consideration, and this can determine the run-time behavior of a Real-Time System. Thus, static techniques give the very loose time bounds which may be retrieved from the test program's structure (source code or object code) [18].
- It is often hard to obtain suitable external user information for static testing as proposed in [19]. Usually this type of information is difficult to assess as the complexity of most systems prohibits the estimation of internal parameter values, loop bounds or the complete dynamic behavior. It is important that the assessment of the run-time behavior, which is particularly tedious, should be undertaken by a testing tool rather than a human tester.
- The information provided is not always precise since it is usually based on an approximation

IV. Static Techniques Classification

Static techniques can be grouped under two categories:

1. Reviews
2. Analysis.

a) *Reviews*

A review is a fundamental technique that must be used throughout the development lifecycle. Basically a review is any of a variety of activities involving evaluation of technical matter by a group of people working together [20]. A major portion in static software testing is carried out with the help of reviews. It is not necessary that all below mentioned reviews will be performed, if more than one type of review is used, the order may vary. For example, an informal review may be carried out before a technical review, or an inspection may be carried out on a requirements specification before a walkthrough with customers. Reviews can be of following types:

i. Informal Review

A review done by peers in an informal fashion without any documented findings or process. An informal review involves two or more people looking through a document or codes that one or the other of them has written. The purpose is still to detect defects, but there are usually no check-lists used and the result

does not need to be documented. [21]. This is what we all do all the time, mostly without thinking about it as a review: asking a colleague or a friend to look at something we have produced. The objectives of informal reviews are very individual, depending on the author's needs. It could be everything from finding spelling and grammar mistakes, to the structure of the product, to the actual contents from a professional point of view. An informal review follows no formal documented process. The participants are normally just the author and one or two reviewers [1]. The reviewer(s) are usually chosen by the author. This type of static testing can be performed on any document at any state during its production. The author decides when he or she would like an informal review to take place. The review is done according to the reviewer's perception. The author typically gets feedback in either pure verbal form or in the form of notes scribbled on the document under testing. The result of an informal review varies and is very much dependent on the review skills of the chosen reviewer(s). The author decides when the review is over. He or she may correct the document as he or she seems fit; there are no obligations following an informal review. There are a few disadvantages connected with informal reviews:

- Dependency of the reviewers' reviewing skill, but that can easily be overcome by finding the right reviewers for the job.
- There are usually no records kept of the reviews and hence no data available for calculation of effectiveness.

The benefits of informal reviews should, however, not be underestimated. Even though the reviews are informal they are very useful. In most companies no material must be delivered to another without having at least been through an informal review.

ii. *Walkthrough*

A form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems" [22]. In a walkthrough, a group of developers—with three or four being an optimal number—performs the review. Only one of the participants is the author of the program. The other participants are selected from different departments and backgrounds. A walkthrough is characterized by the author of the document under review guiding the participants through the document and his or her thought processes, to achieve a common understanding and to gather feedback. The content of the document is explained step by step by the author, to reach consensus on changes or to gather information [1]. It involves the author presenting his or her material to a selected group of participants in order to get them involved in the test basis more quickly as a result. Questions and explanations are welcomed during the meeting. The purpose is more one of creating a common picture, than of identifying defects, but the technique is still counted as static testing. Scenarios and dry runs may be used to validate the content. Within a walkthrough the author does most of the preparation. The participants, who are selected from different departments and backgrounds, are not required to do a detailed study of the documents in advance [23]. Because of the way the meeting is structured, a large number of people can participate and this larger audience can bring a great number of diverse viewpoints regarding the contents of the document being reviewed as well as serving an educational purpose. If the audience represents a broad cross-section of skills and disciplines, it can give assurance that no major defects are 'missed' in the walk-through. A walkthrough is especially useful for higher-level documents, such as requirement specifications and architectural documents. Advantages of walkthroughs are

- This process frequently exposes a batch of errors, allowing the errors to be corrected later as a whole.
- Walkthroughs, result in lower debugging (error-correction) costs, because when an error is found it is usually precisely located in the code.
- Walkthrough is an excellent tool for collaboration where joint decisions can be made by the tea. Each and every member present must be involved in making decisions [24].
- Walkthrough is also useful for training the entire team at one time. Test Scenarios and designs are generally subjected to walkthrough.
- People understand what they can expect from the document/approach under consideration.
- Problems can get recorded and suggestions can be received from the team for improving the work product further.

Despite having so many advantages, Walkthroughs do have following disadvantages [24]:

- Availability of people can be an issue when teams are large. Multi locations and distributed teams are major challenges in walkthrough. It is more formal than peer and procuring people can be a challenge.
- Team members may not be experts in giving comments and may need some training and basic knowledge about the project, artifact under walkthrough and so on.
- Time can be a constraint where schedules are tight.

iii. *Technical Review*

As the name suggests, a technical review focuses on the technical parts of the project like architecture and program design [25]. A peer group discussion activity that focuses on achieving consensus

on the technical approach to be taken. Technical reviews may also provide recommendations of alternatives and examination of various alternatives [26]. Here, primarily the technical experts and the architects take part together with other developers. This type of review is led by a trained moderator and comprises of experts from the team itself. The purpose is both to evaluate choices of solution, and compliance with standards and other documentation. The result is often documented in a log. Technical reviews are less formal and there is little or no focus on defect identification on the basis of referenced documents, intended readership and rules. In practice, technical reviews vary from quite informal to very formal.

iv. *Inspection*

The most formal review technique is called an inspection and is strictly governed. The method was developed by Michael E. Fagan at IBM.30 [26] and has since been improved [27] and adapted by many major software companies. An inspection is a set of procedures and error-detection techniques for group code reading [1]. Led by a trained moderator and involves peers to examine the product. The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists. An exit criterion is defined and a formal follow-up is carried out on the documented findings. Metrics are calculated and analyzed to optimize the process. The participants prepare themselves carefully by examining selected areas according to role descriptions and check-lists. At the review meeting all points of view are recorded and how they are to be addressed can be discussed. On the other hand, the problems are solved afterwards. In the inspection meeting the defects found are logged and any discussion is postponed until the discussion phase. This makes the inspection meeting a very efficient meeting. Since the way of working is formal, only a limited amount of material is examined on each occasion, mostly no more than a dozen pages, depending on the test basis [25]. For this type of review to be effective requires the participants to be trained, check-lists to be compiled, and the review meeting itself to be chaired by an experienced moderator. The reason for carrying out inspections can be explained by using Weinberg's concept of egoless engineering [21]. Weinberg refers to the human tendency to self-justify actions. Because of this tendency, many engineering organizations have established independent test groups that specialize in finding defects. Similar principles have led to the introduction of inspections and reviews in general. Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals. For example, if the time to market is extremely important, the emphasis in inspections will be on efficiency. In a safety-critical market, the focus will be on effectiveness. Advantages of Inspection are [24]:

- Expert's opinion is available as they are present during review. Subject-matter experts can clarify many issues.
- Inspection must be time effective as availability of experts is limited. There is always a time pressure for inspection.
- Defects are recorded but solution is not given by subject-matter experts. This helps the organization to initiate own action plan for fixing the defects.

Inspection has got following disadvantages:

- Manual process with strict guidelines
- Time-consuming (expensive)
- Tedious and error-prone (cannot be done full-time)
- Inspections increase costs early in the software process
- Precise standards or guidelines must be available and inspection team members must be familiar with them
- Not incremental (repeated inspection costs same as first one)

v. *Management Review*

Management reviews are conducted by management representatives to evaluate the status of work done and to make decisions regarding downstream activities [29]. A systematic evaluation of a software acquisition, supply, development, operation, or maintenance process performed by or on behalf of management that monitors progress, determines the status of plans and schedules, confirms requirements and their system allocation, or evaluates the effectiveness of management approaches used to achieve fitness for purpose [26]. The primary objective is to find defects in the documents under static testing. The secondary objective is to monitor progress according to the current plan, to assess status, and to make necessary decisions about any actions to take accordingly, including changes in resources, time, and/or scope/quality and updating the plan accordingly. The scope and the quality are usually expressed in terms of requirements to fulfill. Management reviews are usually planned to take place at certain times in the development life cycle, typically in connection with defined milestones, that is, transfer from one development phase to the next. Management review is a review type performed on management documents. This may be project related plans like

- Project management plans, including schedules and resources
- Quality assurance plans
- Configuration management plans
- Risk management plans

- Contingency plans
- Plans pertaining to the product, such as:
- Safety plans
- Installation plans
- Maintenance plans
- Backup and recovery plans
- Disaster plans
- Reports, such as:
- Progress reports
- Incident reports, including customer complaints
- Technical review reports
- Inspection reports
- Audit reports

A management review is performed using all appropriate information about the status of the project and the product, like progress and status reports concerning both technical and financial aspects and incident reports [30]. The roles that should be filled for a management review are the decision maker (the owner of the plan), the leader, reviewers, and a recorder. The reviewers are relevant stakeholders and should include management and technical staff involved in the execution of the planned activities. For the management review of a test plan the reviewers should include the test manager, the project manager (higher management), and the people involved in the testing activities. The total number of participants should be within 3–10 people. The management review process is also fairly formal. The leader schedules the preparation and the review meeting and presents the plan and any other information to the reviewers. The reviewers are usually expected to be prepared, that is to know the current plan and any deviations from it, before the review meeting is held. At the review meeting the plan is checked for compliance with other plans and consistency with reality [31]. The performance of management procedures being applied may also be assessed. Conclusions about what should be changed in the plan and what should not be are reached before the end of the meeting. A report is written after the review meeting summarizing the action items defined and the issues to be resolved, if any. In some cases measurements related to the time and effectiveness of the review are reported. The benefits of management reviews are many. A plan that is agreed on by all relevant stakeholders has a higher probability of being followed than a plan without such an agreement. The disadvantages of management reviews are few. The outcome depends on the reviewers, but these are usually sufficiently committed. If reporting of measurements is not imposed it is difficult to calculate the effectiveness of the technical reviews, but the measurements are not difficult to obtain [30].

vi. *Audit*

Audit is by far the most formal static testing technique. They are conducted by personnel external to a project to evaluate compliance with specifications standards contractual agreements or other criteria [32]. Audits may be either internal or external. Internal audits are performed by auditors from the organization, but external to the project under audit. External audits are performed by auditors entirely external to the organization, usually from some sort of authority organization. Auditors must have a special education and certification to perform official audits. Audits are usually performed late in the development life cycle, just before release of the product and/or close down of the project. Smaller audits may be performed in connection with other important milestones. The primary objective of an audit is to provide an independent evaluation of an activity's compliance to applicable process descriptions, contracts, regulations, and/or standards [26]. Any document, technical as well as managerial, can be part of an audit. In an audit a number of documents, if not all documents, will usually be evaluated and their consistency as well as their collective compliance to the basis material checked. A lead auditor is responsible for the audit, and he or she also acts as moderator. The lead auditor and any other participating auditors collect evidence of compliance through document examination, interviews, and walkthroughs of documents of special interest. The audit leader decides when the audit should take place, and the documents will be audited in the state they are in at the time of the audit. The result of an audit is a report in which the findings are summarized in the form of lists of observations, deviations, recommendations, and corrective actions to be taken, as well as a pass/fail statement [30]. The disadvantages of audits are that they are expensive and the least effective static testing type. On the other hand, audits are usually performed because they are mandatory in some context and therefore serve a different purpose than pure defect.

Different People are involved in review process at different stages of the review; their names can change as per the review type. Table 1 shows different stages in a review process (formal). Table 2 shows the persons usually involved in a review process.

Stage	Description
Planning	Selecting the personnel, allocating roles; defining the entry and exit criteria for more formal review types (e.g. inspection); and selecting which parts of documents to look at.
Kick-off	Distributing documents; explaining the objectives, process and documents to the participants; and checking entry criteria (for more formal review types).

Individual Preparation	Work done by each of the participants on their own before the review meeting, noting potential defects, questions and comments.
Review Meeting	Discussion or logging, with documented results or minutes (for more formal review types). The meeting participants may simply note defects, make recommendations for handling the defects, or make decisions about the defects.
Rework	Fixing defects found, typically done by the author.
Follow-up	Checking that defects have been addressed, gathering metrics and checking on exit criteria (for more formal review types).

Table 1: Review Process Stages

Role Name	Role
Manager	Decides on the execution of reviews, allocates time in project schedules and determines if the review objectives have been met.
Moderator	The person who leads the review of the document or set of documents, including planning the review, running the meeting, and follow-up after the meeting. If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests.
Author	The writer or person with chief responsibility for the document(s) to be reviewed
Reviewers	Individuals with a specific technical or business background (also called checkers or inspectors) who, after the necessary preparation, identify and describe findings (e.g. defects) in the product under review. Reviewers should be chosen to represent different perspectives and roles in the review process, and should take part in any review meetings.
Scribe (Recorder)	Documents all the issues, problems and open points that were identified during the meeting. Looking at documents from different perspectives and using checklists can make reviews more effective and efficient, for example, a checklist based on perspectives such as user, maintainer, tester or operations, or a checklist of typical requirements problems.

Table 2: Major Roles in Review

b) *Static Analysis*

Static analysis is a fault-detection technique, which evaluates a system or component based on its form, structure, content, or documentation [33]. Static analysis is used to monitor whether an object under analysis meets uniform expectations around security, reliability, performance, and maintainability. Static analysis is performed to improve and ensure the intrinsic quality of software programs. Static Analysis can reduce defects by up to a factor of six [34]. (See figure 2). The important thing for the tester is to be aware that the static analysis measures can be used as early warning signals of how good the code is likely to be when it is finished.

Static Analysis has got following advantages:

- Early detection of defects prior to test execution [14].
- Early warning about suspicious aspects of the code or design, by the calculation of metrics, such as a high complexity measure [14].
- Identification of defects not easily found by dynamic testing.
- Detecting dependencies and inconsistencies in software models, such as links [14].
- Improved maintainability of code and design.
- Prevention of defects provided that engineers are willing to learn from their errors and continuous improvement is practiced of defects.

Typical defects discovered by static analysis include [17]:

- Referencing a variable with an undefined value;
- Inconsistent interface between modules and components;
- Variables that are never used;
- Unreachable (dead) code;
- Programming standards violations;
- Security vulnerabilities;
- Syntax violations of code and software models.

Static Analysis has got following disadvantages:

- Analyzes simple static properties e.g. metrics collected to assess complexity (size, McCabe)
- Halting problem makes it unaccountable to prove non-trivial properties of a program (in general) [35]
- Static analysis computes an approximation of what would happen during execution, such an approximation is bound to have imperfections [35]
  1. some false positives (noise)
  2. some false negatives (didn't find what it should have)

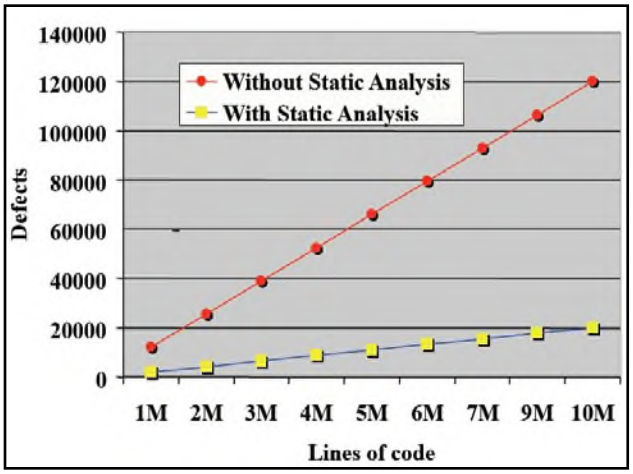

Figure 2: Static Analysis Potential Benefits [34].

Static Analysis is done using following approaches

i. *Coding Standards*

This type of static analysis is conducted to verify adherence to coding standards. The first action to be taken is to define or adopt a coding standard [14]. Usually a coding standard consists of a set of programming rules (e.g. 'Always check boundaries on an array when copying to that array'), naming conventions (e.g. 'Classes should start with capital C') and layout specifications (e.g. 'Indent 4 spaces'). It is recommended that existing standards are adopted. The main advantage of this is that it saves a lot of effort. An extra reason for adopting this approach is that if you take a well-known coding standard there will probably be checking tools available that support this standard. It can even be put the other way around: purchase a static code analyzer and declare (a selection of) the rules in it as your coding standard [36]. Without such tools, the enforcement of a coding standard in an organization is likely to fail. There are three main causes for this: the number of rules in a coding standard is usually so large that nobody can remember them all; some context-sensitive rules that demand reviews of several files are very hard to check by human beings; and if people spend time checking coding standards in reviews, that will distract them from other defects they might otherwise find, making the review process less effective.

ii. *Code Metric*

The code metrics helps to measure structural attributes of the code. When the system becomes increasingly complex, it helps to decide the design alternatives, more so while redesigning portions of the code [14]. Performing static code analysis, usually information is calculated about structural attributes of the code, such as comment frequency, depth of nesting, cyclomatic number and number of lines of code. This information can be computed not only as the design and code are being created but also as changes are made to a system, to see if the design or code is becoming bigger, more complex and more difficult to understand and maintain. The measurements also help us to decide among several design alternatives, especially when redesigning portions of existing code. There are many different kinds of structural measures, each of which tells us something about the effort required to write the code in the first place, to understand the code when making a change, or to test the code using particular tools or techniques.

Experienced programmers know that 20% of the code will cause 80% of the problems [37], and complexity analysis helps to find that all-important 20%, which relate back to the principle on defect clustering. Complexity metrics identify high risk, complex areas.

c) *Code Structure*

There are many different kinds of structural measures, each of which tells us something about the effort required to write the code in the first place, to understand the code when making a change, or to test the code using particular tools or techniques. It is often assumed that a large module takes longer to specify, design, code and test than a smaller one [38]. But in fact the code's structure plays a big part.

There are several aspects of code structure to consider:

- Control flow structure;
- Data flow structure;
- Data structure.

i. *Control Flow Analysis*

A control flow is an abstract representation of all possible sequence of events (paths) in the execution through a component or a system. This aspect of structure reflects the iterations and loops in a program's design [39]. If only the size of a program is measured, no information is provided on how often an instruction is executed as it is run. Control flow analysis can also be used to identify unreachable (dead) code. It is worthy of note that most software code metrics are strongly related to the concept of structure of software code, e.g. number of nested levels or cyclomatic complexity [40]. Control flow goes through basis blocks or nodes of code being executed as an entity, with an entry point in the beginning and only there, and with an exit point in the end and only there. The control goes from the starting point and is transferred from one block to another to the end. It can be very useful to draw a control flow graph from the code (or the requirements). It gives an overview over the decisions points, branches, and paths in a piece of code, and its inherit complexity. Some extracts of control flow graphs are shown here:

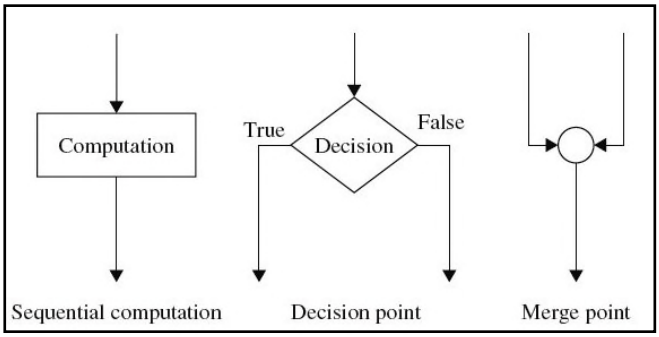

Figure 3: Control Flow Graphs [39]

Static analysis tools may be used to draw control flow graphs of larger pieces of code. It is, however, a good idea to train drawing these, because they give a good understanding of how the code is structured. A control flow graph does not necessarily give an idea of what the code is actually doing, but that is not important anyway. We as testers should concentrate on the structure, not the functionality. Static analysis tools can find faults in the control flow, typically:

- "Dead" code (i.e., code that cannot be reached during execution)
- Uncalled functions and procedures

Both dead code and uncalled functions are quite often found in legacy systems [30]. Undocumented changes and corrections have caused some code to be circumvented but not removed. The cause of the change is since forgotten, but nobody maintaining the code has had the courage (or initiative) to get the unused code removed. Such unused areas do not present direct risks. They do however disturb maintenance, and they may unintentionally be invoked with unknown consequences.

ii. *Data Flow Analysis*

Data-flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program [41]. Data flow structure follows the trail of a data item as it is accessed and modified by the code. Using data flow measures it is shown how the data act as they are transformed by the program. Defects can be found such as referencing a variable with an undefined value and variables that are never used. The normal life cycle for a data item, a variable, consists of the phases:

- Declaration—Space is reserved in memory for the variable value
- Definition—A value is assigned to the variable
- Use—The value of the variable is used
- Destruction—The memory set aside for the value of the variable is freed for others to use

Data-flow analysis derives information about the dynamic behavior of a program by only examining the static code [42].

Static analysis can find anomalies in the data flow in relation to the variable life cycle. Anomalies may, for example, be:

- Use before declaration
- Use before definition
- Redefinition before use
- Use after destruction

If a variable is defined and then redefined before use, is it because the first or the second definition is superfluous, or is it because a usage statement is missing? Some programming languages permit variables to be used before they are declared; in this case the first usage will be treated as an implicit declaration. The phases "definition" and "use" may be repeated many times during the life of a variable.
Two different kinds of usage are defined in data flow analysis (see Figure 4):

- Computation data use = c-use = data not used in a condition [43]
- Predicate data use = p-use = a data use associated with the decision outcome of the predicate portion of a decision statement (predicate = condition = evaluates to T or F) [43]

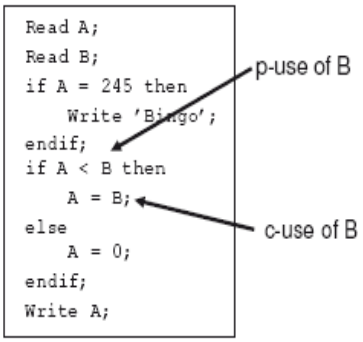

Figure 4: Usage example in Data Flow

To design test cases with the data flow test technique we concentrate on one component and for that component we must [44]:

- Number the lines, if they are not numbered already
- List the variables
- List each occurrence of each variable and assign a category (definition, p-use, or c-use)
- Identify the du-pairs and their type (c-use or p-use)
- Identify all the sub paths that satisfy the pair
- Derive test cases that satisfy the sub paths

When the final test procedures are designed from the test cases, they will of course follow a path in the control flow through the code. There are a few pitfalls that we need to be aware of in connection with data flow analysis. Data items (variables) may be composed of a number of single variables. An

important problem concerning data flow analysis is that sometimes static analysis tools will report a large number of anomalies that are not in fact caused by anything being wrong. These "false alarms" may hide the real problems in the sheer number of alarms.

iii. *Data Structure Analysis*

Data structure refers to the organization of the data itself, independent of the program. When data is arranged as a list, queue, stack, or other well-defined structure, the algorithms for creating, modifying or deleting them are more likely to be well-defined, too. Thus, the data structure provides a lot of information about the difficulty in writing programs to handle the data and in designing test cases to show program correctness. That is, sometimes a program is complex because it has a complex data structure, rather than because of complex control or data flow. The process of identifying data structures requires human interpretation and intuition [45]. Data Structure Analysis is necessary as it greatly affects the performance of the algorithm which accesses it. Efficient storage greatly improves the performance of the overall system.

V. Conclusion

In this paper we presented an in-depth review of static software testing techniques. We observed that using static techniques can be very useful and can yield excellent results if implemented in right manner in SDLC. One thing we have to make sure is that we use correct static technique at correct place (which technique to use, where to use it in SDLC). We should also acquaint ourselves well with details of static techniques (what is technique all about) then only can we select appropriate technique and implement that properly. We should also keep one thing in mind that static techniques and testing (dynamic) are complementary and not opposing verification techniques, so we should use static techniques to supplement testing process; Static techniques are about prevention while dynamic testing is about cure.

References Références Referencias

1. Myers, Glenford J., The art of software testing, Publication info: New York: Wiley, c1979. ISBN: 0471043281 Physical description: xi, 177 p.: ill. ; 24 cm.
2. Hetzel, William C. The Complete Guide to Software Testing, 2nd ed. Publication info: Wellesley, Mass.: QED Information Sciences, 1988. ISBN: 0894352423.Physical description: ix, 280 p.: ill; 24 cm
3. Quadri, S.M.K and Farooq, SU, "Software Testing – Goals, Principles, and Limitations", International Journal of Computer Applications (0975 – 8887) Volume 6– No.9, September 2010.
4. Farooq, SU and Quadri, S.M.K., "Effectiveness of Software Testing Techniques on a Measurement Scale", Oriental Journal of Computer Science & Technology, Vol. 3(1), 109-113 (2010).
5. Chen, T. Y. and Yu, Y. T., "On the expected number of failures detected by sub domain testing and random testing", IEEE Transactions on Software Engineering 22(2):109–119 (1996).
6. Frankl, P. G. and Weyuker, E. J."A formal analysis of the fault-detecting ability of testing methods", IEEE Transactions on Software Engineering 19(3):202–213 (1993).
7. Farooq, SU, Quadri, S.M.K. and Ahmad, Nesar " Decisive Factors for Selecting Software Testing Techniques", In the Proceedings of 6th JK Science Congress, Srinagar, 2010
8. 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society, Digital Object Identifier: 10.1109/IEEESTD.1990.101064
9. Leveson, Nancy."Safeware." Addison-Wesley Publishing Company. 1995
10. Farooq, SU and Quadri, SMK, "Identifying some problems with selection of software testing techniques", Oriental Journal of Computer Science & Technology Vol. 3(2), 266-269 (2010)
11. OULD, M., and UNWIN, C.: 'Testing in software development' Cambridge Univ. Press, 1986
12. Roper, M.,"Software Testing", McGraw-Hill, 1994, 149 pages, ISBN 0-07-707466-1.
13. Sommerville, I, "Software Engineering", Pearson, 2008, 864 pages, ISBN 978-81-317-2461-3
14. Graham, Dorothy, Van Veenendaal, Erik, Evans, Isabel and Black Rex, "Foundations of Software Testing: ISTQB Certification", Thomson, 2006, 258 pages.
15. http://www.softwaretestingmentor.com/types-of-testing/static-testing.php, Accessed on 7 Dec 2010
16. Lerner, Sorin, "Software Reliability Methods", http://cseweb.ucsd.edu/classes/sp06/cse291a/sorin-intro.ppt, Accessed on 11 Dec 2010
17. Sasidhar, K.N., "Static Techniques", http://www.scribd.com/doc/4548375/Static-Testing-Techniques, Accessed on 10 Dec 2010
18. Koza, C. and Puschner, P., " Calculating the maximum execution time of real-time programs.", Real-Time Systems, 1:159-176, 1989.
19. Park, C.Y., "Predicting program execution times by analyzing static and dynamic program paths.", Real-Time Systems, 5:31-62, 1993.

20. Review process, http://www.test-resources.info/iseb-software-testing-review-process.php, Accessed on 17 Dec 2010
21. Weinberg, Gerald M., "The psychology of computer programming", Dorset House Pub., 1998, 292 pages
22. 1028-2008 IEEE Standard for Software Reviews and Audits, IEEE Computer Society, Digital Object Identifier: 10.1109/IEEESTD.2008.4601584
23. Hoffer, J A et.al, "Modern Systems Analysis and Design", Pearson Education India, 2008
24. Limaye, M G, "Software Testing: Principles, Techniques and Tools", Tata McGraw-Hill, 2007
25. Ryber, Torbjörn, "Essential Software Test Design", Fearless Consulting, 2007
26. IEEE Std 1028-1997, "IEEE Standard for Software Reviews, IEEE Computer Society, ISBN 1-55937-987-1
27. Fagan, M.: 'Design and code inspections to reduce errors in program development', IBM Syst. j., 1976, 15, (3), pp. 182–211
28. Gilb, Tom and Graham, Dorothy, "Software Inspection", Addison Wesley Longman, Inc.,1993
29. Types of Review, http://www.geekinterview.com/question_details/70152, Accessed on 22 Dec 2010
30. Hass, A. M.J, "Guide to Advanced Software Testing", ARTECH HOUSE, INC, 2008 ISBN-13: 978-1-59693-285-2
31. Marchewka, Jack T., "Information Technology Project Management", John Wiley and Sons, INC., 2006, ISBN:978-81-265-0804-4
32. Software Audit Review , http://en.wikipedia.org/wiki/Software_audit_review, Accessed on 27 Dec 2010
33. IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.
34. Jones, Capers, "Software quality in 2010: Software Productivity Research LLC A survey of the state of art", www.sqgne.org/presentations/2010-11/Jones-Nov-2010.pdf, Accessed on 5 Jan 2011.
35. Moonen, Leon, "Static Analysis for Software Verification", http://www.uio.no/studier/emner/matnat/ifi/INF4290/v10/undervisningsmateriale/INF4290-StaticAnalysis.pdf, Accessed on 7 Jan 2011
36. http://www.tiobe.com/index.php/content/paperinfo/HowToUseCodingStandards.html, Accessed on 5 Jan 2011
37. Edward, Kit, "Software Testing in the Real World", Pearson Education,2006, ISBN 978-81-7758-572-8
38. Ainapure, B.S. "Software testing and quality assurance", Technical Publications, 2009 - 400 pages
39. Naik, Kshirasagar and Tripathy, Priyadarshi, "Software testing and quality Assurance ", John Wiley & sons, Inc., ISBN 978-0-471-78911-6
40. Pham, Hoang, "Handbook of reliability engineering", Springer-Verlag London, 2003, ISBN: 1-8523-453-3
41. Data flow Analysis, en.wikipedia.org/wiki/Data-flow_analysis, Accessed on 10 Jan 2011
42. Introduction to Data-flow Analysis, http://www.cis.upenn.edu/~cis570/slides/lecture04.pdf, Accessed on 16 Jan 2011
43. Living Glossary ,http://www.testingstandards.co.uk/living_glossary.htm, Accessed on 16 Jan 2011
44. Standard for Software Component Testing, http://www.testingstandards.co.uk/Component%20Testing.pdf, Accessed on 16 Jan 2011
45. Technique - Data Structure Analysis, http://www.gerussell.com/technique_data_structure_analysis.htm, Accessed on 17 Jan 2011