Chapter 4: Static testing

"On November 4th the system did fail. This was caused by a minor programming error that caused the system to "crash", the automatic change over to the back up system had not been adequately tested, and thus the whole system was brought down".

Extract from the main conclusions of the official report into the failure of the London Ambulance Service 's Computer Systems on October 26th and 27th and November 4th 1992.

4.0 STATIC TESTING

4.1 Overview

Static testing techniques is used to find errors before software is actually executed and contrasts therefore with dynamic testing techniques that are applied to a working system. The earlier we catch an error, the cheaper it is, usually, to correct. This module looks at a variety of different static testing techniques. Some are applied to documentation (e.g. walkthroughs, reviews and Inspections) and some are used to analyze the physical code (e.g. compilers, data flow analyzers). This is a huge subject and we can only hope to give an introduction in this module. You will be expected to appreciate the difference between the various review techniques and you will need to be aware of how and when static analysis tools are used.

4.2 Objectives

After completing this module you will:

§ Understand the various review techniques that you can apply to documentation and code.
§ Appreciate the difference between walkthroughs, formal reviews and inspections.
§ Understand how static analysis techniques can detect errors in code.
§ Understand two complexity metrics (lines of code and McCabe's metric).

4.3 REVIEWS AND THE TEST PROCESS

4.3.1 What is a review?

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. The objective of any review is to obtain reliable information, usually as to status and/or work quality.

4.3.2 Some history of reviews

During any project, management requires a means of assessing a measuring progress. The so-called progress review evolved as means of achieving this. However, results of those early reviews proved to be bitter experiences for many project managers. Just how long can a project remain at 90% complete? They found that they could not measure 'true' progress until they had a means of gauging the quality of the work performed. Thus the concept of the technical review emerged to examine the quality of the work and provide input to the progress reviews.

4.3.3 What can be reviewed?

There are many different types of reviews throughout the development life cycle. Virtually any work produced during development can be (and is) reviewed. This includes, requirements documents, designs, database specifications, designs, data models, code, test plans, test scripts, test documentation and so on.

4.3.4 What has this got to do with testing?

The old fashioned view that reviews and testing are totally different things stems from the fact that testing used to be tacked onto the end of the development lifecycle. However as we all now view testing as a continuous activity that must be started as early as possible you can begin to appreciate the benefits of reviews. Reviews are the only testing technique that is available to us in the early stages of testing. At early stages in the development lifecycle we obviously cannot use dynamic testing techniques since the software is simply not ready for testing.

Reviews share similarities to test activities in that they must be planned (what are we testing), what are the criteria for success (expected results) and who will do the work (responsibilities). The next section examines the different types of review techniques in more detail.

4.4 TYPES OF REVIEW

Walk-through, informal reviews, technical reviews and Inspections are fundamental techniques that must be used throughout the development process. All have their strengths and weaknesses and their place in the project development cycle. All four techniques have some ground rules in common as follows:

A structured approach must be for the review process.

Be sure to know what is being reviewed - each component must have a unique identifier.

- Changes must be configuration controlled.
- Reviewers must prepare.
- Reviewers must concentrate on their own specialization.
- Be sure to review the product, not the person.

There must be:

- Ø Total group responsibility.
- Ø Correct size of review group.
- Ø Correct allocation of time.
- Ø Correct application of standards.

Checklists must be used.
Reports must be produced.
Quality must be specified in terms of:

- Ø Adherence to standards.
- Ø Reliability required.
- Ø Robustness required.
- Ø Modularity.
- Ø Accuracy.
- Ø Ability to handle errors.

4.5 REVIEW TEAM SIZE AND COMPOSITION

Problems with small teams must be avoided; bring in extra people, (perhaps use the Testing Team) to bring extra minds to bear on the issues.

Opinion is often divided as to whether or not the author should participate in a review. There are advantages to both scenarios. Since specifications and designs must be capable of being understood without the author present, an Inspection without them tests the document. Another reason for excluding the author from the review is that there should be team ownership of the product and team responsibility for the quality of all the deliverables, maintaining ownership via the author runs against this.

Alternatively, including the author can be a valuable aid to team building. Equally, an author may well be able to clear up misunderstandings in a document more quickly than another team member, thus saving the reviewer valuable time. From a practical perspective however, it is worth remembering that an author is the least likely person to identify faults in the document.

The one person who should not attend reviews is the manager. If, as in some cases, the manager is also a contributor to the deliverables, he should be included but treated the same as other group members. It is important that reviewers are a peer group process.

4.6 TYPE 1, 2 AND 3 REVIEW PROCESSES

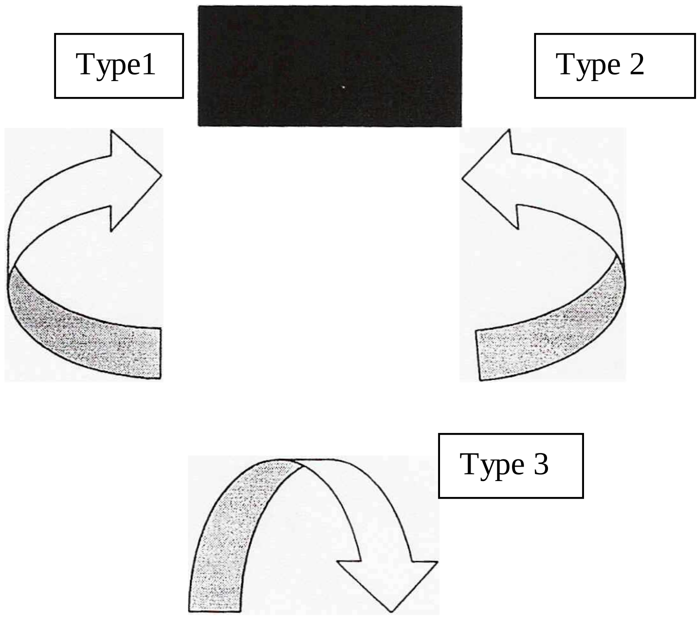
Review process is both most effective and universal test method and management needs to make sure that review process is working as effectively as possible. Useful model for manager is 1,2,3 model.

1, 2, 3 model is derived from work of first working party of British Computer Society Specialist Interest Group in Software Testing and book that came from this work; in Software Development.

Type 1 testing is process of making sure that product (document, program, screen design, clerical procedure or Functional Specification) is built to standards and contains those features that we would expect from the name of that product. It is a test to make sure that produce conforms to standards, is internally consistent, accurate and unambiguous.

Type 2 testing is process of testing to see if product conforms to requirements as specified by output of preceding project stage and ultimately Specifications of Requirements for whole project. Type 2 testing is backward looking and is checking that product is consistent with preceding documentation (including information on change).

Type 3 testing is forward looking and is about specification of certification process and test that are to be done on delivered product. It is asking question - Can we can build deliverables (test material, training material, next stage analysis documentation)?



4.6.1 Make reviews incremental

Whilst use of 1, 2, 3 model will improve review technique, reviewing task can be made easier by having incremental reviews throughout product construction.

This will enable reviewers to have more in-depth understanding of product that they are reviewing and to start construction type 3 material.

4.6.2 General review procedure for documents

Test team will need general procedure for reviewing documents, as this will probably form large part of team's work.

1. Establishing standards and format for document. 2. Check contents list.
3. Check appendix list.
4. Follow up references outside documents.
5. Cross check references inside document.
6. Check for correct and required outputs.
7. Review each screen layout against appropriate standard, data dictionary, processing rules and files/data base access.
8. Review each report layout against appropriate standard, data dictionary, processing rules and files/data base access.
9. Review comments and reports on work andreviews done prior to this review.
10. Documents reviewed will range from whole reports such as Specification of Requirements to pages from output that system is to produce. All documents will need careful scrutiny.

4.6.3 Report on review

Report should categorize products:

Defective	All agreed	Total rework
Defective but soluble	All agreed solution acceptable	Rework and review all material
Possible defect, needs explanation	Some but not all agree	Seek explanation possibly review some of work
Quality issue	Prefer an alternative	Costs compared to standards
Acceptable	All accept	Proceed
Over developed for the budget	Most agree	Proceed but review budgets

Be sensitive to voice of concerned but not necessarily assertive tester in team; this person may well have observed a fault that all others have missed. It should not be that person with load voice or strong personality is allowed to dominate.

4.7 INSPECTIONS

This section on Inspections is based on an edited version of selected extracts from Tom Gib and Dorothy Graham's book [Gib, Graham 93].

4.7.1 Introduction

Michael E. Fagan at IBM Kingston NY Laboratories developed the Inspection technique. Fagan, a certified quality control engineer, was a student of the methods of the quality gurus W. Edwards Deming and J. M. Juran.

Fagan decided, on his own initiative, to use industrial hardware statistical quality methods on a software project he was managing in 1972-74. The project consisted of the translation of IBM's software from Assembler Language to PLS, a high-level programming language assembler. Fagan's achievement was to make statistical quality and process control methods work on 'ideas on paper'. In 1976 he reported his results outside IBM in a now famous paper [Fagan, 1976].

The Inspection technique was built further by Caroline L. Jones and Robert Mays at IBM [jones, 1985] who created a number of useful enhancements:

§ The kickoff meeting, for training, goal setting, and setting a strategy for the current inspection
§ Inspection cycle;
§ The causal analysis meeting;
§ The action database;
§ The action team.

4.7.2 Reviews and walk-through

Reviews and walkthroughs are typically peer group discussion activities - without much focus on fault identification and correction, as we have seen. They are usually without the statistical quality improvement, which is an essential part of Inspection. Walkthroughs are generally a training process, and focus on learning about a single document. Reviews focus more on consensus and buy-in to a particular document.

It may be wasteful to do walkthroughs or consensus reviews unless a document has successfully exited from Inspection. Otherwise you may be wasting people's time by giving them documents of unknown quality, which probably contain far too many opportunities for misunderstanding, learning the wrong thing and agreement about the wrong things.

Inspection is not an alternative to wal1cthroughs for training, or to reviews for consensus. In some cases it is a pre-requisite. The difference processes have different purposes. You cannot expect to remove faults effectively with walkthroughs, reviews or distribution of documents for comment. However, in other cases it may be wasteful to Inspect documents, which have not yet 'settled down' technically. Spending time searching for and removing faults in large chucks, which are later discarded, is not a good idea. In this case it may be better to aim for approximate consensus documents. The educational walkthrough could occur either before or after Inspection.

4.7.3 Statistical quality improvement

The fundamental differences between Inspection and other review methods is that Inspection provides a tool to help improve the entire development process, through a well-known quality engineering method, statistical process control, [Godfrey, 1986].

This means that the data which is gathered and analyzed as part of the Inspection process - data on faults, and the hours spent correcting them - is used to analyze the entire software engineering process. Widespread weakness in a work process can be found and corrected. Experimental improvement to work processes can be confirmed by the Inspection metrics - and confidently spread to other software engineers.

4.7.4 Comparison of Inspection and testing

Inspection and testing both aim at evaluating and improving the quality of the software engineering product before it reaches the customers. The purpose of both is to find and then fix errors, faults and other potential problems.

Inspection and testing can be applied early in software development, although Inspection can be applied earlier than test. Both Inspection and test, applied early, can identify faults, which can then be fixed when it is still much cheaper to do so.

Inspection and testing can be done well or badly. If they are done badly, they will not be effective at finding faults, and this causes problems at later stages, test execution, and operational use.

We need to learn from both Inspection and test experiences. Inspection and testing should both ideally (but all too rarely in practice) produce product-fault metrics and process improvement metrics, which can be used to evaluate the software development process. Data should be kept on faults found in Inspection, faults found in testing, and faults that escaped both Inspection and test, and were only discovered in the field. This data would reflect frequency, document location, security, cost of finding, and cost of fixing.

There is a trade-off between fixing and preventing. The metrics should be used to fine-tune the balance between the investment in the fault detection and fault prevention techniques used. The cost of Inspection, test design, and test running should be compared with the cost of fixing. The faults at the time they were found, in order to arrive at the most cost-effective software development process.

4.7.5 Differences between Inspection and testing

Inspection can be used long before executable code is available to run tests. Inspection can be applied mud earlier than dynamic testing, but can also be applied earlier than test design activities. Test can only be defined when a requirements or design specification has been written, since that specification is the source for knowing the expected result of a test execution.

The one key thing that testing does and Inspection does not, is to evaluate the software while it is actually performing its function in its intended (or simulated) environment. Inspection can only examine static documents and models; testing can evaluate the product working.

Inspection, particularly the process improvement aspect, is concerned with preventing software engineers from inserting any form of fault into what they write. The information gained from faults found in running tests could be used in the same way, but this is rare in practice.

4.7.6 Benefits of Inspection

Opinion is divided over whether Inspection is a worthwhile element of any product 'development' process. Critics argue that it is costly; it demands too much 'upfront' time and is unnecessarily bureaucratic. Supporters claim that the eventual savings and benefits outweigh the costs and the short-term investment is crucial for long-term savings.

In IT-Start's Developers Guide (1989), it is estimated that 'The cost of non-quality software typically accounts for 30% of development costs and 50% to 60% of the lifecycle costs'. Faults then are costly, and this cost increases the later they are discovered. Inspection applied to all software is arguably the prime technique to reduce defect levels (in some cases to virtually zero defects) and to provide an increased maturity level through the use of Inspection metrics. The savings can be substantial.

Direct savings

Development productivity is improved.
Fagan, in his original article, reported a 23% increase in 'coding productivity alone' using Inspection [Fagan, 1976, IBM Systems Journal, p 187]. He later reported further gains with the introduction of moderator training, design and code change control, and test fault tracking.

Development timescale is reduced.
Considering only the development timescales, typical net savings for project development are 35% to 50%.

Cost and time taken for testing is reduced.
Inspection reduces the number of faults still in place when testing starts because they have been removed at an earlier stage. Testing therefore runs more smoothly, there is less debugging and rework and the testing phase is shorter. At most sites Inspection eliminates 50% to 90% of the faults in the development process before test execution starts.

Lifetime costs are reduced and software reliability increased.
Inspection can be expected to reduce total system maintenance costs due to failure reduction and improvement in document intelligibility, therefore providing a more competitive product.

Indirect savings

Management benefits.
Through Inspection, managers can expect access to relevant facts and figures about their software engineering environment, meaning they will be able to identify problems earlier and understand the payoff for dealing with these problems.

Deadline benefits.

Although it cannot guarantee that an unreasonable deadline will be met, through quality and cost metrics Inspection can give early warning of impending problems, helping avoid the temperature of inadequate correction nearer the deadline.

Organizational and people benefits.

For software professionals Inspection means their work is of better quality and more maintainable. Furthermore, they can expect to live under less intense deadline pressure. Their work should be more appreciated by management, and their company's products will gain a competitive edge.

4.7.7 Costs of Inspection

The cost of running an Inspection is approximately 10% - 15% of the development budget. This percentage is about the same as other walkthrough and review methods. However, Inspection finds far more faults for the time spent and the upstream costs can be justified by the benefits of early detection and the lower maintenance costs that result.

As mentioned earlier, the costs of Inspection include additional 'up front' time in the development process and increased time spent by authors writing documents they know will be Inspected. Implementing and running Inspections will involve long-term costs in new areas. An organization will find that time and money go on:

- Ø Inspection leading training.
- Ø Management training.
- Ø Management of the Inspection leaders.
- Ø Metric analysis.
- Ø Experimentation with new techniques to try to improve Inspection results.
- Ø Planning, checking and meeting activity: the entire Inspection process itself.
- Ø Quality improvement: the work of the process improvement teams.

The company may also find it effective to consider computerized tools for documentation and consistency checking. Another good investment might be improved meeting rooms or sound insulation so members of the Inspection team can concentrate during checking.

4.7.8 Product Inspection steps

- The Inspection process is initiated with a request for Inspection by the author or owner of a task product document.
- The Inspection leader checks the document against entry criteria, reducing the probability of wasting resources on a product destined to fail.
- The Inspection objectives and tactics are planned. Practical details are decided upon and the leader develops a master plan for the team.
- A kickoff meeting is held to ensure that the checkers are aware of their individual roles and the ultimate targets of the Inspection process.
- Checkers work independently on the product, document using source documents, rules, procedures and checklists. Potential faults are identified and recorded.
- A logging meeting is convened during which potential faults and issues requiring explanations, identified by individual checker, are logged. The checkers now work as a team aiming to discover further faults. And finally suggestions for methods of improving the process itself are logged.
- An editor (usually the author) is given the log of issues to resolve. Faults are now classified as such and a request for permission to make the correction and improvements to the product is made to the document's owner. Footnotes might be added to avoid misinterpretation. The editor may also make further process improvement suggestions.
- The leader ensures that the editor has taken action to correct all known faults, although the leader need not check the actual corrections.
- The exit process is performed by the Inspection leader who uses application generic and specific exit criteria.
- The Inspection process is closed and the product made available with an estimate of the remaining faults in a 'warning label'.

Exercise

Comparison between Various Techniques

Take a few moments to complete the following table.

TECHNIQUE	Primarily Used for	INVOLVES	LED BY	FORMALITY
Walkthroughs	Education; Dry runs	Peer group	Author	Fairly informal
Informal reviews	Fault detection	Anyone	Individual	Undocumented; Cheap. Useful
Technical reviews	Fault detection	Peer group; Technical experts; No managers	Individual	Formal; Documented; No metrics kept
Inspections	Fault detection; Process improvement	Defined roles	Trained moderator (not the author)	Very formal Rules, checklists Metrics kept

4.9.2 McCabe's complexity metric

McCabe's complexity metric is a measure of the complexity of a module's decision structure. It is the number of linearly independent paths and therefore, the minimum number of paths that should be tested. The metric can be calculated in three different ways. The number of decisions plus one, the number of 'holes' or connected regions (bearing in mind that there is a fictitious link between the entry and exit of the program), or thirdly the equation:

$M=L-N+2P$

Where: L = the no. Of links in graph

N = the no. Of nodes in the graph

P = the no. Of disconnected parts of the graph

Despite its simplicity the McCabe metric is based on deep properties of program structure. The greatest advantage is that it is almost as easy to calculate as the 'lines of code' metric, and results in a considerably better correlation of complexity to faults and the difficulty of testing.

McCabe advises partitioning programs where complexity is greater than la and this has been supported by studies such as Walsh who found that 23% of the routines with an M value of greater than 10 contained 53% of the faults. There does appear to be a discontinuous jump in the fault rate around M = 10. As an alternative to partitioning, others have suggested that the resources for development and testing should be allocated in relation to the McCabe measure of complexity, giving greater attention to modules ht exceed this value.

The weakness of the McCabe metric is found in the assumption that faults are proportional to decision complexity, in other words that processing complexity and database structure, amongst other things, are irrelevant. Equally it does not distinguish between different kinds of decisions. A simple "IF-THEN-ELSE" statement is treated the same as a relatively complicated loop yet we intuitively know that the loop is likely to have more faults. Also CASE statements are treated the same as nested IF statements which is again counter intuitive