

PHILOSOPHY and PRACTICAL IMPLEMENTATION of STATIC ANALYZER TOOLS

White paper describing the philosophy and practical implementation of
tools to perform static analysis on source code.

December, 1998

Dirk Giesen, QA Systems Technologies BV

TABLE OF CONTENTS

1. Introduction	3
2. Static Analysis, a General Introduction	4
Dimensions of Testing	4
<i>Source code versus Executable code</i>	4
<i>Software development phases</i>	4
<i>White box versus Black box</i>	4
<i>Functional and Performance</i>	4
Dynamic Testing and Intrinsic Quality	4
<i>Buying a second hand car – Analogy</i>	5
Static Analysis	5
<i>Coding Standard</i>	6
3. Static Analyzer Tool, a Concept	7
4. Integrating the Static Analyzer in the Software Development Process	8
Example Software Development Process	8
Static Analysis during Development and Unit Testing	8
Static Analysis during Integration and System Testing	9
Static Analysis during Acceptance Testing and QA	9
Absolute and Relative Checking	9
5. Commercial Solution	10
Have the Knowledge available	10
Have the Technology available	10
Can support Cultural Change	10

1. Introduction

Bloor Research Ltd., a well known, UK based, consultants and research company for Information Technology stated in their CAST Tools report of 1996 [Bloor 1996] that 60% of the software faults and errors that were found in released software products could have been detected by means of static analysis.

Dr. Thomas Liedtke and Dr. Christian Ebert of Alcatel Sel AG in Stuttgart, Germany explained the tight relation between static analysis and the reduction of support efforts on released software products [Liedtke, 1995].

This white paper describes the philosophy of static analysis and the justification for the adaptation of commercial static analyzer tools. The targeted audience are managers of companies that develop safety critical and high integrity software programs as a part of the products that are commercially produced (i.e. Defense and Aerospace, Telecommunication, Consumer Electronics, Medical Systems and Automotive branches) or that support their primary business processes (i.e. Financial, Retail, Airline Carrier and Government branches).

The white paper has the following set-up;

- Static Analysis, a General Introduction,
- Static Analyzer Tool, a Concept
- Integrating the Static Analyzer in the Software Development Process
- Commercial Solution

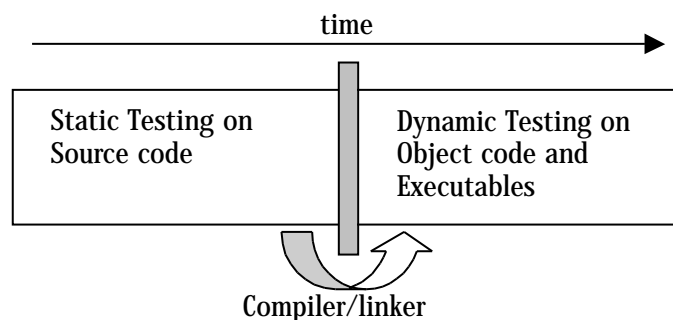
These topics will be covered in the next paragraphs.

2. Static Analysis, a General Introduction

Dimensions of Testing

Source code versus Executable code

Static analysis (also referred to as *static testing*) covers the area of performing all kinds of tests on the source code of software programs. *Dynamic testing* (the counterpart of static testing also known as runtime testing) covers the area of performing all kinds of tests on the object code and executables that is determined from the source code. The difference between these types of testing is determined by the status of the software program. In this context the status of a software program is changed by the usage of the compiler/linker.



Before compilation the testing is targeted to improve and ensure the quality of the source code. After compilation the testing is targeted to improve and ensure the quality of the object code and the executable. Both areas of testing are targeted to improve and ensure the quality of the software program.

Software development phases

Another dimension of testing is seen during the traditional phases of a software development process. Testing is for instance done by individual developers (module- or unit testing), by integrators (integration- and/or system testing), by end-user representatives (Beta- and/or acceptance testing) and before releasing a software product by quality managers (quality assurance testing). In general it can be stated that companies perform all kinds of testing during the software development phases. During every phase static testing as well as dynamic testing can be performed. Paragraph 4 discusses the adaptation of static testing in an example software development process.

White box versus Black box

White box and Black box testing are terms that refer to the knowledge the tester has about the internal structure of the tested system. The first, white box testing offers the tester all the knowledge, black box testing does not. Static testing is white box testing. Most areas of dynamic testing are black box testing.

Functional and Performance

The areas of for instance functional testing and performance testing are mostly performed in the dynamic testing area. These tests are done with executable code.

Dynamic Testing and Intrinsic Quality

At the moment of compilation the internal (static) quality of the software program is frozen. In the context of this paper the frozen internal quality is referred to as the *intrinsic quality* of the software program.

Dynamic testing is often performed by means of CAST (Computer Aided Software Testing) tools like debuggers, memory leakage detectors, code coverage analyzers, record and playback tools, etc. In some cases where the software program is embedded code

(implemented on a target hardware device) dynamic testing has to be non-intrusive and is therefore very difficult and expensive (the usage of simulators etc. is often required).

Independently of how much effort and money is put in dynamic testing, a software program with an unknown intrinsic quality will remain to have unknown intrinsic quality. When faults and errors are detected and corrected the intrinsic quality of the software program is likely to grow, but to what level remains to be unknown. To test the correction the dynamic test cycle has to be restarted (also to test if no other faults or errors have been introduced). This edit-compile/link-debug cycle will continue until the amount of errors and faults have reached an acceptable level. After that the software program is “promoted” to the next phase of development or to be released to the customer. Still with unknown intrinsic quality!

The traditional edit-compile/link-debug cycle is highly inefficient in case the intrinsic quality of the software system is unknown. Furthermore software programs with unknown intrinsic quality will not ensure the reliability that is intended to be proven by the dynamic testing.

These are the fundamentals of this white paper. It justifies the necessity to invest in the area of defining intrinsic quality and performing static analysis. The next section will address the definition of intrinsic quality. But first an example in our daily life is given.

Buying a second hand car – Analogy

A person has the intention to buy a nice second hand car, preferably the good looking silver one he saw at the dealer in town. His only problem is that the car is quite expensive. But at a certain day he has the money available to buy it and he makes an appointment with the dealer. After walking round the car a few times, sitting in it, shifting the gear, kicking the tires and listening to the never ending sales speech of the salesman he eventually asks him if he can take it along for a test ride. The buyer wants to know how the car drives before he buys it. The salesman agrees, and after an hour the buyer comes back, and buys the car. In this situation the buyer bought a car after testing it “dynamically”; the car looked good, had no visible rust and the tires etc. looked all right to him. On top of that it drove very well. All of the things he could do at that moment reside in the area of dynamic testing. But, at the moment he bought the car he hadn't the slightest idea what its intrinsic quality was. How does the engine and the gearbox look like internally, and how long would the exhaust pipe and the brake system last. The answers to all these kinds of questions would give the buyer much more information than he had now. Probably he will find out later, after using the car for a few months.

Static analysis in the car-world is often done by technical inspections etc.

Static Analysis

As said before static analysis covers the area of performing all kinds of tests on the source code of software programs. Static analysis is performed to improve and ensure the intrinsic quality of software programs.

Examples of static analysis are; source code reading, code reviews and also compilation (which is a part of static testing). These examples target to test the quality of the source code. To make sure that the above examples are performed in an effective and efficient way static testing must be based on (company specific) programming *rules* and *guidelines*. Rules describe what is mandatory and guidelines describe what is recommended.

The rules and guidelines are described in a *coding standard*. A coding standard therefore defines the way a programmer must and should program. The result of code reading and code review sessions is source code that adheres to the coding standard. This source code has a defined (known) intrinsic quality and is therefore ready to be compiled.

Coding Standard

The rules and guidelines as described in a coding standard define the level of intrinsic quality that a company wants to achieve. The intrinsic quality definition can be divided into several quality area's. These areas can directly be translated from the quality improvement issues that are (or should be) defined by the senior management of the organization. Improving quality often means improving reliability as seen before, but it also means (more or less indirectly) improving the area's of maintainability, testability, portability and readability of the software programs. These five quality area's will be addressed in the coding standard, and every area will have its specific rules and guidelines, translated in tests (checks) that can be performed on the source code.

Measuring maintainability means measuring the results of all the checks that are related to the maintainability aspect of the coding standard. After measurement it will be clear if improvement is necessary to adhere to the maintenance part of the coding standard. This method of measurement and improvement is valid for every separate quality area.

It is explained below what results can be achieved by improving;

Reliability Will result in a lower amount of static bugs in the software program, and therefore higher customer satisfaction. Besides it results in less develop/test cycles, shorter dynamic testing time and therefore shorter delivery time of releases.

Maintainability Will result in better understanding of the code by others, shorter maintenance times, shorter renewed dynamic testing time and therefore shorter delivery time of new releases.

Testability Will result in shorter dynamic testing time and therefore shorter delivery of releases. Besides that it will make dynamic testing more effective, one knows how much testing is required, and what modules should be tested more extensive. This all results in improved reliability after dynamic testing.

Portability Will result in flexibility when the software program has to run on different hardware platforms. The amount of work for porting the code will be much less. This results in a shorter delivery time of new releases.

Readability Will result in better understanding of the code by others and therefore shorter review times and clearer code reading.

Improvement of all these aspects will result in higher customer satisfaction, shorter delivery times and decreasing of costs. But if improvement is required one must be able to measure. To measure is to know, and to know is the means to improve.

The next paragraph will cover the topic of automated code inspection, a means to partly automate the code reading and review sessions.

3. Static Analyzer Tool, a Concept

A static analyzer Tool is a tool that automates the code inspection process and offers all kinds of intrinsic quality information to the user.

The idea behind automated code inspection is that a lot of the tasks done during the code reading and reviewing; controlling the produced source code against the coding standard are automated.

This implies that certain parts of the manual inspections will be done by means of a commercially used and supported tool, that will be a part of the software development process (refer also to paragraph 4.)

A direct result of this approach will be that the time required to perform code reading and reviewing will decrease. Another result is that, because of automation all code can be inspected. This coverage practically never is reached with manual inspections.

For this purpose the static analyzer has to be able to parse and understand the target source code as if it were a compiler. This is the first requirement of the analyzer. For a certain language, it has to be able to understand the syntax of that language. This feature is the basic requirement of the tool, it is also the basic requirement for a compiler. To be able to test the parsed source code against the defined (company specific) rules and guidelines - as defined in a coding standard - it has to offer the ability to perform checks on the parsed code. Both elements are necessary for automating the code inspection functionality. If on top of that the tool offers user friendly presentation and analysis capabilities based on the checks that are implemented, a static analyzer is created.

By means of this static analyzer tool developers can control the level intrinsic quality of every line of source code against the coding standard on an automated way, before the code is compiled. With this approach intrinsic quality control will be done on the earliest stage of coding possible.

In practice this could mean that no code will be compiled without conformance to the company's coding standard. And in fact this is the goal of static testing, because every dynamic test situation will be done on software programs that have defined intrinsic quality! Paragraph 4 will describe how this feature should be embedded in the software development process.

4. Integrating the Static Analyzer in the Software Development Process

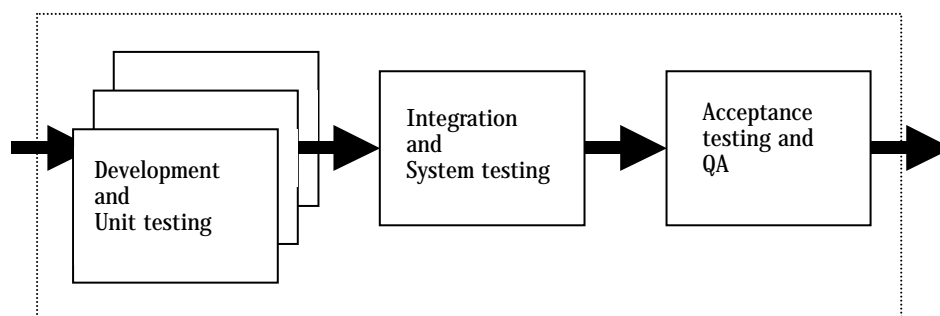
The static analyzer can be embedded in the software development process in several stages, for instance;

- Development and Unit testing
- Integration and System testing
- Acceptance testing and QA

As stated in paragraph 2 static testing can be done in several stages, right before the software system will be built (compiled/linked) for the purposes of that stage.

Example Software Development Process

The following process phases can be seen in an example software development environment.



Input to this software development process are requirements, bugfixes, change requests etc. (plus its technical specifications) and output is a released product.

The process between the dashed lines has the following phases; Multiple developers perform project specific development work on certain parts of the software program. One developer is for example maintaining code, the other is fixing bugs and a third developer is adding functionality. At a certain moment all changes have to be integrated for the new release of the software program, and system testing has to be performed. When system testing has finished (customer) acceptance testing will be done, and the quality manager can perform quality inspections (metrics, comparisons etc.). All actions will be taken in this order before the software program can be released to the customer.

For every process phase the static analyzer should be used. The developer that fixes a bug should use the static analyzer before the compilation for the dynamic unit test is performed. Only if this is performed successfully the fix can be “checked-in” for the Integration phase. The integrator (or build manager) is responsible for combining all the checked-in sub-systems of the software product in order to produce one release for the system test. The integrator again should use the static analyzer before the system will be compiled for performing the dynamic system tests. After the tested system has entered the acceptance test phase the QA manager should use the static analyzer to perform quality analysis before the software product will finally be offered to the acceptance testers, and before it will be released.

Static Analysis during Development and Unit Testing

During the unit testing phase the static analyzer can be integrated in the make facility and/or the check-in function. If the developer intends to build the sub-system for testing, the static analyzer is implicitly (under water, or interactively) used. In case of conformance to the codingstandard, the make utility will finish the build automatically. If not the developer first has to reach conformance first. It is also possible that the static analyzer is

run implicitly with the check-in function. After the “promotion” the developer can assure conformance to the coding standard for his sub-system.

Static Analysis during Integration and System Testing

During the system testing phase the static analyzer can be integrated in the integration make facility implicitly. The integrator combines the checked-in sub-systems and starts making the new release. Implicitly the static analyzer is started (under water or interactively) for conformance checking. In case of conformance to the coding standard, the make will finish the build automatically. If not, the integrator probably has to reject the non-conform sub-system and finish the build with the conforming sub-systems.

With this step the Integrator can assure conformance to the coding standard of the new release. After this “promoting” the build system no code changes can be made anymore.

Static Analysis during Acceptance Testing and QA

Finally when the software program is promoted from the integration and system test phase, the quality manager can perform quality analysis using the static analyzer. It can be very well possible that metric information is generated for further analysis. It also is possible to retrieve and combine information to compare the overall intrinsic quality of this release with the overall intrinsic quality of earlier releases (or even with other releases of different software programs). The goal here is that this data is used for further project planning and fine tuning of the software development process and offering feedback to the people who work on it.

The information acquired at this stage is also very useful for reviewing the level of quality of the current release of the coding standard. Maybe it is necessary to modify or improve the intrinsic quality rules and guidelines of this standard.

Absolute and Relative Checking

The level on which conformance will be achieved can differ from stage to stage and can also differ through time. For instance companies could have large pieces of existing source code that are not conforming a newly introduced coding standard. In these situation it will be very difficult to state that at a certain stage in the software development process the code has to be full conforming to the standard. In cases were software programs are developed from scratch this measurement is much easier.

Absolute checking means that every release of a (sub-)system has to conform the entire coding standard. Relative checking means that is every release of a (sub-) system has to have improved quality compared with the former release of that (sub-) system. In relative checking environments one could state that every new release should for instance have one warning less than the former release. One could also say that a new release should always have zero reliability warnings and fewer other warnings than the former release.

In situations where existing (non conforming) code is used relative checking will be performed.

The quality manager then could define the level of quality that should be reached at the end of a project or at a certain release.

5. Commercial Solution

The introduction of the concept of static analysis, using automated tools requires more than just the implementation of the tool. Three tightly related requirements to a supplier are required in order to make the introduction successful;

- Have the Knowledge available
- Have the Technology available
- Can support Cultural Change

Have the Knowledge available

The knowledge of the supplier has to address the area static analysis. The supplier of the tool has to have knowledge and experience with the related programming languages, the performance software audits and zero-level assessments, with the technique of defining coding standards and with the ways software development processes work.

Have the Technology available

The supplier has to have the technology available to produce commercial available (whether or not custom made) static analyzers for programming languages. The supplier also needs to have the means to implement the product as described in this white paper, and to support it commercially for a practically unlimited period.

Can support Cultural Change

The organization that will start using the concept of static analysis, and the related automated inspection tools have to be aware why this kind of testing is required and what can be achieved doing so. Therefore the supplier has to be able to explain these topics to the organization – from management to the developers, by means of conceptual training- and supporting sessions.