An Approach to Code Analysis in Students' Diploma Projects

Ognian Nakov[1] and Daniela Gotseva[2]

Abstract – **In this paper we focused on three ASA tools for C/C++ programs. After short ASA tools description, we examine the faults identified by ASA tools, manual inspections and system failure testing. Additionally we categorize raw output from ASA tools that help us to make conclusions about the efficiency of static analysis for software fault detection in students' projects. We analyze 700 student diploma projects during the last 13 years.**

Keywords – **Quality code analysis, Bug reviews C/C++ programming, Software engineering.**

I. Introduction

One of the possible fault-detection techniques is static analysis. This analysis concerned evaluating a system and its components based on a code, forms, data structures, documentation without program execution. Inspections are an example of static analysis that relies on code rewires. The other possibility is using of automated tools for that purpose. These tools help us to reduce code errors such as runtime exception, redundant code, inappropriate use of variables, division by zero and potential memory leaks. We defined the use of automated static analysis (ASA) tools and Inspections that mean manual code review. ASA may help software engineers to fix faults in software test process. In this paper we report the result of using static analysis procedures as a fault detection technique in students' projects.

The study was a research that analyzed 700 students' diploma projects in Technical University of Sofia, Computer System Department. Since 1996 we collect, inspect and analyze by ASA tools over 6 million lines of code (LOC). In our research we examine software projects written in C/C++ that underwent various combinations of inspection and ASA. We used Goal-Question-Metric (GQM) to motivate and focus our data collection and analysis.

Structure of the present paper is as followed: automated static analysis tools are described in section 2, section 3 covered data collection, tests and their results are shown in section 4, and conclusion is given in section 5.

II. Automated Static Analysis Tools

ASA can be used as an added fault-detection filter in software development process. ASA tools automate the identification of certain types of anomalies, as discussed above, by scanning and parsing source text of a program to look for a fixed set of patterns in the code. ASA utilizes control flow and data flow analysis, interface and information flow analysis of the source code. There are some errors that are never detected by ASA tools [3, 4]. Additionally every ASA tool generates different, sometimes no overlapping, errors [5].

The important benefit of ASA is that they do not require code execution for bug tracking. In this case ASA is opposite to the language compilers. C language doesn't have strong type checking and the compiler can omit some errors. They can be trapped by ASA tools.

There are range of ASA tools and services deployed for C/C++ programs. One of these products is FlexeLint [1]. It will check C/C++ source code and find bugs, glitches, inconsistencies, non-portable constructs, redundant code, and much more. It looks across multiple modules, and so, enjoys a perspective your compiler does not have. FlexeLint is a Unix-based tool and there is also Windows-based version: PC-Lint.

The other good ASA tool is Reasoning [2]. Its services boost the productivity of development teams by uncovering security vulnerabilities and reliability defects before they become costly problems. This tool finds defects in C/C++ applications. Reasoning's Discovery Mapping Analytics Service (DMA) is an analysis of users' source code using static analysis techniques and Reasoning's expertise in identifying Implementation Defects. The tool process users' code through various static analysis engines and analyze the results to benchmark the quality level of each component of the application.

The followed metrics are embedded into Reasoning:

- Metrics for Prioritization – Using Reasoning's Discovery Mapping Analytics service, the engineering managers will have the information to recognize and prioritize the most problematic application modules and direct engineering resources for improved effectiveness and more predictable results.
- Metrics for Risk Management – Reasoning's Discovery Mapping Analytics service maps implementation errors that cause system crashes or security vulnerabilities. With this mapping, the user can significantly enhance risk assessment and management capability, with improved prioritization, calibration and predictability. Improvements in process for discovering defects will improve the ability to manage the risk of killer defects getting to customers.
- Metrics for Code Integrity Benchmarking – Reasoning's Discovery Mapping Analytics service is a unique addition to your management dashboard, providing quality benchmarks. Reasoning's DMA

[1]Ognian Nakov is with the Faculty of Computer Systems and Controls at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mail: nakov@tu-sofia.bg.

[2]Daniela Gotseva is with the Faculty of Computer Systems and Controls at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, e-mail: dgoceva@tu-sofia.bg.

service provides an unbiased, third-party assessment of code reliability and vulnerability. The DMA results show how users' code characteristics compare to those of some of the world's largest development organizations. Once measured, the user can compare: results of a quality initiative over the lifecycle steps of an application, use of different methodologies, team structures, training protocols, etc.

Reasoning DMA service provides metrics, giving the dashboard measurements that let manage quality initiatives effectively. Some of the possible error tracking with Reasoning are: NULL pointer assignment, out of array access bounds, memory leaks, bad deallocation and uninitialized variables.

III. Data Collection

We collected and analyzed faults into 700 students' diploma projects during last 13 years. Data analysis consists of faults for above 6 million LOC written in C/C++, developed by bachelor and master of computer science students. In our analysis we discussed three cases: projects with inspections, projects with ASA tools testing at first step and inspections at second, and projects with inspections first and ASA tool testing after that. For our research purpose we used different ASA tools, as shown in Table I. In this research the number of errors found by FlexeLint is four times than errors found by Reasoning. Therefore we based our analysis on FlexeLint results.

We denoted students' diploma projects as SP and defined two product versions: SP1.0 that have only handle inspections, SP1.1 that has ASA tool testing and inspections too.

TABLE I
DATA ANALYSIS

Project Version	ASA	Inspections
SP1.0	Not performed	Yes
SP1.1	FlexeLint, Reasoning	Yes

IV. Results

In this section we provided the achieve results. We divided them into five categories, given in subsections A - D. The basic goal is to determine whether ASA tools can help to students to improve their programming techniques and to see what kind of errors are most frequently occurred in students' projects. Each section started with a base question and short explanation of a used metrics to answer to the question in it. Using of GQM in each section help us to collect and analyze data. All of data analysis, implication of it are posted and discussed.

A. Student's Diploma Project Quality

The question that is important in this section is:

"Will a student's diploma project be of higher quality if ASA tool is using in development process?"

To answer to this question we used:

- quantity of defects found by system testing
- quantity of defects found by our testing

Divided by churned thousand lines of code (KLOC).The results were shown in Table II. Project quality comparison based on a number of total failures per churned KLOC (KLOCC). In the table we used SP1.0 as a baseline project for comparison. We normalized the failures per KLOCC metrics relatively to the SP1.0 projects. This gives us relative quality of SP projects.

There is a wide variance in the relative quality of the projects. As a result, *our analysis didn't provide conclusive results about whether ASA tools will help to increase the SP projects quality*.

TABLE II
RELATIVE SP DIPLOMA PROJECTS QUALITY

Project	Relative Quality (failures/ KLOCC)	Process step 1	Process step 2
SP1.0	1.0	Inspections	
SP1.1	1.32	ASA	Inspections

B. Fault Detection Yield

The question that we asked in this section is:

"How effective is ASA at detecting faults compared with inspections and testing?"

To answer to this question we used:

- quantity of ASA faults
- quantity of inspection faults
- quantity of test failures

Fault detection yield (FDY) refers to the percentage of defects, present in the code at the time of fault detecting practice [6, 9, 12]. FDY can't be precisely computed until the project is used extensively by the users. This measure decreased as more bugs are found. Additionally we calculated defect removal efficiency (DRE) [7] as a measure of how well bugs are removed. Software defect removal efficiency is percentage of total bugs eliminated in the code. High level of defect removal efficiency is corresponding to high level of user satisfaction.

TABLE III
DEFECT REMOVAL YIELD

Project Version	Phase ASA	ASA (%)	Inspe-ctions (%)	Test (%)	DRE (%)
SP1.0	Not performed	Not preformed	42.31	96.73	98.10
SP1.1	Prior to inspections	35.00	20.48	98.18	99.05

For SP1.1 ASA performed prior to inspection. No ASA test is done for SP1.0. The results are shown in Table III. Research indicates that the user can receive high quality

project, if the result is greater than 95% [7, 8, 13, 14]. The value of DRE is higher than industrial benchmark and this fact indicated high quality of software project.

These results indicate that defect removal yield of ASA isn't significantly different from that of inspections. The defect removal yield of execution-based testing is two times higher than that of ASA and therefore may be more effective at finding the defects.

C. Classes of Faults and Failures

The main questions discussed in this section are:

"What classes of faults and failures are most often detected by ASA, by inspection and by system testing? What classes of defects are escaped to the customers?"

To answer to them we used Orthogonal Defect Classification (ODC) [10, 11]. ODC is a scheme to capture the semantics of each software defect quickly. It is the definition and capture of defect attributes that make mathematical analysis and modeling possible. Analysis of ODC data provides a valuable diagnostics method for evaluating the various phases of the software life cycle (design, development, test and service) and the maturity of the product. ODC makes it possible to push the understanding and use of defects well beyond quality.

We measured:

- quantity of ASA faults by ODC type
- quantity of inspection faults by ODC type

We counted faults according to ODC type classification. Here we will present the results from each metrics.

ASA Faults

Each fault was documented with a problem explanation and detailed information such as: description, location, precondition, impact, severity, suggestion and code fragment. Then every fault was manually classified according to ODC types. Finally faults were counted and percentages are calculated. A summary of the results are shown in Table IV. Only FlexeLint is included in comparison.

TABLE IV
ASA FAULTS CLASSIFICATION ACCORDING TO ODC TAXONOMIES

ODC taxonomy	SP1.1 (%)
Assignment	
• All tools	80
• FlexeLint including	61
Checking	
• All tools	20
• FlexeLint including	50
Other ODC taxonomies	0

The result shown in Table IV indicated that ASA tools are effective for identifying two ODC types: Assignment and Checking. Checking defects are happen in low level design or coding phases and Assignments were occurred only in coding phase. These problems are due to logical than static analysis.

Inspection Faults

All inspection faults are documented in text file. Every inspection file was manually created and classified according ODC types. The result of this classification is shown in TableV. Note that in handle inspection some additional properties are documented: readability of code, maintainability, naming convention, coding standards and programming style. These comments are approximately 25% of statements in inspection records. They are not included in ODC taxonomies.

The results show that inspection identifies Algorithm, Documentation and Checking faults. Approximately 85% of all faults belong to these three categories and the distribution is constant regardless of whether or not ASA tests are performed.

TABLE V
INSPECTION FAULTS CLASSIFICATION

Defect Type	SP1.0 (%) No ASA	SP1.1 (%) After ASA
Algorithm	30.40	38.12
Documentation	29.01	35.13
Checking	26.62	17.85
Assignment	6.33	5.02
Function	1.26	1.74
Interface	2.21	1.02
Build/Package/Merge	4.17	1.12

D. Programming Errors

The questions discussed in this section are:

"What kind of programmer errors is most frequently identified by ASA? How often does ASA find these errors?"

To answer to them we used:

- quantity of ASA faults by defect type

To avoid differences in defect types among different tools, only one ASA is used. We chouse FlexeLint, because it identified most defect types from examined ASA tools. Then we merged the same and very similar static analysis fault to performed result aggregation. All data are ranked with most frequently faults at the top of the list. FlexeLint can detect more than 800 bugs, but only 33 were found in students' diploma projects. The faults were given one of the following severity levels, based on potential failure:

- **Critical** – this fault can cause application dump, service outage, system reboot;
- **Major** – this fault can cause segmentation fault, memory leaks, resource leaks, data corruption;
- **Minor** – this fault may result in unexpected behavior;

The results are consistent with the 80-20 rule/Pareto Principle, i.e. a great majority of the faults identified by few key programmer errors, as shown in Table VI. "Possible use of NULL pointer" is most frequently error, identified by ASA – approximately 47% of all faults. About 92% of faults are focused on 10 fault types. To improve the code quality we will used this information in future educations to point the students what kind of programmer errors are most often happen in their projects.

There are some additional limitations on this research. First ASA tool outputs are screening. Second, assigning of

severity level is a manually operation and is subjective.

TABLE VI
PARETO EFFECT IN ASA FAULTS

	% all faults	% critical faults	% major faults	% minor faults
Top 1 fault: *Possible use of NULL pointer*	45.53	63.12	37.23	39.86
Top 5 faults: *Possible use of NULL pointer* *Possible access Out-Of-Bounds* *Pointer not freed or returned* *Memory leak* *Variable not initialized before using*	72.63	83.43	59.12	73.64
Top 10 faults: *Possible use of NULL pointer* *Possible access Out-Of-Bounds* *Pointer not freed or returned* *Memory leak* *Variable not initialized before using* *Inappropriate deallocation* *Suspicious use of ;* *Data overrun* *Type mismatch with switch expression* *Control flows into case/default*	92.23	89.45	87.31	93.04

V. CONCLUSION

To examine the quality of automated static analysis tools, we inspect two ASA tools. In this research we gather information about ASA tools fault detection, manually inspection faults and system testing failures in students' diploma projects. Our analysis provides some results that are shown in Section 4. Using the received results we can conclude:

- The defect removal yield of ASA isn't significantly different from that of inspections. The defect removal yield of execution-based testing is two times higher than that of ASA and therefore may be more effective at finding the defects.
- The ASA tools are effective for identifying two ODC types: Assignment and Checking.
- The inspection identifies Algorithm, Documentation and Checking faults.
- The great majority of the faults identified by few key programmer errors.
- "Possible use of NULL pointer" is most often fault, identified by ASA – approximately 47% of all faults.
- About 92% of faults are focused on 10 fault types.
- The ASA tools can be used to find security vulnerable errors.

In conclusion results indicate that ASA tools are economical complement to other testing techniques.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the students support of the research.

REFERENCES

[1] http://www.gimpel.com/html/products.htm
[2] http://www.reasoning.com
[3] M. Young and R.N. Taylor, "Rethinking the Taxonomy of Fault Detection Techniques," Proc. Int"l Conf. Software Eng., pp. 53-62, 1989
[4] Osterweil, "Integrating the Testing, Analysis, and Debugging of Programs," Proc. Symp. Software Validation, 1984
[5] N. Rutar, C.B. Almazan and J.S. Foster, "A Comparison of Bug Finding Tools for Java," Proc. IEEE Int"l Symp. Software Reliability Eng. (ISSRE), pp. 245-256, 2004
[6] C. Jones, "Software Defect Removal Efficiency," Computer, vol. 29, no. 4, pp. 94-95, Apr. 1996.
[7] C. Jones, Software Assessments, Benchmarks, and Best Practices. Addison-Wesley, May 2000.
[8] B. Chess, "Improving Computer Security Using Extended Static Checking," Proc. IEEE Symp. Security and Privacy, pp. 160-173, 2002.
[9] V.R Basili, S. Green, oth. "The Empirical Investigation of Perspective-Based Reading", Empirical Software Eng. —An Int"l J., Vol. 1, No. 2, 1996.
[10] R. Chillarege, Bhandari, I.S, oth., "Orthogonal Defect Classification—A Concept for In-Process Measurements", IEEE Trans. Software Eng., vol. 18, no. 11, pp. 943-956, Nov. 1992.
[11] IEEE, "IEEE Standard Classification for Software Anomalies", IEEE Standard 1044-1993, 1993.
[12] Humphrey, W.S. A Discipline for Software Engineering. Addison Wesley, 1995.
[13] Chess B., McGraw, G. "Static Analysis for Security," IEEE Security & Privacy, vol. 2, no. 6, pp. 76-79, 2004.
[14] http://www.securityfocus.com