I.J. Wireless and Microwave Technologies, **2012, 6, 17-22**
Published Online December 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijwmt.2012.06.03

Available online at http://www.mecs-press.net/ijwmt

Software Defect Detection-oriented Static Analysis Techniques

Hua ZHANG

Department of Computer and Communication, Weifang University, Weifang261061, China

Abstract

This paper mainly studies the method of static analysis techniques; discuss static analysis techniques status and progress, as well as the characteristics of static analysis.

Index Terms: Software Quality Assurance; Defect Detection; Static Analysis

1. Introduction

Nowadays, the application of software extends rapidly and the application environment is becoming more and more complicated. The number of accidents caused by poor quality of software is increasing, which, in turn, is doing more and more damage and leading to more serious results, such as the failure of IBM360 operating system and the explosion of Ariane 5. That is the reason why the quality of software is attracting more and more attention.

The insurance of software quality is a long-term task, which runs through the whole process of software development. One of the significant principles is to find the defects as soon as possible. If the defects are ignored and left to the next stage, the cost to restore the defects will expend to 5 to 10 times larger, or even become unreasonable. Well control of software defects can bring high quality software.

The static analysis evaluates software system in terms of the structure, the content and the document of the software without the need to perform any program. In this way, the defects in program code can be easily identified and in the final stage of software development, efforts can be concentrated on the analysis of complex functions and the algorithm mistakes. Static analysis can either search for the defects previous to software engineers' investigation and test, or consist in the whole process of software development.

2. The Comparison of Static Analysis, Manpower Examination and Dynamic Test

There are many methods to detect code defects, among which manpower examination is an effective one. It also has disadvantages: it is time-consuming and the detectors have to know what the defects are before conducting good examination of the software code. Compared with this, static analysis gathers more speed. The

Foundation item: Year 2009 Science & Technology Development Plan of Weifang

static analysis tool can detect the defects by itself and safe much man power. Furthermore, the users do not necessarily acquire professional knowledge as the detectors. A good static tool can also help the users find out the defects of the software.

Another way of examining software defects is dynamic test during the program, such as Purify[1], Valgrind[2], CCured[3] and so on.. These tools can provide precise examination results, but meanwhile, can cause over-cost during the process. Most of them can only find bugs instead of the defects in operating system. The static tools , on the contrary, can identify the defects off the regular execution path without increasing cost.

3. Frequently Used Static Analysis Tools and Their Features

Static analysis tools can automatically identify the abnormality of certain types. Frequently used method is to scan and analyze the source code so as to search for certain model collection in the code. The largest advantage of static tools lies in deducing the results can be used on all execution paths with no need to perform target program. Fig. 1 describes the general system flow of static analysis tools.

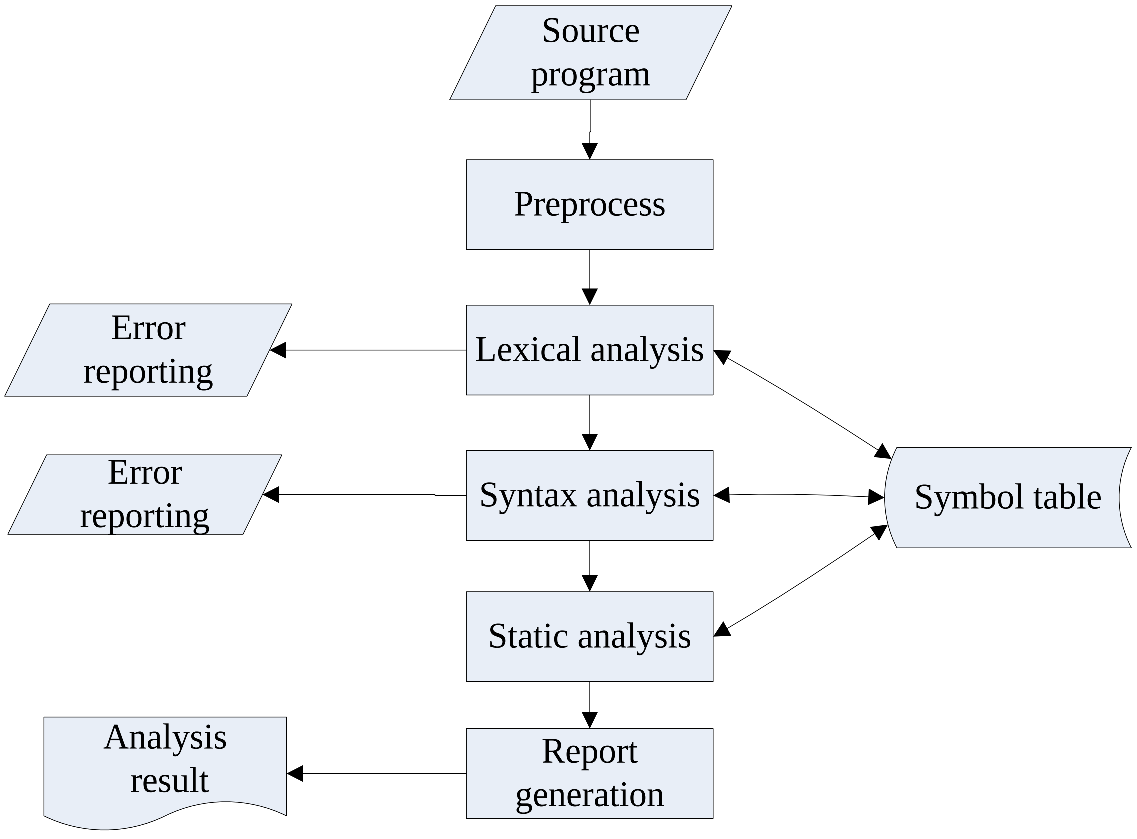

Fig 1. System flow of static analysis tool

Static tools can be divided into four categories in terms of the technologies being applied.

Firstly, tools based on the string matching techniques, such as Unix utility grep. Grep has a string-searching list, which can detect the defects of library function call. It matches notes, characters, statements and function calls all as flows while fails to understand the scanned files and thus causes its low preciseness.

Secondly, the basic morphology analytical method, such as ITS4 (www.cigital.com/its4/), FlawFinder(www.securesoftware.com) and RATS(www.securesoftware.com). They preprocess the source files as token flow and then match the token flow with the defect structure in the base. Although this method has made great progress compared to grep, it still fails to consider the semantic of source code and understand the executive behavior of the program, which lead to high error detecting rate.

Thirdly, method of Program Semantic Analysis, it mainly considers the basic semantics of program through analyzing the control flow and data flow as well as defining and invoking function relationship. This sort of tool specially features on defect inspection, while some flexible ones can be used for inspection through reading a pre-defined supported defection inspection mode.

The main semantic-based static analysis tools are as follows:

a) BOON [4] adopts integer range analysis to determine whether there is an array bound in program C. Although it can detect the defects that many lexical tools cannot identify, yet it neglects the sequence of statement and pointer aliasing, nor can do modeling for the inter-process dependence.

b) CQual [5]uses type validation to examine type identifier as well as to check character string usage defects and lock defects in program C. It requires programmer to annotate some variables.

c) xg + + [6] uses a template-driven compiler expansion to search the defects in kernel of Linux and OenBSD. It can be used for inspecting memory without releasing dynamic allocation, kernel deadlock and some other defects.

d) MOPS [7] uses model checking method to search for deficiencies related to security attributes, such as priority-level management errors, file access competition, etc.

e) Splint [8] uses annotation-aided program analysis provided by programmer to search for un-initialized variables, array subscript bounds and other defects.

f) FindBug [9,10] uses the defect model checker to collect defects in Java programs, such as abnormal null pointer, deadlocks and thread error, etc.

g) PREfix [11] makes analysis over the program call graph to track variables and memory status by analyzing plenty of paths in program. It can also be used for searching the common dynamic error to be happened in program C and C + +.

h) SLAM [12] can abstract a program into Boolean and study its accessible state. If the Boolean program reaches an error state, it will re-analyze the Boolean program. SLAM can be used to search for flaws in Windows drivers.

i) Flexelint (www.gimpel.com / html / products.htm) can be used to check the errors in C / C + + source code, inconsistency, non-portable constant, redundant code and others. Reasoning's Illuma (www.reasoning.com) can be used to search for defects in C / C + +. Klocwork (www.klocwork.com) covers two static analysis tools: inForce and GateKeeper. inForce implements the source code static analysis to identify potential defects and security vulnerabilities. GateKeeper makes analysis over the strong and weak points of the source code architecture. We can also use this tool to evaluate the quality of code, defects and maintenance costs as well.

j) CodeSurfer [13] uses a variety of semantics including call graph and dependence graph to assist the software review. The pointer analysis of program can ensure the accuracy of the pointer alias analysis expression. The inquiry in dependency graph makes it possible to answer back the semantic problems of program. The software defects can be searched through the model inspection of the program. Yet the deficiency is the huge cost in creating dependency graph and is unsuitable for the application for large-scale program.

k) PG-Relief [14] is an automated static analysis system co-developed by Nanjing University and Fujitsu. It can search for potential deficiencies hidden in programs through source code and header files of C / C + +. It also can count up the degree of complexity and scale as well. The analytical defects cover potential logic errors, i.e, reliability defects, program comprehension defects resulting from involved and abstruse design ,i.e, maintenance defects, possible different interpretation defects due to different complier,i.e, migration defects and possible increasing of program objective code size and execution steps, i.e., defects in efficiency.

The fourth tool is the combination of data mining techniques and program analysis. It doesn't need to pre-define patterns or rules, but mine programming model with data mining methods. Some tools are as follows:

a) CP-Miner [15] uses data mining technology to identify copy – paste code of large-scale software and to inspect software defects related to copy - paste code. Firstly, find the corresponding copy - paste code, including those codes ever modified, through introduction of sequential pattern mining techniques. Secondly, test defects through checking the consistency between these codes and corresponding identifiers.

b) PR-Miner [16]uses frequent itemsets mining to extract implicit programming rules. In addition, it also provides an effective algorithm to automatically detect those codes that run counter to the proposed programming rules.

The method of combining data mining and program analysis is more flexible than other testing methods. Programmers needn't do anything during the process, nor requiring any priori knowledge about software. Yet they also cannot recognize any type of software defects. Right now, they can only identify the copy-paste code that dismissing from modification or those deficiencies related to some common programming rules.

4. Limitations of Static Analysis

No single software detection technology can do all software defects detection[17]. Static analysis tool can search for some certain patterns or rules. Although some advanced tools are admitted to gradually add new rules, yet the corresponding rules haven't been compiled and the corresponding errors cannot be detected. Although the combination of data mining and program analysis can automatically dig out some defect detection modes, yet it can not dig out all of the defect modes. Therefore, some software bugs can be automatically identified by static analysis tools to, while some will never be detected.

The test results of static analysis tools still require manual assessment. No any tools will recognize which issue is more important to users. Therefore, the modification will be done against manual judgement. There are also some false dropping for the static analysis tools, so manual work is required to determine if the testing results are the real software defects.

Although the method is not quite perfect, yet it is still useful in detecting software defects. Jiang Zheng [18] and his colleagues once made analysis over the automatic analysis for its roles in developing high-quality commercial products. They made comparison over three large-scale commercial softwares developed by Notel Network for static analysis error, dynamic testing and user reporting error. The results show that the static analysis was quite effective in identifying code defects, which make it possible for the software to emphatically analyze more complex functional or algorithms errors. Their statistical results show that the automated static analysis can be used as a supplement to other defect detection techniques as well as the production for the high-quality software products.

5. Research Status of Static Analysis

Static analysis has been continuously developing from the earlier character string matching and lexical-oriented analysis to semantic-oriented analysis and even to the combination of data mining technology and program analysis. The recent research mainly focuses on semantic-oriented static analysis, which makes analysis over the control flow and data flow as well as the function call relationship. The false rate is greatly decreased since the method has been extended to the implication of program during inspection. The application of data mining is a new method recently being used in defect detection. The flexibility of detection can be increased since data mining does not require a priori knowledge, that is, there's nothing we need to pre-define defect mode or inspection rules. Yet the current existing methods just combine data mining and lexical analysis. The inspection category is only in a limited range and the false testing still exist. If the data mining and semantic analysis can be used together, the method will be more powerful and greatly contribute to the defect detection.

6. Forecast

To sum up, it’s been a very good research direction to apply the static analysis-oriented defect detection to software development. At present, there are still several difficult issues for further study:

a) How to integrate the defect detection tools into software development process in a better approach. For example, the Unit testing has been applied to all stages of software development, or has been combining the static analysis and dynamic testing together.

b) One important issue for the existed static analysis tools is that they can not avoid a large number of false alarms. Some static analysis tools possibly report 50 false alarms when report a real error. How to reduce the number of false alarms will allow to concentrate on correction of real software defects.

c) How to minimize cost of defects analysis.

d) A good static analysis tool should be easier to use and can help point out or eliminate common software defects. It also needs to hold a better rule sets and powerful data mining methods to dig out more models. However, it’s still been a very difficult issue in specifically evaluating the effectiveness of a false tool since it is difficult to know the real number of defects in programs especially large-scale programs.

We will focus on these difficult issues in the current or future research.

References

[1] R. Hastings, B. Joyce. Purify: Fast Detection of Memory Leaks and Access Errors. Proc. Winter USENIX Conf., 158-185, 1992

[2] N. Nethercote, J. Seward. Valgrind: A Program Supervision Framework. Proc. Third Workshop Runtime Verification, 2003.

[3] J. Condit et al.. CCured in the Real World. Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, 232-244, 2003

[4] D. Wagner et al.. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. Proc. 7th Network and Distributed System Security Symp, Internet Soc.,. 3–17, 2000.

[5] J. Foster et al.. Flow-Sensitive Type Qualifiers. Proc. ACM Conf. Programming Language Design and Implementation (PLDI 02), ACM Press, pp. 1–12, 2002.

[6] K. Ashcraft, D. Engler. Using Programmer-Written Compiler Extensions to Catch Security Holes. Proc. IEEE Symp. Security and Privacy, IEEE CS Press, 131–147, 2002.

[7] H. Chen, D. Wagner. MOPS: An Infrastructure for Examining Security Properties of Software. Proc. 9th ACM Conf. Computer and Communications Security (CCS 02), ACM Press, 235–244, 2002

[8] D. Larochelle, D. Evans. Statically Detecting Likely Buffer Overflow Vulnerabilities. Proc. 10th Usenix Security Symp. (Usenix 01), Usenix Assoc., 177–189, 2001

[9] D. Hovemeyer, W. PughDec. Finding Bugs is Easy. ACM SIGPLAN Notices. 39(12): 92-106, 2004

[10] D. Hovemeyer, J. Spacco, W. Pugh. The 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. 13-19, 2006

[11] W. R. Bush, J. D. Pincus, D. J. Sielaff. A Static Analyzer for Finding Dynamic Programming Errors. Software—Practice & Experience, 30:775–802, 2000

[12] T. Ball, S. K. Rajamani. The SLAM project: Debugging System Software via Static Analysis. In Proceedings of the 29th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, 1–3, 2002

[13] P. Anderson, T. Reps, T. Teitelbaum. Design and Implementation of a Fine-Grained Software Inspection Tool . IEEE Transactions on Software Engineering. 29(8): 721-733, 2003

[14] Cai Zhimin, Research and realization of software static analysis, NanJing University, Master Dissertation, 2002(in Chinese)

[15] Z. Li, S. Lu, S. Myagmar, Y. Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. IEEE Transactions on Software Engineering. 32(3): 176-192, 2006
[16] Z. Li, Y. Zhou. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. ACM SIGSOFT Software Engineering Notes. 30(5): 306-315, 2005
[17] M. Young, R.N. Taylor. Rethinking the Taxonomy of Fault Detection Techniques. Proc. Int'l Conf. Software Eng., 53-62, 1989
[18] J. Zheng et al.. On the Value of Static Analysis for Fault Detection in Software. IEEE Transactions on Software Engineering. 32 (4): 240-53, 2006