Towards SOA-based Code Defect Analysis

Qianxiang Wang, Na Meng, Zhiyi Zhou, Jinhui Li, Hong Mei

School of Electronics Engineering and Computer Science, Peking University

Key Laboratory of High Confidence Software Technologies (Peking University) - Ministry of Education

Beijing, China, 100871

{wqx, mengna06, zhouzhy06, lijh06, meih}@sei.pku.edu.cn

Abstract

Static code analysis is the analysis of software that is performed to acquire information concerning the dynamic behavior of programs built from that software, without actually executing the programs. Currently, most analysis techniques are implemented as independent tools, or plugins for integrate development environment (IDE, e.g., Eclipse). However, in this paper, we introduce a new way to release the analyzing ability: a Web Service based approach, which can integrate multiple analysis tools, and provides analyzing ability by way of standard Web Service interface. The user can benefit code analyzing ability by just submit the code to be analyzed to the service, without download any analyzing tools, and then get the merged result. The experiment shows that the proposed approach is feasible and efficient.

1. Introduction

Static code analysis is the analysis of software that is performed to acquire information concerning the dynamic behavior of programs built from that software even without actually executing the programs. It can be used in a variety of software engineering tasks, such as architecture recovery, assertion discovery, debugging, automotive software engineering, clone detection, program comprehension and reverse engineering [1].

This paper focuses on how to use code analysis to find code defect. From the view of the defect finding knowledge, we can divide defects into three levels: 1) Defects that can be found with knowledge of language grammar, which is usually specified using BNF, and can be found easily by compiler; 2) Defects that can be found with knowledge of code defect patterns [3-11], which are gained from analysis experience, and independent with the specific application requirement; 3) Defects that can be only found with knowledge of application requirement, which is application specific, and often used to guide the process of software testing.

The proposed SOA based approach aims to find the second kind of defects. Some of such well-known tools include Coverity [3], FindBugs [7, 8], PMD [9], Fortify [10], and JLint [11]. All these tools focus on Java defect, and try to find defects by scanning code against the pre-defined defect patterns.

1.1 Challenges

As is mentioned in [6], there is no single “best” defect-finding tool to discover all defects contained in checked programs but not provide any false report. Therefore, in order to benefit from different analysis tools when searching for as many useful defects as possible, usually people need to download multiple tools to their own computers and then install them patiently since most of these tools are currently issued as independent software or plugins for IDE.
Additionally, what intensifies people’s demand for convenience as well as simplicity of the tools is the fast growth in the number of defect patterns. Owing to the spreading use of various middleware by companies and organizations, defect patterns coming along with intermediate software are puzzling more and more programmers. Consequently, users always have to update their tools so as to keep abreast with the present state of defect pattern libraries, much like the way they deal with the Antivirus software.

Confronted with people’s urgent want for conveniency as well as easiness of code defect analysis techniques, we try to exploring a new approach to meet the need.

1.2 Approach overview

The proposed approach integrates multiple defect checking tools into a Web Service, named as Code Defect Analysis Service (CODAS), which is based on SOA and focuses on the matching of multifarious defect patterns. The user can visit CODAS from browser or Plugins inside IDE.

Compared with existed defect pattern based static code analysis tools, this approach integrates vantages of different analysis technologies employed by various tools and thus releases people from troubles concerning downloading and installing of different tools. Besides, the network-enabled service is easy to utilize for end users. Furthermore, the published service may facilitate collection of analysis results and get feedback from users, mining the collected data, and adapt the analyzing implementation according to mined knowledge. This may help to improve the quality of analyzing service provided.

The main limitations of CODAS include: 1) The response time depends highly on the network; 2) If the size of the code to be analyzed is too large, this may also lead to the long submitting time and thus low response time; 3) How to merge multiple results to a single one and make users easy to understand is a hard problem still under exploring.

1.3 Paper organization

The rest of the paper is organized as follows. In section 2, we introduce some important static code analysis tools, which enlighten our research work. Section 3 describes the design of CODAS. Section 4 gives some experiments with a prototype of CODAS to demonstrate the feasibility of our approach. Section 5 discusses related issues and future work. Finally, Section 6 concludes the whole paper.

2. Related work

The motivation of our approach is sparked by several static code analysis tools. The most significant three ones among them include FindBugs [7,8], PMD [9], Fortify[10], and JLint [11].

FindBugs analyzes Java bytecode and applies a series of ad-hoc techniques designed to balance precision, efficiency and usability. The distinct predominance of this tool is that it contains a considerable number of defect patterns not covered by any other tool. Moreover, it is released as both independent software and Eclipse plugin for users with different appetites. Especially, with a 1.4 or better Java Runtime Environment (JRE) installed, users are sponsored to try the graphic user interface (GUI) version of FindBugs on their projects using Java Web Start. Such funny trial renders us inspiration for the approach proposed in this paper. Besides, the proprietary solution composed with FindBugs—Fortify [9] once encouraged users to submit code to its web site, and gave back analysis results to them several days later. This attempt also contributes to our research.

PMD, unlike FindBugs, performs defect checking on Java source code. In such a manner, the tool can discover some suspicious or abnormal coding practice which implies serious defects but should have been dropped by analysis tools dealing with Java bytecode due to the optimization conducted by compilers. Additionally, the tool has already been published as independent software as well as plugins for various IDEs.
**JLint** analyzes Java bytecode with syntactic checks and dataflow analysis. What distinguishes the tool from other Java code defect checkers is its interprocedural and inter-file component, which finds deadlocks by building a lock graph and ensuring that there are never any cycles in the graph. Up till now, this tool has only been published as independent installable programs for different operating systems.

With the summary of tools aforementioned, we notice again that various tools are preeminent in dealing with different kinds of defect pattern matching. Naturally, an integrated tool converging benefits of them all will certainly free users from disturbance of installing analysis tools one by one.

Therefore, we hope to encapsulate the checking capabilities of various tools into a Web Service. This manner can not only guarantee that users always have access to its latest version—just as Java Web Start does, but also ensures that its developers are able to have enough analysis results to evaluate and thus decide how to improve the service provided—simply like what Fortify does.

Based on the consideration remarked above, we bring forward our SOA-based code defect analysis approach, which is to be expatiated on in the following section.

### 3. Code Defect Analysis Service

In this section, we firstly introduce the overview of CODAS, then introduce one part of the service implementation (CODA), and finally discuss the service maintenance.

#### 3.1 CODAS Overview

As a kind of Web Service, CODAS owns interface definition to facilitate users when they are submitting code. The internal of CODAS involves multiple analyzing tools, all of them can analyze code independently, and the analyzing results of different tools are merged to a single result, so as to make the result easy to users [12]. Figure 1 illustrates the servicing scenario. Currently, we have integrated Findbugs, PMD, and CODA (Code Defect Analyzer, which is still under further construction, by the authors).

**Figure 1. Servicing Scenario of CODAS**

- **Code**
  The code to be uploaded can be source code or byte code. Certainly, the size of byte code is usually smaller than that of its corresponding source code and thus requires less bandwidth. However, as mentioned in Section 2, the upload of source code is not always undesirable since it may supply more hints of defects and thus do customers an important favor.

- **Auxiliary Information**
  Some additional information may assist the server to better carry out code defect analysis. For example, the user may provide information about programming language used in the inspected code, compiler’s name and version if the code has already been compiled, or middleware used if the code is applications based on intermediate software.

#### 3.2 Implementation of CODA

As is shown in Figure 2, CODA consists of the following major components:

**User Preference Customization** is exploited by users to customize some information according to their own preference. The information may involve what kind of
defects users would like the service to concern and in what manner they expect the service to present discovered defects. For instance, with the help of Defect Examination module, users may deselect defect patterns about concurrency if they are aware that the checked code has nothing to do with multithread programming. In this way, the service is configured to focus on only defect patterns customers may be interested in and thus work out results sooner. On the other hand, by virtue of Defect Presentation module, users can also decide whether defects should be organized according to their seriousness or the checked code they enclose, as well as whether defect reports should be presented directly or stored into a designated file.

**Code to be Analyzed Uploading** is a component used for customers to upload the code to be analyzed. It can accept various forms of code supported by back end analysis engines, such as source code and object code.

**Analysis Engine Factory** contains a number of core analysis engines which implement different analysis techniques applied to various code forms of diverse languages aiming at a wide range of defects. For example, an analysis engine may function to construct hierarchy of inspected programs in order to find inconsistency in it, while another analysis engine will perform dataflow analysis to look for dead code or null pointer reference.

**Detector Category Collection** includes a system of detector categories, which is formed up in accordance with analysis engines depended on. Accordingly, detectors sharing the same analysis engine are generally put into the same category.

**Pattern Set Repository** comprises diverse defect pattern sets. The partition of these pattern sets relies on the detectors applied. In other words, if two defect patterns can be discovered with the same analysis logic, they may belong to the same defect pattern set. Additionally, for more information about defect patterns as well as their category hierarchy, please refer to [12].

**Sub Task** is a concept introduced to reduce time spent on program analysis. Since there is almost no data dependency between each two analysis engines, we may divide the overall task of defect analysis into several sub tasks for separate engines—so as to increase parallelism and improve response time of the service.

### 3.3 Service maintenance

With the continuous emergence of new software, especially with the endless appearance of novel software framework and middleware, a growing number of defect patterns come forth and are playing an increasingly momentous role. A rising number of programmers are trapped by defects when developing their own software on the basis of framework and middleware. Consequently, it is a crucial task to frequently update the Pattern Set Category displayed in Section 3.1 so that it may cover more defect patterns.

In addition, with an eye to the development of static code analysis, we also strive to endow extensibility of Analysis Engine Factory discussed in Section 3.2 in order that new analysis engines may be appended when necessary.

### 4. Experiments

To verify the feasibility of our approach, we implemented a prototype of CODAS on our web site.
http://as.pku.edu.cn. Although its analysis capability is still limited, the cheerful results of experiments performed on it fortify our confidence in the proposed approach.

### 4.1 Design of experiments

We design two sets of experiments to substantiate the effectiveness of our approach through comparison.

In the first set of experiments, we analyze code defects existent in Apache Tomcat 5.5.26 with FindBugs, PMD, and CODA, all of them are installed on a client computer and function to examine for bugs locally. Meanwhile, in the second set of experiments, we check the same code with CODAS. Encapsulating the capabilities of FindBugs, PMD, and CODA, CODAS is provided by a server for users who favor remote code analysis.

By recording and comparing the time spent by each tool, we can deduce whether the cost CODAS pays for its availability and facility is worthwhile. Specially, so as to elude inaccuracy caused by accidental factors, we conduct analysis with each tool for ten times and then take the average time span as a final result shown in Table 1.

### 4.2 Experiment Results

The server side application CODAS is deployed on a personal computer (PC) with a Pentium 4 2.26G CPU, a 1G memory as well as Windows XP Professional Edition SP2 operating system. And the network used is 100MB local area network (LAN).

Running time of the two analyzing modes—local mode with FindBugs as well as PMD and remote mode with CODAS—is displayed in Table 1.

In this table, we recorded the time spent by FindBugs and PMD separately when they check defects locally. On the other hand, we also time CODAS when the same code is examined remotely.

As is shown in Table 1, the total running time used for comparison consists of time taken for uploading as well as analyzing checked code, and downloading defect reports. In local analysis mode, however, since there is no need to upload or download data from a remote server; the time spent on these two tasks is zero.

<table>
<thead>
<tr>
<th>Analysis Mode</th>
<th>Up-load</th>
<th>Analyze</th>
<th>Down-load</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>Bin</td>
<td>0</td>
<td>42047</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Src</td>
<td>0</td>
<td>1030781</td>
<td>0</td>
</tr>
<tr>
<td>Remote</td>
<td>Bin</td>
<td>3781</td>
<td>42844</td>
<td>422</td>
</tr>
<tr>
<td></td>
<td>Src</td>
<td>3891</td>
<td>1039459</td>
<td>4234</td>
</tr>
</tbody>
</table>

In our experiments, we get 408 defects from FindBugs and 72286 from PMD. Correspondingly, CODAS provides the same number of defects as FindBugs does when accepting bytecode and the same as PMD when getting source code.

Under the circumstances of maintaining checking capabilities owned by encapsulated techniques, CODAS usually requires more running time to function well. This fact is sound in each respect. For one thing, as a sort of service provided by a central server, CODAS needs time to accept uploaded checked programs from users and transport analysis reports as feedback. For the other thing, SOAP Message Transmission Optimization Mechanism encoding and decoding account for the additional cost occurring in analyzing time of remote mode.

What is more important, the extra time cost of CODAS is no more than twelve percent of the overall interval necessary for analysis. Therefore, the additional expense is acceptable and thus our approach is feasible.

### 5. Discussion and future work

Despite of convenience and simplicity of the service, CODAS can not substitute analysis tools published in other forms, such as independent software or plugins for IDE.

Actually, the service has significant dependence on
transportation conditions of network. In figure 1, the customer submits code CODAS, and get analyzing result from CODAS. Another servicing way of CODAS is to provide customers with only new defect patterns. The IDE of customers integrate engines and which can accept defect patterns and generate new detectors.

Additionally, as is well known to all, no static code analysis is powerful enough to simulate the exact execution of any program or thoroughly excel testing when looking for defects. Therefore, we also plan endeavor to improve the combination between static analysis and dynamic execution information retrieval (such as online monitoring which is discussed a lot in [13]) so as to improve accuracy of analysis and reliability of final results.

In future, we will enhance CODAS by adding new analysis engines as well as new tools and new defect patterns, reducing its dependence on performance of network as well as servers.

Furthermore, the philosophy of developing software by the aid of SOA can be even extended to other phases of the whole lifecycle of software, such as requirement analysis, design, testing and maintenance. In this way, we may import a new conception of “SOA based software engineering”.

### 6. Conclusion

In this paper, we introduced a web service based code analysis approach, which encapsulates multiple analyzing tools, and facilitates users’ adoption of static code analysis when searching for defects in their own code. We discussed the design of CODAS, which embodies the central philosophy of our approach; we also expounded the experiments carried out on a prototype implementing CODAS’ design to verify the feasibility.

This paper is supported by the National High-Tech Research and Development Plan of China, No. 2006AA01Z175, and National Natural Science Foundation of China, No. 60773160.

### References


[12] Na Meng, Qianxiang Wang, Qian Wu, Hong Mei, An Approach to Merge Results of Multiple Static Analysis Tools, The 8th International Conference on Quality Software (QSIC), August 2008.