Identifying Fault-Prone Files Using Static Analysis Alerts Through Singular Value Decomposition

Mark Sherriff[1,2], Sarah Smith Heckman[1,2], Mike Lake[1], Laurie Williams[2]

[1]IBM, 3901 S. Miami Blvd., Durham, NC, USA

[2]NC State University, 890 Oval Dr., Raleigh, NC, USA

Abstract

Static analysis tools tend to generate more alerts than a development team can reasonably examine without some form of guidance. In this paper, we propose a technique for leveraging field failures and historical change records to determine which sets of alerts are often associated with a field failure using singular value decomposition. We performed a case study on six major components of an industrial software system at IBM over six builds spanning eighteen months of development. Our technique identified fourteen alert types that comprised sets of alerts that could identify, on average, 45% of future fault-prone files and up to 65% in some instances.

1 Introduction

Static analysis is the process of evaluating a system or component based on its form, structure, content, or documentation [2] without execution of the code. Static analysis tools search for implementation problems identified by a predefined set of rules of potential anomalies commonly found in source code. The static analysis rule types range from possible mistypes in the code (e.g. = instead of ==) to more complex errors in the system logic (e.g. memory leaks). We term the use of static analysis tools as *automated static analysis (ASA)*. An *ASA alert* is a single report from ASA, indicating an area of the code base that has broken an ASA rule. Each ASA alert has an *ASA alert type*, which describes the ASA rule that has been broken.

Our research goal is to provide a methodology for highlighting files that contain groups of static analysis alerts historically associated with field failures. To address this goal, we have developed a technique that leverages historical field failure information and change records in conjunction with ASA alerts to generate ASA alert signatures. An *ASA alert signature* is a set of static analysis alert types that has historically been associated to one or more field failures in a particular project. We generate ASA alert signatures using singular value decomposition (SVD). SVD provides a means for associating files with field failures and ASA alerts with those files. A set of files that has changed together is identified as potentially failure-prone if a future version of the set of files contains all of the alert types in an ASA alert signature.

Our hypothesis is that automated static analysis alert signatures generated from historical information through singular value decomposition can be used to identify failure-prone files. To test our hypothesis, we performed a post-hoc experiment on six components of a large IBM software system across six builds over a period of eighteen months. We further investigate how the data set size and values in the singular vectors affect the efficacy of our technique

2 Automated Static Analysis

Tools can be used to automate the process of performing static analysis. ASA may be run through-

out the development process since this analysis does not require execution [1]. However, static analysis tools suffer from several problems. The main problem with static analysis is that many of these tools have a high rate of false positives due to approximations made to the analysis [1]. Because ASA tool generates false positive alerts, developers must inspect the alerts generated from ASA tools to verify the accuracy of the alerts for fault fixes [1].

3 ASA Alert Signatures

In this section, we will describe the steps in our methodology for generating ASA alert signatures from historical data. We will also describe how to apply the generated alert signatures to a set of ASA alerts types and system files of a future build.

3.1 Gathering Field Failure Data and Performing SVD

We begin by gathering the source code change records and fault information to populate a matrix *M* that indicates when files have changed together in response to a field failure. The rows and columns of the matrix are comprised of every file in the system. The values within the matrix indicate the number of times that the files assigned to that row and column have changed together to repair a specific field failure.

After the change records and field failure information has been gathered and put into the matrix *M*, we perform a SVD on the matrix to determine what files tend to change together in response to field failures. We are interested in these associations because our overall goal is to find out what sets of ASA alerts are correlated with field failures. To detect the association between ASA alerts and field failures, we need to analyze the files that are common between the field failures and the ASA alerts.

Performing SVD on the matrix M generates three matrices: *U*, *S*, and *V*. The columns of the *U* and *V* matrices provide information as to the structure of the association clusters and the rows of *U* and *V* are the files in each cluster. An *association cluster* is a group of files that tend to change together. The SVD generates the same number of association clusters as the rank of the original matrix. The singular values, found on the diagonal of the *S* matrix, represent the amount of variability each association cluster contributes to the original analysis matrix.

3.2 Generating ASA Alert Signatures through SVD

Once we know what files are strongly associated with field failures, we can then determine how the ASA alerts compare with these file clusters. In this step, we will create a new matrix *M*. However, this matrix will be an asymmetric matrix with the previously generated file clusters as the columns and the different types of static analysis alerts as the rows. The values in the matrix *M* will be the difference in the number of ASA alerts found between two builds. We are interested in the difference between two baselines because this will highlight any possible correlation between the addition, removal, and change of ASA alerts with the repair of field failures and visa-versa.

Performing a SVD on the new matrix *M* yields another set of *U, S,* and *V* matrices. We can interpret these matrices in much the same way as before. However, now the columns of *U* indicate clusters of ASA alerts and the rows of *U* indicate alert types. We use the singular values along the diagonal of the *S* matrix to identify the most significant ASA alert signatures. In clusters involving ASA alerts, a high singular value indicates that that set of alerts appears numerous times within the code base.

In some instances, however, an alert signature will be comprised of a single alert type with a high singular value. This phenomenon is indicative of an alert type that is so pervasive in the code base that it can match with nearly every file, thus eliminating much of the benefit of our technique. These types of alerts can be identified if they appear in a signature by themselves and have a singular value that is a factor of ten higher than the next value. We eliminate these signatures from our technique, since this would significantly increase the number of potentially failure-prone files identified by our technique that may not contain actual faults.

An opposite problem can occur with ASA alert signatures with low singular values. For example, an ASA alert signature consisting of a subset of alerts from another ASA alert signature with a lower singular value can appear. In this instance, the signature with the lower singular value would match with fewer field failures, thus identifying only a few files as being failure-prone.

Often, these files will have already been identified by another ASA alert signature with a higher singular value. Thus, the goal then is to choose a set of signatures that are distinct from each other, but also not too general or pervasive (as indicated by the singular value).

After the ASA alert signatures have been generated and identified, each subset of ASA alert types found in a given signature can be compared to a full set of ASA alerts from a code base. However, since these ASA alert signatures were generated based on clusters of files, the signatures need to be applied in a similar fashion. Thus, we must finally build clusters of files based upon all changes in the system on which to apply the signatures.

4 IBM Case Study

We performed a case study of our technique on six components of a large IBM software system. In this section, we will discuss our research methodology and results from the case study.

4.1 Case Study Setup

We performed our case study on six components of a large software project over six builds. The six builds span eighteen months of the development of the system. We gathered builds two to four months apart between October 2005 and March 2007. An IBM internal ASA tool was run on each build of the software post hoc, and we gathered information on the files, alert types, and line numbers where the alerts appeared.

The ASA tool used reports 74 different ASA alerts and classifies the ASA alerts into five separate categories: error, mistake, warning, security, and portability. An ASA error alert is a high priority alert, with mistake and warning as medium and low priority, respectively. Security alerts indicate areas where the program may be subverted, such as unverified inputs. Portability alerts are for problems that would only appear if the code is ported to another machine with a different bit depth (such as 32-bit to 64-bit). Each classification of alert can be enabled or disabled according to the developer's preferences.

4.2 ASA Alert Signatures

We followed the steps outlined in Section 3 for this experiment. We repeated our technique using every combination (i.e. builds 1 and 3, 1 and 4, 2 and 4, etc.) of the six build dates to determine if time or the number of change records affected our technique. We generated a total of 74 ASA alert signatures for each experiment set because there are 74 alert types in the ASA tool we selected. Nearly 95% of the overall variability in the original matrix is found in the first five singular values. We found that the first singular value is a factor of ten greater than the second in nearly every case. This large separation of values is an indication that the ASA alert signature associated with the first singular value would have an extremely high match rate with sets of ASA alerts. An ASA alert signature associated with the first and highest singular value consisted of only one or two ASA alerts that were pervasive in the system and would not provide any value to the set of signatures for minimizing the total number of files to examine.

Across the fifteen experiment sets, we gathered four signatures for each, for a total of 60 ASA alert signatures. The 60 alert signatures were comprised of only 14 of the 74 total alert types, providing some initial indication as to which alerts were most likely to be associated with field failures. We highlight these two signatures to give examples as to how a grouping of ASA alerts can provide an indication of the potential field failures in the system.

4.2.1 ASA Alert Signature 1: Taking a Wrong Turn

This ASA alert signature was common among 14 of the 15 experiment sets and often had the second or third highest singular value. This set of ASA alerts focuses on missing parts of an execution path. The signature consists of four alert types. The alert types are as follows:

- *Warning 9: Return value of function is not used*
- *Mistake 5: Expression always evaluates true or false*
- *Warning 14: Possible type mismatch*
- *Warning 15: then/else/loop not surrounded by braces*

4.2.2 ASA Alert Signature 2: Memory Leaks

This set of three ASA alert types focuses on memory leaks and appeared in approximately half

of the experiment sets. The alert types are as follows:

- *Error 23h: Heap memory leak*
- *Warning 2: Value of variable with local scope is never used*
- *Mistake 21: Advisory has been issued for this function*

4.3 Applying the Signatures

Once we built our ASA alert signatures, we applied them to ASA alerts associated with clusters of files generated from all changes in the system found up until the current set of changes in the system to predict future field failures. When we examined each ASA alert signature's precision and accuracy independently, we determined that 86% of the failure-prone files identified by the signatures were identified by more than one signature. Each signature did identify new failure-prone files that were not identified by the other signatures. However, there was a point of diminishing returns as new signatures with lower singular values were applied. Using the ASA alert signatures, our technique reduced the total raw number of ASA alerts to examine by 74% and the total number of files to examine by 43%. The set of identified potentially failure-prone files by the ASA signatures encompassed 40% of the actual failure-prone files. If a development team examined every file that contained at least one static analysis alert, only 79% of the field failures would be detected.

The main limitation of our technique is in selecting an appropriate data source. The analyst must be convinced that when a set of changes are attributed to repairing a given fault or failure, those changes are indeed associated with that fault or failure and are not associated with another set of changes. Traceability from field failures to the files they affect is required to build the failure trends that are then associated with ASA alerts. Another limitation of this technique is that even at its optimal rate at identifying failure-prone files, it cannot exceed the overall number of failure-prone files that could be found by the static analysis tool selected. Further, our technique is specifically based on historical data. Therefore, files that are recently created and have little to no change history or files that have not in the past been failure-prone cannot be isolated effectively through our technique

5 Conclusions

In this paper, we have presented a technique for combining a project's historical field failure information, change records, and static analysis alerts to generate ASA alert signatures. These alert signatures consist of groupings of ASA alert types that have been directly linked to field failures in previous builds. By applying these signatures to a current set of ASA alerts, developers can isolate specific files and alert types that historically have been associated with field failures.

Acknowledgements

Partial funding was provided for NCSU authors by the National Science Foundation.

About the Author

Mark Sherriff is a fifth year PhD candidate at NC State University. His email address is mark.sherriff@ncsu.edu.

Sarah Smith Heckman is a third year PhD student at NC State University and a Software Engineer interning at IBM. Her email address is sarah_heckman@ncsu.edu.

Mike Lake is a Senior Technical Staff Member working as a software architect in IBM Tivoli Monitoring.

Laurie Williams is an Associate Professor in the Computer Science department at NC State University. Her email address is williams@csc.ncsu.edu.

References

[1] Chess, B. and West, J., *Secure Programming with Static Analysis*. Upper Saddle River, NJ: Addison-Wesley, 2007.

[2] IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.