Using Groupings of Static Analysis Alerts to Identify Files Likely to Contain Field Failures

Mark S. Sherriff, Sarah Smith Heckman
NC State University, IBM
890 Oval Drive
Raleigh, NC, USA
+1-919-513-5082
{mssherri, sesmith5}@ncsu.edu

J. Michael Lake
IBM
3901 S. Miami Blvd.
Durham, NC, USA
johnlake@us.ibm.com

Laurie A. Williams
NC State University
890 Oval Drive
Raleigh, NC, USA
+1-919-513-4151
williams@csc.ncsu.edu

ABSTRACT
In this paper, we propose a technique for leveraging historical field failure records in conjunction with automated static analysis alerts to determine which alerts or sets of alerts are predictive of a field failure. Our technique uses singular value decomposition to generate groupings of static analysis alert types, which we call alert signatures, that have been historically linked to field failure-prone files in previous releases of a software system. The signatures can be applied to sets of alerts from a current build of a software system. Files that have a matching alert signature are identified as having similar static analysis alert characteristics to files with known field failures in a previous release of the system. We performed a case study involving an industrial software system at IBM and found three distinct alert signatures that could be applied to the system. We found that 50% of the field failures reported since the last static analysis run could be discovered by examining the 10% of the files and static analysis alerts indicated by these three alert signatures. The remaining failures were either not detected by a signature which could be an indication of a new type of error in the field, or they were on areas of the code where no static analysis alerts were detected.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Symbolic execution, Testing tools*

General Terms
Management, Measurement, Reliability, Experimentation

Keywords
Static Analysis, Singular Value Decomposition, Field Failures

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 associated with a predefined set of rules of potential anomalies in the 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 *automated static analysis (ASA)*. An *ASA alert* is a report from the ASA tool indicating an area of the code base that has broken a specific type of ASA rule.

Copyright is held by the author/owner(s).
ESEC-FSE'07, September 3–7, 2007, Cavtat near Dubrovnik, Croatia.
ACM 978-1-59593-812-1/07/0009.

Research has shown that ASA alerts can identify certain classifications of faults and field failures [8]. However, the number and pervasiveness of certain alert types might indicate that nearly all the files in the system are potentially failure-prone[1]. The number of static analysis alerts reported by the static analyzer could overwhelm the development team. Certain alert types, and certain combinations of alerts found together, could be used to reduce the number of identified failure-prone files to those that are most likely to actually contain failures based upon previous versions of the system.

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 failures 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 with one or more field failures in a particular project. We generate ASA alert signatures by using singular value decomposition (SVD). The SVD provides a means for associating files with field failures and ASA alerts with those files. A set of files that has changed together are identified as 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 identify files that failure-prone. To test our hypothesis, we performed an experiment on three components of a large industrial software system. Over a year of static analysis and field failure information was analyzed from an industrial software system in this research.

2. RELATED WORK
In this section, we will discuss related work and background literature in automated static analysis.

2.1 Automated Static Analysis
ASA may be run throughout the development process since this analysis does not require execution [1]. However, static analysis tools suffer from several problems. The main problem 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].

[1] A *failure-prone file* is any file that contains a field failure [8].

We used an internal IBM ASA tool in our investigation. One of the main goals of this tool is to avoid as many false positives as possible while not requiring any extra specification from the user. The tool spends extra execution cycles traversing paths that it identifies as leading to an error to ensure that the path is indeed executable. This extra computation increases the runtime of the tool in comparison to other ASA tools.

The ASA tool classifies its 74 different ASA alerts into five categories: error, mistake, warning, security, and portability. An 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 alert category can be enabled or disabled according to the developer's preferences.

2.2 Using ASA Alert to Separate High Quality Components

Other studies have analyzed the ability of ASA alerts to narrow the focus on fault- or failure prone areas of code. Static analysis alerts were used to predict the pre-release fault density of Windows Server 2003 [4]. The research demonstrated a positive correlation between the ASA fault density and pre-release testing fault density and that discriminant analysis of ASA faults could be used to separate high- from low-quality components with 83% accuracy. Additionally, a study was conducted of the use of static analysis at Nortel [5, 8]. ASA and failure data from three products (over three million lines of code) that underwent ASA during test were analyzed [5, 8]. The data demonstrated a statistically-significant correlation between the number of ASA alerts and field failures in a module (a grouping of files). These results indicate that when a module has a large quantity of ASA faults, the module is likely to be problematic in the field. Finally, discriminant analysis indicated that ASA faults could be used to separate fault-prone from non-fault prone modules with 87.5% accuracy. In both of these studies, only the quantity of ASA alerts was used. In our study, we use information about the types of the alerts and about historical relationships of sets of alerts that historically appear together in code with field failures.

3. GENERATING ALERT SIGNATURES

Historical records of field failures, change records, and static analysis results are all required to generate accurate ASA alert signatures. First, we must identify appropriate data sources and be reasonably confident that the data is accurate in associating distinct code changes with specific failures. With these data sets, we can find associations between files and static analysis alerts based upon what files changed together due to repairing field failures. We outline the steps in the algorithm for generating signatures in Figure 1.

We begin by gathering source code change records and fault information to populate a matrix *M* that indicates how many times 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 combination have changed together to repair a specific fault. The values on the diagonal of the matrix represent the total number of times that a file has changed because of a field failure.

```
1  Pick two builds separated by enough time
that field failures have been reported
between the two builds.
2   Run ASA on the two builds and record
results.
3  M = [Files in system x Files in system]
s.t. M = where the values in the matrix
indicate the number of times two files have
changed together to repair a field failure
between the two builds.
4       File/Field Failure Clusters =
gatherClusters(M)
13 M' = [File/Field Failure Clusters x Alert
Types] s.t. M' = delta in the number of ASA
alerts reported between the two builds.
5     Clusters of Alerts (Signautres) =
gatherClusters(M')
6  Set of clusters = gatherClusters(M)
7    [U, S, V] = svd(M');
8    for i:size of U
9      Gather signature i information
10     for j:size of U
11       if |U(j, i)| > threshold
12         Store element of cluster i
13       end
14     end
15   end
16 end
```

Figure 1. Algorithm for generating signatures.

After matrix *M* has been populated, we perform SVD on the matrix *M* to determine what files tend to be associated with field failures. When we perform SVD on the matrix *M*, matrices *U, S,* and *V* are generated. The columns of the *U* and *V* matrices provide information as to the structure of the association clusters, while the singular values from the *S* matrix represent the amount of variability each association cluster contributes to the original analysis matrix. An association cluster is formed by taking the files in each column of the matrix *U* is over a given threshold. We are interested in these association clusters because our overall goal is to find out what sets of ASA alerts are associated 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.

Using the singular values from the *S* matrix, we can determine how many of the association clusters we will use in our analysis. We do not use all of the generated clusters due to cluster duplication and because clusters that have relatively small singular values are not strongly linked and thus have less value. A cluster's strength, represented by the size of the singular value coupled with it, indicates the amount of variability that the association cluster provides to the original analysis matrix [7]. Osinski used a threshold of 90% of the variability to determine the appropriate number of clusters to examine [6].

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 clusters on one axis and the different types of static analysis alerts on the other. The values in the matrix *M* will be the difference in the number of ASA alerts found between the ASA runs on the two

selected system builds. We are interested in the difference between two baselines because this will highlight any possible correlation between the removal of ASA alerts with fewer 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, where the columns of *U* indicate clusters of ASA alerts.

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 future version of the code base. However, since these ASA alert signatures were generated based on clusters of files, the signatures need to be applied to clusters of files instead of individual files. We can generate clusters of files to apply the signatures to in a way similar to how we generated the signatures. However, in this case, we will use all changes in the system that were made to modify all faults, as opposed to just those changes made to repair field failures, so that we can examine all areas of the system for potential faults. Once these clusters have been generated, the alerts contained in each cluster can be gathered based upon the files within each cluster. ASA alert signatures can then be compared with alerts associated with each file cluster to determine which areas of the system may require further V&V efforts.

4. IBM CASE STUDY

During the spring of 2007, we performed a case study of our technique with a large IBM software system. In this section, we will describe our case study experience and our results.

4.1 Case Study Setup

We selected Matlab 7.2 R2006a as our SVD tool and used an internal ASA tool for generating ASA alerts. We performed our case study on three modules of a large industrial project. We selected these particular modules because they were primarily written in C and C++, which are two of the languages that this particular version of the ASA tool could analyze. We generated ASA signatures from data from two builds of the system, one from late October 2005 and the other from mid-December 2006. ASA was run on each build of the software, and we gathered information on the files, alert types, and line numbers where the alerts appeared. All 74 alert types from the five categories of alerts were included. ASA alerts were associated with clusters of files and the difference between the two releases was calculated for generating the ASA alert signature clusters.

4.2 ASA Alert Signatures

Using our technique, three ASA alert signatures were created in this case study. The three signatures were:

ASA Alert Signature 1: A Misstep in the Path

- *M5: Expression always evaluates true or false*
- *W5: Operator "=" in the Boolean expression should possibly be "=="*
- *W13: Function never used*
- *P2: The cast (int)long will cause truncation on the portability target machine*
- *S2: Passing untrusted input to argument*

ASA Alert Signature 2: Common Errors

- *W15: then/else/loop not surrounded by braces*
- *W16: Function accesses the same variable through two parameters*
- *M18: Comparing pointers to strings*
- *E18: Function lacks a return statement with a value*

ASA Alert Signature 3: Memory Leaks

- *E23h: Heap memory leak*
- *W9: Return of function not used*
- *M21: Advisory has been issued for this function*

4.3 Applying the ASA Alert Signatures

We examined the ASA alerts that were generated on the December 2006 release of the software system using the alert signatures. In this release, the tool generated ASA alerts on 2,448 files. We then collected field failure information from December 2006 to March 2007 to determine failure-prone files. A summary of the effects of applying the ASA alert signatures can be found in Table 1.

Table 1. Summary of Effects of Applying Alert Signatures.

	Before Applying ASA Signatures	**After Applying ASA Signatures**
# of Files	2,448	393
% Reduction in Files to Examine	N/A	70%
% Field Failures Covered	79.6%	49.5%
% Absolute False Positive Rate Improvement	N/A	20%

Using the ASA alert signatures, there was a significant reduction in the number of files and ASA alerts that need to be analyzed. As mentioned, the ASA tool reported at least one alert each of the 2,448 files that it was run against. After applying the ASA alert signatures, 393 of the 2,448 files were identified as belonging to a file clusters containing alert types similar to alerts associated with a previous field failure. There is, however, a reduction in the number of field failures found verses checking every file that contained at least one static analysis alert. Note that our technique highlights areas of the code base that may contain field failures based upon previous development efforts and field failure reports. The field failures that could not be identified by the ASA alert signatures do not match any previous alert patterns in reported field failures. Nearly 50% of the field failures fall under a similar ASA alert pattern from previous releases, indicating that a large percentage of field failures come from a relatively common and consistent set of mistakes. If every file that contained a static analysis alert was examined, only 79.6% of the field failures could be detected. Research has shown that ASA tools can only find certain types of programmer errors [8] and, thus, cannot be expected to find all faults that lead to field failures.

4.4 Comparison to Other Techniques

We also examined the efficacy of our technique against the models proposed by Zheng et al. [8] and Nagappan and Ball [3] Both Zheng et al. and Nagappan and Ball proposed that ASA alert density could be a predictor of pre-release fault density. The main difference between the research presented in this paper and these two studies is the granularity level throughout the work. Zheng et al. and Nagappan and Ball use the overall number of alerts or the alert density to identify potential fault-prone modules, while we focus on

fault-prone files. Therefore, we also investigated whether there was a correlation between the number ASA alerts and field failures at the file level to better compare our technique to theirs. The results of the correlation analysis at the file level can be found in Table 2.

Table 2. Summary of Spearman Rank Correlation for Files

		# ASA alerts	# test faults	# field failures	# total failures
# ASA alerts	**Correlation Coefficient**	1	**.182**	**.139**	**.212**
	Sig.	.	**.001**	**.001**	**.001**
# test failures	**Correlation Coefficient**		1	.143	.924
	Sig.		.	.001	.001
# field failures	**Correlation Coefficient**			1	.510
	Sig.			.	.001

As shown in Table 2, we did not find a strong correlation between the number of non-clustered ASA alerts and the number of field failures (.139). We also used a discriminant analysis to predict the fault-prone files in a similar fashion to Zheng. We found that our technique had a 15.4% improvement in true positive rate over their discriminant analysis at the file level for this data set. From this we conclude that while we found similar results as Zheng and Nagappan at the module level, our technique showed some improvement at the file level.

Another difference in the techniques is that the previous two studies focus on identifying fault-prone modules, the boundaries of which are specifically identified by the development teams. By using SVD to isolate areas of change, we are comparing our ASA alert signatures against files that tend to changed together in response to failures, regardless of the files' location or module. Functionality can often be spread among various modules, and thus by examining how ASA alerts affect these functional areas, we are targeting specific, customer-facing requirements where field failures are likely to be found and could cause difficulties in the field

5. CONCLUSIONS AND FUTURE WORK

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 releases. By applying these signatures to a current set of ASA alerts, developers can isolate specific files and alert types that historically have led to field failures.

We performed a case study with an industrial software system at IBM to evaluate our technique. Field failure information, change records, and ASA alerts were gathered on two releases of the system over a 14-month period. The data from these releases were used to build ASA alert signatures that correspond to field failures found between those two releases. We then applied these ASA alert signatures to the alert set from the latest release to predict the failure-prone files for the following three months. We found that 50% of the field failures could be discovered by examining the 10% of the files and their associated alert signature(s). The remaining failures were either not detected by a signature which could be an indication of a new type of error in the field, or they were on areas of the code where no static analysis alerts were detected.

The analyses presented in this paper using SVD, both in the background section and in the current work, show examples of how relationships between files can be detected using software development artifacts, such as faults, field failures, and ASA alerts. We are currently continuing to examine the various different types of development artifacts that could be used to help drive development decisions.

6. REFERENCES

[1] B. Chess and G. McGraw, "Static Analysis for Security," in *IEEE Security and Privacy*, November/December 2004 ed, 2004, pp. 32-35.

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

[3] N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicator of Pre-Release Defect Density," in *International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 580-586.

[4] N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," in *International Conference on Software Engineering (ICSE)*, St. Louis, MO, 2005.

[5] N. Nagappan, L. Williams, M. Vouk, J. Hudepohl, and W. Snipes, "A Preliminary Investigation of Automated Software Inspection," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, St. Malo, France, 2004, pp. 429-439.

[6] S. Osinski, J. Stefanowski, and D. Weiss, "Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition," in *Advances in Soft Computing, Intelligent Information Processing and Web Mining*, Zakopane, Poland, 2004, pp. 359-368.

[7] T. Will, "Introduction to the Singular Value Decomposition." vol. 2006: UW-La Crosse, 1999.

[8] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. Hudepohl, and M. Vouk, "On the Value of Static Analysis for Fault Detection in Software," *IEEE Transactions on Software Engineering,* vol. 32, no. 4, pp. 240-253, April 2006.