Toward a Learned Project-Specific Fault Taxonomy: Application of Software Analytics

A Position Paper

Billy Kidwell
Computer Science Department
University of Kentucky
Lexington, USA
bill.kidwell@uky.edu

Jane Huffman Hayes
Computer Science Department
University of Kentucky
Lexington, USA
hayes@cs.uky.edu

***Abstract*—This position paper argues that fault classification provides vital information for software analytics, and that machine learning techniques such as clustering can be applied to learn a project- (or organization-) specific fault taxonomy. Anecdotal evidence of this position is presented as well as possible areas of research for moving toward the posited goal.**

Keywords—fault taxonomy; machine learning; software repositories; clustering

I. Introduction

Any viable, useful software product is characterized by at least two "proof of life" indicators: feature requests and bug reports from users and/or developers. Our software-driven society is known for its appetite for more and better features, all while shunning buggy, slow, or just "ugly" software products. Development organizations that want to remain competitive in this environment must develop high quality software that is also feature rich and reaches the market before competitors. How can this be accomplished?

Studies have shown that history repeats itself in software products, just as in life. Bugs/faults (called faults hereafter) or types of faults that occurred in the past are likely to occur again in the future (or are likely to reappear as latent errors). In addition, there is evidence that fault taxonomies, when used conscientiously, can assist a development organization in understanding the types of faults that tend to occur: and when, and where, and for whom. This information can greatly assist in preventing errors, in finding and correcting errors, and in supporting other applications such as clone detection. There has been prior work on automatically assigning categories to a fault based on a provided fault taxonomy.

In this paper, we argue that classified faults contain vital information for software analytics, and that both the development of a fault taxonomy and the assignment of fault taxonomy categories to an individual fault or bug report must be automated. We describe the utility of fault classification data in the context of decisions that can be aided by software analytics. Then, we argue for an automated approach to learning a fault taxonomy for a specific project based on project data. Based on our experience in the traceability area, we understand the difficulties of getting practitioners to adopt a new practice. This makes full automation desirable. We believe, further, that recovering a fault taxonomy is analogous to recovering traceability links from software engineering artifacts or discovering the architecture style of a given software system using its source code.

The paper is organized as follows. Section II presents background information and our position. Section III presents evidence that fault classification data is vital to software analytics. Section IV provides research results to date to defend our position for a learned fault taxonomy. Section V suggests research goals/areas to move toward a realization of the position and Section VI concludes.

II. Background and Position

This section presents background on fault classification and fault taxonomies as well as our position.

A. Background

IEEE defines a software *fault* as an "incorrect step, process or data definition in a computer program" [1]. The term *fault* and *defect* are used synonymously in this paper. A fault leads to a *failure* when the software does not perform to specifications.

A *fault taxonomy* provides a scheme of classification for software faults. Many attributes of a fault may be classified, including severity, when the fault was found, the type of failure, and the type of fault. A good example of a fault taxonomy is the *Orthogonal Defect Classification* scheme, developed by Chillarege et al. [2].

Clustering is a machine learning technique that groups data instances into natural groups [3]. Clustering is therefore useful when a training set is not available. Given a sufficiently large training set, classification learning algorithms, e.g. Bayesian networks, decision trees, and others, can be used to classify new faults.

B. Position

Position: That fault classification provides data that is essential to many types of decisions that could be aided by software analytics.

<u>Further:</u> That machine learning can be used to learn a fault taxonomy for a given software project.

III. Fault Classification and Software Analytics

Buse and Zimmermann surveyed 110 professional software engineers and managers to determine their information needs with regard to software analytics [4]. Failure information and Bug Reports were the top two indicators for decision making. Buse and Zimmermann also present themes for the types of questions and decisions in which analytics could support software engineers and managers. In this section we argue that many of these types of decisions can be enhanced with fault classification data.

A. Targeting Testing

Miller et al. developed detailed taxonomies for both faults and for verification and validation (V&V) techniques, which include different testing approaches and inspections [5]. Their research maps the fault type to the V&V types that can detect it and includes cost benefit analysis on each technique. Applying software analytics to this approach of choosing the most applicable V&V technique could expand the reach and utility of such a method due to the availability of additional data.

Vegas et al. present a characterization process for testing technique selection [6]. The characterization schema includes the defect (fault) type. This history can then be used to determine which types of faults are found by a particular technique. Since components often exhibit similar types of faults as they have in the past, it supplies helpful empirical data about the selection of the most effective testing technique.

Misirli et al. present a retrospective study of software analytics projects in industry [7]. Among the feedback on a case study for defect prediction was the need for information about "defect causes, such as the phases introduced, categories and severity levels" [7]. Defect prediction models with this additional information can provide recommendations about *where* defects will occur, along with *what kind* of defects can be expected, and thus *how* they can be most effectively detected.

B. Release Planning

One of the relevant factors for release planning is the number of outstanding faults in the software [4]. Managers want to know that the fault arrival rate is declining to determine that the software is approaching a stable point, and can be considered ready for release. Chillarege et al. describe how the defect type attribute of the Orthogonal Defect Classification (ODC) can be used to assess the state of the software with regard to release [2].

Consider, for example, that the fault arrival rate slows. This is normally an indication that the software is stabilizing. However, what if the faults that have been logged indicate that functionality is missing? In that case, this could represent a local minimum on the reliability growth curve. More faults are just around the corner, but the functionality has not been tested because it is not yet fully implemented. As software processes become more iterative and incremental, with iterations as short as two weeks, this type of analysis becomes even more important, and requires automation.

C. Judging Stability

The same process to measure the stability of a release, described above in Release Planning, can also be applied to components or subsystems that are being developed independently. By using a fault classification scheme, we can build a profile for different stages of stability, and then compare the current state of a project to past profiles. This provides a quantitative assessment of the project's progress that can be used to aid in decisions about status and course correction.

D. Targeting Training

As noted by Buse and Zimmermann, software development is primarily a human endeavor that can benefit from considering individual and team collaboration [4]. It is often difficult to monitor the quality of work by individuals, in order to recommend ways to improve performance. We believe that fault classification plays a role here as well.

Yu reports on a software fault prevention program at Lucent Technologies [8]. A crucial finding reported by Yu was that nearly half of the faults were introduced during coding, and many of the faults were preventable. Yu goes on to describe the fault prevention guidelines that were developed. Yu estimates that the 34.5% reduction in coding faults saved approximately US$7M (published in 1998) in product rework and testing [8].

Yu's classification of faults is similar to other fault classification schemes, and we feel that availability of this data could make the development of such fault prevention guidelines more efficient and effective. As a result, software developers get valuable training on best practices that can increase the quality of their source code.

E. Targeting Inspection

Inspection is an important practice in verification and validation of software. It is not always clear, however, when it should be applied, and to what extent. Runeson et al. analyzed several empirical studies to answer this question, and provided some practical findings [9]. They find that inspections are more efficient and effective at finding design specification defects. Code defects are more effectively found by functional or structural testing, but some studies suggest that these activities find different kinds of defects.

Hayes et al. describe a method to improve code inspections through the use of *fault links* [10]. A fault link is a relationship between the type of code fault and the types of components in which they occur. In this experiment they demonstrate that use of fault link information to customize code review checklists can improve the number of faults that are found by 170-200% and the number of hard to find faults by 200-300%. This demonstrates the use of fault classification data, along with properties of the software, to improve code inspections.

Historical data that includes classified fault data can help a manager assign resources appropriately to these activities for a given project. Given historical data about the types of faults that were found using inspection and testing, we could better understand the effectiveness of each activity for a given project or organization. Software components tend to have the same types of faults that have occurred in the past. The combination of a component's history and the effectiveness of past techniques provide quantitative data that can be used to allocate resources on these activities.

F. Summary

In this section we have presented evidence to support the use of fault classification data for five types of decisions that can be supported by software analytics. In each case prior research provides examples of how fault classification data can be used, and its importance to the decision-making process.

These areas may be improved through the use of software analytics while using classified fault data to aid in the decision making process. In many cases, we find that classified fault data is not available. In the next section, we present our position regarding the automatic construction of a fault taxonomy.

IV. FAULT TAXONOMIES FROM CHANGE DATA

This section describes current research efforts to automatically classify software faults through the use of clustering. We present these results as anecdotal evidence that more granular fault data can be extracted using machine learning.

Our current investigation of a fault taxonomy focuses on the classification of the fix for the fault as a proxy for the nature of the fault. This is similar to the *Defect Type* attribute in the Orthogonal Defect Classification scheme [2]. We extract the syntactical differences from the source code changes that repaired each fault. This data is arranged in a feature vector that is used for clustering and analysis. We describe this approach next.

To extract the syntactical changes from the source code, we extend the fine-grained source code changes introduced in the *ChangeDistiller* tool [11]. The algorithm for this tool compares the abstract syntax trees for two revisions of a file [12]. The tool and taxonomy were created for change impact analysis. We were able to extend the taxonomy using contextual information that is captured by the tool [13]. Additional changes were made to the ChangeDistiller project in order to handle problematic constructs such as the presence of anonymous classes. The changes are available as open source code[1].

For each fault we construct a vector with each possible syntactic change as a feature, and the frequency of that change as the value of the feature. As an example, a fault may have three changes to *condition expressions*, one inserted *if statement*, and one inserted *return statement*. The set of vectors from all faults in a version form a dataset. The dataset is provided as input to the clustering algorithm.

Clustering is performed with the CLUTO clustering toolkit [14]. The cosine similarity is used as the distance measure. A repeated bisection clustering algorithm was used with the **I1** criterion function [13]. CLUTO reports the features in the vector that contribute to the internal similarity of each cluster [14]. This provides a quantitative way to name clusters based on the dominant traits. The use of the **I1** criterion function maximizes the tightness of the resulting clusters by providing one poor quality cluster for outliers. Zhao and Karypis conclude that this property may be useful for noisy data sets [15]. Our analysis of this cluster supports their claim.

[1] https://bitbucket.org/bill_kidwell/tools-changedistiller/

Initial results were encouraging in multiple areas. The frequency of occurrence for our extended change types indicate that these changes occur frequently for fault repairs. In addition, the changes were consistent between two versions of Eclipse that were released two years apart, and that included re-architecture from a proprietary runtime to a runtime based on the OSGi® specifications [16]. The clusters that were found for these versions were also consistent, suggesting a possible consistency in the patterns of change. Initial manual analysis indicates that the nature of faults within a cluster is indeed similar: i.e., we have identified a fault type that is part of a larger fault taxonomy. For more details about the resulting clusters, data collection, and validation please refer to our previous publication [13].

In addition, this process can be implemented as part of a continuous integration system. This allows fault data to be automatically collected and updated as code is integrated, builds are completed, and automated tests are running. We believe that the automatable nature of the data collection will increase the likelihood of technology transfer to industry.

V. WHAT NEXT?

This section suggests research areas/goals for using classified faults/fault taxonomies as a data source for software analytics.

A. Fault Classification and Software Analytics

In Section III we provide information about the application of fault classification data in the context of software analytics questions and decisions. Each of these areas provides a starting point for further study. A first step is to implement systems that provide the information for these decisions. Further work can incrementally expand on each of these areas by adding additional information that is not easily available to decision-makers today, thus improving the state of the art for these practices through software analytics.

B. Fault Taxonomies from Change Data

Based on the results to date from Section IV, we found that the syntactical change patterns in fault fixes exhibit consistency, and provide insight into the nature of faults.

Therefore, we feel that we have provided initial validation of our position: that software analytics can assist with the learning of fault taxonomies, and that it is important to integrate these methods into development tools.

To further work in this area, we suggest these research areas/goals:

- Apply additional machine learning algorithms and techniques to improve the automatic classification of faults.
- Study additional projects from multiple domains to understand variance in faults.

- Enhance tools to support multiple programming languages.
- Study the application of recovered fault taxonomies to process improvement, e.g., and verification and validation, with comparisons to manual classification approaches.
- Automate other attributes of fault taxonomies, such as failure type or severity, using machine learning.

C. New Applications for Software Analytics

In addition to the applications of fault classification data that we have discussed in Section III, and the automation of the fault classification process, there are numerous problems that can benefit from classified fault data. Classified fault data provides additional precision to the measurement of fault occurrence that is not possible without it.

As an example of additional applications, we consider the problem of evaluating the success of refactoring efforts. One of the motivations of refactoring is to improve the quality of a software component. Measuring the number of faults after the refactoring provides coarse data, but measuring faults of a particular type can provide more useful data.

As an example, consider a refactoring effort that reduces the complexity of the source code in a highly volatile area of the software. The expected impact of this change is likely to include a reduction in the number of logic faults that are introduced. After the refactoring has occurred, there will still be numerous faults in the component due to the fact that it is a volatile area of the code. If the refactoring was successful, the number of logic faults will be reduced. However, faults due to incomplete or ambiguous requirements are likely to occur here as well. The classification of the faults separates these measurements in a way that allows progress to be tracked more precisely.

VI. Summary

In this paper we provided a two part position about software fault classification and software analytics. First, we believe that fault classification provides useful data for software engineering decisions. In Section III we examine five types of decisions that others have identified for the application of software analytics. For each of these areas we submit evidence from previous studies about the use of classified fault data to aid in the decision making process.

Our second position involves the manner in which software faults are classified. Fault classification has traditionally been a manual task, and there have been many studies that discuss the benefits of the practice. However, industry adoption has been limited to mature organizations. We present initial results from our research supporting our position. In addition, we provide source code changes that were used in our experiment to allow others to contribute to this area.

Finally, we provide our suggestions for future work in three parts. First, we provide suggestions on the use of classified fault data for software analytics. Then, we present future directions for learning fault taxonomies from change data. Finally, we suggest how software analytics and fault classification data can be used to solve new problems, using the assessment of refactoring as an example.

In conclusion, a fault taxonomy provides improved precision in the measurement of software quality. This improved precision can be valuable in the application of software analytics. In order to promote the value of fault classification, we feel that research in automation of the task is warranted.

Acknowledgment

Work on this topic was previously partially funded by NASA under grant NNG05GQ58G.

References

[1] "IEEE Standard Glossary of Software Engineering Terminology," IEEE Computer Society, IEEE Std 610.12-1990, Dec. 1990.

[2] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification-A Concept for In-Process Measurements," *IEEE Trans Softw Eng*, vol. 18, no. 11, pp. 943–956, 1992.

[3] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*, 2nd ed. Morgan Kaufmann, 2005.

[4] R. P. L. Buse and T. Zimmermann, "Information Needs for Software Development Analytics," in *Proceedings of the 34th International Conference on Software Engineering*, Piscataway, NJ, USA, 2012, pp. 987–996.

[5] S. M. Mirsky, J. E. Hayes, and L. A. Miller, "Guidelines for the Verification and Validation of Expert System Software and Conventional Software: Project Summary. Volume 1," Nuclear Regulatory Commission, Washington, DC (United States). Div. of Systems Technology; Electric Power Research Inst., Palo Alto, CA (United States). Nuclear Power Div.; Science Applications International Corp., McLean, VA (United States), NUREG/CR--6316-Vol.1; SAIC--95/1028-Vol.1, Mar. 1995.

[6] S. Vegas, N. Juristo, and V. Basili, "Packaging experiences for improving testing technique selection," *J. Syst. Softw.*, vol. 79, no. 11, pp. 1606–1618, Nov. 2006.

[7] A. T. Misirli, B. Caglayan, A. Bener, and B. Turhan, "A Retrospective Study of Software Analytics Projects: In-Depth Interviews with Practitioners," *IEEE Softw.*, vol. 30, no. 5, pp. 54–61, Sep. 2013.

[8] W. D. Yu, "A software fault prevention approach in coding and root cause analysis," *Bell Labs Tech. J.*, vol. 3, no. 2, pp. 3–21, 1998.

[9] P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, "What do we know about defect detection methods? [software testing]," *IEEE Softw.*, vol. 23, no. 3, pp. 82–90, May 2006.

[10] J. H. Hayes, I. R. Chemannoor, and E. A. Holbrook, "Improved code defect detection with fault links," *Softw. Test. Verification Reliab.*, vol. 21, no. 4, pp. 299–325, Dec. 2011.

[11] H. C. Gall, B. Fluri, and M. Pinzger, "Change Analysis with Evolizer and ChangeDistiller," *IEEE Software*, vol. 26, no. 1, pp. 26–33, 2009.

[12] B. Fluri, M. Wuersch, M. Pinzger, and H. Gall, "Change Distilling:Tree Differencing for Fine-Grained Source Code Change Extraction," *IEEE Trans. Softw. Eng.*, vol. 33, no. 11, pp. 725–743, 2007.

[13] B. Kidwell, J. H. Hayes, and A. P. Nikora, "Toward Extended Change Types for Analyzing Software Faults," in *2014 14th International Conference on Quality Software (QSIC)*, 2014, pp. 202–211.

[14] G. Karypis, "CLUTO: A Clustering Toolkit," University of Minnesota, Department of Computer Science, Minneapolis, MN, Technical Report #02-017, Nov. 2003.

[15] Y. Zhao and G. Karypis, "Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering," *Mach Learn*, vol. 55, no. 3, pp. 311–331, Jun. 2004.

[16] D. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, and T. Watson, "The Eclipse 3.0 platform: Adopting OSGi technology," *IBM Syst. J.*, vol. 44, no. 2, pp. 289–299, 2005.