An Interview with Prof. Andreas Zeller,

(Saarland University, Germany):

Mining Your Way to Software Reliability

Interviewed by Walter Tichy, Karlsruhe Institute of Technology,

Interview conducted April 20, 2010

Editor's Introduction

In 1976, Les Belady and Manny Lehman published the first empirical growth study of a large software system, IBM's OS 360. At the time, the operating system was twelve years old and the authors were able to study 21 successive releases of the software. By looking at variables such as number of modules, time for preparing releases, and modules handled between releases (a defect indicator), they were able to formulate three laws: The law of continuing change, the law of increasing entropy, and the law of statistically smooth growth. These laws are valid to this day.

Belady and Lehman were ahead of their time. They understood that empirical studies such as theirs might lead to a deeper understanding of software development processes, which might in turn lead to better control of software cost and quality. However, studying large software systems proved difficult, because complete records were rare and companies were reluctant to open their books to outsiders.

Three important developments changed this situation for the better. The first one was the widespread adoption of configuration management tools, starting in the mid 1980s. Tools such as RCS and CVS recorded complete development histories of software. These tools stored significantly more information, in greater detail, than Belady and Lehman had available. The history allowed the reconstruction of virtually any configuration ever compiled in the life of the system. I worked on the first analysis of such a history to assess the cost of several different choices for smart recompilation (ACM TOSEM, Jan. 1994). The second important development was the inclusion of bug reports and linking them to offending software modules in the histories. This information proved extremely valuable, as we shall see in this interview. The third important development was the emergence of open source, through which numerous and large development histories became available for study. Soon, workers began to analyze these

repositories. Workshops on mining software repositories have been taking place annually since 2004.

I spoke with Prof. Andreas Zeller (shown on the previous page) about the nuggets of wisdom unearthed by the analysis of software repositories. Andreas works at Saarland University in Saarbrücken, Germany. His research addresses the analysis of large, complex software systems, especially the analysis of why these systems fail to work as they should. He is a leading authority on analyzing software repositories and on testing and debugging.

Walter Tichy
Editor

An Interview with Prof. Andreas Zeller,

Saarland University, Germany:

Mining Your Way to Software Reliability

Interviewed by Walter Tichy, Karlsruhe Institute of Technology,

Interview conducted April 20, 2010

Walter Tichy: Analyzing software repositories is attracting a lot of attention these days. Why is this happening? Why is it interesting?

Andreas Zeller: Repositories are the lifeblood of modern programming practice. Programmers use software repositories all day—to store versions and changes (version archives), to track and assign problems (problem databases), or to keep track of special issues (vulnerability databases). In the past decade, usage of automated repositories has increased dramatically. Today, researchers can leverage these repositories to reconstruct and understand the evolution of a software system—all changes, all problems, and all fixes. Repositories are great sources of unbiased data on how a product came to be—something that's very valuable and hard to find.

WT: What are some of the results that you and others have come up with?

AZ: From an industrial perspective, most important is the ability to locate weak spots in a system, which we can do by figuring out where in a system the most bugs have been fixed in the past. By mining the Firefox history, for instance, we found that most security issues were located in the Javascript engine—which confirmed the intuition of programmers and managers. We also found severe problem spots in places like the HTML layout engine or the document object model, though, which the developers did not know about. Generally, mined information is most valuable where it deviates from intuition.

Once you know where bugs have been in the past, you can try to predict future ones. Microsoft has had great successes relating bug densities to the amount of change, complexity metrics, or recently, the structure of the development organization. When Microsoft realized that bugs with no clear ownership took too long to fix, it reorganized its Windows 7 development teams according to the empirical evidence.

WT: In what way did Microsoft reorganize the development teams, and was that successful?

AZ: This is well described in an article in the German *c't* magazine for computer technology 23/2009: "Compared to Vista, the new management structure became much flatter. Instead of a single hierarchical structure with two teams (development and testing), the Windows 7 development team was separated into 25 individual subteams, each of them responsible for a part of the final product in terms of code, functionality, and quality assurance. This new structure allowed for a much better team management—teams that would fit into a single meeting room, with short communication paths, and who could all have a beer together without filling the entire bar." All this reorganization was directly influenced by relating the Vista structure to defect densities.

WT: I don't see how defect densities lead to a flat team structure. Please clarify.

AZ: In 2008, Nachi Nagappan, Brendan Murphy, and Vic Basili carried out a study at Microsoft, entitled "The influence of organizational structure on software quality." The major result of this study was that of all factors, organizational complexity metrics were the strongest defect predictors. This is just reasonable: If you have too many people who only feel semi-responsible about an issue, you won't get it fixed. Or, in more layman terms: Too many cooks spoil the broth. Consequently, Microsoft reorganized the subteams around features, resulting in a clear responsibility (or "one cook") for each feature.

WT: Is it correct to say that most likely bugs will appear where bugs were in the past?

AZ: Yes, this is correct. It is like fishing: Just like fish will appear at the same spot again and again, bugs also tend to cluster in certain spots. For fish, though, we have a theory on how they reproduce. The weird thing about bugs is that the more you fix, the more you will still find. Bugs adhere to the Pareto principle: 80 percent of all defects will be found in 20 percent of the locations. As soon as you identify these "usual suspects," you can focus your efforts towards them and thus increase effectiveness.

WT: With the information from past projects, is it possible to predict the location or density of bugs in new projects?

AZ: All these prediction methods are based on similarity: New modules similar to defect-prone ones will likely be defect-prone, too. In our own research, one of the major similarity factors

was the usage of specific interfaces. In Eclipse, for instance, modules that interacted with compiler internals were seven times more likely to have post-production defects than code that dealt with the user interface. Such properties will carry over to new projects, too—and tell us which parts will be easy and which ones will be hard to get right.

WT: Are there other insights that can be gained from analyzing software repositories besides identifying bug-prone components?

AZ: Several. You'd like to know where the effort goes to, for instance. Unfortunately, few people record the effort in a way that could be attributed to specific code locations. You'd also like to know where most change takes place, or how change propagates within your system. By correlating defects with test coverage, you'll find out whether your testing effort is well directed. You can check for compliance with specific processes. There are many things your data can tell you.

WT: I can see that large companies can benefit, but what about smaller software outfits? How can they profit from analyzing their software and bugs?

AZ: Even a single developer can profit from knowing where the most problems have been in the past—by refocusing her testing or maintenance efforts, for instance. All it takes is a certain discipline in recording your changes and your bugs—but this is pretty standard today even in the smallest of outfits.

WT: What are your recommendations for individual software developers?

AZ: Go and collect data; analyze it; and finally, learn from it.

WT: What's the future of this area for research and practice?

AZ: Five years ago, the discovery that all this data was there for exploring created a gold rush in research; lots of people mined the data and indeed found a few nuggets. Today and in future, the challenge will be to find not only correlations, but actual insight into how to develop software. Mining thus becomes an important technique in empirical software engineering, which still must be complemented with qualitative and experimental research.

In practice, there's still a lot of potential in systematically collecting development data -- it will make you more productive, more efficient, and more systematic, too. Your decisions will be

guided by evidence, complementing and occasionally rechanneling your intuition. And we will see much better tool support: future versions of Visual Studio will integrate such empirical predictions.

WT: Are there any tools that you would recommend that practitioners could use today for extracting interesting facts from their software repositories?

AZ: Unfortunately, there are few publicly available tools simply because there are so many ways the data can be organized. In our experience, it takes between a few days and a few weeks to access and map central data such as changes and defects.

WT: Where can readers find out more about results in this area?

AZ: As a starting point, I'd recommend *Predicting Bugs from History* by Zimmermann et al., describing the basics of mining and prediction. It has appeared as a chapter in the book *Software Evolution* by Springer. The chapter is available for free at the book's Web site.

For the latest and greatest in terms of research, I recommend the symposium on mining software repositories (MSR) as well as the workshop on recommendation systems (RSSE). The proceedings will give you an idea of what's going on in the field. The venues are great meeting places, too!

DOI: 10.1145/1880066.1883621

