Week 1 Journal

Rodger Byrd

I. Course Goals

My primary goal for this course are to determine whether I want to pusue a Doctor of Philosophy degree. My other goals include determining whether I'm interested in reasearch, and I would particularly like to learn how to analyze data and show statistical relevance to support a hypothesis. I'd like to learn what kind of tests you can perform on the data to support your hypothesis. I'm studying Computer Science and this will be the last class I take for my master's degree. I completed my undergraduate studies in Electrical Engineering, but I've mostly worked in the fields of software development and cyber security.

Fig. 1. Rodger Byrd

I work full time for an engineering consulting firm in the field of cybersecurity. My main motivation for graduate school was to become credentialed in the specific field that I've been working in since I completed my Bachelor's Degree. I would also like to open up more career options. I have two young children and with career, my available free time is constrained, so the decision to pursure more education after I complete my Master's Degree is a difficult one. I'm hoping this course will help to inform that decision. I've included a picture of myself above in figure 1.

II. Git Repository

The files for this latex document are in the github repository located at `https://github.com/rodger79/CS6000`. I used Texmaker and BibTeX to build the pdf from the latex project files.

The first problem I encountered was getting my latex tool to work with the IEEEtran plugin. I also learned that doing references for 25 papers can be very time consuming. I hadn't used a separate .bib file in a latex project before so that took a little troubleshooting to get to work. I also had to remember how to use math mode in Latex. It's been a while since I've used it.

III. Research

I'm particularly interested in anti-patterns in code so most of the papers I found were related to that field.

The first of the five papers I read was about very small patterns in source code and how they can be used to identify potential vulnerabilites in the code. The authors[1] referred to these as micro or nano patterns. Nano patterns were defined as a method level pattern and they showed that they could better predict vulnerable methods in code than with software metrics.

The next paper I read looked at anti-patterns in exception handling. They[2] used data analysis to identify multiple anti-patterns in exception handling that could cause software failures.

The third paper[3] I looked at studied whether software design patterns were actually useful in training new software developers. They claim that novice software developers misuse design patterns, but they didn't have good data to support their claim.

The next paper [4] came up with a technique to use unit testing to identify anti-patterns in code. They came up with a tool to detect known anti patterns in source code.

The last paper [5] looked at how anti-patterns can redcue the benefits of continuous integration practices. They used the following expression $(\Delta_{Breaks} < 0) \wedge (\Delta_{Runs} < 0 \vee \Delta_{Skipped} > 0)$ to determine whether a specific unit test was skipped in a following build.

The other papers are cited in the bibliography. [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [3] [25]

References

[1] K. Z. Sultana, B. J. Williams, and A. Bosu, "A comparison of nano-patterns vs. software metrics in vulnerability prediction." 25th Asia-Pacific Software Engineering Conference, 2018, p. 10.

[2] G. B. de Pdua and W. Shang, "Studying the prevalence of exception handling anti-patterns." IEEE 25th International Conference on Program Comprehension, 2017.

[3] A. Ghazarian, "Work in progress: Transitioning from novice to expert software engineers through design patterns: Is it really working?"

[4] H. Kaur and P. J. Kaur, "A gui based unit testing technique for anti-pattern identification."

[5] C. Vassallo, S. Proksch, H. C. Gall, and M. D. Penta, "Automated reporting of anti-patterns and decay in continuous integration." IEEE/ACM 41st International Conference on Software Engineering, 2019, p. 11.

[6] F. Jaafar, F. Khomh, Y.-G. Guhneuc, and M. Zulkernine, "Anti-pattern mutations and fault-proneness." 14th International Conference on Quality Software, 2014, p. 10.

[7] W. Ma, L. Chen, Y. Zhou, and B. Xu, "Are anti-patterns coupled? an empirical study." IEEE International Conference on Software Quality, Reliability and Security, 2015, p. 10.

[8] B. Chen and Z. M. J. Jiang, "Characterizing and detecting anti-patterns in the logging code." IEEE/ACM 39th International Conference on Software Engineering, 2017, p. 11.

[9] W. Fenske, "Code smells in highly configurable software." International Conference on Software Maintainence, 2015, p. 4.

[10] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana, and R. Oliveto, "How do community smells influence code smells?" ACM/IEEE 40th International Conference on Software Engineering: Companion Proceedings, 2018, p. 2.

[11] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews." IEEE International Conference on Software Maintenance and Evolution, 2017.

[12] ——, "Confusion in code reviews: Reasons, impacts, and coping strategies." IEEE International Conference on Software Analysis, Evolution and Reengineering, 2019.

[13] N. Nahar and K. Sakib, "Acdpr: A recommendation system for the creational design patterns using anti-patterns." IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, 2016.

[14] C. Denzler and D. Gruntz, "Design patterns: Between programming and software design." 30th International Conference on Software Engineering, 2008.

[15] M. Huaxin and J. Shuai, "Design patterns in software development."

[16] J. S. Fant, "Building domain specific software architectures from software architectural design patterns."

[17] M. Aoyama, "Evolutionary patterns of design and design patterns."

[18] D. J. Ferreira, "Work in progress: Exploring programming anti-patterns as emphasis on creativity in the teaching of computer programming."

[19] C. Izurieta and J. M. Bieman, "How software designs decay: A pilot study of pattern evolution." First International Symposium on Empirical Software Engineering and Measurement.

[20] R. F. G. Antoniol and L. Cristoforetti, "Using metrics to identify design patterns in object-oriented software."

[21] T. Feng, J. Zhang, H. Wang, and X. Wang, "Software design improvement through anti-patterns identification."

[22] F. ZENG, A. CHEN, H. Wang, and X. TAO, "Study on software reliability design criteria based on defect patterns."

[23] K. Z. Sultana, "Towards a software vulnerability prediction model using traceable code patterns and software metrics."

[24] Z. Zemin, "Study and application of patterns in software reuse." IITA International Conference on Control, Automation and Systems Engineering, 2009.

[25] A. Tahmid, N. Nahar, and K. Sakib, "Understanding the evolution of code smells by observing code smell clusters." IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, 2016.