To: Engineering Communications

From: Patrick Austin (10-1)

Date: April 27, 2017

Subject: Professional Development

**Foraging and Navigations, Fundamentally: Developers’ Predictions of Value and Cost - Foundations of Software Engineering Conference Paper**

Past studies of professional software developers have shown that simply navigating through existing source code can consume up to 50% of a developer’s time. The authors attempt to expand on this finding by studying how software engineers make estimates about the cost and value of their navigation decisions within source code. They use Information Foraging Theory, a theoretical foundation which describes how people seek information in a software environment, to study ten professional developers at Oracle and their programming habits over a two hour coding session. The authors found that the programmers overestimated the value of a code search 50% of the time more than the programmers predicted they would, and underestimated the cost 40% more often than they predicted they would. In order to improve these figures, which would hopefully lead to more productive programmers who are better capable of making estimates of time and manpower needed for programming projects, the authors recommend the use of programming tools such as Sketch Link and Team Tracks, which help visualize and provide links within source code so the relationships between different areas of code can be understood more easily.

**How do Fixes Become Bugs? - Foundations of Software Engineering Conference Paper**

A common phenomenon in software development is that fixing a bug can introduce more bugs, even ones that are more severe than the bug originally being addressed. In order to analyze this phenomenon, the authors conducted a study of 970 randomly selected fixes for major operating systems such as Linux and FreeBSD. The researchers found that between 14.8% and 24.4% of all fixes, depending on the operating system, were incorrect and introduced additional bugs that impacted end users. These bugs ranged in effect from system crashes, to security vulnerabilities, to data corruption. By studying the pattern of these incorrect fixes, the authors observed that bugs dealing with concurrency, relating to the maintenance of multiple programs running simultaneously and sharing computer resources, were the ones that most commonly led to an incorrect fix, constituting 39% of all incorrect fixes. The authors also studied the code knowledge of the developers of these fixes, and found that 27% of all incorrect fixes were made by developers who had not previously been exposed to the code being fixed. As a recommendation, they encourage software developers to assign programmers with more past experience with a given piece of code to bugfixes for that code.

**Learning Natural Coding Conventions - Foundations of Software Engineering Conference Paper**

Programmers often learn or develop their own idiosyncratic styles for formatting code. This can be problematic when a group of programmers come together on a team for a large, professional software project, which almost always requires that the code be easily readable and maintainable for all current members of the team as well as future programmers who might maintain the software. The authors of this paper have attempted to address this problem by creating a suite of software called Naturalize, which learns the style of a codebase and is then capable of suggesting formatting changes to new pieces of code to match the learned standard. The authors go into considerable detail on the algorithms they have designed for the Naturalize framework, which include components that handle conventions for naming variables, matching syntax, and creating patterns of spacing. The suggestions of Naturalize come within a window of confidence the user can modify in order to receive a broader range of suggested changes. The authors tested the Naturalize framework and found that it achieved 94% accuracy in suggestions for identifier names. They also used the Naturalize framework to generate 18 patches for five open source projects to improve code formatting consistency. Of these, 14 were accepted.

**Temperature Management in Data Centers: Why Some (Might) Like It Hot - SIGMETRICS Conference Paper**

The authors note that data centers consume nearly 3% of the world’s electricity, and nearly a third of that energy is spent on cooling. They also note previous studies suggesting that allowing a small increase in temperature within a datacenter can significantly reduce energy consumption. To the end of reducing energy costs and the carbon footprints of data centers, the authors study the setpoint temperature for various hardware components. The setpoint temperature is a safe temperature for operation of a hardware component suggested by the hardware manufacturer. However, the authors experiment with exceeding the setpoint temperature for hard disks, DRAM, and other components and find that the suggested setpoint temperatures are very conservative. DRAM setpoint temperatures can be greatly exceeded with no noticeable consequences, hard disk temperatures are found to be correlated more strongly to variability in temperature than average temperature, and still other components are found to become more likely to fail with higher temperature in a linear fashion, rather than the generally believed exponential fashion. These observations suggest that data centers can allow higher temperatures in order to reduce cooling costs without incurring significantly higher failure rates for hardware components. However, various factors would have to be taken into account to implement this solution, including more careful management of fans within systems and the introduction of measures to more quickly act on cooling system failures, which would produce adverse consequences more quickly in a hotter data center.

**Vandalism Detection in Wikidata - Conference on Information and Knowledge Management Conference Paper**

Wikipedia is commonly known as an online encyclopedia that is free for all to edit, but vandalism and defacement of Wikipedia pages by anonymous contributors has proved to be a persistent problem. Manually reversing vandalism on Wikipedia is a time consuming process; an algorithm to automatically detect and reverse vandalism would increase the productivity of Wikipedia’s legitimate editors. The authors used machine learning to analyze 24 million Wikipedia revisions which had each been marked as legitimate or vandal in hopes of finding telltale properties of vandal revisions that could be used to detect submissions vandal that should be declined or reversed. In all, the machine learning algorithm analyzed 47 characteristics of Wikipedia edits that correlated to vandalism, which are then used in an algorithm that asserts the probability that an edit is legitimate or vandalism. This algorithm significantly outperforms the existing Wikidata Abuse Filter in its ability to detect vandal edits. The authors suggest future research that could further explore ways to avoid false positives. They also suggest the idea of providing user-friendly explanations of the algorithm’s reasoning when edits are denied, in order to facilitate manual review of denials and in order to provide more information to legitimate users who may have accidentally tripped the filter.

**Why We Refactor? Confessions of GitHub Contributors - Foundations of Software Engineering Conference Paper**

Refactoring is a term used to refer to improving the design of code without changing its behavior. The authors attempt to study the motivations behind refactoring code by studying a selection of 124 popular Java projects hosted on the code repository website GitHub. The authors attempt to understand why instances of refactoring were undertaken in these projects by asking the software developers for their motivation, then applying a principle of thematic analysis to categorize the responses into different types. The most common type of refactoring was extracting a method, with the most common motivation being to take a piece of code used in a single code location and make it capable of being used independently in multiple different locations. The authors examine and characterize the survey results and conclude that most refactoring is driven by the addition of new requirements for the code rather than merely following best coding practice, and that manual refactoring is prevalent even though powerful supporting tools exist that can perform automatic refactoring. The authors recommend future research in systems that analyze code and then recommend and guide the refactoring process. They suggest that improvements in such systems would lead to more developers adopting them, which would lead to more productivity via less time spent refactoring by hand.

**ASME Critical Thinking Podcast**

The narrator of the podcast reviews concepts such as arguments, perception, and problem solving that contribute to what is commonly called critical thinking. In exploring the concept of perception, the narrator explores the way in which people can have their beliefs and biases shaped by life experience. As an example, the narrator discusses two workers who might perceive a third coworker entirely differently solely on the basis of the random events each experienced with him. In exploring the concept of an argument, the narrator breaks down the concepts of a premise and a conclusion, and gives steps to identify and resolve fallacious arguments. In the discussion of problem solving, the narrator emphasizes the importance of a problem solving approach compared to relying on past knowledge alone. The approach the narrator suggests is to identify the problem, list the unknowns, list the knowns, and then relate the knowns and the unknowns. The narrator discusses an example where people were asked how to cure a tumor using radioactive lasers by comparing the situation to a castle under siege. Some were able to connect and analogize between these scenarios, but many others failed to transfer information between these cases due to false preconceptions and misunderstandings about the transfer of concepts and use of terms such as beam and dose.

**ASME Safety and Risk Podcast**

The narrator of the podcast discusses concepts relating to safety, risk analysis, and risk management. The narrator contrasts how average citizens view risk, qualitatively, with how engineers should view risk, as a quantitative measure of the likelihood of an event and its impact. The narrator discusses the two most popular methods for identifying risk, hazard and operability analysis and failure modes. The former is used to generate scenarios in which problems might arise due to an otherwise unanticipated combination of causes and effects. For example, a hazardous situation might arise if some system experiences low pressure and high temperature simultaneously. The latter method, failure modes, is used to generate scenarios where the item being assessed for risk is operating in a non-ideal state. By analyzing the failure modes one can assess possible causes for failure and the potential for risk in the first place. The narrator also discusses the process of estimating the impact of consequences of failure, so as to accurately assess the total risk. These impacts can include damage to property, life, and image.

**ASME Team Building Podcast**

The narrator discusses strategies that can be used to make teams as effective as possible, so as to improve productivity and happiness in the workplace. The narrator compares different types of teams, such as functional teams, cross-functional teams and autonomous teams. Each type of team has a different power structure, and different advantages and weaknesses. The narrator describes the means to make teams effective via effective communication and reduction of conflict. He recommends understanding opposing viewpoints and working towards a consensus solution that all parties in the team can accept. He discusses when teams should be formed and how to choose members for a team by balancing different types of personalities, such as introverts and extroverts, in order to achieve team synergy. Successful teams develop in a four stage process of form, storm, norm, and perform, concluding in an elevated level of performance and synergy created by the group coming together. Ways to overcome common team failures, such as lack of trust or communication, are discussed to prevent failures and problems from recurring. This is especially relevant for global teams or teams that primarily communicate online via email. One such way to overcome team failure discussed by the narrator is to have clearly defined team roles and responsibilities.

**References**

Allamanis, M., Barr, E., Bird, C., and Sutton, C. "Learning natural coding conventions." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014.

ASME Professional Practice Curriculum (website). Retrieved from

http://php.aist.org/ela/training\_series.htm

El-Sayed, N., Stefanovici, I., Amvrosiadis, G., Hwang, A., and Schroeder, B. "Temperature management in data centers: why some (might) like it hot." ACM SIGMETRICS Performance Evaluation Review Vol. 40, Ed. 1, 2012.

Heindorf, S., Potthast, M., Stein, B., and Engels, G. "Vandalism detection in Wikidata." Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, 2016.

Piorkowski, D., Henley, A., Nabi, T., Fleming, S., Scaffidi, C., and Burnett, M. "Foraging and navigations, fundamentally: developers' predictions of value and cost." Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016.

Silva, D., Tsantalis, N. and Valente, M. "Why we refactor? confessions of GitHub contributors." Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016.

Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. "How do fixes become bugs?." Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 2011.