



# Quality, Nontechnical Skills, Blind Programmers, and Deep Learning

Jeffrey C. Carver, Birgit Penzenstadler, Alexander Serebrenik, and Mirosław Staron

**THIS ISSUE'S "PRACTITIONERS' Digest"** department reports from the 34th International Conference on Software Maintenance and Evolution, the 44th Euromicro Conference on Software Engineering and Advanced Applications, and the 12th International Symposium on Empirical Software Engineering and Measurement (ESEM). Feedback or suggestions are welcome. In addition, if you try or adopt any of the practices included in the column, please send Jeffrey C. Carver and the authors a note about your experiences.

## The Pareto Principle for Defects

"Are 20% of Files Responsible for 80% of Defects?" by Neil Walkinshaw and Leandro Minku reports on an empirical study investigating industrial anecdotes stating that the Pareto principle holds for the relationship between source code and the number of defects in a system.<sup>1</sup> The Pareto principle (also known as

the *80/20 rule*, the *law of the vital few*, or the *principle of factor sparsity*) states that, for many events, roughly 80% of the effects come from 20% of the causes. Management consultant Joseph M. Juran suggested the principle and named it after Italian economist Vilfredo Pareto, who noted the 80/20 connection in 1896 when he found that approximately 80% of the land in Italy was owned by 20% of the population. In this study, the authors analyzed 100 systems from active GitHub repositories. They analyzed the relationships among files, basic metrics, such as code churn and lines of code (LOC), and defect fixes. The analysis shows that the Pareto principle holds only if each fix counts as an individual defect—in other words, if the bug exists in multiple files, it counts as multiple defects. In addition, code churn was a more reliable indicator of defect proneness than was LOC but only for extremely high-churn values. The overall conclusion is that it is difficult to identify the "most fixed" 20% of files using only basic metrics. However, even if those files

could be identified, focusing only on those files would be insufficient because fixes often involve multiple files, including those fixed less frequently. Access this paper at [http://bit.ly/PD\\_2019\\_March\\_1](http://bit.ly/PD_2019_March_1).

## Nontechnical Skills and Agile Development

"Non-Technical Individual Skills Are Weakly Connected to the Maturity of Agile Practices," by Lucas Gren and colleagues, reports on a developer survey that studies the belief that the nontechnical skills of individual developers are able to predict team-level performance in relation to collaboration.<sup>2</sup> Because agile approaches emphasize people and their skills, many believe that both technical and nontechnical individual competencies contribute to team capabilities. This survey of 113 agile developers from six organizations in The Netherlands and Brazil found a different result by asking developers how personally satisfied they were with their competency in a specific nontechnical skill. In this case, personal satisfaction was more important than simply rating their level of competency because personal satisfaction

Digital Object Identifier 10.1109/MS.2018.2883874  
Date of publication: 22 February 2019

relates to putting the skill into practice in a team. Interestingly, the results of the survey showed that individual nontechnical skills had little power in predicting (i.e., explaining the variance in) how mature the agile practices were within a team. The authors concluded that, to advance the maturity of agile practices, it is more important to focus on assessing and improving the capacity of the team relative to nontechnical skills rather than ensuring that individual team members possess those skills. Access this paper at [http://bit.ly/PD\\_2019\\_March\\_2](http://bit.ly/PD_2019_March_2).

authors believe that it is often cheaper to prevent TD than to repay the debt later, the goal of this survey was to understand the most common causes and effects of TD-to-TD prevention. The results of the survey showed that the survey respondents were familiar with the concept of TD. Some of the most likely causes of TD were deadlines, inappropriate planning, lack of knowledge, and lack of a well-defined process. Some of the most impactful effects of TD were low quality, delivery delay, low maintainability, rework, and financial loss. The authors plan to

cannot take advantage of these tools. The AudioHighlight tool begins to bridge this gap by providing code navigation support for blind programmers and rendering the code in a browser or IDE with Hypertext Markup Language (HTML) tags on structural elements, such as classes, functions, and control flow structures. Using a virtual cursor, these HTML tags allow a blind programmer to quickly navigate all methods in a class rather than having to go line by line with a traditional screen reader. An evaluation with 10 blind programmers showed that AudioHighlight was faster and easier to use than the state-of-the-art tools, without reducing accuracy. The tool also promoted faster and easier program comprehension. Access this paper at [http://bit.ly/PD\\_2019\\_March\\_4](http://bit.ly/PD_2019_March_4).

Because agile approaches emphasize people and their skills, many believe that both technical and nontechnical individual competencies contribute to team capabilities.

### Technical Debt

“The Most Common Causes and Effects of Technical Debt: First Results From a Global Family of Industrial Surveys,” by Nicolli Rios and colleagues, reports the views of technical debt (TD) from a survey of 107 Brazilian software practitioners.<sup>3</sup> TD contextualizes the tradeoffs between the short-term benefit of a software development choice, e.g., increased productivity or shorter release time, and the long-term “debt” incurred by that choice, e.g., later tasks become more time-consuming or error-prone. The concept is that the debt incurred for the short-term benefit must be paid back, with interest, later in the development process. Software projects commonly incur TD, which brings risks and management difficulties. Because the

repeat this survey in the coming years to continue to gain a better and more generalizable understanding of TD, based on empirical results. This paper appears in the Industry Track of ESEM 2018. Access this paper at [http://bit.ly/PD\\_2019\\_March\\_3](http://bit.ly/PD_2019_March_3).

### Supporting Blind Programmers

“AudioHighlight: Code Skimming for Blind Programmers,” by Ameer Armary and colleagues, addresses the challenges of making IDE- and web-based tools more accessible for blind programmers.<sup>4</sup> Currently, IDEs and web-based platforms such as GitHub contain tools to support the daily tasks of sighted programmers. Unfortunately, blind programmers often

### Quality Metrics Misperceptions

“Improving Code: The (Mis)perception of Quality Metrics,” by Jevgenija Pantiuchina and colleagues, focuses on the ability of software metrics to capture how software developers perceive source code quality.<sup>5</sup> The authors extracted 1,282 commits from 986 GitHub projects in which the comment explicitly referred to the developers’ intention to improve quality attributes such as cohesion, coupling, code readability, or code complexity. They then determined whether the code quality, as measured by state-of-the-art software metrics, reflected this improvement. Results showed that the quality metrics often did not reflect the developer’s perception of improved quality. This mismatch suggests that the developer’s perception of quality is multifaceted and the metrics might be, at best, reflecting only some of these facets. Therefore, the



**JEFFREY C. CARVER** is a professor in the University of Alabama's Department of Computer Science. Contact him at [carver@cs.ua.edu](mailto:carver@cs.ua.edu).



**BIRGIT PENZENSTADLER** is an assistant professor of software engineering at California State University, Long Beach. Contact her at [birgit.penzenstadler@csulb.edu](mailto:birgit.penzenstadler@csulb.edu).



**ALEXANDER SEREBRENIK** is an associate professor in Eindhoven University of Technology's Department of Mathematics and Computer Science. Contact him at [a.serebrenik@tue.nl](mailto:a.serebrenik@tue.nl).



**MIROSLAW STARON** is a professor of software engineering in the University of Gothenburg's Department of Computer Science and Engineering. Contact him at [miroslaw.staron@gu.se](mailto:miroslaw.staron@gu.se).

authors call for caution with building and using software quality metrics and, thus, the applications built upon those metrics, such as code smell detectors and refactoring recommenders. Access the paper at [http://bit.ly/PD\\_2019\\_March\\_5](http://bit.ly/PD_2019_March_5).

### Deep-Learning Challenges

“Software Engineering Challenges of Deep Learning,” by Anders Arpteg and colleagues, shares their experiences with building deep-learning

systems.<sup>6</sup> While production-level deep-learning systems are increasing in popularity through their use in applications, including weather prediction, house pricing prediction, and autonomous driving software, their construction is challenging. Production-level deep-learning systems present a number of difficulties, including choice of the proper algorithm, data filtering, and run-time quality. Using experiences from seven industrial deep-learning projects, the authors of

this paper identified three categories of 12 specific challenges:

- development challenges, which are related to the software engineering of deep learning, include experience management, limited transparency, troubleshooting, resource allocation, and testing.
- production challenges, which are related to postdeployment life-cycle phases, include dependency management, monitoring and logging, unintended feedback loops, and glue code.
- organizational challenges, which are related to organizing the development of deep-learning systems, include effort estimation, privacy and safety, and cultural differences.

**T**his paper provides a nice overview of the challenges developers must address to be prepared for large-scale software development methodologies for deep-learning systems built to analyze big data. Access this paper at [http://bit.ly/PD\\_2019\\_March\\_6](http://bit.ly/PD_2019_March_6).

### References

1. N. Walkinshaw and L. Minku, “Are 20% of files responsible for 80% of defects?” in *Proc. 12th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement (ESEM 18)*, 2018.
2. L. Gren, A. Knauss, and C. Johann Stettina, “Non-technical individual skills are weakly connected to the maturity of agile practices,” *Inf. Softw. Technol.*, vol. 99, pp. 11–20, 2018.


(continued on page 136)

simply maximize code health, we must instead contribute to a making a good decision.

This article has laid out my way of thinking about decisions. There are topics that are relevant to good decision making that, in my experience, are rarely discussed, such as the cultural dynamics that affect code quality and how well the code expresses theories about the problem domain and the architecture.

Long-term health of the code depends on the decision makers having the right information and knowing the implications of their decisions. As someone who reads and writes software for a living, you have a special role: you must inform the others about what's happening in the code because what they know about the code comes only from you. If you are able to collaborate with the decision makers and bring the information about tradeoffs happening in the code, then they will avoid the temptation to decide based simply on features and timelines, a

👤 ABOUT THE AUTHOR



**GEORGE FAIRBANKS** is a software engineer at Google. Contact him at [gf@georgefairbanks.com](mailto:gf@georgefairbanks.com).

short-sighted approach that can lead us to never change the oil in our cars because we simply must get to our appointments. 🙏

## References

1. M. Fowler, *Refactoring*. Reading, MA: Addison-Wesley, 2018.
2. M. Feathers, *Working Effectively With Legacy Code*. Englewood Cliffs, NJ: Prentice Hall, 2004.
3. A. Tsakiris, "Managing software interfaces of on-board automotive controllers," *IEEE Softw.*, vol. 28, no. 1, pp. 73–76, Jan.-Feb. 2011. doi: 10.1109/MS.2011.11.
4. T. Wolfe, *From Bauhaus to Our House*. New York: Farrar, Straus, and Giroux, 1981.
5. P. Naur, "Programming as theory building," *Microprocessing Microprogramming*, vol. 15, no. 5, pp. 253–261, May 1985. doi: 10.1016/0165-6074(85)90032-8.
6. W. Cunningham, "The WyCash portfolio management system," in *OOPSLA '92 Addendum to the Proc. Object-Oriented Programming Systems, Languages, and Applications*, Vancouver, British Columbia, Canada, 1992, pp. 29–30.

## PRACTITIONERS' DIGEST (continued from page 129)

3. N. Rios, R. Oliveira Spínola, M. Mendonça, and C. Seaman, "The most common causes and effects of technical debt: First results from a global family of industrial surveys," in *Proc. 12th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement (ESEM 18)*, 2018.
4. A. Armaly, P. Rodeghero, and C. McMillan, "AudioHighlight: Code skimming for blind programmers,"

in *Proc. 2018 IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, pp. 206–216.

5. J. Pantiuchina, M. Lanza, and G. Bavota, "Improving code: The (mis) perception of quality metrics," in *Proc. 2018 IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, pp. 80–91.
6. A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning,"

in *Proc. 2018 44th Euromicro Conf. Software Engineering and Advanced Applications (SEAA)*, pp. 50–59.



IEEE COMPUTER SOCIETY

**DIGITAL LIBRARY**

Access all your IEEE Computer Society subscriptions at

**[computer.org/mysubscriptions](http://computer.org/mysubscriptions)**