Editor: Jeffrey C. Carver
University of Alabama
carver@cs.ua.edu

Software Quality, Energy Awareness, and More

Jeffrey C. Carver, Aiko Yamashita, Leandro Minku, Mayy Habayeb, and Sedef Akinli Kocak

FOLLOWING ON last issue's column, this column reports on six more papers from the 2015 International Conference on Software Engineering (ICSE) and its satellite events.

"Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality," by Mathieu Lavallée and Pierre Robillard, identified 10 organizational factors that can decrease software quality. To identify the factors, Lavallée and Robillard observed 10 months of weekly status meetings about an in-house software project at a large telecommunications company. Representative factors include these:

- *Internal dependencies*. Many dependencies exist between software modules, and conflicts on scheduling deployments exist between projects.
- *External dependencies*. Long-term dependencies on third-party libraries exist, and change requests to those libraries cause delays.
- *Organically grown processes*. Processes emerge as needed, usually after a crisis, and are often introduced locally rather than organization-wide.
- *Budget protection*. Developers feel it's cheaper in the short term to build a wrapper than to solve a problem.
- *Scope protection*. Rather than prioritizing a global scope, teams prioritize a local scope and transfer as many requirements as possible to other projects.
- *Undue pressure*. Managers and senior developers circumvent team policies to give direct orders to the team and threaten it.

Better training and process management can avoid operational faults and misuse.

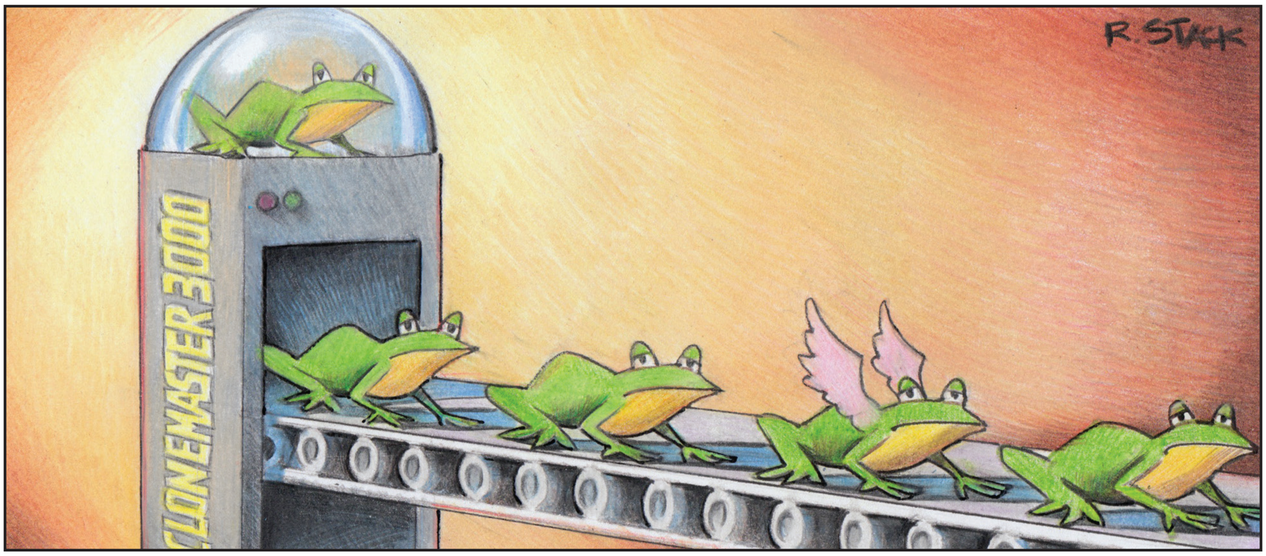
For each factor, the authors suggested corrective actions.

Although the authors' findings indicate that these problems might not affect project success, they do affect software quality, which in turn increases software maintenance costs over time. Other key takeaways include these:

- The lack of centralized strategies for making key decisions might be reflected through conflicting software modules.
- Practitioners can use these organizational antipatterns as hints to warn stakeholders, customers, managers, and team members when their actions might result in code that's costlier to maintain.

This paper appeared in the main research track of ICSE '15; access it at http://goo.gl/yrCVHh.

"The Last Line Effect," by Moritz Beller and his colleagues, discussed the phenomenon in which the last line or statement of a micro-clone (a very short segment of duplicated code) is much more likely to be faulty than any other line or statement. (The last-line effect has caught the reddit community's attention; see www.reddit.com/r/programming/comments/270orx/the.)



To detect such microclones, which traditional clone detection tools miss, Beller and his colleagues used the PVS-Studio static-analysis tool (www.viva64.com/en/pvs-studio). They analyzed 202 microclones from 208 well-known open source systems. They found that, when a fault was present, the last statement was 17 times more likely to be faulty than all other statements combined. For faulty microclones consisting of only two statements, the second statement was always the faulty one.

These results suggest that developers must be extra careful when reading, modifying, reviewing, or creating the last line or statement of a microclone, especially after performing copy-and-paste. As code quality consultant Thomas Kinnen said, "I perform tons of code reviews. After having read 'The Last Line Effect,' I check the last line or statement in a microclone extra carefully." This paper appeared in the International Conference on Program Comprehension; access it at http://goo.gl/bbkGH9.

"An Empirical Study on Quality Issues of Production Big Data Platform," by Hucheng Zhou and his colleagues, described an analysis of 210 incidents from Microsoft ProductA (anonymized), a company-wide multitenant big data computing platform serving thousands of customers from hundreds of teams. System and customer factors led to the highest proportion of incidents. This suggests that

- broader testing, especially online testing in real production, is vital to detect problems early and
- better training and process management can avoid operational faults and misuse.

Another important aspect of this research is the catalog of telemetry data, which other providers of big data frameworks can use. The metrics in this catalog include job- or vertex-specific metrics (for example, latency metrics and task I/O metrics), performance counters (for example, CPU usage and memory usage), and job or vertex logs (for example, log entries of interesting execution points). The paper also described mitigation strategies that other providers of big data frameworks can use. It was part of the ICSE '15 Software Engineering in Practice track; access it at http://goo.gl/AYcGhR.

"Mining Energy-Aware Commits," by Irineu Moura and his colleagues, examined techniques developers use to save energy, which is especially important for devices with limited battery life, such as mobile devices. Users often factor energy efficiency into their choice of mobile apps. Even though this quality is important to users, little is known about the strategies adopted to minimize energy consumption or their impact on software quality.

The information mined from 371 energy-aware commits from GitHub identified these energy-saving techniques:

- altering the frequency and voltage of the CPU and peripherals such as Wi-Fi,
- using power-efficient libraries,
- disabling features,

- fixing energy-related bugs,
- implementing low-power idling, and
- manipulating time-outs to stop computation.

The authors also identified possible negative side effects, including corruption of serial transmission and low responsiveness or performance. Furthermore, the results showed that developers were often unsure whether their energy-aware strategies were effective. There is a need for better-documented energy libraries. This paper was part of the 12th Working Conference on Mining Software Repositories; access it at http://goo.gl/AcRzDq.

"An Empirical Study of Architectural Change in Open-Source Software Systems," by Duc Minh Le and his colleagues, reported on an analysis of several hundred versions of 14 open source Apache systems. The authors' key findings include these:

- A semantic (conceptual) view reveals notably different aspects of system evolution than the corresponding structural view.
- Architectural changes can occur inside components even when the overall architecture remains stable.
- The package structure provides only a limited indication of the system architecture.
- Dramatic architectural changes tend to occur both between the end of one major version and the beginning of the next and across one or more minor versions.

These findings provide insight into how architectures change over time. This paper was part of the 12th Working Conference on Mining Software Repositories; access it at http://goo.gl/YM4cPT.

"Supporting Physicians by RE4S: Evaluating Requirements Engineering for Sustainability in the Medical Domain," by Birgit Penzenstadler and her colleagues, presented Requirements Engineering for Sustainability (RE4S). RE4S is a method that uses checklists and reference models to guide software engineers in including sustainability throughout requirements engineering, from identifying stakeholders, to analyzing the domain, to defining a usage model, and finally to specific requirements. To help software engineers identify sustainability concerns, the checklist starts with four questions about the system's purpose, impact, stakeholders, and goals and constraints. RE4S also provides reference models for sustainability goals and stakeholders.

The paper included a case study of Cognatio, a system that supports communication between patients and physicians. Patients can track prescribed medications and observed symptoms; physicians can send reminders and review patient data. Using RE4S to consider sustainability during requirements engineering improved the software's social aspects (for example, interaction between user groups analyzed from different perspectives) and environmental aspects (for example, avoiding overprescription and misaligned prescriptions). So, most of the developed artifacts reflected sustainability concerns that would have been missed otherwise. This example illustrates how using RE4S can help developers systematically integrate sustainability goals and requirements with other requirements, and refine them into software-specific constraints considered during design and implementation.

An online version of RE4S is at http://birgit.penzenstadler.de/se4s. This paper was part of the 4th International Workshop on Green and Sustainable Software; access it at http://goo.gl/RuyBJ0.

Feedback or suggestions are welcome. In addition, if you try or adopt any of the practices included in the column, please send the paper authors and Jeffrey Carver (carver@cs.ua.edu) a note about your experiences.

JEFFREY C. CARVER is an associate professor in the University of Alabama's Department of Computer Science. Contact him at carver@cs.ua.edu.

AIKO YAMASHITA is a data analyst and entrepreneur at Yamashita Research and is an adjunct associate professor at Oslo and Akershus University College of Applied Sciences. Contact her at aiko.fallas@gmail.com.

LEANDRO MINKU is a lecturer (assistant professor) in the University of Leicester's Department of Computer Science. Contact him at l.l.minku@cs.bham.ac.uk.

MAYY HABAYEB is a master's student in Ryerson University's Data Science Laboratory. Contact her at mayy.habayeb@ryerson.ca.

SEDEF AKINLI KOCAK is PhD candidate and research assistant in Ryerson University's Data Science Laboratory. Contact her at sedef.akinlikocak@ryerson.ca.