GitHub, Technical Debt, Code Formatting, and More

Jeffrey C. Carver, Jordi Cabot, Rafael Capilla, and Henry Muccini

**THIS ISSUE’S COLUMN** reports on papers from the 19th International Conference on Model Driven Engineering Languages and Systems, the 2016 ACM SIGPLAN International Conference on Software Language Engineering, the 12th International ACM SIGSOFT Conference on the Quality of Software Architectures, and the 13th Working IEEE/IFIP Conference on Software Architecture. Feedback and suggestions are welcome. In addition, if you try or adopt any of the practices included in the column, please send Jeffrey Carver and the paper authors a note about your experiences.

**GitHub and Open Source**

“The Quest for Open Source Projects That Use UML: Mining GitHub,” by Regina Hebig and her colleagues, reports on how frequently open source projects use UML. Although several empirical studies have investigated the use of UML in industrial projects, little is known about UML’s penetration into open source.

Hebig and her colleagues systematically mined 10 percent of all GitHub projects to collect UML diagrams stored as .xmi or .uml files (file extensions that UML tools typically use). They also used their image classifier to identify UML diagrams (in image formats such as .bmp and .jpeg). Their search discovered 3,295 open source projects, which included 21,316 UML models. Most of these projects had very few UML files, with two-thirds including just one file.

For those projects that used UML, the authors investigated when the UML models were created and how often they were updated. In most projects, the UML files were uploaded during a project’s early phases and rarely updated. The authors plan to classify the maturity of the modeling practices used in these projects. You can access this paper at goo.gl/52LebA.

The dataset they used is at oss.models-db.com. Since the paper’s publication, the dataset has been extended to include links to more than 93,000 UML files spread across 24,000 GitHub repositories. This dataset will enable further studies of UML’s relationship and importance to open source.

**Technical Debt in Model-Driven Engineering**

“Technical Debt in MDE: A Case Study on GMF/EMF-Based Projects,” by Xiao He and his colleagues, explores to what extent MDE (model-driven engineering) software fares better than non-MDE software in terms of maintainability, especially regarding technical debt. Technical debt is the cost of refactoring software...
artifacts that don’t satisfy the desired quality requirements, plus the interest incurred to maintain those artifacts. In MDE projects, most code is typically generated from some higher-level software models. A key (supposed) benefit of generated code is that its internal quality is higher than that of the handwritten code in non-MDE projects.

Xiao He and his colleagues analyzed 16 nontrivial MDE projects to identify bad smells (patterns in the code denoting violations of fundamental design principles). These smells often indicate technical debt.

In most cases, code generators incurred more total and average technical debt than developers did. The most frequent bad smells were Duplicate Code, Cyclomatic Complexity, Too Many Methods, Excessive Imports, and God Class. MDE brings interesting benefits to software development, but code with little technical debt doesn’t seem to be one of them, at least with current MDE tools. You can access this paper at goo.gl/5qzunA.

**A Universal Code Formatter**

“Towards a Universal Code Formatter through Machine Learning,” by Terence Parr and Jurgen Vinju, proposes a novel approach to build a code formatter specific to a given organization, no matter the preferred language or coding convention. Many code formatters are available, but configuring their rules to suit an organization’s needs is cumbersome and might require a language expert. Instead, Parr and Vinju introduce CodeBuff, a tool that uses machine learning to abstract formatting rules from a representative corpus of sample programs that follow the target coding style. That is, the formatter is configured by example.

CodeBuff operates in two phases. The training phase builds a statistical model of the formatting rules from the corpus. The formatting phase applies those rules to format other documents using the same style.

The authors have tested CodeBuff with a variety of grammars including Java, SQL, and ANTLR (Another Tool for Language Recognition). Training requires approximately 10 files. The generated formatters are fast and accurate, and the results are generalizable to other languages. You can access this paper at goo.gl/jkjo7S.

**Assuring Architectural Quality**

“AQAF: An Architecture Quality Assurance Framework for Systems Modeled in AADL,” by Andreas Johnsen and his colleagues, describes a formal, automated solution to verification of architecture engineering of critical embedded systems modeled in the Architecture Analysis and Design Language (AADL). In such systems, it’s important to perform verification during architecture engineering to ensure dependability and system safety. Johnsen and his colleagues discuss a framework of techniques that use a common formalism, flow graphs, and timed automata to support automated, traceable verification.

An analysis of AQAF’s effectiveness and efficiency on Bombardier Transportation’s safety-critical train control system indicated that AQAF could effectively and efficiently detect and report on faults that were injected during architecture design, implementation, and maintenance. AQAF detected each injected fault and reduced the time and memory consumption of regression verification by approximately 25 percent compared with a rerun-all approach.

You can access this paper at goo.gl/t6uc6q.

**Continuous Architecting**

“A Multiple Case Study of Continuous Architecting in Large Agile Companies: Current Gaps and the CAFFEA Framework,” by Antonio Martini and Jan Bosch, introduces CAFFEA (Continuous Architecture Framework for Embedded and Agile Software Development), which helps agile organizations use software architecture appropriately to ensure long-term system qualities.

Martini and Bosch developed the framework on the basis of an investigation of three large embedded-software companies. They identified architectural gaps that occurred when the companies used agile development, including risk management, change or decision management, documenting architectures, and monitoring.

Mapping these gaps to roles indicated which roles suffered from a lack of specific architectural practices. CAFFEA describes the results of such analysis specific to the agile environment, including architectural roles, virtual teams, and architectural practices.

The authors first evaluated CAFFEA in a workshop with five large embedded-software companies, including 40 participants. This workshop resulted in the use of CAFFEA in several companies. An empirical evaluation of CAFFEA after one year of use in one of these companies found that it improved risk management, decision making, communication across teams, and the ability to manage architectural technical debt.

A CAFFEA user at Grundfos has used it to create a “development runway” around architectures, processes, and tools. The runway supports the
development organization by defining a common way of working and means to continually improve it. CAFFEA is a major enabler for bridging the gap between development units divided by geography, culture, and scale. Another user from Axis Communications stated, “The CAFFEA model has given us a framework to organize both domain-specific as well as feature teams, all driven in an agile way … [including] maintenance of code … and how to communicate on risks and architectural problems. The framework … has also helped us look into details like different kinds of technical debt. … We now know that we don’t have any huge technical-debt risks.”

You can access this paper at goo.gl/olLq90.

References

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

JORDI CABOT is an ICREA (Catalan Institution for Research and Advanced Studies) research professor at the Interdisciplinary Internet Institute. Contact him at jordi.cabot@icrea.cat.

RAFAEL CAPILLA is an associate professor of computer science at Rey Juan Carlos University of Madrid. Contact him at rafael.capilla@urjc.es.

HENRY MUCCINI is an associate professor of software engineering in the University of L’Aquila’s Information Engineering and Computer Science and Mathematics Department. Contact him at henry.muccini@univaq.it.

2017 B. Ramakrishna Rau Award
Call for Nominations

Honoring contributions to the computer microarchitecture field

New Deadline: 1 May 2017

Established in memory of Dr. B. (Bob) Ramakrishna Rau, the award recognizes his distinguished career in promoting and expanding the use of innovative computer microarchitecture techniques, including his innovation in compiler technology, his leadership in academic and industrial computer architecture, and his extremely high personal and ethical standards.

WHO IS ELIGIBLE? The candidate will have made an outstanding innovative contribution or contributions to microarchitecture and use of novel microarchitectural techniques or compiler/architecture interfacing. It is hoped, but not required, that the winner will have also contributed to the computer microarchitecture community through teaching, mentoring, or community service.

AWARD: Certificate and a $2,000 honorarium

PRESENTATION: Annually presented at the ACM/IEEE International Symposium on Microarchitecture

NOMINATION SUBMISSION: This award requires 3 endorsements. Nominations are being accepted electronically: www.computer.org/web/awards/rau.

CONTACT US: Send any award-related questions to awards@computer.org.

www.computer.org/web/awards/rau