Product Lines, Energy Conservation, Use Cases, Agile Development, and Infotainment

THIS ISSUE’S COLUMN reports on papers (and a talk) from the 19th International Conference on Software Product Lines, the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, the 23rd IEEE International Requirements Engineering Conference, and the 9th European Conference on Software Architecture. Feedback or suggestions are welcome. In addition, if you try or adopt any of the practices mentioned in the column, please send Jeffrey Carver and the paper authors a note about your experiences.

Modeling Product Lines
“Modeling Aerospace Systems Product Lines in SysML,” by Jesús Gaeta and Krzysztof Czarnecki, presents a method and pattern catalog for modeling avionics product lines in SysML (Systems Modeling Language).1 The method divides a product line model into a family model and variant models. The family model represents

- the parts of the system common to all family members and
- the parts of the system that can vary from product to product.

It combines features, text requirements, and structural and behavioral aspects. A variant model defines a specific system variant’s architecture. It contains the variable parts of the family model and introduces parts unique to that variant.

To evaluate the method, Gaeta and Czarnecki performed a case study that created a configurable, reusable system model covering parts of a fuel control unit used at the subject company. The developers found the feature model (which contained 20 common and 23 variable features) useful because it covered the relevant aspects of variability. With this method, the researchers specified more than 60 requirements, including mandatory, optional, and variant-specific ones. You can access this paper at http://goo.gl/zlU80K.

Reconciling Energy Conservation and Aesthetics
“Optimizing Energy Consumption of GUIs in Android Apps: A Multi-objective Approach,” by Mario Linares-Vásquez and his colleagues, describes an approach to automatically choose visually appealing yet energy-conscious color combinations for Android app GUIs.2 Because energy consumption is a key feature users consider when choosing a mobile app, reducing energy consumption is important for producing a competitive app.

Existing approaches to reduce energy consumption usually try to control how mobile apps access energy-greedy components, including the screen. The energy consumed by the popular organic light-emitting diode (OLED) screens depends on the displayed colors. However, manually finding color combinations that balance energy consumption and visual appeal is difficult. This difficulty arises from

- the requirement of staff with expertise in color design and OLED screens,
- the high number of possible color combinations, and
• trade-offs between visual appeal and energy efficiency.

To overcome this difficulty, Linares-Vásquez and his colleagues use a multiobjective evolutionary algorithm combined with OLED power models, pixel-based engineering, color theory, and dynamic analysis. Using a developer’s original design, this approach automatically provides the developer with a set of color alternatives, each of which differently balances energy consumption, contrast, and similarity to the original.

Evaluation of the approach on 25 Android apps demonstrated significant improvements in energy consumption and contrast, while producing designs similar to the original. A survey of 85 app users showed that the approach successfully generated energy-saving, visually appealing color alternatives. You can access this paper at http://goo.gl/CjdTYn.

How Use Cases Change
“Understanding Changes in Use Cases: A Case Study,” by Mohammad Basirati and his colleagues, argues that we need a deeper understanding of which changes in requirements artifacts, such as use cases, are problematic in terms of difficulty or risk. Basirati and his colleagues manually inspected more than 400 changes in 32 use cases over 15 months in a software project at Munich Re, a reinsurance company. This research provided empirical evidence about specific maintenance risks and suggests the need to continuously analyze local and temporal dispersion.

The results answer three important questions. First, which use cases changed and where? The most frequently changing use cases were badly structured and strongly depended on each other (that is, they belonged to the same workflow). In addition, alternative flows (variations or extensions of the main flow) were especially prone to changes relative to their size.

Second, which types of changes occurred? Changes in content (semantic changes) and content presentation (syntactic changes) happened with a similar frequency. Over time, phases with a higher proportion of syntactic changes coincided with rather calm periods in the project. That means, when the project was calmer, engineers took the time to perform quality assurance and improve existing use cases. Tracking the proportion of syntactic and semantic changes over time can indicate the effort going into quality assurance of use cases.

Finally, what types of changes were problematic? Problematic changes originated from essential complexity (in the domain) and accidental complexity (in the requirements artifacts themselves). For the latter, dispersed changes were particularly prone to inconsistency, aggravated by inconsistent UI details or improper referencing. Changes in the domain taxonomy were particularly difficult.

These last results suggest the need to monitor dispersed changes and quality defects. They also provide empirical evidence about quality defects leading to poor maintainability, such as inconsistent UI details or improper references. So, practitioners can facilitate evolving essential complexity and identifying accidental complexity (quality defects) by analyzing the changes over time in their requirements documents. You can access this paper at http://goo.gl/E4v1SM.

Combining Architecture with Agile Development
“Distilling Best Practices for Agile Development from Architecture Methodology: Experiences from Industrial Application,” by Dominik Rost and his colleagues, reports the lessons learned from industrial experience to identify software architecture best practices for use in agile methods. Scrum and Extreme Programming agile practices don’t provide engineering-style support for architecture. The authors’ experiences show that integrating software architecture practices and agile development improves product quality significantly and predictably. They also learned that developers should reserve time for planning and architectural work, should check compliance with architectural design decisions, and shouldn’t introduce too many additional meetings (common in many agile approaches). You can access this paper at http://goo.gl/xgrUkL.

Automotive Infotainment
“Connected Infotainment Systems: The Internet of Things in the Car,” an industry talk by Thomas Kropf (Bosch Car Multimedia GmbH), provided an overview of the challenges and solutions for infotainment systems, one of the most complex and challenging software systems in a car. Such software contains over 30 million SLOC and is highly connected within the car and to various cloud services. Also, these systems face challenges related to stringent security demands and functional safety.

Overall, the Internet of Things in the car is an exciting research area with promising impacts on the automotive industry. Connected infotainment systems and services, mobility
solutions for driver assistance, intelligent security, energy management, and connectivity to a variety of sensors all provide smart, adaptive behavior. Market trends and user needs require adequate variability models and high usability and high interoperability of infotainment systems. These requirements have a clear impact on the software production and update cycles and the software architecture. You can access slides of this talk at https://extras.computer.org/extras/mso2016030029s1.pdf.

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.

EDUARDO SANTANA DE ALMEIDA is an assistant professor of software engineering at the Federal University of Bahia. Contact him at esa@dcc.ufba.br.

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

LEANDRO L. MINKU is a lecturer (assistant professor) in the University of Leicester’s Department of Computer Science. Contact him at leandro.minku@leicester.ac.uk.

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.

BIRGIT PENZENSTADLER is an assistant professor of software engineering at California State University, Long Beach. Contact her at birgit.penzenstadler@csulb.edu.