Variability in Software: State of the Art and Future Directions (Extended Abstract)

Krzysztof Czarnecki

Generative Software Development Lab, University of Waterloo, Canada
kczarnec@gsd.uwaterloo.ca

Variability is a fundamental aspect of software. It is the ability to create system variants for different market segments or contexts of use. Variability has been most extensively studied in software product lines [10], but is also relevant in other areas, including software ecosystems [4] and context-aware software [15]. Virtually any successful software faces eventually the need to exist in multiple variants.

Variability introduces essential complexity into all areas of software engineering, and calls for variability-aware methods and tools that can deal with this complexity effectively. Engineering a variable system amounts to engineering a set of systems simultaneously. As a result, requirements, architecture, code and tests are inherently more complex than in single-system engineering. A developer wanting to understand a particular system variant has to identify the relevant parts of requirements and code and ignore those that pertain solely to other variants. A quality assurer verifying a variable system has to ensure that all relevant variants are correct. Variability-aware methods and tools leverage the commonalities among the system variants, while managing the differences effectively.

This talk will analyze how variability affects the software lifecycle, focusing on requirements, architecture and verification and validation, review the state of the art of variability-aware methods and tools, and identify future directions.

Variability modeling at the requirements level helps explaining a variable system and its evolution to all stakeholders, technical and nontechnical ones. A popular technique is feature models [19,11], which are hierarchically composed declarations of common and variable features and their dependencies. At the requirements level, system features represent cohesive clusters of requirements [7,8], providing convenient “mental chunks” to all stakeholders to understand the system, its variability, and its evolution. For example, changes at the code level can be explained as feature additions, removals, merges, or other kinds of modifications at the feature level [27]. As feature models provide only declarations of features, an obvious direction is to enrich them with feature specifications, such as behavioral ones (e.g., [30]). Feature models can also be enhanced with rich forms of dependencies, such as representations of feature interactions (e.g., [24]), and with quality attributes, such as performance (e.g., [2,26]). Feature-oriented software development (FOSD) aims at treating features as first-class citizens,

that is, modularizing them throughout the entire lifecycle [1]. Although promising, empirically quantifying the benefits of feature modularization is future work.

Variation points accommodate variability at the architectural level. They are locations in software artifacts where variability occurs. Variability engineering classifies variation points according to their variability type [29], such as value parameters, optional elements, element substitution or repeated instantiation, and their binding time. The Common Variability Language (CVL) [25], which is an effort by the Object Management group to standardize variability modeling, provides a comprehensive taxonomy of variation points. Many existing design and architectural patterns [16,6] can be used to implement variation points, as they are often about allowing certain system aspects to vary easily. For example, the strategy pattern is used to vary algorithms within a system. Planning and scoping variability can guide and ground architectural design: varying parts need to be localized and hidden behind common interfaces. But variability also impacts established modularity concepts. For example, variable modules need to expose configuration interfaces and their functional interfaces may need to be variable. This combination leads to the notion of variability-aware module systems [21], with implications on compositionality and type consistency. Tools and languages using such module systems also need to support staged configuration of such modules [12]. Further, variability affects other modularity topics, such as cohesion, coupling, crosscutting, and tangling. Understanding the impact of variability on modularity is a topic for future research.

Variability also affects verification and validation. Variability-aware methods leverage commonalities to reduce time and effort to check variable systems. The reduction is useful for product lines with large numbers of products, but is indispensable for ecosystem platforms, whose users may decide to instantiate any variant out of the exponential number of supported variants. Existing work includes variability-aware forms of standard analysis techniques, such as type checking (e.g., [20,13]), dataflow analyses (e.g., [5]) and model checking (e.g., [9]). In testing, variability engineering can be applied to tests in order to derive test suits along with a variant. For example, the variability model and the build system of the eCos operating system allow users to derive a system variant along with suitable tests. Researchers have also proposed the concept of shared execution of tests on multiple variants [23,22] and adapted combinatorial testing to generate sets of variants covering certain feature-combination heuristics for platform testing (e.g., [28]). However, testing variable systems offers plenty opportunities to be explored in future research, including test selection, prioritization, and generation at unit, integration, and system levels, and potentially in the context of specific changes.

Indisputably, variability has become a fundamental aspect of software engineering. Variability engineering is practiced today both in open-source (e.g., see a study of variability engineering in open-software systems software, including the Linux kernel [3]) and industry (e.g., see the experiences of Danfoss [18], GM [14], and Rolls-Royce [17]). Variability engineering also offers abundant opportunities for software researchers to explore new directions in a wide range of areas within

requirements, architecture, verification and validation, and beyond, including reverse engineering, programming languages, performance, reliability, and security. Today's and future application domains pose new challenges to variability engineering to be addressed in research. They include scaling variability to systems of systems in automotive and aerospace engineering and dealing effectively with software ecosystems, cloud computing, and context-aware applications in enterprise and consumer domains.

References

1. Apel, S., Kästner, C.: An overview of feature-oriented software development. *Journal of Object Technology* 8(5), 49–84 (2009)
2. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Reasoning on Feature Models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) *CAiSE 2005*. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
3. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: Variability modeling in the systems software domain. Technical report, gsdlab-tr. Generative Software Development Laboratory, University of Waterloo (July 06, 2012), <http://gsd.uwaterloo.ca/node/465>
4. Bosch, J.: From software product lines to software ecosystems. In: *Proceedings of the 13th International Software Product Line Conference, SPLC 2009*, Carnegie Mellon University, Pittsburgh, PA, USA, pp. 111–119 (2009)
5. Brabrand, C., Ribeiro, M., Tolêdo, T., Borba, P.: Intraprocedural dataflow analysis for software product lines. In: *Proceedings of the 11th Annual International Conference on Aspect-Oriented Software Development, AOSD 2012*, pp. 13–24. ACM, New York (2012)
6. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York (1996)
7. Chen, K., Zhang, W., Zhao, H., Mei, H.: An approach to constructing feature models based on requirements clustering. In: *Proceedings of the 13th IEEE International Conference on Requirements Engineering, RE 2005*, pp. 31–40. IEEE Computer Society, Washington, DC (2005)
8. Classen, A., Heymans, P., Schobbens, P.-Y.: What's in a *Feature*: A Requirements Engineering Perspective. In: Fiadeiro, J.L., Inverardi, P. (eds.) *FASE 2008*. LNCS, vol. 4961, pp. 16–30. Springer, Heidelberg (2008)
9. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011*, pp. 321–330. ACM, New York (2011)
10. Clements, P., Northrop, L.: *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., Inc. (2001)
11. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wasowski, A.: Cool features and tough decisions: a comparison of variability modeling approaches. In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS 2012*, pp. 173–182. ACM, New York (2012)
12. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration Using Feature Models. In: Nord, R.L. (ed.) *SPLC 2004*. LNCS, vol. 3154, pp. 266–283. Springer, Heidelberg (2004)

13. Czarnecki, K., Pietroszek, K.: Verifying feature-based model templates against well-formedness OCL constraints. In: *Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE 2006*, pp. 211–220. ACM, New York (2006)
14. Flores, R., Krueger, C., Clements, P.: Mega-scale product line engineering at general motors. In: *Proceedings of the 16th International Software Product Line Conference, SPLC 2012*, vol. 1, pp. 259–268. ACM, New York (2012)
15. Fortier, A., Rossi, G., Gordillo, S.E., Challiol, C.: Dealing with variability in context-aware mobile software. *J. Syst. Softw.* 83(6), 915–936 (2010)
16. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
17. Habli, I., Kelly, T.: Challenges of establishing a software product line for an aerospace engine monitoring system. In: *Proceedings of the 11th International Software Product Line Conference, SPLC 2007*, pp. 193–202. IEEE Computer Society, Washington, DC (2007)
18. Jepsen, H.P., Beuche, D.: Running a software product line: standing still is going backwards. In: *Proceedings of the 13th International Software Product Line Conference, SPLC 2009*, Carnegie Mellon University, Pittsburgh, PA, USA, pp. 101–110 (2009)
19. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, CMU (1990)
20. Kästner, C., Apel, S., Thüm, T., Saake, G.: Type checking annotation-based product lines. *ACM Trans. Softw. Eng. Methodol.* 21(3), 14 (2012)
21. Kästner, C., Ostermann, K., Erdweg, S.: A variability-aware module system. In: *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA 2012*, pp. 773–792. ACM, New York (2012)
22. Kästner, C., von Rhein, A., Erdweg, S., Pusch, J., Apel, S., Rendel, T., Ostermann, K.: Toward variability-aware testing. In: *Proceedings of the 4th International Workshop on Feature-Oriented Software Development, FOSD 2012*, pp. 1–8. ACM, New York (2012)
23. Kim, C.H.P., Khurshid, S., Batory, D.: Shared execution for efficiently testing product lines. In: *IEEE International Symposium on Software Reliability Engineering (ISSRE 2012)*. IEEE (2012)
24. Lee, K., Kang, K.C.: Feature Dependency Analysis for Product Line Component Design. In: Bosch, J., Krueger, C. (eds.) *ICSR 2004*. LNCS, vol. 3107, pp. 69–85. Springer, Heidelberg (2004)
25. Object Management Group: Common variability language (CVL). OMG Revised Submission, Document (August 05, 2012), www.omgwiki.org/variability
26. Olaechea, R., Stewart, S., Czarnecki, K., Rayside, D.: Modeling and multi-objective optimization of quality attributes in variability-rich software. In: *International Workshop on Non-Functional System Properties in Domain Specific Modeling Languages (NFPinDSML 2012)*, Innsbruck, Austria (2012)
27. Passos, L., Czarnecki, K., Sven, A., Wąsowski, A., Kästner, C., Guo, J., Hunsen, C.: Feature-oriented software evolution. In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS 2013*. ACM, New York (2013)

28. Perrouin, G., Sen, S., Klein, J., Baudry, B., Traon, Y.L.: Automated and scalable t-wise test case generation strategies for software product lines. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation (ICST 2010), pp. 459–468. IEEE Computer Society, Washington, DC (2010)
29. Schmid, K., John, I.: A Customizable Approach to Full-Life Cycle Variability Management. *Science of Computer Programming* 53(3), 259–284 (2004)
30. Shaker, P., Atlee, J.M., Wang, S.: A feature-oriented requirements modelling language. In: Proceedings of the 20th IEEE International Requirements Engineering Conference (RE 2012). ACM (2012)