Extractive Multi Product-Line Engineering

Kamil Rosiak

Technische Universität Braunschweig

Braunschweig, Germany

k.rosiak@tu-bs.de

Abstract—Cloning is a general approach to create new functionality within variants as well as new system variants. It is a fast, flexible, intuitive, and economical approach to evolve systems in the short run. However, in the long run, the maintenance effort increases. A common solution to this problem is the extraction of a product line from a set of cloned variants. This process requires a detailed analysis of variants to extract variability information. However, clones within a variant are usually not considered in the process, but are also a cause for unsustainable software. This thesis proposes an extractive multi product-line engineering approach to re-establish the sustainable development of software variants. We propose an approach to re-engineer intra-system and inter-system clones into reusable, configurable components stored in an integrated platform and synthesize a matching multi-layer feature model.

Index Terms—clone detection, variability mining, refactoring, multi product-line

I. Introduction

With an increasing demand for custom-tailored software systems, ad-hoc reuse strategies are often used [1]. On the one hand, system functionality is reused within a variant, known as intra-system cloning [2]. On the other hand, entire variants are copied and adapted for new requirements, also known as clone-and-own or inter-system cloning [3]–[5]. While code cloning comes with practical benefits [6]–[8], research has shown that, with a growing number of clones, they become a significant source of faults and cause problems during development, evolution, and maintenance, unless special care is taken to identify and track existing code clones and their evolution [9]–[13]. Development and maintenance costs increase with the number of variants [14]. The extraction of a product line is a common approach to solve this problem and a relevant industrial case [3], [15], [16]. The state of the art of extractive software product-line adoption and reverse-engineering deals with clones at the granularity of entire system variants [17]–[22].

However, we argue that clones can appear at any level of granularity. At the coarsest level of entire variants, a set of cloned variants can be integrated into a software product line. Clones of finer granularity within variants can also be extracted into configurable components. Thus, they can be considered as sub-product lines, which can again have further sub-product lines of their own at even finer granularity levels. Generalizing this concept, the detection and integration of clones into configurable components at any granularity level can be considered as the extraction of a multi product-line [23].

This thesis aims to develop an extractive multi product-line engineering approach to consolidate intra- and inter-system clones to reestablish sustainable development and maintenance of variants. We propose an automatic refactoring of system variants into an integrated platform, configured with a multi-layer feature model to derive system variants and include configurable components.

Extractive multi product-line engineering enables systems to be re-engineered and transitioned from ad-hoc reuse with clone-and-own to multi product-line engineering with an integrated platform [24].

II. Proposed Approach

We propose a technique for extracting multi product-lines from implementation artifacts containing clones at different levels of granularity. This approach is applicable to implementation artifacts of different types, such as Java [25] or IEC 61131-3 [26]. Figure 1 shows an overview of the proposed multi-product line extraction approach using Java as artifact type, which we explain in detail from left to right. First, artifacts are parsed into a generic graph structure ①. Artifact adapters translate the respective types of artifacts into the graph structure. At least one existing variant is required to build the integrated platform ②. An n-way clone detection approach is applied to the graph structure to identify intra-clones. Cloned artifacts are identified and refactored into configurable components automatically. Components are integrated variant-wise into an integrated platform. We use a pair-wise, iterative, order-invariant variability-mining technique to detect commonalities and differences between already stored artifacts and new variants to be integrated. The integrated platform can be refactored and annotated with feature information. Based on the variability and feature information stored in the integrated platform, we generate a fully constrained multi-layer feature model, representing the software product-line of the input system ③. The integrated platform allows deriving system variants and components ④. They can be used to evolve system variants as well as to fix bugs across all variants. This allows sustainable reuse of code within and between variants. Evolved components and system variants are integrated in the same way as described above ⑤.

III. Planned Contributions

The specific contributions of this work are the following:

Extractive Multi Product-Line Engineering: Krueger [27], Almeida [28] as well as Frankes and Kang [29] provide an overview of software reuse and strategies for the adoption of software. However, traditional extractive product-line engineering and clone detection approaches only focus on either
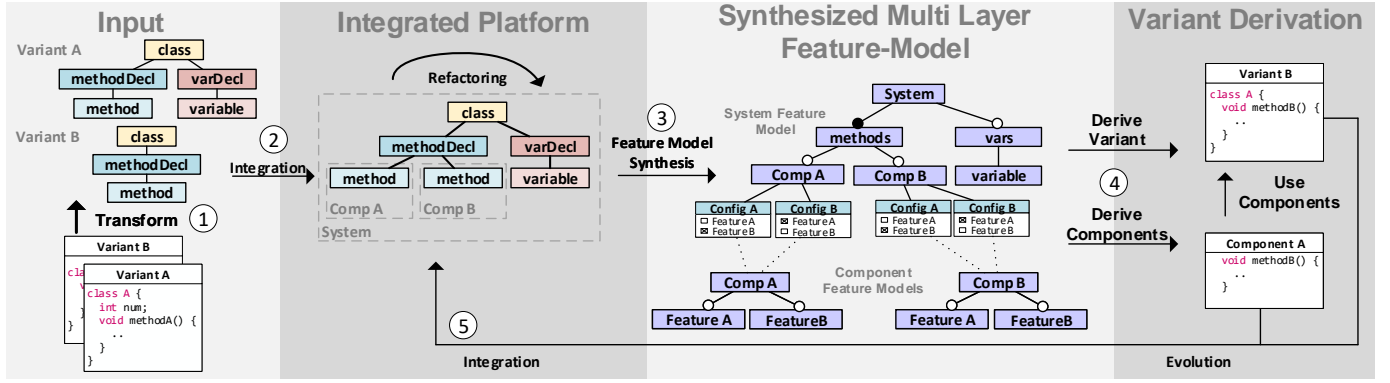
Fig. 1. Overview of the extractive multi-product line engineering approach.

inter-system or intra-system clones. To bridge the gap and reestablish sustainable development, we propose a conceptual process to re-engineer every granularity level of cloned artifacts into a multi product-line.

Clone Detection for Any Clone Granularity: Research has shown approaches for the detection of clones for different types of artifacts, e.g, trees [30], [31], text [32], [33] and tokens [34], [35], but also hybrid approachs that use a mix of those artifacts [36]. Bellon et al. [37] showed an approach on abstract syntax trees that can detect exact and near-miss clones. Göde et al. [38] developed an incremental clone detection approach based on suffix trees that can analyze software revisions. However, independent of the technique applied to the subject system to identify clones, most research focuses either on the detection of intra-system or inter-system clones [39]. We propose a clone detection approach, which allows detecting clones at any granularity level. Moreover, we show an automatic refactoring approach, which allows migrating clones into configurable components.

Merging Approach: After differencing a set of artifacts, they are post-processed, and an output is generated. Rubin et al. [40], [41] proposed an n-way model merge approach to merge multiple input models into one. Other research used an iterative pair-wise merge for block-based languages [42], [43]. However, existing n-way merging approaches require a comparison of all variants at once, which means that the matching must be recomputed for every new variant. State of the art pair-wise merging with iteration is not order-invariant. In order to achieve ideal results, the correct order between variants needs to be known. First, we contribute a taxonomy mining approach to determine the order in which variants should be merged to improve the iterative pair-wise merge approach's quality. Second, we propose an iterative pair-wise order-invariant model merge approach for cases where not all variants are available upfront.

Feature Model Synthesis: Extracting variability information from variants is a commonly used approach to generate a software product line. Schlie et.al. [44] proposed incremental feature model synthesis for clone-and-own software systems in models. Their synthesis only considers feature model elements without cross-tree constraints. We generalize this approach

based on an annotation language to allow feature model synthesis on any artifact type and extend it to consider cross-tree constraints.

IV. EVALUATION PLAN

For the evaluation, we derived the following main research questions:

> **RQ1:** Is our approach for re-engineering of clones of any granularity into a multi product-line *correct*, *scalable*, and *useful*?
>
> **RQ2:** Is our iterative pair-wise order-invariant model merge approach *correct* and *scalable*?
>
> **RQ3:** Is our synthesized constrained feature model *correct* and *useful*?

Lack of case studies with existing ground truth often limits the evaluation of clone and variability analysis approaches. To close the gap, we plan to develop a seed-based synthetic clone data set generator that simulates cloning on varying granularity levels and tracks the ground truth. This technique will allow a precise analysis of precision and recall of our clone detection approach. This generator is based on the automatic mutation framework for evaluating code clone detection tools from Roy et al. [45] and extends it with cloning of variants. We plan to evaluate our approach based on synthetically created scenarios, which allows measuring precision and recall of our clone detection approach for arbitrary granularity level. Those are key measures to show the correctness of clone detection approaches [45]. In a second step, we aim to establish software product-lines from known variant-rich case studies such as the pick and place unit (PPU) [46] or ArgoUML [47]. For evaluation of the multi product-line extraction, we derive as a base-line all integrated variants and compare if all variants could be derived correctly and the number of required artifacts. To determine the performance and scalability of our approach, we measure run-time and memory consumption. To determine our feature model synthesis usability, we measure how many operations need to be applied manually and how usable our approach is. Moreover, we plan a user study with practitioners from local industry to determine our approach's usefulness in real-world scenarios.

REFERENCES

[1] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics." in *icsm*, vol. 96, 1996.

[2] R. Koschke, "Large-scale inter-system clone detection using suffix trees," in *IEEE 2012 16th European Conference on Software Maintenance and Reengineering*.

[3] K. Schmid and M. Verlage, "The economic impact of product line adoption and evolution," *IEEE software 2002*, vol. 19, no. 4.

[4] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Enhancing clone-and-own with systematic reuse for developing software variants," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014.

[5] R. Lapeña, M. Ballarin, and C. Cetina, "Towards clone-and-own support: locating relevant methods in legacy products," in *20th International Systems and Software Product Line Conference*, 2016.

[6] C. J. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful: patterns of cloning in software," *Empirical Software Engineering*, vol. 13.

[7] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," in *IEEE 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, 2007.

[8] J. Krinke, "A study of consistent and inconsistent changes to code clones," in *14th working conference on reverse engineering (WCRE 2007)*. IEEE, 2007.

[9] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *Queen's School of Computing TR*, vol. 541, no. 115, 2007.

[10] R. Koschke, "Survey of research on software clones," in *Dagstuhl Seminar*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[11] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *2009 IEEE 31st International Conference on Software Engineering*.

[12] N. Gode and J. Harder, "Clone stability," in *2011 15th European Conference on Software Maintenance and Reengineering*. IEEE, 2011.

[13] K. Hotta, Y. Sano, Y. Higo, and S. Kusumoto, "Is duplicate code more frequently modified than non-duplicate code in software evolution? an empirical study on open source software," in *ERCIM Workshop on Software Evolution (EVOL)*, 2010.

[14] J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik, "Managing forked product variants," in *16th International Software Product Line Conference-Volume 1*, 2012.

[15] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski, "A survey of variability modeling in industrial practice," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, 2013.

[16] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 25–34.

[17] C. Krueger, "Eliminating the adoption barrier," *IEEE Software*, vol. 19, no. 4, 2002.

[18] J. Krüger, W. Mahmood, and T. Berger, "Promote-pl: a round-trip engineering process model for adopting and evolving product lines," in *24th ACM Conference on Systems and Software Product Line: Volume A-Volume A 2020*.

[19] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Bottom-up technologies for reuse: automated extractive adoption of software product lines," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017.

[20] J. Martinez, W. K. Assunção, and T. Ziadi, "ESPLA: a catalog of extractive SPL adoption case studies," in *21st International Systems and Software Product Line Conference-Volume B*, 2017.

[21] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "The ECCO tool: Extraction and composition for clone-and-own," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015.

[22] D. Wille, T. Runge, C. Seidl, and S. Schulze, "Extractive software product line engineering using model-based delta module generation," in *Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, 2017.

[23] G. Holl, P. Grünbacher, and R. Rabiser, "A systematic review and an expert survey on capabilities supporting multi product lines," *Information and Software Technology*, vol. 54, no. 8, 2012.

[24] M. Antkiewicz, W. Ji, T. Berger, K. Czarnecki, T. Schmorleiz, R. Lämmel, S. Stănciulescu, A. Wąsowski, and I. Schaefer, "Flexible product line engineering with a virtual platform," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014.

[25] K. Arnold, J. Gosling, D. Holmes, and D. Holmes, *The Java programming language*. Addison-wesley Reading, 2000, vol. 2.

[26] M. Tiegelkamp and K.-H. John, *IEC 61131-3: Programming industrial automation systems*. Springer, 1995, vol. 14.

[27] C. W. Krueger, "Software reuse," *ACM Computing Surveys (CSUR)*, vol. 24, no. 2, 1992.

[28] E. S. De Almeida, A. Alvaro, D. Lucrédio, V. C. Garcia, and S. R. de Lemos Meira, "A survey on software reuse processes," in *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005*.

[29] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE transactions on Software Engineering*, vol. 31, no. 7, 2005.

[30] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *IEEE 29th International Conference on Software Engineering (ICSE'07)*.

[31] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," in *IEEEE 2006 13th Working Conference on Reverse Engineering*.

[32] C. K. Roy and J. R. Cordy, "NiCad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *2008 16th iEEE international conference on program comprehension*.

[33] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*.

[34] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, 2002.

[35] H. A. Basit and S. Jarzabek, "Efficient token based clone detection with flexible tokenization," in *6th joint meeting of the European software engineering conference*, 2007.

[36] H. Kim, Y. Jung, S. Kim, and K. Yi, "MeCC: memory comparison-based clone detector," in *33rd International Conference on Software Engineering*, 2011.

[37] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *IEEE International Conference on Software Maintenance (Cat. No. 98CB36272)*, 1998.

[38] N. Göde and R. Koschke, "Incremental clone detection," in *IEEE 2009 13th european conference on software maintenance and reengineering*.

[39] J. Svajlenko and C. K. Roy, "Evaluating clone detection tools with BigCloneBench," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015.

[40] J. Rubin and M. Chechik, "N-way model merging," in *2013 9th Joint Meeting on Foundations of Software Engineering*, 2013.

[41] J. Rubin, K. Czarnecki, and M. Chechik, "Managing cloned variants: a framework and experience," in *Proceedings of the 17th International Software Product Line Conference*, 2013, pp. 101–110.

[42] D. Wille, "Managing lots of models: the famine approach," in *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.

[43] D. Wille, S. Schulze, C. Seidl, and I. Schaefer, "Custom-tailored variability mining for block-based languages," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016.

[44] A. Schlie, A. Knüppel, C. Seidl, and I. Schaefer, "Incremental feature model synthesis for clone-and-own software systems in MATLAB/Simulink," in *24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, 2020.

[45] C. K. Roy and J. R. Cordy, "A mutation/injection-based automatic framework for evaluating code clone detection tools," in *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2009.

[46] B. Vogel-Heuser, C. Legat, J. Folmer, and S. Feldmann, "Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit," Institute of Automation and Information Systems, Technische Universität München, Tech. Rep., 2014.

[47] M. Zhang, "ArgoUML," *Journal of Computing Sciences in Colleges*, vol. 21, no. 5, 2006.