ulm university universität
uulm

Empirical Evaluation of Feature Trace Recording on the Edit History of Marlin

Bachelor's Thesis Proposal

Sören Viegener 02.02.2021

Supervisor: M.Sc. Paul Maximilian Bittner

Reviewer: Prof. Dr. Thomas Thüm

1 Introduction

When working on software projects with a large amount of variability, software product lines are the software engineering method of choice. When developing different variants of software that share common artefacts, software product lines yield many advantages, such as enhanced quality of code, reduced development costs, and shorter time to market [1], [2], [3], [4]. Because of the great initial investment needed for software product lines and because for some projects not all requirements are known at the beginning of development, ad-hoc solutions, such as clone-and-own, are commonly used in practice [5]. Clone-and-own is a term used to describe the development of different variants (i.e., clones) of software in parallel. When a new requirement creates the need for a new variant, an existing variant is cloned and modified. Clone-and-own is simple but effective when there are only a few variants, but becomes cumbersome, when the amount of variants increases [5], [6], [7].

For clone-and-own software, there is often a point at which support for new variants or the maintenance of existing clones is not feasible anymore. Thus, the migration of software projects to software product lines has been a focus of research [8], [9], [10], [11], [12]. Between full software product lines and clone-and-own projects lies a large spectrum of different software engineering methods [6]. One intermediate state, which is proposed and discussed in current research [13], [14], [15], is a clone-and-own project with a targeted synchronization between clones. Differences and commonalities between clones are assessed in terms of features, known from software product-line engineering. Changes to one clone are transferred to other clones containing the same feature that was modified. The goal of this method is to avoid duplicate work because bug fixes or new requirements concerning more than one variant can be synchronized. Before any synchronization can take place, knowledge about the feature implementations (i.e., feature traces) is needed.

Feature trace recording is a new method of tracking the implementations of features during the development of a software project [16]. When writing code for a specific variant, the developer specifies the features which they are currently working on as a propositional formula called the feature context. When editing source code, feature traces are recorded for the edited artefacts depending on the feature context and the type of the per-

formed edit (i.e., insertion, deletion, move, or update). The feature context is explicitly allowed to be unspecified (i.e., set to a null value) which accounts for existing or new code without knowledge of the implemented features.

As feature trace recording is in the early stages of research, there have not yet been any empirical studies evaluating its usage. To test its usability in real software, empirical data is needed. Throughout research, this is usually done by analyzing an existing software product line because such projects already contain information about feature implementations [17], [18].

The goal of this thesis is the empirical evaluation of feature trace recording by using data gathered from the software product line *Marlin* [19]. Building a tool to reverse-engineer the history of this software product-line repository will be the main subject. The idea is to extract line-based edit patterns from commits using pattern matching. This information can then be used to detect which feature contexts would have been necessary to record the feature traces present in the product line after the commit. The tool can then be used to empirically evaluate feature trace recording. This can range from an evaluation of how feature trace recording is able to handle specific edits in practice, to more general information about edits performed on software product lines. Information such as the complexity of feature contexts or the edit patterns used may be of interest. The existing theoretical analysis of feature trace recording [16] will thus be expanded by an empirical evaluation of the subject.

2 Related Work

Software product lines have been a part of research for many years [3], [20], [21]. Various studies found many positive aspects such as an increased development efficiency, a reduced time-to-market, and higher profit margins [1], [2], [3], [4]. An empirical analysis by Krüger and Berger [22] and case studies by Rubin et al. [7] and Ardis et al. [23] have confirmed some and sometimes all of these positive aspects in practice. As Krüger and Berger also show, there is often not a clear distinction between the usage of software product lines and a clone-and-own approach in practice [22]. Examples such as the open-source software project *Marlin* [19] show how a software product line can be implemented while also using cloning [18].

In research, there have been many contributions on the continuum between clone-and-own and software product lines and especially about the process of migrating to software product lines [6], [8], [9], [10], [11], [12]. There have been multiple approaches ranging from reverse engineering of software product lines [24] to fully functional migration tools [8]. Rubin et al. describe the steps of the migration process and present case studies on three different companies, each at a different stage of the transitioning process [7]. They also introduced a set of operators that can be applied to the scenarios. Another case study on a successful transition is presented by Krueger et al. [25]. A large study covering migrations to software product lines was presented by Laguna and Crespo [11]. Feature trace recording can simplify a transition to a software product line because feature traces can be gradually recorded during development. An intermediate state, where only some feature traces are known, is explicitly supported.

When a full migration is not feasible or not wanted, there are still ways in which a classic clone-and-own soft-

ware project can be improved [6], [10], [26], [27], [28], [29]. A case study by Dubinsky et al. outlines why developers might often rather use cloning in their software projects [5]. They find that developers value the simplicity and flexibility of clone-and-own approaches [5]. Fischer et al. define three informal steps for improving clone-and-own projects [30]. First, *extraction* gets all reusable parts out of existing variants. *Composition* is the process of combining the extracted artefacts. Lastly, *completion* adds the final code segments needed to finalize the software variant.

The step they refer to as *extraction* has been an extensive part of research on the topics *feature location* [31], *clone detection* [32], or *variability mining* [9]. For developers, finding implementations of features can be tedious and require a lot of time [33] so tools to assist or automate this process are of great need. Rubin and Chechik [34] and Dit et al. [31] present an overview of different approaches. Some semi-automated methods only assist developers and still require a lot of user interaction [9], while fully automated methods [12], [14], [35] can often only find a fraction of the features [34]. Another way of addressing the feature traceability problem is to keep track of feature implementations beforehand. Ji et al. presented a way of tracing features using annotations in the source code, known as embedded annotations [13]. They concluded that embedded annotations can possibly save time when recorded during development instead of recovered afterwards. Their approach still requires manual annotation by the developer, which differs from feature trace recording.

There have been several other methods similar to feature trace recording [16]. An approach by Schwägerl and Westfechtel is a tool for filtered model-driven product-line engineering [36]. Here, only a filtered part (i.e., a partial configuration) of the whole product line is edited, whilst changes are propagated to other variants that also implement the modified features. Nguyen et al. present *JSync* which is similar to Feature Trace Recording because it also works with abstract syntax tree-based edits [37]. *JSync* is capable of clone detection and aids in targeted synchronization of edits with other clones of the same software.

While version control systems handle chronological differences between software, variation control systems manage parallel variants of software. Linsbauer et al. present a comparison of many different variation control systems [15]. One of these is ECCO [14], a tool capable of managing feature-oriented clone-and-own projects. Similarly, VTS provides support for feature-oriented projects which use the C preprocessor to define feature implementations [17]. Feature trace recording differs from these variation control systems because it specifically allows code to not be part of any feature [16]. During development, the developer specifies the current feature context and edits are then recorded together with this feature context. Using this information, a mapping of features to their implementations is created, whilst accounting for missing information. As the name hints, feature trace recording can only be used to record feature traces and is not a full variation control system. It could be used in combination with variation control systems.

The edit patterns that will be analyzed in this thesis were presented by Stănciulescu et al. [17]. These line-based edit patterns were already shown to be applicable to edits on an older version of *Marlin* [17]. The edit patterns are for feature-oriented software with feature implementations using the C preprocessor which is quite common [38]. Especially the usage of disciplined annotations which are handled in feature trace recording has been discussed and advantages have been found for real software projects [39], [40], [41].

3 Problem Statement

This thesis aims to provide an extensive empirical evaluation of feature trace recording. The theoretical analysis of feature trace recording [16] on the basis of the edit patterns described by Stănciulescu et al. [17] will be reused and extended if proven incomplete. These edit patterns will be used to reverse-engineer the edits from the history of the software product line *Marlin* [19].

For this evaluation, a tool will be implemented. The tool should be able to take a repository of a software product line as input to analyze its commits. The tool should match edit patterns on each commit. Depending on the commit there may be multiple (different) edit patterns. To verify the tool's correctness, the *Marlin* repository[1] should be used as the dataset, if possible, to see if the results of Stănciulescu et al. can be reproduced. Initially, we expect the edit patterns by Stănciulescu et al. to suffice for the *Marlin* repository. If further repositories are evaluated or during the evaluation of the *Marlin* repository, new edit patterns may be found. For each matched edit pattern, the feature context should be determined. The reverse-engineered edit patterns and feature contexts can then be analyzed in regard to the research questions described in the next section.

4 Research Questions

The thesis aims to answer as many of the following research questions as possible. During the work on this thesis, further questions may emerge and may also be answered.

RQ1: Can the results of the edit pattern analysis on *Marlin* by Stănciulescu et al. be reproduced?
The first goal of the evaluation will consist of finding edit patterns in the commit history of the *Marlin* repository, if possible. The amount of edit patterns can then be compared to the results of Stănciulescu et al. [17] to gain information about the correctness of the tool.

RQ2: Evaluation of Feature Trace Recording
The following questions are of interest and should be answered using the *Marlin* repository.

1. **How many different feature contexts are there per commit?**
   By inspecting how often the feature context usually needs to be switched, usability for developers could be assessed.

2. **How complex would the feature contexts have to be?**
   The complexity could be determined by the amount of literals and/or the amount of negations.

3. **How often is an empty feature context (i.e., null) feasible?**
   In practice, it may be simpler for developers to not have to specify a feature context. As this is explicitly supported by feature trace recording, information about cases, where null is feasible, may be helpful.

4. **How similar is the feature context to the target feature traces?**
   With feature trace recording, the feature context given by the developer should be simpler than the feature

[1] https://github.com/MarlinFirmware/Marlin

traces that are recorded. There may also be differences between the feature context and the target feature traces depending on the edit pattern used.

RQ3: Optional Further Evaluation

In the following, we present optional research questions that may be answered in the thesis.

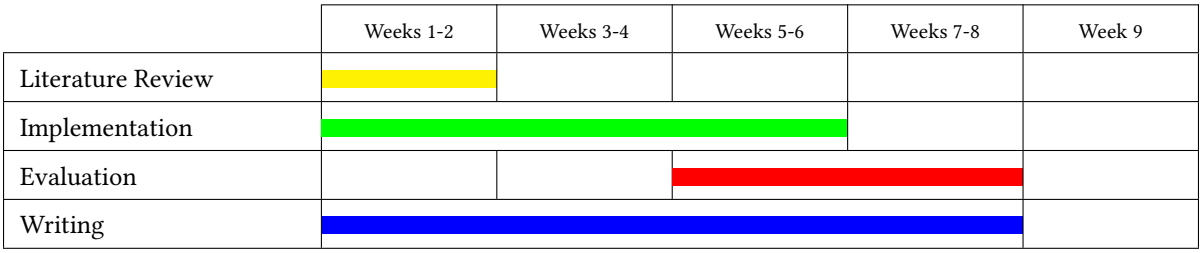
1. **Are there changes in commits that do not fit any edit pattern?**
   In the newer commits of *Marlin*, which were not evaluated by Stănciulescu et al., there might be changes in commits that do not fit any of the given edit patterns.

2. **Are there previously unknown edit patterns?**
   If there are commits with changes that do not fit any pattern, it might be possible to define new edit patterns.

3. **Is the edit pattern correlated to the complexity of the feature context?**
   Some edit patterns might require more complex feature contexts.

5 Time Schedule

This is the planned time schedule for the thesis. The last week is reserved as a time buffer.



6 References

[1] Charles W Krueger. "Introduction to the emerging practice of software product line development". In: *Methods and Tools* 14.3 (2006), pp. 3–15.

[2] Linda Northrop. *Software product lines essentials*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2008.

[3] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[4] David M Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*. Vol. 12. Addison-Wesley Reading, 1999.

[5] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. "An Exploratory Study of Cloning in Industrial Software Product Lines". In: *Proc. Europ. Conf. on Software Maintenance and Reengineering (CSMR)*. IEEE, 2013, pp. 25–34.

[6] Wenbin Antkiewicz Michałand Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. "Flexible Product Line Engineering with a Virtual Platform". In: *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, 2014, pp. 532–535.

[7] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. "Managing Cloned Variants: A Framework and Experience". In: *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 2013, pp. 101–110.

[8] Wolfram Fenske, Jens Meinicke, Sandro Schulze, Steffen Schulze, and Gunter Saake. "Variant-Preserving Refactorings for Migrating Cloned Products to a Product Line". In: *Proc. Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 316–326.

[9] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. "Variability Mining: Consistent Semiautomatic Detection of Product-Line Features". In: *IEEE Trans. on Software Engineering (TSE)* 40.1 (2014), pp. 67–82.

[10] Rainer Koschke, Pierre Frenzel, Andreas P. Breu, and Karsten Angstmann. "Extending the Reflexion Method for Consolidating Software Variants into Product Lines". In: *Software Quality Journal (SQJ)* 17.4 (2009), pp. 331–366.

[11] Miguel A. Laguna and Yania Crespo. "A Systematic Mapping Study on Software Product Line Evolution: From Legacy System Reengineering to Product Line Refactoring". In: *Science of Computer Programming (SCP)* 78.8 (2013), pp. 1010–1034.

[12] David Wille, Sandro Schulze, Christoph Seidl, and Ina Schaefer. "Custom-Tailored Variability Mining for Block-Based Languages". In: *Proc. Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2016, pp. 271–282.

[13] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. "Maintaining Feature Traceability with Embedded Annotations". In: *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 2015, pp. 61–70.

[14] Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. "The ECCO Tool: Extraction and Composition for Clone-and-Own". In: *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 2015, pp. 665–668.

[15] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. "A Classification of Variation Control Systems". In: *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)*. ACM, 2017, pp. 49–62.

[16] Paul Maximilian Bittner, Alexander Schultheiß, Thomas Thüm, Timo Kehrer, Lukas Linsbauer, and Jeffrey Young. "Feature Trace Recording". unpublished paper.

[17] Stefan Stănciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wąsowski. "Concepts, Operations, and Feasibility of a Projection-Based Variation Control System". In: *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 323–333.

[18] Stefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. "Forked and Integrated Variants in an Open-Source Firmware Project". In: *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 151–160.

[19] Erik van der Zalm. *Marlin Firmware*. Accessed at December 02, 2019.

[20] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.

[21] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[22] Jacob Krüger and Thorsten Berger. "An empirical analysis of the costs of clone-and platform-oriented software reuse". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 432–444.

[23] Mark Ardis, Nigel Daley, Daniel Hoffman, Harvey Siy, and David Weiss. "Software product lines: a case study". In: *Software: Practice and Experience* 30.7 (2000), pp. 825–847.

[24] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. "Towards a Language-Independent Approach for Reverse-Engineering of Software Product Lines". In: *Proc. ACM Symposium on Applied Computing (SAC)*. ACM, 2014, pp. 1064–1071.

[25] Charles W Krueger, William A Hetrick, and Joseph G Moore. "Making an Incremental Transition to Software Product Line Practice". In: *Methods & Tools* (2006), pp. 16–27.

[26] Eddy Ghabach, Mireille Blay-Fornarino, Franjieh El Khoury, and Badih Baz. "Clone-and-Own software product derivation based on developer preferences and cost estimation". In: *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE. 2018, pp. 1–6.

[27] Raúl Lapeña, Manuel Ballarin, and Carlos Cetina. "Towards Clone-and-Own Support: Locating Relevant Methods in Legacy Products". In: *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 2016, pp. 194–203.

[28] Lukas Linsbauer, Stefan Fischer, Roberto E. Lopez-Herrejon, and Alexander Egyed. "Using Traceability for Incremental Construction and Evolution of Software Product Portfolios". In: *Proc. Int'l Symposium on Software and Systems Traceability (SST)*. IEEE, 2015, pp. 57–60.

[29] Thomas Schmorleiz and Ralf Lämmel. "Similarity Management of 'Cloned and Owned' Variants". In: *Proc. ACM Symposium on Applied Computing (SAC)*. ACM, 2016, pp. 1466–1471.

[30] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. "Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants". In: *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 391–400.

[31] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. "Feature Location in Source Code: A Taxonomy and Survey". In: *J. Software: Evolution and Process* 25.1 (2013), pp. 53–95.

[32] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh. "Software Clone Detection: A Systematic Review". In: *J. Information and Software Technology (IST)* 55.7 (2013), pp. 1165–1199.

[33] Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. "How Developers Perform Feature Location Tasks: A Human-Centric and Process-Oriented Exploratory Study". In: *J. Software: Evolution and Process* 25.11 (2013), pp. 1193–1224. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1593.

[34] Julia Rubin and Marsha Chechik. "A Survey of Feature Location Techniques". In: *Domain Engineering: Product Lines, Languages, and Conceptual Models*. Ed. by Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin. Springer, 2013, pp. 29–58.

[35] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. "Variability Extraction and Modeling for Product Variants". In: *Software and System Modeling (SoSyM)* 16.4 (2017), pp. 1179–1199.

[36] Felix Schwägerl and Bernhard Westfechtel. "SuperMod: Tool Support for Collaborative Filtered Model-Driven Software Product Line Engineering". In: *Proc. Int'l Conf. on Automated Software Engineering (ASE)*. ACM, 2016, pp. 822–827.

[37] Hoan Anh Nguyen, Tung Thanh Nguyen, Nam H. Pham, Jafar Al-Kofahi, and Tien N. Nguyen. "Clone Management for Evolving Software". In: *IEEE Trans. on Software Engineering (TSE)* 38.5 (2012), pp. 1008–1026.

[38] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. "An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines". In: *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 2010, pp. 105–114.

[39] Jörg Liebig, Christian Kästner, and Sven Apel. "Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code". In: *Proc. Int'l Conf. on Aspect-Oriented Software Development (AOSD)*. ACM, 2011, pp. 191–202.

[40] R. Malaquias, M. Ribeiro, R. Bonifácio, E. Monteiro, F. Medeiros, A. Garcia, and R. Gheyi. "The Discipline of Preprocessor-Based Annotations - Does #ifdef TAG n't #endif Matter". In: *Proc. Int'l Conf. on Program Comprehension (ICPC)*. 2017, pp. 297–307.

[41] Sandro Schulze, Jörg Liebig, Janet Siegmund, and Sven Apel. "Does the Discipline of Preprocessor Annotations Matter?: A Controlled Experiment". In: *SIGPLAN Not.* 49.3 (2013), pp. 65–74.