# Migrating the Android Apo-Games into an Annotation-Based Software Product Line

Jonas Åkesson
Chalmers University of Technology
Gothenburg, Sweden

Sebastian Nilsson
Chalmers University of Technology
Gothenburg, Sweden

Jacob Krüger
Otto-von-Guericke University
Magdeburg, Germany

Thorsten Berger
Chalmers | University of Gothenburg
Gothenburg, Sweden

## ABSTRACT

Most organizations start to reuse software by cloning complete systems and adapting them to new customer requirements. However, with an increasing number of cloned systems, the problems of this approach become severe, due to synchronization efforts. In such cases, organizations often decide to extract a software product line, which promises to reduce development and maintenance costs. While this scenario is common in practice, the research community is still missing knowledge about best practices and needs datasets to evaluate supportive techniques. In this paper, we report our experiences with extracting a preprocessor-based software product line from five cloned Android games of the Apo-Games challenge. Besides the process we employed, we also discuss lessons learned and contribute corresponding artifacts, namely a feature model and source code. The insights into the processes help researchers and practitioners to improve their understanding of extractive software-product-line adoption. Our artifacts can serve as a valuable dataset for evaluations and can be extended in the future to support researchers as a real-world baseline.

## CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software reverse engineering**; *Maintaining software.*

## KEYWORDS

Software product line, Extraction, Case study, Feature model, Antenna, Apo-Games

**ACM Reference Format:**
Jonas Åkesson, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger. 2019. Migrating the Android Apo-Games into an Annotation-Based Software Product Line. In *23rd International Systems and Software Product Line Conference - Volume A (SPLC '19), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3336294.3342362


*SPLC '19, September 9–13, 2019, Paris, France*

ACM ISBN 978-1-4503-7138-4/19/09…$15.00
https://doi.org/10.1145/3336294.3342362

## 1 INTRODUCTION

In practice, organizations often start to develop a family of similar software systems by copying an existing one and adapting it to new requirements. This approach is called clone-and-own [6, 21], operating at the granularity of whole systems—or, in our case, Android apps [4]. While it is a simple and quick approach to software reuse, it also has severe limitations. All changes, such as bug fixes and updates, must be propagated to ensure the correct behavior of all clones, each of which can have individual side effects. Thus, maintaining a large number of clones becomes an error-prone and costly activity [6, 18, 24].

When these problems become too severe, organizations often start to extract [9] a *software product line* from the cloned legacy systems [1, 5]. A software product line promises benefits considering the quality, development effort, and time-to-market, mainly through the reusable features that are implemented in a common platform [8, 23]. Instead of having separated code clones for each system, developers can configure features (i.e., select or deselect them) to define a variant that is automatically created.

Despite the practical importance of migrating from cloned systems into a software product line [3, 7, 24], we still lack detailed insights into best practices, and most automated techniques are of limited applicability [10, 11, 13, 14, 19, 22]. To address these shortcomings, more and more researchers propose to systematically collect real-world case studies and datasets, for instance, in the EPSLA catalog [16]. Besides collecting experiences of researchers and practitioners, openly accessible data does also provide ground-truths to evaluate and benchmark automated techniques [13, 22]. In this spirit, Krüger et al. [12] provide a set of 25 games (20 Java, five Android) that have been implemented by a single developer using the clone-and-own approach. The authors asked the research community to work on five challenges to provide artifacts that can later be used for evaluation purposes. These challenges are concerned with reverse engineering (i.e., feature models, feature locations, architecture recovery), code analysis (i.e., code smells), and extracting an actual software product line.

In this challenge solution, we describe a case study that we conducted to address the last challenge. For this purpose, the first two authors of this paper extracted a software product line from the Android versions of the Apo-Games. As variability mechanism for our platform, we used the Antenna preprocessor. In this paper, we report our analysis and migration methodology, as well as experiences we gained during this process. More precisely, we make the following contributions in this paper:

**Table 1: Systems of the Apo-Games that we migrated.**

| Name | Year | SLOC | Game Type |
|---|---|---|---|
| ApoClock | 2012 | 3,615 | Arcade/puzzles |
| ApoDice | 2012 | 2,523 | Level-based puzzles |
| ApoSnake | 2012 | 2,965 | Snake |
| ApoMono | 2013 | 6,487 | Level-based puzzles |
| MyTreasure | 2013 | 5,360 | Level-based puzzles |

**Table 2: Legacy compared to software-product-line games.**

| | #F | #A | #C | # Files | SLOC |
|---|---|---|---|---|---|
| Legacy | — | — | — | 117 | 20,950 |
| SPL | 61 | 19 | 14 | 43 | 10,278 |
| Reduction | | | | -63.25% | -50.94% |

F: Features; A: Abstract Features; C: Constraints

- We explain how we analyzed and migrated five Android games into an annotation-based software product line.
- We discuss our experiences on problems and open challenges that we faced during this process.
- We contribute a repository,[1] including the feature model and source code of our software product line. To this end, we provide a FeatureIDE [17] project that allows to configure and instantiate variants.

The results provide insights for researchers and practitioners alike to understand migration processes from cloned systems to a software product line. In addition, our artifacts can be used as baselines to evaluate and compare automated techniques. To this end, they can also serve as starting points to extend the artifacts further and integrate them into suitable datasets.

## 2 METHODOLOGY

Before the conduct of our case study, the first two authors of this paper performed a literature survey to familiarize with software-product-line techniques and tools. We remark that we did not employ a specific re-engineering process, such as those described by Assunção et al. [2], but employed commonly mentioned activities. To migrate the cloned Android games into a software product line, we also tackled additional challenges, mainly constructing a feature model. Within this section, we describe our methodology, which includes reports on our *subject systems*, the *domain analysis*, *feature recovery*, and the actual *transformation*.

### 2.1 Subject Systems

The Apo-Games case [12] includes a set of five different Android games for which the source code is publicly available. We provide an overview of these games in Table 1. As we can see, these five games have been published within two years and comprise between roughly 2,000 and 6,500 source lines of code. Each of these games uses a customized version of a third-party game engine.

### 2.2 Domain Analysis

The first step of our case study was to perform a domain analysis. We started by installing all games from Google Play and documented the games' functionalities and visible entities (e.g., buttons, player character, game logic). Based on this documentation, we gained knowledge about the commonalities and variations within the games, which we can characterize as follows:

- **ApoClock** provides two different game modes, arcade and puzzle. In both modes, the player has to hit clocks with a ball before any of the clocks runs out of time. While the arcade mode is endless and accumulates a score, the goal in the puzzle mode is to clear a designed level.
- **ApoDice** is a puzzle game that shows dices to the player, each dice having a number indicating how often it can move. The player has to move each dice to a black square on the board within the given number of moves. A total of 30 levels is predefined, but users can add their own ones.
- **ApoMono** requires the player to move an avatar from a starting position to a goal. Throughout the avatar's path, there are different obstacles, for example, missing tiles or walls. The player can move some stones on the field to overcome such obstacles.
- **ApoSnake** is similar to the original snake game where the player controls a snake to collect candy, increasing its size. However, in ApoSnake, the amount of candies in a level is fixed and they have different colors, which colors the snake in the same way. The snake can then move through any wall that has the same color as itself.
- **MyTreasure** is a puzzle game within a two dimensional maze. The player has to collect a golden coin in this maze, for which they has to rotate the maze, allowing the avatar to move according to gravity. In addition, yellow blocks in the maze also move according to the rotation.

All of these games also have an editor that allows the user to create own custom levels for that specific game. Custom levels are stored together with all levels other users created, and all levels can be retrieved and played by using a button in the menu. However, we found that this button crashes some games (except ApoMono and MyTreasure), because the server on which the games were stored is not available anymore.

In addition to playing the games, we performed a lightweight architecture analysis. For this purpose, we reverse engineered the class diagrams of all games using IntelliJ IDEA Ultimate. We manually inspected the classes that were shown in the models and compared the code and models. This helped us to identify feature locations and understand that all games share a common core and comprise some unique classes, for example, for parts of the game logic or branding. Moreover, we found that the systems were not solely cloned, but already designed for reuse, for instance, there are common classes to define the basic functions of all buttons.

Finally, we built the source code using Android Studio[2] to make sure that the games build and run as expected. We found that ApoSnake did not start, due to an issue in the external games engine. Thus, while we migrated its features, we were not able to start it, neither as legacy nor as software-product-line game.

[1] https://bitbucket.org/Jacob_Krueger/splc2019_antenna_apo-games_spl

[2] https://developer.android.com/studio/index.html

## 2.3 Feature Recovery

Our domain analysis helped us to obtain a general understanding of the games and their features. To improve this understanding, we analyzed the source code of the games based on clone detection and pairwise comparison, an idea proposed by other researchers [7, 14].

**Clone Detection.** For code clone detection, we used the CPD tool.[3] We then analyzed the identified clones on file level, using the pairwise comparison of Meld.[4] To allow Meld to work properly, we first had to address the naming conventions of the Apo-Games [14], removing the game-specific prefixes. We considered identical code to be part of common features, while we considered variations for variable child features. Such a pair-wise comparison does not scale for many systems, but is feasible for comparing five games [7, 14].

**Feature Modeling.** We used FeatureIDE [17] to model the identified commonalities and variability in a feature model. To derive the dependencies between features, we relied on our domain knowledge and the presence or absence of features in the legacy games (e.g., there are game-specific features that have to be in alternative groups). In Figure 1, we display the final feature model that we constructed after the previous steps.

## 2.4 Transformation

We used the preprocessor Antenna from FeatureIDE to mark optional features in the software product line with `#ifdef` annotations. Considering the actual code transformation, we incrementally integrated the most similar games (in terms of identified code clones) into a common platform. To this end, we relied on the pairwise diffing of Melt that helped us to compare classes and to identify the exact differences between games. Initially, we focused on integrating two games (i.e., ApoDice and ApoSnake) and getting them to build and run, allowing us to test our setup and the source code. So, we first integrated only the common base and the main differences of these two games in few features. Afterwards, we annotated additional features to increase the variability of the software product line. We remark that we could still not get ApoSnake to run. After we ensured that we could build and run ApoDice as a variant of the software product line, we continued with integrating the other legacy games, in the following order: ApoClock, ApoMono, and MyTreasure. Unfortunately, we repeatedly had problems with building and running the configured games in Android Studio, which we could not fully resolve.

During our the transformation, we constantly performed quality assurance to maintain the quality of the resulting software product line. For this purpose, we configured the games in FeaturesIDE to then build and execute them in Android Studio. Besides testing for flaws in the code that may results in a bug or errors, we also compared the variants to the legacy games. With this comparison, we aimed to identify any unintended differences, for example, because we used Antenna incorrectly or lost functionalities.

# 3 RESULTS

In this section, we describe the artifacts we created. We summarize the characteristics of the feature model and source code of the software product line compared to the legacy games in Table 2.

[3] https://pmd.github.io/pmd/pmd_userdocs_cpd
[4] http://meldmerge.org/

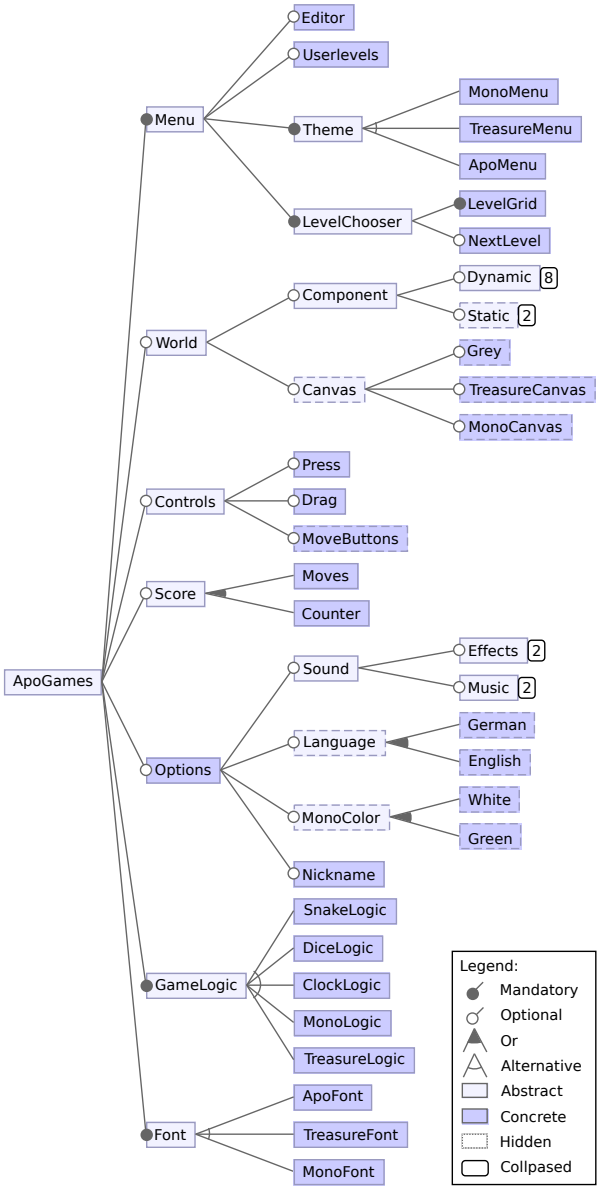


**Figure 1: Excerpt of the feature model of the extracted software product line. Hidden features are not yet configurable.**

**Feature Model.** In total, we identified 61 features, of which 42 are concrete features, and we implemented 27 of them in the software product line. Our implementation is a reduce set of all features, due to time constraints and unexpected efforts (cf.Section 4). We defined a group of child feature below each feature that required game-specific adaptations. As we can see in Figure 1, we defined few mandatory features (i.e., *GameLogic*, *Menu*, *Theme*, *LevelGrid*, *Font*), which are needed to set up the rough architecture of a game. In particular, the feature *GameLogic* is the most basic one that contains the more advanced game algorithms. Other features are optional and provide additional functionalities that were available in some games, such as an *Editor* and *Options*.

We defined 14 cross-tree constraints in the model. These ensure that a user can only configure a game that will work, even though not all configurations will result in a reasonable game. Thus, these constraints ensure that the right game elements (like a branding), have to be selected together. For example, *TreasureGameLogic* is implied if the corresponding *TreasureMenu* is selected. We included hidden features that we did not annotate with Antenna, yet.

**Source Code.** Considering the implementation, we migrated all selected Apo-Games into a common platform, but did not annotate all optional features. However, we achieved a considerable reduction in code size of approximately 51%. We remark that this number results from large restructurings and merges during the transformation that reduced the redundant code clones, but is not completely correct, due to tooling differences and the fact that Antenna uses comments for its annotations. Moreover, we could reduce the number of files from a total of 117 to 43. Considering that the largest legacy game had 28 classes on its own, these statistics show that there is an increase in the architectural complexity for the software product line, which we expected as it integrates multiple games. Nonetheless, the reduced size highlights the potential for reuse even for smaller cloned systems, such as Android games.

## 4 LESSONS LEARNED

During the analysis and transformation of the Android games, we experienced some challenges that we discuss in the following. Some points are strongly connected to Android and technical issues during software evolution, both of which may appear in any organization or open-source project at some point.

**Dissimilarity due to Libraries.** When we analyzed the architectures of the games, we expected that the games would be quite similar, due to the similar structure. However, despite ApoDice and ApoClock sharing almost all classes and the same look-and-feel, the actual implementations differ quite a lot. We found that these differences are due to the game engine that was used in two different versions, resulting in changes in, for example, the rendering technique. In our experience, such external libraries cannot only conflict expectations, but were also hampering variability.

**Unexpected Effort of Branding.** All of the games share some common elements, such as menus. We expected that these would comprise a lot of cloned code and only small variations to change the look-and-feel of individual games. However, we experienced that these views were fundamentally different in their source code. Thus, we re-designed common features and implemented them from scratch, only introducing branding as child features. This way, we reduced the redundancy that may have been in the software product line, but it required unexpectedly high effort and we had to stop at some point to focus on integrating features.

**Deciding about Features.** We experienced that it was challenging to decide whether we should try to extract a common feature or not. As aforementioned, merging can be an effortful task, because some features differ heavily in the legacy games, for instance, due to the different game types and evolution (e.g., the external game engine). After the migration, we are still not sure if extracting a software product line from the Android games was worth the effort. In our opinion, the systems need to be much closer to each other (i.e., not outliers [15]) and this may be a common problem with games.

**Readability of Source Code.** A common argument against annotations for variability is that they obfuscate the source code, which can become lengthy and hard to read [1, 20]. We experienced exactly this issue while transforming the variants, as we had to merge various implementations of the same method into one. This problem can be tackled with additional refactoring, but it still hampers the integration of variants and is a source for errors.

**Available Tooling.** We used FeatureIDE for most parts of our case study and found it most useful, too. In the initial phase, we identified other tools that could support the analysis of the legacy games. However, few of such research tools were available in a usable state or provided more help compared to other open-source solutions that we used. We found several specialized tools that could have been helpful with additional artifacts besides the source code. Overall, we found no established tool for software product lines that sufficiently supports the extraction process that we applied.

## 5 THREATS TO VALIDITY

**Internal Validity.** Unfortunately, we could not cooperate with the original developer of the Apo-Games for this case study. Thus, we had to analyze the legacy games and scope features ourselves. While we were careful and checked the results constantly, other researchers or practitioners would arguably still derive slightly different results. However, we could run and execute the legacy games form our software product line and our experiences as well as artifacts are nonetheless valuable baselines for future work.

**External Validity.** We had only access to five rather small Android games to migrate them. So, our process and the results may not be fully comparable to real-world scenarios. Still, considering that only few subject systems for the extraction of software product lines are available, the Apo-Games are a valuable dataset. Moreover, our case study already revealed important insights into problems that will only increase for larger systems, wherefore our experiences and artifacts are helpful for future research and practitioners.

## 6 CONCLUSION

In this paper, we described a case study in which we extracted a software product line from five Android games of the Apo-Games dataset. From this case study, we contributed the resulting artifacts (i.e., feature model, implemented software product line) and reported our experiences. Our main insight is that merging clones is a challenging process, even if we rely on annotation-based approaches that are often argued to only require simple additions of variability into the code [1]. However, locating, scoping, and merging of features remained challenging tasks that require considerable efforts. These problems are more conceptual and independent of the used variability mechanism.

In future work, we want to extend and refine our artifacts to enable researchers to use them as suitable ground truths for evaluations. Moreover, we plan to conduct further case studies to verify our insights. Especially, we aim to understand what factors (e.g., the variability mechanism) have what impact on processes and efforts.

**Acknowledgments.** This work is supported by the ITEA project REVaMP$^2$ funded by Vinnova Sweden (2016-02804), and the Swedish Research Council Vetenskapsrådet (257822902). We thank Jennifer Horkoff for valuable comments on this work.

# REFERENCES


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

[2] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering Legacy Applications into Software Product Lines: A Systematic Mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.

[3] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 7:1–7:8.

[4] John Businge, Openja Moses, Sarah Nadi, Engineer Bainomugisha, and Thorsten Berger. 2018. Clone-Based Variability Management in the Android Ecosystem. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 625–634.

[5] Paul C. Clements and Linda M. Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley.

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

[7] Slawomir Duszynski, Jens Knodel, and Martin Becker. 2011. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *Working Conference on Reverse Engineering (WCRE)*. IEEE, 303–307.

[8] Peter Knauber, Jesus Bermejo, Günter Böckle, Julio C. S. do Prado Leite, Frank J. van der Linden, Linda M. Northrop, Michael Stark, and David M. Weiss. 2002. Quantifying Product Line Benefits. In *International Workshop on Software Product-Family Engineering (PFE)*. Springer, 155–163.

[9] Charles W. Krueger. 2002. Easing the Transition to Software Mass Customization. In *International Workshop on Software Product-Family Engineering (PFE)*. Springer, 282–293.

[10] Jacob Krüger, Thorsten Berger, and Thomas Leich. 2019. *Software Engineering for Variability Intensive Systems*. CRC Press, Chapter Features and How to Find Them: A Survey of Manual Feature Location, 153–172.

[11] Jacob Krüger, Wolfram Fenske, Jens Meinicke, Thomas Leich, and Gunter Saake. 2016. Extracting Software Product Lines: A Cost Estimation Perspective. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 354–361.

[12] Jacob Krüger, Wolfram Fenske, Thomas Thüm, Dirk Aporius, Gunter Saake, and Thomas Leich. 2018. Apo-Games - A Case Study for Reverse Engineering Variability from Cloned Java Variants. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 251–256.

[13] Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. 2019. Where is my Feature and What is it About? A Case Study on Recovering Feature Facets. *Journal of Systems and Software* 152 (2019), 239–253.

[14] Jacob Krüger, Louis Nell, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2017. Finding Lost Features in Cloned Systems. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 65–72.

[15] Crescencio Lima, Wesley K. G. Assunção, Jabier Martinez, William Mendonça, Ivan C Machado, and Christina F. G. Chavez. 2019. Product Line Architecture Recovery with Outlier Filtering in Software Families: The Apo-Games Case Study. *Journal of the Brazilian Computer Society* 25, 1 (2019), 7.

[16] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 38–41.

[17] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.

[18] Tristan Pfofe, Thomas Thüm, Sandro Schulze, Wolfram Fenske, and Ina Schaefer. 2016. Synchronizing Software Variants with Variantsync. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 329–332.

[19] Julia Rubin and Marsha Chechik. 2013. A Survey of Feature Location Techniques. In *Domain Engineering*. Springer, 29–58.

[20] Janet Siegmund, Christian Kästner, Jörg Liebig, and Sven Apel. 2012. Comparing Program Comprehension of Physically and Virtually Separated Concerns. In *International Workshop on Feature-Oriented Software Development*. ACM, 17–24.

[21] Ştefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. 2015. Forked and Integrated Variants in an Open-Source Firmware Project. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 151–160.

[22] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems. In *International Systems and Software Product Line Conference (SPLC)*. ACM. Accepted.

[23] Frank van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action*. Springer.

[24] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. 2006. Assessing Merge Potential of Existing Engine Control Systems into a Product Line. In *International Workshop on Software Engineering for Automotive Systems (SEAS)*. ACM, 61–67.