Analysis of a Clone-and-Own Industrial Automation System: An Exploratory Study

- Work in Progress -

Nick Lodewijks
University of Amsterdam,
The Netherlands
nicklodewijks@gmail.com

Abstract

In industry the development of similar products is often addressed by cloning and modifying existing artifacts. This so called "clone-and-own" approach is often considered to be a bad practice, but is still perceived as a favorable and natural software reuse approach by many practitioners. In this paper, we present the research direction, context and related literature of an exploratory study on the analysis of an industry system developed using the clone-and-own approach. The main objective of our study is to understand and quantify the benefits and drawbacks of this approach, by analyzing the evolution and change characteristics of this system. We intend to present the initial results of this study at the seminar.

1 Introduction

Cloning is often considered to be a practice harmful to the quality of source code, and potentially a cause of maintainability problems (Kapser and Godfrey, 2006; Thummalapenta et al., 2010). Yet, in industry the development of similar products is often addressed by cloning and modifying existing artifacts. This so called "clone-and-own" approach is perceived as a favorable and natural software reuse approach by many practitioners, mainly because of its simplicity and availability (Dubinsky et al., 2013).

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org). 07-09 June 2017, Madrid, Spain.

Several tools and techniques for dealing with cloned product variants have been proposed. Some of them advocate elimination of all clones by merging the variants into a single platform, and others propose to maintain multiple variants as-is (Rubin, Czarnecki, and Chechik, 2013). What approach works best for a given situation depends on the domain and context of that situation. For example, in some cases completely eliminating all clones and adopting an integrated platform is not possible nor beneficial (Antkiewicz et al., 2014). Eliminating clones will increase coupling, and changing shared code may require re-testing of all variants that use it (Dubinsky et al., 2013). If the success of your product highly depends on the stability of the code-base, than completely moving away from a clone-and-own approach without considering its merits can be a reckless decision.

In order to determine the applicability of tools and techniques in a particular situation, it is essential to understand the context, environment and characteristics of that situation. Therefore, the main objective of this study is to understand and quantify the benefits and drawbacks of the clone-and-own approach used for the development of an industrial automation system. We aim to gather quantitative data by analyzing how this system evolved, and study how this approach has affected ongoing project development and maintenance.

2 Subject System

The system studied in this work is the MES-Toolbox; a proprietary Java-based factory automation system developed by ENGIE. Development of this system started 17 years ago and it has grown to contain more than 6500 Java files, with a total of approximately 1 Million Lines Of Code (MLOC). The MES-Toolbox is designed for industrial automation of batch and continuous production processes. It can visualize, control and register every step of an entire production process; from the intake of raw material (unloading from trucks, ships, bags, pallets, containers), preparation (dosing, weighing, heating), processing (pressing, grinding, mixing), storage, to distribution of end products to customers. Depending on what customers require for their production process, the system performs (e.g.) article and recipe management, quality registration, production planning, tracking and tracing of materials used in production, stock control, shift registration, production performance analysis and communicates with ERP systems.

This functionality is spread out over a number of services that run in parallel on multiple JVM's on a dedicated server in the factory. Most of these services can safely be rebooted or reconfigured without disturbing the production process. To monitor and control physical production equipment (e.g.: conveyors, mixers, weigher, buttons, lights), the MES-Toolbox communicates with PLC's that perform the actual low-level control of these physical devices. Since the MES-Toolbox is usually configured to contain detailed information about the physical equipment, structure and processes, much of this PLC code is generated.

The system has a modular structure, and its design aims to separate *common* and *core* code from *customer-specific* implementation code as much as possible. In practice however, this has proven to be very challenging due to specificity and high degree of variation of customer requirements in the domain of industrial automation (Schrock, Fay, and Jager, 2015). These ever changing requirements have led to a variant rich and highly configurable platform.

Within the organization there is a clear distinction between platform development and application development, this distinction is often found in a Software Ecosystem (SECO) (Lettner, Angerer, Prähofer, et al., 2014). A small team of five developers is responsible for the overall design, development and maintenance of this system. The founder and writer of the first line of code of this system is also still part of this team. Work of this team is mainly focused on maintenance of the core platform, development of complex customer specific features, standardization of functionality, development of product configuration tools, and provide support to application engineers. The author of this paper is also part of this team, thus is familiar with its architecture and the industrial automation domain.

While the platform contains a constantly growing set of reusable core components and ready-to-use standard solutions, for every new factory, a clone of the latest platform release is realized by creating a branch with the *Subversion* version control system. The clone are then configured and changed in any possible way by Application Engineers to add, modify or remove functionality. Most of the application engineers co-located with the platform engineers.

3 Research Direction

We will study this case from the perspective of the following research questions:

- What are the change characteristics, and challenges related to clone-and-own in the development of this industrial automation system?
- How does this practice affect ongoing project development and maintenance?

In our study we will use both qualitative and quantitative methods to gain insight into these questions. We will study existing literature to gather known characteristics and challenges of clone-and-own, product line engineering, software evolution and industrial automation. We then conduct in-depth interviews to determine whether and to what extent these findings may apply to this case.

While the experience of the author as a developer of the system may provide a detailed interpretation of fine grained changes, this can cause some bias. We aim to reduce this threat as much as possible by providing quantitative data to support our findings, and collaboration with an external supervisor.

For the quantitative data we will use the source code repository and code differencing tools as the main sources of data. To get insight into how this practice affects ongoing project development and maintenance, we will investigate how and to what extent changes are propagated between releases and projects.

4 Related Literature

Clone Evolution Patterns

Thummalapenta et al. (2010) proposed an approach for the identification of the evolution of cloned code fragments over time, and categorized the evolution patterns as (a) Consistent Evolution, (b) Late Propagation, (c) Delayed Propagation, and (d) Independent Evolution. In our study we may be able to use these patterns to characterize some of the change patterns in the evolution of the product family. For example, Delayed Propagation may be used as a strategy to validate the correctness of changes on some variants, before propagating them to all variants. Or, Independent Evolution may be used to keep the variant as-is after the project has been commissioned and the testing phase has already finished.

Similar characteristics were found by Stanciulescu, Schulze, and Wąsowski (2015) in a study on the advantages and disadvantages of forking using the case of Marlin, an open source firmware for 3D printers. They found that important bug-fixes were not propagated and functionality was sometimes developed more than once. Intuitively you may consider these findings to be bad practices, and drawbacks of clone-and-own. But, there are situations where this may be desirable, as the authors found that "Once the firmware is configured and running on the printer, new changes are not desired".

In an environment where the potential cost of an error can be large, systems are changed as little as possible when maintained (Cordy, 2003). In a clone-an-own based system, this characteristic can be detected by looking for patterns like Independent Evolution, the lack of synchronization with the origin, or redundant code. This is in line with some of the cloning patterns described by Kapser and Godfrey (2006). They argued that code duplication can also have benefits, and described the pro's and con's in a catalog of cloning patterns used in real-world systems.

Software Ecosystem Characteristics

Lettner, Angerer, Prähofer, et al. (2014) studied the relevance of characteristics of Software Ecosystems in the domain of industrial automation, and found some additional characteristics that according to them are particularly important in the industrial automation domain. For example, platform quality characteristics like stability and backwards compatibility, and long-term platform evolution seemed to be essential to the success of the studied system. One of the reasons for this conclusion was that *"application engineer B reported that he had to update a ten-year-old version of the platform software, because an important customer had decided to leave out several platform releases and then requested a new feature. This led to significant difficulties in merging the old software version with the new functionality."*. Developers of the system we study have reported similar issues with upgrading customer systems to a new release.

In a later study by Lettner, Angerer, Grünbacher, et al. (2014), the *change characteristics* and *software evolution challenges* of the same ecosystem were investigated. The software change taxonomy of (Buckley et al., 2005) was used to describe qualitatively when, where, and how changes were made in different parts of the system and what was affected by changes. The authors found that the ecosystem is subject to both continuous and periodic evolution. The core platform is continuously changed to include new features and bug-fixes, while those changes are only periodically released to platform users. The granularity of these changes is reportedly primarily coarse for customer requirements, and fine for bug fixes. Propagation of changes is done by hand, and change impact analysis is performed manually, based on expert knowledge.

The system we study is in the same domain and seems to be developed in a similar manner. Our study will be different in a sense that we aim to support qualitative findings with quantitative data by analyzing the evolution of the system. For example, we know that in this case changes are also propagated by hand, so we intend to study when this is done, how frequently, and what factors influence the decision on whether or not to propagate changes from project to a release, between releases or between projects.

Crosscutting Concerns

A possible area of interest in the analysis of clone-and-own evolution, is the presence and development of *crosscutting concerns* in the system. A crosscutting concern is a feature whose implementation is spread across many different modules (Marin, Deursen, and Moonen, 2007). If product variants, or clones, exhibit a high degree of variation in the implementation of crosscutting concerns, we expect that this may also affect the extent to which changes are propagated, and how the code-bases diverge.

Marin, Moonen, and Deursen (2005) propose a classification system for crosscutting concerns in terms of *sorts*, where a *sort* is a description based on a number of distinctive properties. A *sort* we expect to find often in this case study is *Entangled Roles.* In Object Oriented terminology this sort is defined as: *Implement a method with (entangled) functionality that belongs to a different concern than main concern of that method.* A characteristic of clone-and-own is that it allows application engineers to make these kind of fine-grained changes quickly. For example, a customer wants to be notified when stock levels exceed a certain value for a specific product. If there is no such monitoring system in place, then the fastest solution can be to add this functionality to a method that deals in some way with stock-control. Implementation of a generic solution may exceed the level of expertise of the application engineer, and waiting for a platform engineer to develop the solution may take too much time.

Figueiredo et al. (2009) describe 13 patterns of crosscutting concerns identified in three case studies, one of which was a software product line. The authors found that some patterns consistently emerged in situations with frequent use of inheritance. They found that this was often the case in product lines, because "*Program families rely extensively on the use of abstract classes and interfaces in order to implement variabilities. The inappropriate modularisation of such crosscutting concerns might lead to future instabilities in the design of the varying modules*"

Detection of crosscutting concerns is called *aspect mining*. Various aspect mining techniques have been proposed (Kellens, Mens, and Tonella, 2007; Tourwé and Mens, 2004; Ceccato et al., 2006). For example, fan-in analysis looks for crosscutting functionality by detecting methods that are explicitly invoked from many different methods scattered throughout the code (Marin, Deursen, and Moonen, 2007). History-based concern mining techniques analyze change-history to detect which program entities change together frequently (Breu and Zimmermann, 2006; Adams, Jiang, and Hassan, 2010). Hashimoto and Mori (2012) developed a tool that improves history-based concern mining by combining it with fine-grained change analysis based on abstract syntax tree differencing.

The goal of our study is not to investigate the effectiveness of these tools and techniques, but they may provide insight into the characteristics of change patterns we will find.

Clone-and-Own in Product Line Engineering

Dubinsky et al. (2013) studied the processes and perceived advantages and disadvantages of the clone-and-own approach of six industrial software product lines. They show that cloning is perceived as a favorable and natural reuse approach by the majority of practitioners in the studied companies, mainly because of its simplicity and availability. They found that practitioners lack the awareness and knowledge about forms of reuse, and many alternative approaches fail to convince them that they yield better results.

Rubin, Czarnecki, and Chechik (2013) proposed a framework to organize knowledge related to the development, maintenance and merge-refactoring of product lines realized via cloning. This framework is a step towards a recommender system that can assist users in selecting tools and techniques that are useful in their situation.

Hetrick, Krueger, and Moore (2006) report on the experience of a structured, incremental transition from a clone-and-own approach to software product line practices. They show that it is possible to make this transition without a large upfront investment and without disruption of the ongoing production schedules. The authors show that the *file branch factor* gradually reduced during the transition, to a point where all branches from product line core assets were completely eliminated. In our study we intend to use similar metrics to study the extent to which variants have diverged from their origin. We think that this metric may be suitable to indicate what parts of the system require attention, and can aid the prediction of how tools and techniques may affect the situation.

Antkiewicz et al. (2014) propose an incremental and minimally invasive strategy for adoption of product-line engineering. The strategy is called *virtual platform*, and should allow organizations to obtain incremental benefits from incremental changes to the development approach.

By studying the development practices of our industry case, we gain insight into an industry context and the needs of practitioners. This may serve as input for recommender systems, requirements for the *virtual platform*, and can be helpful to practitioners, researchers and tool developers.

5 Conclusion

In this paper, we have outlined the research direction, context, and literature that may be of interest in our exploratory study; the analysis of an industry system developed using a clone-and-own approach. The main objective of our study is to understand and quantify the benefits and drawbacks of this approach, by analyzing the evolution and change characteristics of this system.

We intend to present the initial results of our study at the seminar, where we hope to get feedback on the research direction, and pointers to literature that may be relevant to us.

Acknowledgements

We thank prof. dr. J.J. Vinju for his helpful inputs on related literature and the direction of this study.

References

Adams, B., Z. M. Jiang, and A. E. Hassan (2010). "Identifying Crosscutting Concerns Using Historical Code Changes". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. Vol. 1. ACM, pp. 305–314.

Antkiewicz, M. et al. (2014). "Flexible Product Line Engineering with a Virtual Platform". In: *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*. ACM, pp. 532–535.

Breu, S. and T. Zimmermann (2006). "Mining Aspects from Version History". In: *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. IEEE, pp. 221–230.

Buckley, J. et al. (2005). "Towards a Taxonomy of Software Change". In: *Journal of Software Maintenance and Evolution: Research and Practice* 17.5, pp. 309–332.

Ceccato, M. et al. (2006). "Applying and Combining Three Different Aspect Mining Techniques". In: *Software Quality Journal* 14.3, pp. 209–231.

Cordy, J. R. (2003). “Comprehending Reality - Practical Barriers to Industrial Adoption of Software Maintenance Automation”. In: *Program Comprehension, 2003. 11th IEEE International Workshop on.* IEEE, pp. 196–205.

Dubinsky, Y. et al. (2013). “An Exploratory Study of Cloning in Industrial Software Product Lines”. In: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pp. 25–34.

Figueiredo, E. et al. (2009). “Crosscutting Patterns and Design Stability: An Exploratory Analysis”. In: *IEEE International Conference on Program Comprehension*, pp. 138–147.

Hashimoto, M. and A. Mori (2012). “Enhancing History-Based Concern Mining with Fine-Grained Change Analysis”. In: *2012 16th European Conference on Software Maintenance and Reengineering.* IEEE, pp. 75–84.

Hetrick, W. A., C. W. Krueger, and J. G. Moore (2006). “Incremental Return on Incremental Investment: Engenio’s Transition to Software Product Line Practice”. In: *International Conference on Object-Oriented Programming, Systems, Languages and Applications.* ACM, pp. 798–804.

Kapser, C. and M. Godfrey (2006). “”Cloning Considered Harmful” Considered Harmful”. In: *2006 13th Working Conference on Reverse Engineering.* IEEE, pp. 19–28.

Kellens, A., K. Mens, and P. Tonella (2007). “A Survey of Automated Code-Level Aspect Mining Techniques”. In: *Transactions on Aspect-Oriented Software Development IV.* Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 143–162.

Lettner, D., F. Angerer, P. Grünbacher, et al. (2014). “Software Evolution in an Industrial Automation Ecosystem: An Exploratory Study”. In: *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on.* IEEE, pp. 336–343.

Lettner, D., F. Angerer, H. Prähofer, et al. (2014). “A Case Study on Software Ecosystem Characteristics in Industrial Automation Software”. In: *Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014.* ACM, pp. 40–49.

Marin, M., A. van Deursen, and L. Moonen (2007). “Identifying Crosscutting Concerns Using Fan-In Analysis”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 17.1, pp. 1–37.

Marin, M., L. Moonen, and A. van Deursen (2005). “A Classification of Crosscutting Concerns”. In: *21st IEEE International Conference on Software Maintenance (ICSM’05).* IEEE, pp. 673–676.

Rubin, J., K. Czarnecki, and M. Chechik (2013). “Managing Cloned Variants: A Framework and Experience”. In: *Proceedings of the 17th International Software Product Line Conference - SPLC ’13.* ACM, p. 101.

Schrock, S., A. Fay, and T. Jager (2015). “Systematic interdisciplinary reuse within the engineering of automated plants”. In: *Systems Conference (SysCon), 2015 9th Annual IEEE International*, pp. 508–515.

Stanciulescu, S., S. Schulze, and A. Wąsowski (2015). “Forked and Integrated Variants in an Open-Source Firmware Project”. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, pp. 151–160.

Thummalapenta, S. et al. (2010). “An Empirical Study on the Maintenance of Source Code Clones”. In: *Empirical Software Engineering* 15.1, pp. 1–34.

Tourwé, T. and K. Mens (2004). “Mining Aspectual Views using Formal Concept Analysis”. In: *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on.* IEEE Comput. Soc, pp. 97–106.