Towards a Formalised Approach for Integrated Function Updates of Mechatronic Systems

Tim Warnecke, Karina Rehfeldt, Andreas Rausch

Technische Universität Clausthal
38678 Clausthal-Zellerfeld, Germany
email: {tim.warnecke, karina.rehfeldt, andreas.rausch}@tu-clausthal.de

David Inkermann, Tobias Huth, Thomas Vietor

Technische Universität Braunschweig
38106 Braunschweig, Germany
email: {d.inkermann, tobias.huth, t.vietor}@tu-braunschweig.de

***Abstract*—Looking at different everyday products, we are facing the situation that they are replaced although their technical life time has not ended. The main reason for this is that customers often replace products like smart phones or household devices, because there are new ones available providing new and additional functions and features. These functions and features are predominately based on software and follow shorter development and innovation cycles. From the resource point of view the mismatch between technical life time and use period of products leads to great disposal. In order to address this challenge, a common concept is to update existing products. To provide substantial new functions, such updates have to concern both hardware and software components. Due to the complexity of dependencies between these components, it is not an easy task to come up with these integral and verified updates. As a first step to tackle this problem, we propose a formalized approach to describe integrated hardware and software upgrades. Based on this formalism, we present our ongoing research and preliminary results in the fields of functions and systems modeling.**

***Index Terms*—Complex Systems; Design for Maintainability; Release Management; Software Product Lines; Software Evolution.**

I. Introduction

In most of modern products like vehicles, household devices or machine tools, functions are realized by a combination of hardware, electronics and software components. These components are the results of development within different domains and are often differing with regard to the time needed for their realization, their innovation cycles and their specific technological advancements. Fast technological developments, for instance in the fields of comfort or communication interfaces, are often driven by software engineering. However, engineering and development of hardware components like a car body or drive train are missing the speed of these advancements. At the same time, implementation of new software often requires specific hardware like sensors or actors. Therefore, common practice is to implement and release new functions only within new released product generations. This leads to the situation that the actual use period of products compared to their technical life span is greatly shortened [1], [2]. Products are shut down or disposed of, although their functionality is still given from the technical point of view [3]. This is caused by the customers buying decisions which are tremendously influenced by features like comfort, assistance or multimedia-based functions. Furthermore, the gap between technical life span and actual use period leads to the disposal of still valuable resources like materials but energy - for instance of the manufacturing process stored within the mechanical components - as well. In order to counteract this trend and increase the ecological sustainability of products, there are several restrictive laws planned. For instance, it is planned to define a minimum service life span for electric components [4]. Another expedient and less restrictive strategy is a so called planned product upgrade of an existing product. Within this paper we follow the second strategy.

A. Challenge of Integrated Product Updates

There is a great body of literature dealing with the adaptability and changeability of products. General approaches referred to as Design for Changeability [5], Design for Flexibility [6] or modularization [7] provide basic strategies to adapt system properties and functions during the life cycle. However, these approaches are focusing on the classical life cycle understanding and do not consider upgrades of existing products. The focus of these approaches is to support the initial design of products.

In the domain of software engineering, product updates are generally possible if the software was designed accordingly. However, when it comes to systems, including both hardware and software like embedded systems, software upgrades are limited by the installed hardware components. Furthermore, software updates are often hindered because new software

 ISBN: 978-1-61208-532-6

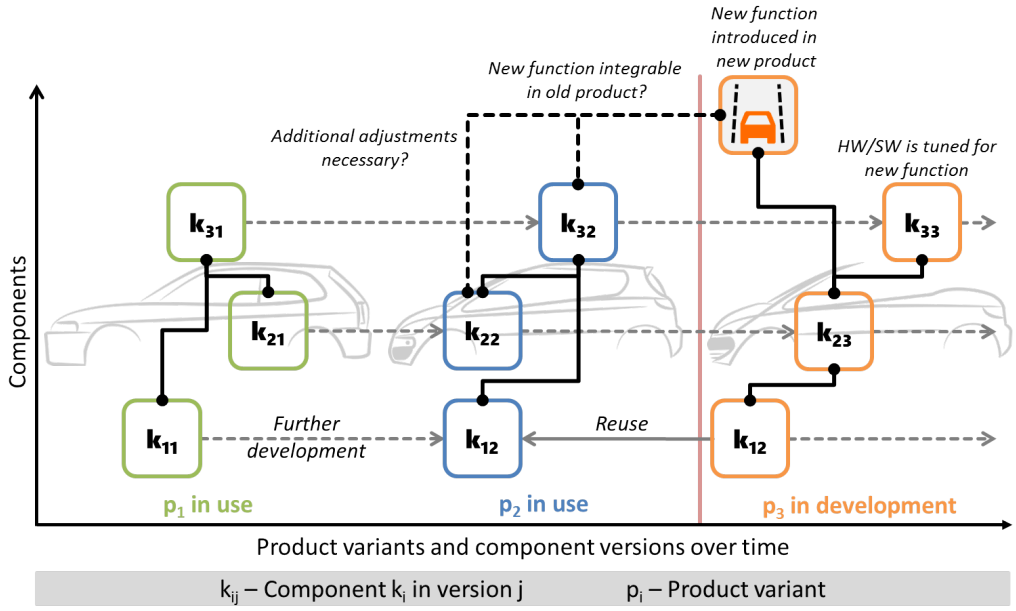

Fig. 1. Illustration of the question whether an existing product variant can be updated with a new function.

is developed with current hardware components in mind and does not consider limitations of older hardware components. Therefore, compatibility with existing hardware components of previous product variants is not guaranteed. This forward-looking development drives progress, but makes the integration of new software functions into existing products even harder. In the domain of hardware engineering, it is even worse, since most of the used hardware components can't be easily changed after the initial handover to the customer. Furthermore, it is hard to identify and define required extensions within existing hardware configurations to enable the implementation of new software functions.

In order to support an integrated function update of mechatronic systems, it is therefore essential to analyze the interrelations between functions and features as well as the components realizing these functions and features.

B. Focus of Research

Figure 1 highlights the problem addressed in this contribution by schematically representing the development of vehicle variants in product lines and their constituting components. The development is ongoing over time and new products are based on either existing, updated or newly developed components. To support the update of products, it is essential to understand how their components can be adapted. This question is directly linked to the time scale we present in Figure 1. Here, the first two vehicles p_1 and p_2 represent vehicles, which are already in use. Vehicle p_1 of the first generation uses three components $k_{11} - k_{31}$. Note, here k_{ij} is the component k_i used in development version j. The vehicle p_2 in the middle is an advanced development of p_1. Product p_2 uses the same component identities $k_1 - k_3$ but in development version 2, here components $k_{12} - k_{32}$. On the right hand side of Figure 1 a vehicle in development version, p_3, is illustrated. Variant p_3 uses the components k_2 and k_3 in version 3. In this case, the component k_{12} used in vehicle p_2 is reused directly, because an improvement was not necessary. Additionally, p_3 receives a new functionality, in this case a lane keeping system (LKS), illustrated by the top most component. Because of a proper executed development process of vehicle p_3, we can be certain that the LKS will work correctly in the component configuration of this vehicle. Additionally, during its own development, the previous vehicle p_2 was already tested and checked for functional correctness. In order to support the integrated update of the vehicle p_2, we are facing the question whether the LKS system of vehicle p_3 can be also integrated and which changes have to be made with regard to existing software and hardware components.

Our research is focusing on methods to model and analyze the compatibility and interaction between hardware and software components. Primary objective is to provide methods supporting estimation and evaluation of required changes of both, hardware and software, components. Furthermore, we aim to reduce the effort required for safeguarding of changed product configurations after updating these. This research will contribute to efficient upgrades of existing mechatronic systems and, therefore, the extension of their use periods. The research is guided by the following questions:

- How do we model the structure, behavior, functions and requirements of systems to identify required adaptations or updates with regard to hardware and software?
- How do we reduce test effort of a new system configuration when its functions and structure are already partly tested?
- How do we support the design of evolution friendly

 ISBN: 978-1-61208-532-6

system structures?

In this contribution, we describe a formalism to specify the first question. Furthermore, we state the areas to work on further more precisely. Therefore, in Section II we discuss the state of the art and basic understanding of releases and design for maintainability as well as approaches for systems modeling, both from the mechanical and the software point of view. In Section III, we define a simple formalization to describe functions, components and their connecting structure. Based on this, we introduce our approach to the stated problem in Section IV. The paper is summarized by a short conclusion in Section V.

II. Background and State of the Art

Based on the focus of research introduced, in this section an overview and definitions of essential terms are given. The focus is on basic strategies to extend the use period of complex systems and their effects on hardware and software components.

Release Management (RM) is originated from software development and describes the process and activities conducted to develop and deploy releases as a result of change requests. A major task of RM is to maintain the integrity and minimize the disruption of the original system during and after the deployment release of new features. This is done by prior planning and testing of a release [8]. A release in this context, is a collection of "one or more changes to a service that are built, tested and deployed together" [8]. At the beginning of the RM process, a subset of changes (sometimes also referred to as requirements) is selected as the scope of the release [9]. Besides the development and implementation of the release, RM also covers the estimation of effort and the resource management needed for planning, design and implementation of releases. In information technology, RM is well-established. The transfer to "hardware" products is still subject of research, cf. [10].

Design for maintainability (DfM) comprises the consideration of aspects regarding for instance serviceability, repairability or supportability, minimizing the effort during the use phase of the system to keep it in - or to restore it to - a usable condition [11]. According to the IEEE Standard Glossary of Software Engineering Terminology, maintainability is defined as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" [12]. DfM is closely linked with the modularity of products whose foundation is laid during the early design phases in form of the product's architecture [13].

There is a lot of research dealing with effort estimation in software changes, design of maintainable software systems and processes and methods for software engineering. Regarding software changes, in the last years more and more approaches were made in feature-oriented software evolution and changes. [14] proposes a feature-based software evolution with automatic traceability, analysis and recommendations. Passos et. al assume that managing changes at the level of features can help system designers in estimating costs and efforts while having a better understanding of the impacts of a change on the system.

Ferreira et. al [15] studied whether feature-oriented programming really benefits variety management in software product lines (SPL). They argue feature-oriented programming is indeed well suited for software product lines but still has drawbacks. The SPL tends to be more stable under certain aspects with feature-oriented programming. Nevertheless, feature-based software evolution does not have a general answer how new features can be integrated into whole systems dealing with both hardware and software.

The difficulty of the considered mechatronic systems and their updates are the deep interaction of hardware and software components. The software is built with a specific hardware concept in mind and vice versa. To our current knowledge there is no model or paradigm to generally address the problem of modeling a complex system to identify required adaptations for upgrade of functionalities. As a first step towards a better understanding and basis for our further research, in the following section we propose a formalism to describe integrated function updates of existing mechatronic systems. Based on the formalism, we identify further research needs.

III. Definitions

To give an idea on how we are tackling the mentioned questions, we formalize the concepts of product variants, hardware and software components as well as functions. Without loss of generality, we term hardware and software components simply as components and formalize them with the same definition. Later, a distinction between hardware and software components might become necessary. But for now, we are interested in the general idea of including new functions through new components whether hardware or software into existing products.

First, we define that every product variant consists of a restricted number of **software and hardware components** k_{ij} with i is the identity and j is the development version of component k. Furthermore, with regard to a development process we denote k_{ij-1} as the **predecessor version** of component k_{ij} and k_{ij+1} the **successor version** of component k_{ij}. So, we define the set of **all usable components** as

$$K = \{k_{ij}\} \text{ with } i, j \in \mathbb{N}$$

We define a **product variant** as

$$p_n := (K_m, v_m := K_m \times K_m) \text{ with } n, m \in \mathbb{N}$$

and the set of all possible product variants as

$$P := \{p_n\} \text{ with } n \in \mathbb{N}$$

This highlights that every product variant $p_i \in P$ consists of a subset of all usable components $K_m \subseteq K$ and a **structure** v_m. v_m describes the connections between the different

components K_m to form a fully functional system. We do not differentiate the connections between components of a product variant. In case of a connection between two software components, it might be an interface usage. The connection between two hardware components might be a physical link while the connection between hardware and software are, e.g., signals.

Every product variant p_n is developed with the objective to fulfill a pre-defined set of functions taking into account specific requirements. The set of **all possible functions** can be defined as

$$F = \{f_n\} \text{ with } n \in \mathbb{N}$$

The function set $F_q \subseteq F$ denotes a subset of all possible functions. If a function set F_q is **fulfilled** by a product variant p_n we denote this as a satisfiability relation

$$F_q \models p_n$$

In the next section, we show our formalized approach to the problem described in the introduction.

IV. A Formalized Approach

If products are developed as product lines, we expect that a previous product variant p_{old} exists. p_{old} fulfills a known set of functions F_{old}. Additionally, a new product variant p_{new} exists, which fulfills F_{new}. In the proposed notation, we write $F_{old} \models p_{old}$ and $F_{new} \models p_{new}$. We now formalize our question whether p_{old} may be upgraded to fulfill the new function set F_{new} of our new product. Moreover, which changes in p_{old} are necessary to be able to fulfill the functions F_{new} completely or with limitations. We assume that a product variant called p'_{old} derived from p_{old} exists and fulfills the set of functions F_{new} from p_{new} - that is $F_{new} \models p'_{old}$. We define the product variant p'_{old} as

$$p'_{old} := (\{K_m\}, v_m : K_m \times K_m)$$
$$\text{with } \forall k_{xa} \in \Pi_1(p_{new}) \exists k_{xb} \in \Pi_1(p'_{old})$$

With other words, we are looking for a product variant, which contains for every component identity k_i in version a from p_{new} a corresponding component identity k_i in version b. Figure 2 shows the construction of the updated product based on the old and new product. The upper two product variants are the old product p_{old}, which will be updated and the new product p_{new} with a new functionality. There are components from p_{old}, which are kept in p'_{old}. Those are components not touched by the update (blue) and components, which are equal to the ones used in p_{new} and, therefore, kept (gray). Components existing in different version in p_{old} and p_{new} are updated (violet). Also, in p_{old} there are components that are removed in p'_{old} (red). Additional components for the new functionality in p_{new} are added to p'_{old} (green). The bottom product variant shows a possible outcome of the updated p_{old}, our p'_{old}.

Coming back to our initial question, we want to find a way to determine whether a new function can be integrated

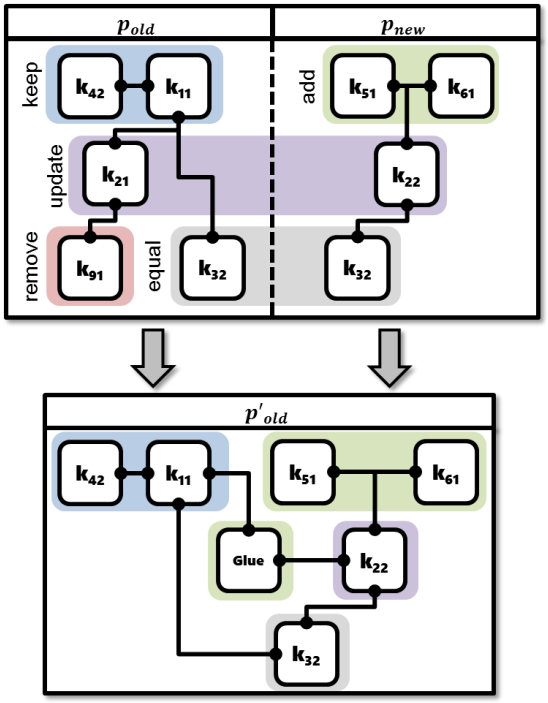

Fig. 2. Illustration of the construction of an updated product based on p_{old} and p_{new}.

into an old product and which changes to the old product are necessary. On basis of our formalization, we can now formulate essential questions regarding the necessary changes between p_{old} and p'_{old} and the changes' relation to p_{new}.

1) Which components of p_{new} are already included in p_{old} and, therefore, do not need to be changed in p'_{old}?

This questions asks for components in p_{new}, which were already used in the same version in p_{old}. Therefore, we do not need to change p_{old} regarding these components and p'_{old} remains unchanged. In Figure 2 these are the components with gray background.

2) Which components of p_{new} are not included in p_{old} and, therefore, have to be added to p'_{old}?

This question asks for components introduced with the new functions in p_{new}. These components cannot be part of p_{old} and, therefore, have to be added to p'_{old} to fulfill the new functions. In Figure 2 these are components with green background.

3) Which components k_{ia} of p_{new} are included in p_{old} but in a different version b and which version is included in p'_{old}?

This question asks for component identities, which are used both in p_{old} and p_{new} but in different versions. In other words, the component used in p_{new} is either the predecessor or successor of the component used in p_{old}. The question is, which of these version will be included in the updated product p'_{old}. In Figure 2 these are the components with violet background.

4) Which component identities from p_{old} are not included in p_{new} but in p'_{old}?

This question asks for the components in p_{old} that are not touched by the update. These components are not involved in the new or any associated functions. In Figure 2 these are the components with blue background.

5) Are there any components in p'_{old}, which are neither included in p_{new} nor p_{old}?

This question asks for components that were introduced in p'_{old} as a kind of adapter for components from p_{new}. In other words, the necessary components from p_{new} could not be added to p'_{old} because of an incompatible interface. However, an adapter component could resolve this incompatibility. In Figure 2 this is the glue component with green background in p'_{old}.

To tackle these questions we are convinced that a new modeling and engineering approach for complex systems has to be developed. Moreover, even after integrating and developing new components to construct a p'_{old} from p_{old} and p_{new}, it is not obvious that the new product will work like expected. Usually, after the integration of a new function, the entire system has to be retested and verified to guarantee that no unexpected side effects can occur. This includes testing functions without direct interactions with the new one because most side effects occur indirectly.

Naturally, it is not possible to fully test and verify already developed products in the field. Testing is a very time and cost consuming task, which is even harder to accomplish for older products that no longer have a development team. So, it is preferable that we don't need to test and verify the whole product with all its functions again. Instead, it would be preferable to just focus on parts of p'_{old}, which have changed in comparison to p_{old}.

We ask the question, whether it is possible to achieve much more simple retests for the new functions with the test results and data and control flows of p_{old} and p_{new} and the changes made to construct p'_{old}. We are convinced that these informations enable us to slice the architecture of p'_{old} in such a way that only new functions and some of their dependencies to components from p_{old} need to be retested.

V. Conclusion

We have presented the current mismatch between technical life time and the actual use period of modern products and highlighted the need for an integrated functional update approach for mechatronic systems. One of the main reasons for customers to replace a product, are extended functionalities implemented in newer product generations. In order to address this challenge, a common concept is to provide updates for existing products. Due to complex dependencies between software and hardware components, it is not an easy task to come up with these integral updates. Therefore, it is essential to understand the interrelations between the different components as well as the functions and components. We have shown that a critical improvement has to be made regarding modeling approaches for complex systems, addressing both the structure of the system as well as the functional view upon the system. While there exist methods to design both mere software and hardware systems for maintainability, the interaction of both domains in complex mechatronic systems asks for new approaches.

The formalization introduced will be used as a starting point for further research. The next research steps will be to develop an integrated modeling language for requirements, structures and components in mechatronic systems. This modeling language will be designed with the goal to identify needs for adaptation in case of functional upgrades. Another step will be the development of a method to evaluate effort of integral updates.

Our last research question is how to minimize the necessary test cases when a product was updated. Based on slicing techniques we want to reuse the results of formerly done tests to reduce the necessary retests when a product was only slightly changed.

References

[1] K. Ishii, "Incorporating end-of-life strategies in product definition," *Proc. of EcoDesign'99*, pp. 364–369, 1999.

[2] Y. Umeda, T. Daimon, and S. Kondoh, "Life cycle option selection based on the difference of value and physical lifetimes for life cycle design," *Proc. of the International Conference on Engineering Design, ICED 2007*, 2007.

[3] K. Watanabe, Y. Shimomura, A. Matsuda, S. Kondoh, and Y. Umeda, "Upgrade planning for upgradeable product design," in *Quantified Eco-Efficiency*. Springer, 2007, pp. 261–281.

[4] "Directive 2009/125/ec of the european parliament and the council of 21 october 2009 establishing a framework for setting of ecodesign requirements for energy-related products," 2009.

[5] E. Fricke and A. P. Schulz, "Design for changeability (dfc): Principles to enable changes in systems throughout their entire lifecycle," in *Systems Engineering, Vol. 8, No. 4, 2005*, 2005, pp. 342–359.

[6] A. Bischof, "Developing flexible products for changing environments," Dissertation, Technische Universität Berlin, 2010.

[7] K. Ulrich and S. Eppinger, *Product Design and Development*. New York: McGraw-Hill, 1995.

[8] "IEEE standard - adotion of the ISO/IEC 20000-2:2012, information technology - service management - part 2: Guidance on the application of service management systems," 2013.

[9] P. Carlshamre, "Release planning in market-driven software product development: Provoking an understanding," *Requirements engineering*, vol. 7, no. 3, pp. 139–151, 2002.

[10] G. Schuh and W. Eversheim, "Release-engineeringan approach to control rising system-complexity," *CIRP Annals-Manufacturing Technology*, vol. 53, no. 1, pp. 167–170, 2004.

[11] R. F. Stapelberg, *Handbook of reliability, availability, maintainability and safety in engineering design*. Springer Science & Business Media, 2009.

[12] "IEEE standard glossary of software engineering terminology," 1990.

[13] G. Schuh, S. Aleksic, and S. Rudolfs, "Module-based release management for technical changes," in *Progress in Systems Engineering: Proc. of the twenty-third International Conference on Systems Engineering*, 2015, pp. 293–298.

[14] L. Passos, K. Czarnecki, S. Apel, A. Wasowski, C. Kästner, and J. Guo, "Feature-oriented software evolution," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. ACM, 2013, p. 17.

[15] G. C. S. Ferreira, F. N. Gaia, E. Figueiredo, and M. de Almeida Maia, "On the use of feature-oriented programming for evolving software product linesa comparative study," *Science of Computer Programming*, vol. 93, pp. 65–85, 2014.

 ISBN: 978-1-61208-532-6