Variability in Time — Product Line Variability and Evolution Revisited

Christoph Elsner*, Goetz Botterweck†, Daniel Lohmann‡, Wolfgang Schröder-Preikschat‡
* Siemens Corporate Technology & Research, Erlangen, Germany
christoph.elsner.ext@siemens.com
† Lero – The Irish Software Engineering Research Centre, Limerick, Ireland
goetz.botterweck@lero.ie
‡ Friedrich-Alexander University Erlangen-Nuremberg, Erlangen, Germany
{wosch,lohmann}@cs.fau.de

***Abstract*—In its basic form, a variability model describes the variations among similar artifacts from a structural point of view. It does not capture any information about *when* these variations occur or *how* they are related to each other in time. This abstraction becomes problematic as soon as time-related aspects become essential for the modeling purpose, e.g., when providing long-term support for a product line or when planning its future strategy.**

In this paper, we provide an overview of approaches that deal with time-related aspects in variability, which is summarized under "variability in time". In contrast, the modeling of structural commonalities and differences is referred to as "variability in space". We take an inductive approach and survey different uses of the term "variability in time", which turn out to be orthogonal. We generalize the uses and identify three different types: variability of linear change over time (maintenance/evolution), multiple versions at a point in time (configuration management), and binding over time (product derivation). We validate the types by using them to describe complex product line evolution scenarios where they exhibit expressive and discriminatory power. Finally, we go into depth for the first type (maintenance/evolution) and identify the tasks and aspects to be considered when building a detailed evolution research agenda in the future.

I. Introduction

Variability can be seen as "the ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context." [30]. Similarly, variability modeling techniques "aim at representing the variability in a product family". [27] Well-known techniques are feature modeling [14], orthogonal variability modeling [22], COVAMOF [28], and decision modeling [2].

These techniques, however, describe the variability of a product line for one particular moment in time only. Given the inevitable change of a software system over time, several authors added the time dimension to variability. Whereas the classical notion of variability is referred to as "variability in space" the new dimension is called "variability in time"[1] [22], [9], [24], [32], [16], [19], [4]. For ease of reading, when we use the term variability without any qualification in the following, we actually refer to "variability in space".

[1]We regard the terms "variability in time" and "variability over time", both in singular or plural, as synonyms.

The topic of "variability in time" is of considerable importance and has also been discussed in projects and workshops ([21], [33], [15]). However, recent uses and definitions of the term refer to widely differing time-related variability issues. Authors claiming to address "variability in time"—according to their definition—therefore actually only address a *subset* of time-related problems. Whereas there exists substantial work and a common understanding on what "variability in space" is about, this is not the case for "variability in time". For time-related variability issues in product line engineering, an established body of knowledge, which allows for decomposing and structuring the problem in an appropriate way, is still missing. This paper performs first steps towards such a common body of knowledge.

We take an inductive approach and survey different uses of "variability in time" (Section II). They turn out to be orthogonal, so we generalize them and identify three types of variability in time: variability of linear change over time (maintenance/evolution), multiple versions at a point in time (configuration management), and binding over time (product derivation) (Section III). This gives an expressive terminology for characterizing complex time-related variability interrelations, which we illustrate for several product line evolution scenarios (Section IV). Finally, we go into depth for the first type (maintenance/evolution), for which we define tasks and aspects (future planning, present modeling and tracking, past analysis) (Section V), serving as a prerequisite for defining a research agenda in the future.

II. Variability in Time in the Literature

In this section we report on the uses of the term "variability in time" in recent work. Interestingly, it only appears in papers within the product line context. The term has been used to describe three different problem areas: product line evolution, product line versioning, and product line binding variability.

A. Product Line Evolution Variability

One definition recited several times [19], [12], [34] is of Pohl et al. [22]: "Variability in time is the existence of different versions of an artefact that are valid at different times." This definition is not without problems. Having different versions of an artifact is obviously not specific to software product lines. It holds for every evolving software system. However, even for single system development, it is common that one product is released in different versions and, thus, one has to deal with different artifact versions at the *same* time. This definition, however, implicates that there is a straight flow of artifact versions, each new version actually invalidating its predecessor.

Other authors [24], [7] use the term "variability in time" to describe not only the change of the artifact versions over time, but also of their *variability dependencies* (denoting the "variability in space") over time. As requirements change over time, the product line must evolve as well. For a product line this means adding, removing, or changing features, as well as adding, removing, or changing *variability dependencies* (e.g., mandatory, optional, alternative). Still, however, they do not address multiple artifact versions valid at the *same* time.

B. Product Line Versioning Variability

In [32], in turn, the problem of "variability in time" is seen as a problem of dealing with multiple valid versions at the current moment in time. Maintaining one product in different versions in parallel is already a challenge for single product development, as each product consists of a multitude of artifacts of specific versions. For product lines this problem is even more difficult; a multitude of products, each possibly having several released versions, must be related to the correct artifact version for maintenance.

C. Product Line Binding Variability

Other authors, finally, use the term "variability in time" to describe the creation time and binding time of variability [4], [6], [12], [13]. During domain engineering, at certain phases of system development (e.g., analysis, design, coding compilation, linking, distribution, installation, start-up, or run-time) variation points are "made explicit", meaning foreseen, designed, or implemented in a dedicated way. In application engineering, these variation points are successively bound to specific variants, decreasing the variability until at the end one specific system is the end product.

III. Assigning the Different Notions to Types

As can be seen from the previous section, the notion of the term "variability in time" varies considerably, each adopting a different viewpoint. However, the uses actually are orthogonal. Thus, in this section, we assign the different notions of the term to types, which will help to reason about product line variability over time providing a reasonable terminology. We generalize the three above notions and subordinate more time-related variability research, both from general software engineering and product line research, to which we will give exemplary literature references. We distinguish between the following three types of variability in time: variability of change over time (maintenance/evolution), multiple versions at a moment in time (configuration management), and binding over time (product derivation).

A. Variability of Linear Change over Time: Maintenance and Evolution

Meta-studies indicate that 50 to 90 percent of costs refer not to development of software products but to their changing [17]. As described in Section II, the popular definition of "variability in time" in [22] denoting the existence of different artifact versions at different times does not address this issue appropriately in the context of software product lines.

The discipline of software engineering providing comprehensive concepts, methods, and tools for changes is called *software maintenance* or *software evolution*. For this paper, we will not strictly distinguish between both terms. However, commonly, the term maintenance has a rather "reactive" connotation (i.e., corrective or adaptive maintenance according to [18]), whereas evolution tends to be used in a more "proactive" way (i.e., perfective or preventive maintenance in [18]). In the following we will use the more appropriate term, respectively.

Evolving a product line is much more complex as in the case of single systems, since the variability (in space) changes over time as well. Formalizing the different types of product line changes to ensure consistency is a big challenge specific to product line engineering. A taxonomy for the different types of product line evolution operators with a special focus on requirements level changes is given in [26], for example. A more empirical approach categorizing the findings of two case studies into an approach relating the reasons of product line changes to their architectural impact can be found in [29]. Finally, a framework for decentralized evolution of product line feature models is presented in [23]. It facilitates specifying explicitly which change operations are permitted on a feature model by defining allowed deviations with respect to a reference feature model.

B. Variability of Multiple Versions at a Moment in Time: Configuration Management

From a simplified point of view, software maintenance may be seen as a continuous flow of versioning over time, each new version invalidating its predecessor. In this form, it does not concern multiple versions of the same artifact at the same time. This is a task of configuration management.

Software configuration management (SCM) is often defined as a holistic discipline comprising tools, techniques, and processes for tracking and controlling everything related to versioning and changing of software-related artifacts (e.g., [20]). Version management tools such as SVN or ClearCase constitute the technical side of SCM. In this paper, we will use the term configuration management in a narrower sense: managing different versions of the same artifact throughout the software lifecycle, or, as stated informally in [20]: "eliminating the confusion and error brought about by the existence of different versions of artifacts". Thus, it deals with maintaining the integrity of products considering that they may be comprised of artifacts of different versions.

	Linear change over time	Multiple versions at a moment in time	Binding over time
Focus	Change as continuous flow	Managing artifact versions in parallel	Binding VPs as process in time
Off focus	Multiple artifact versions	Evolving artifacts	Evolution and versioning of VPs
Software engineering field	Maintenance/Evolution	Configuration management	Product derivation
Exemplary PL challenge	Consistent evolution of ViS	Consolidating versions and ViS	Time flexibility for binding ViS

PL: product line | VPs: variation points | ViS: variability in space

TABLE I
OVERVIEW OF THE THREE TYPES OF "VARIABILITY OVER TIME".

Once a product is delivered to a customer, keeping it in sync with the product line core assets is often not part of the contract or even feasible (e.g., due to hardware changes). However, the released products have to be maintained as well, for instance by bug-fixing (corrective maintenance) or for possible future update requests (perfective maintenance). Therefore, the configuration of the specific product (i.e., all its artifacts in their specific version) must on the one hand be frozen in time. On the other hand, the product must keep in contact with the evolving core asset base to support the product's maintenance.

Consolidating versioning and software product lines variability is an important research challenge of product line engineering. It has both been addressed in research prototypes by integrating product line functionality into version management tools (e.g., [31]) or vice versa (e.g., [32]).

1) Versions over Time and Continuity: According to Aoyama [1] evolution can be *continuous or discontinuous*. Continuous evolution corresponds to a set of requirement changes that is small enough, so that only little architectural reengineering is required. Above a certain threshold of changes, it is necessary to re-engineer the architecture, leading to substantial architectural and implementational differences. When the continuity is interrupted, a new product line generation arises and, for a certain period of time, both product lines have to be maintained in parallel. Major architectural reengineering leads to differences that are difficult to track. Dealing with those problems is a further challenge for configuration management.

C. Variability of Binding in Time: Product Derivation

A considerable amount of variability of a product line is planned early in the software product line lifecycle, during scoping [25]. The actual implementation of the variability (foreseeing and implementing variation points) can be performed at different moments in domain engineering, as well as the binding of these variation points to variants for product derivation in application engineering [4]. Various binding techniques ranging from coding time to run-time may be applied.

Although the term "variability in time" has been used in the context of variation point creation and binding, one might argue that it is a rather trivial observation that this is usually done in a process over time. However, we believe that it is possible to gain some remarkable insights if seen in a broader scope. Creation and binding of variability successively over time is also known as multi-level and staged configuration, respectively [8]. Such an approach is required, if it is necessary to model so-called "software ecosystems", this is when software product lines themselves pass through several distinct organizations until the variability is completely bound [3].

A further variability aspect is introduced if the variation points are designed in a way that it is possible to bind them at different stages (e.g., either at compile time or at run-time). Then, additional variability, so-called "timeline variability" [10] is introduced, leading to an even more complex product configuration over time.

D. Intermediate Summary

Table I summarizes the three aforementioned types of "variability in time". They differ in what is in and off their focus and in the fields of classical software engineering that address them. To indicate that "variability in time" has specific product-line–related challenges, we also mention one exemplary challenge tackled by recent product line research. Providing a complete list of all relevant challenges is beyond the scope of this paper.

IV. SCENARIOS

In the following, we apply the three different types of "variability in time" to describe several complex product line scenarios where they exhibit expressive and discriminatory power, giving evidence that they provide the right abstraction for talking about variability over time.

Scenario 1: Interaction of Product Line and Products. Figure 1 shows a product line scenario containing each of the types of "variability in time". The first type—linear change over time (maintenance)—develops in parallel with the time flow (horizontal axis). The vertical axis, in turn, corresponds to the binding of variability. Interaction of the upper product line evolution and the lower product maintenance corresponds to configuration management, which is the third type. It becomes necessary, for example, when a product bug-fix is fed back into the product line core assets (bottom-up), or, vice versa, a product receives a feature upgrade from the core asset base (top-down).

The figure may be interpreted as follows: For the version of the product line in 2009, binding of variability (the derivation of the product A and B) is done in a single step. The product line evolves and, in 2010, product C (V1) is derived. This product has to be maintained in parallel to the evolving core

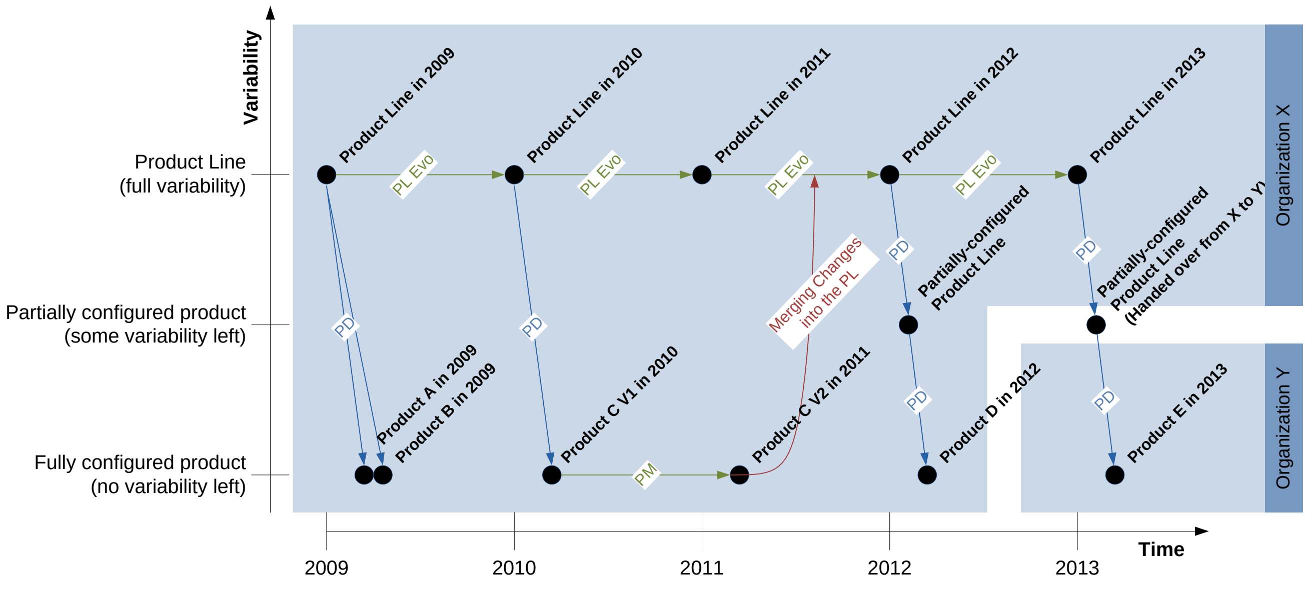

Fig. 1. Product line evolution scenario.

asset base. After creation of a maintenance release in 2011 C (V2), the adaptations are merged back into the product line, which is a configuration management task. Product D is bound in separate steps (staged configuration [8]). Finally, in 2013, product E results from staged configuration involving multiple parties. The supplier (organization X) hands over the partially-configured product line to its customer (organization Y), which binds the remaining variation points and completes product derivation. This is also referred to as *software ecosystems* [3].

Scenario 2: Interaction of Product Lines. The following scenario (Figure 2) illustrates continuity and discontinuity of product lines. Again, maintenance corresponds to the horizontal axis, configuration management, here not between products and their product line, but between distinct product lines, to the vertical axis. Product derivation is not shown in the figure. Subsequently, we distinguish between product line generations, releases, and revisions using version numbering (`<gen>.<rel>.<rev>`). This terminology is similarly used in other publications (e.g., [29], [1]) and can be found likewise in various software projects.

Starting in 2009 with development of the product line, version `n.m.0` is released in 2010, and, one year later in an updated revision (`n.m.1`). In 2011, a new release is split up from `n.m.2`. Both releases now have to be maintained in parallel. Usually, the older release then primarily experiences corrective and maybe adaptive maintenance, whereas the newer one keeps evolving further (perfective, preventive maintenance). Until the end of maintenance of release `m`, considerable interaction between `m` and `m+1` may occur (e.g., backporting of functionality), making complex configuration management (*"release management"*) tasks necessary between the product lines.

An even more profound break is the change of generations. According to [1] this is the case when the prospected changes are beyond a level of tolerance, so that it is necessary to reengineer the software architecture completely. This is the case starting from year 2012 where generation `n+1` is launched. The current state of the art in development and the lessons learnt from the previous generation build the input for creating a new product line architecture. Although the overall architecture is build from scratch, successful core assets of the prior generation may be ported (year 2014). Similarly as in a case of release change, maintenance operations performed on one product line generation may need to be transferred to the other one (e.g., forwardporting). This task is more challenging though, as, due to the differing product line architectures, the previously common core assets may have drifted apart considerably. This could be called *"product line generation management"*.

V. Product Line Evolution in Depth

In the previous sections, we identified three types of "variability over time" and applied them to two evolution scenarios. Now, it is necessary to further decompose and to identify the relevant tasks and aspects to be considered within each type. This will serve as a foundation to define a research agenda for giving appropriate support with tools and methods in the future. For the type of product derivation, there already exists substantial research, even on advanced topics (e.g., staged configuration, software supply chains and ecosystems) [8], [11], [3]. For this paper, we will only perform this decomposition for "product line evolution"; addressing versioning-related variability issues is out of scope for this paper and will be future work.

Product line evolution just is starting to get a foundation suitable for modeling. In [26] a set of evolution operations is defined, and [5] presents initial concepts for evolution-enabled

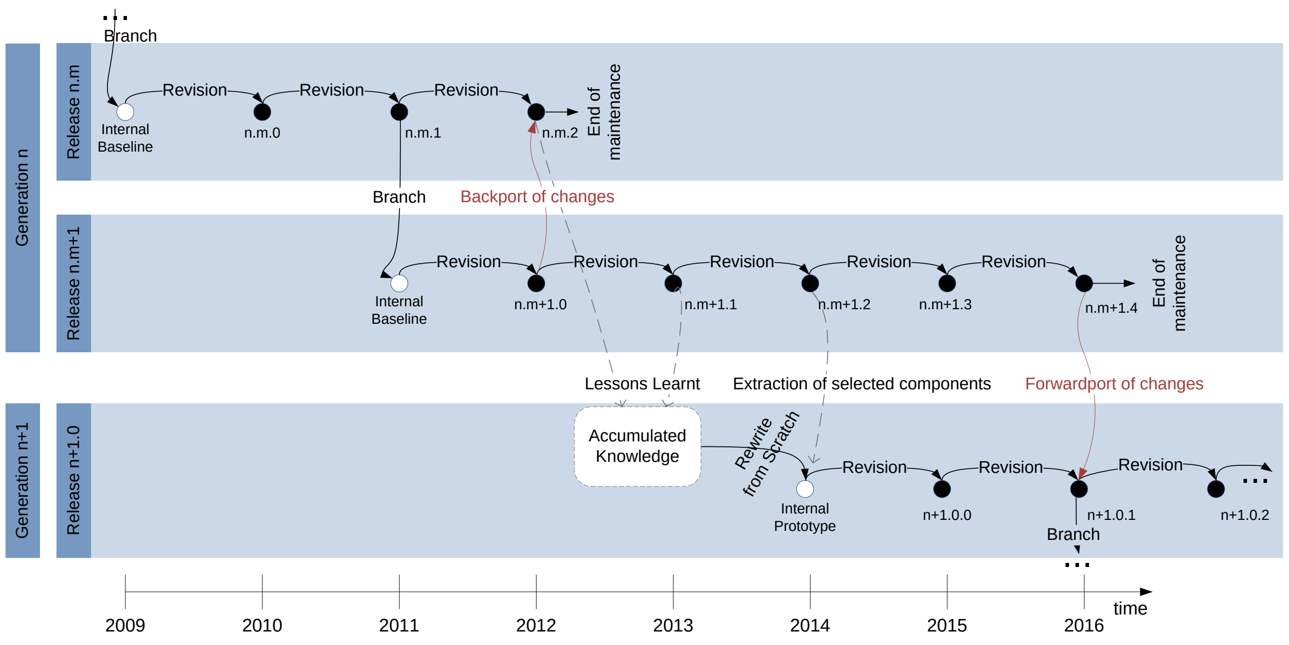

Fig. 2. Evolution paths of product lines (revisions, releases, and generations).

feature modeling. For defining the tasks and aspects relevant for product line evolution, we run through a complete iteration of what may be called the "evolution lifecycle". Depending on the intention of an observer, product line evolution has the following lifecycle phases: future *planning*, present *tracking*, present/past *analysis*, and *correction/realignment*. Figure 3 illustrates this.

Proactive Planning. Scoping [25] is a crucial task for a product line; its future development has to be planned. This means, in the first case, planning the features and their variability for a certain moment in the future. This is however not the only relevant planning task. Additionally, it is necessary to plan the steps, how to reach the future plan (product line evolution). Next to the (proactive) evolution of the product line core assets, the maintenance of released products and previous product line releases and generations must be planned and aligned.

Tracking. Tracking a product line includes both logging its current state and the changes performed on it. When considering the state of a product line as its current features and their dependencies, this can be done by using variability modeling (e.g., feature modeling with versioned features and dependencies) and change recording. This tracks the maintenance/evolution of the product line variability. However, changes to released products (bug-fixes, updates) should consequently also be tracked. This might be difficult, for example in the context of software ecosystems and supply chains [3], [11], as a product line supplier might not even know about the actual products derived from its base product line for the end customers.

Analysis. Given that the product line evolution has been appropriately planned and tracked, we envision three types of analyses: evolution state analysis, evolution step analysis, evolution conformance analysis.

- Analyzing the *evolution state* means checking the consistency at a moment in time. This may involve that each feature on model level must have assigned implementation artifacts, or that dependencies on feature level may not contradict to those on implementation level.
- An *evolution step* is valid if it transforms one valid evolution state to another valid one. For one concrete evolution state as input, this is easy to check (simply applying the evolution step and checking the result for consistency). However, it might be possible to prove that an evolution step produces always a valid result given a valid input.
- Analyzing *evolution conformance*, finally, means checking whether the current and past evolution states and the transition steps performed comply to the evolution how it has been planned. The result of the evolution conformance analysis may not only comprise the actual deviation but also recommendations about evolution steps to perform to get on track again.

Each of these analyses will usually comprise the following tasks: gathering the data, performing the analysis, and reporting of the outcomes, for example by visualization. This also holds true for the maintenance of released products, which must be analyzed as well.

Correction and Realignment. Based on the results of the analyses it might be necessary to correct and realign the development efforts or the product line evolution plan.

As we address a complete iteration of the evolution life-

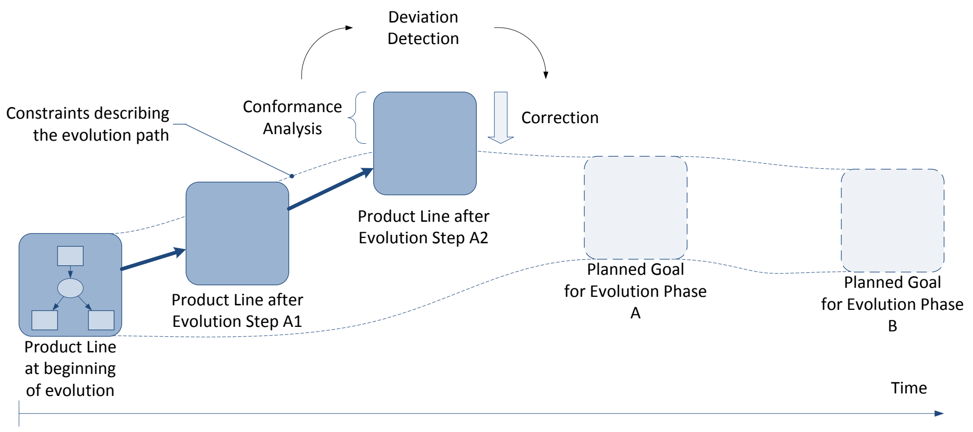
Deviation Detection
Constraints describing the evolution path
Conformance Analysis
Correction
Product Line after Evolution Step A2
Product Line after Evolution Step A1
Planned Goal for Evolution Phase A
Planned Goal for Evolution Phase B
Product Line at beginning of evolution
Time

Fig. 3. Planning, tracking, analysis, and realignment of evolution.

cycle, we are confident that we cover the crucial tasks and aspects to be considered when defining a research agenda for product line evolution in the future.

VI. Discussion

The aim of the paper is to perform first steps towards a common body of knowledge for variability over time. Validation of the results achieved is difficult, as a common terminology and understanding on concepts is missing. Addressing this problem best possible, we base our research both on existing work and exemplary validation. More precisely, our typification of "variability in time" is based on a literature survey and we referred to related work and assigned it to the corresponding types wherever appropriate. Second, we exemplarily validated the identified types by applying them to complex product line scenarios, where they exhibited expressiveness.

VII. Conclusion

Current authors address different time-related variability issues when talking about "variability in time". Without a common framework of concepts, it will be difficult to relate advancements to each other, finally hindering progress. In this paper, we performed the first steps to approach this problem. We surveyed the different uses and found out that their notions are actually orthogonal. By generalization, we received three types of "variability in time": variability of linear change over time (maintenance/evolution), multiple versions at a point in time (configuration management), and binding over time (product derivation). We applied the three types to two product line scenarios, where they exhibited expressive and discriminatory power, giving evidence for their usefulness as part of a body of knowledge for time-related variability issues.

As a first step towards further refinement, we extracted the necessary tasks and aspects relevant for the type maintenance/evolution, thereby covering a complete iteration of the "evolution lifecycle". These results can be used for defining a research agenda for developing tools and methods supporting product line evolution. Obviously this paper can only be seen as a first step towards a common body of knowledge, which has to be developed in vital discussion with the research community. However, we hope that our work stimulates further researchers to engage with the various dynamic properties of product lines and the challenges imposed by this fact.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero – the Irish Software Engineering Research Centre, http://www.lero.ie/.

We thank Christa Schwanninger for her valuable comments on a draft of this paper.

References

[1] M. Aoyama, "Continuous and discontinuous software evolution: aspects of software evolution across multiple product lines," in *Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE '01)*. New York, NY, USA: ACM Press, 2001, pp. 87–90.

[2] C. Atkinson, *Component-Based Product Line Engineering with UML*. Addison-Wesley, 2001.

[3] J. Bosch, "From software product lines to software ecosystems," in *Proceedings of the 13th Software Product Line Conference (SPLC '09)*, 2009, iSBN 978-0-9786956-2-0.

[4] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl, "Variability issues in software product lines," in *Proceedings of the 4th International Workshop on Software Product-Family Engineering*. Heidelberg, Germany: Springer-Verlag, 2001.

[5] G. Botterweck, A. Pleuss, A. Polzer, and S. Kowalewski, "Towards feature-driven planning of product-line evolution," in *Proceedings of the 1st International Workshop on Feature-Oriented Software Development (FOSD'09)*. New York, NY, USA: ACM Press, 2009.

[6] R. Capilla, A. Sánchez, and J. C. Dueñas, "An analysis of variability modeling and management tools for product line development," in *Proceedings of the Workshop on Software and Services Variability Management. Concepts, Models and Tools*, 2007.

[7] J. O. Coplien, "Multi-paradigm design," Dissertation, Vrije Universiteit Brussel, 2000.

[8] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[9] S. Deelstra, M. Sinnema, J. Nijhuis, and J. Bosch, "COSVAM: A technique for assessing software variability in software product families," in *Proceedings of the 20st IEEE International Conference on Software Maintainance (ICSM'04)*. Washington, DC, USA: IEEE Control Systems Magazine, 2004.

[10] E. Dolstra, G. Florijn, M. de Jonge, and E. Visser, “Capturing timeline variability with transparent configuration environments,” in *Proceedings of the International Workshop on Software Variability Management*, May 2003.

[11] H. Hartmann, T. Trew, and A. Matsinger, “Supplier independent feature modelling,” in *Proceedings of the 13th Software Product Line Conference (SPLC ’09)*, 2009, iSBN 978-0-9786956-2-0.

[12] A. Hubaux and A. Classen, “Taming time in software product lines,” University of Namur, Rue Grandgagnage, 21, B-5000 Namur, Belgium, Tech. Rep., July 2008.

[13] M. Jaring and J. Bosch, “Representing variability in software product lines: A case study,” in *Proceedings of the 2nd Software Product Line Conference (SPLC ’02)*. London, UK: Springer-Verlag, 2002, pp. 15–36.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, “Feature-oriented domain analysis (FODA) feasibility study,” Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, Tech. Rep., Nov. 1990.

[15] P. Knauber and J. Bosch, “Icse workshop on software variability management,” in *Proceedings of the 25th International Conference on Software Engineering (ICSE ’03)*, vol. 0. Los Alamitos, CA, USA: IEEE Control Systems Magazine, 2003, p. 779.

[16] P. Knauber and S. Thiel, “Session report on product issues in product family engineering,” in *Proceedings of the 4th International Workshop on Software Product-Family Engineering*. Heidelberg, Germany: Springer-Verlag, 2001.

[17] J. Koskinen, “Software maintenance costs. updated: Sept. 28, 2004.” http://users.jyu.fi/~koskinen/smcosts.htm, P.O. Box 35, 40014-Jyväskylä, Finland, 2004.

[18] B. P. Lientz and B. E. Swanson, *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

[19] D. Nestor, L. O’Malley, A. Quigley, E. Sikora, and S. Thiel, “Visualisation of variability in software product line engineering,” in *Proceedings of the 1st International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, 2007. [Online]. Available: http://www.vamos-workshop.net/2007/files/VAMOS07_0038_Paper_7.pdf

[20] L. Northrop and P. Clements, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[21] K. Pohl, G. Böckle, P. Clements, H. Obbink, and D. Rombach, Eds., *Product Family Development Seminar No. 01161*, April 2001.

[22] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[23] M.-O. Reiser and M. Weber, “Multi-level feature trees: A pragmatic approach to managing highly complex product families,” *Requirements Engineering*, vol. 12, no. 2, pp. 57–75, 2007.

[24] J. Savolainen and J. Kuusela, “Volatility analysis framework for product lines,” in *Proceedings of the 2001 Symposium on Software Reusability (SSR ’01)*. New York, NY, USA: ACM Press, 2001.

[25] K. Schmid, “A comprehensive product line scoping approach and its validation,” in *Proceedings of the 24th International Conference on Software Engineering (ICSE ’02)*. New York, NY, USA: ACM Press, 2002.

[26] K. Schmid and H. Eichelberger, “A requirements-based taxonomy of software product line evolution,” *Electronic Communication of the EASST*, vol. 8, 2007.

[27] M. Sinnema and S. Deelstra, “Classifying variability modeling techniques,” *Information & Software Technology*, vol. 49, no. 7, pp. 717–739, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2006.08.001

[28] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, “COVAMOF: A framework for modeling variability in software product families,” in *Proceedings of the 11th Software Product Line Conference (SPLC ’07)*. Heidelberg, Germany: Springer-Verlag, 2007.

[29] M. Svahnberg and J. Bosch, “Evolution in software product lines: Two cases,” *Journal of Software Maintenance*, vol. 11, no. 6, pp. 391–422, 1999.

[30] M. Svahnberg, J. van Gurp, and J. Bosch, “A taxonomy of variability realization techniques,” *Software - Practice and Experience*, vol. 35, no. 8, pp. 705–754, 2006.

[31] C. Thao, E. V. Munson, and T. N. Nguyen, “Software configuration management for product derivation in software product families,” in *Proceedings of the 15th Annual IEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems*. Washington, DC, USA: IEEE Control Systems Magazine, 2008, pp. 265–274.

[32] A. van der Hoek, “Design-time product line architectures for any-time variability,” *Science of Computer Programming. Special Issue on Software Variability Management*, vol. 53, no. 3, pp. 285–304, 2004.

[33] F. van der Linden, “Software product families in Europe: The Esaps & Café projects,” *IEEE Software*, vol. 19, no. 4, pp. 41–49, 2002.

[34] ——, “Applying open source software principles in product lines,” *UPGRADE – European Journal for the Informatics Professional*, vol. 10, no. 3, pp. 32–40, 2009.