

CHANGE MANAGEMENT PATTERNS IN SOFTWARE PRODUCT LINES

Establishing effective change management practices to prevent uncontrolled evolution of the product platform.

Software product lines (SPLs) undergo numerous changes to accommodate evolving needs of various market segments [1]. While some changes may apply to specific members, others permeate throughout the product line, often finding their way into the platform. In effectively managing evolution, product line development organizations are often faced with the following questions:

- Are there general patterns of changes that are common in product lines?
- If so, what change management practices can be used to effectively manage these patterns?

We investigate these questions through a case study in a product line development organization (see the sidebar on how we conducted our study), identify three interesting patterns of change management, and suggest effective practices for addressing them.

CHANGE MANAGEMENT IN PRODUCT LINES

The evolution of several variants along with the product platform poses several challenges in SPLs. Even sophisticated change management systems are inadequate for managing the complex, interdependent changes that are incorporated in different product variants [6]. Changes in customer requirements and errors detected in product variants may require changes to the platform itself. As all the members of the product line will be affected by changes incorporated in the platform, it is often allowed to evolve independently for different market segments such that only the product variants that are immediately affected by the change are tested and rebuilt. Thus, the platform is branched into a family of platforms. Left unchecked, this practice could potentially undermine the advantages of using a platform and lead to a fragmented product line.

Motivated by this problem, we offer recommendations to help understand and control the ramifications of changes in product lines. We present three commonly faced patterns of changes incorporated in product lines and change management practices that mitigate their adverse effects. Though several techniques have been used by the software engineering community to address these challenges, our recommendations are novel in that they are tailored to product line engineering. Techniques like feature modeling, proposed by the SPL community, only partly address the problem (see the sidebar on related research) [4]. The recommendations we present here encompass different operationalizations that are complementary to such feature-based techniques.

PATTERNS IN CHANGE MANAGEMENT

1. Interdependencies among changes in variants: As product variants evolve, changes incorporated in product variants become dependent on one another. We use a simplified example to understand the complexity of these dependencies. Consider a core product platform that is used across three product variants (see Figure 1). The three variants evolve into different versions due to specific changes—for example, version 1 of variant 1 is subject to two changes to derive version 2.¹ Some examples of such changes observed at SCMCo are shown in Table 1.

In Table 1, C3 refers to a change in an algorithm. It depends on C1, as it was significantly changed when C1 was incorporated. A customer who uses variant 3 also requests C3, in Version 3. Now, before implementing C3, (specific parts of) C1 on which

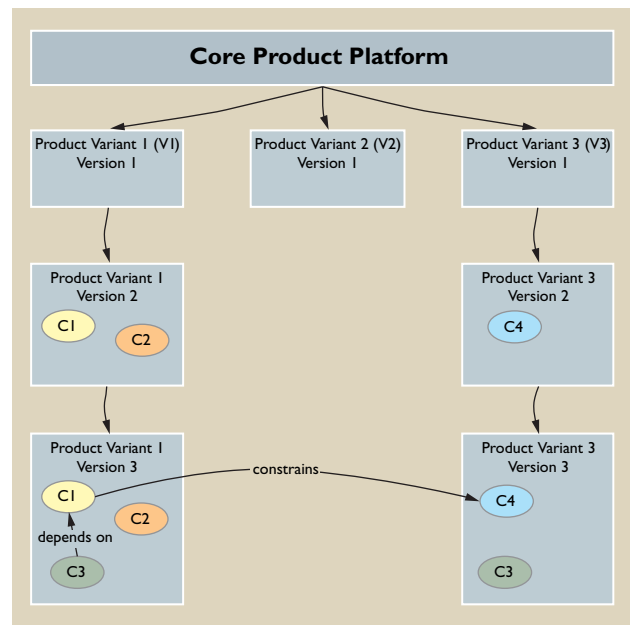


Figure 1. Intertwined changes in product variants.

C3 is dependent should also be implemented. However, C1 conflicts with or constrains C4, which is also part of variant 3.

This pattern (typically more complex than described here) is common in the product line at SCMCo. In summary, interdependencies among changes incorporated in different variants pose a problem in managing product line evolution.

HOW THE STUDY WAS CONDUCTED

We conducted a case study in an organization (hereafter referred to as SCMCo) that develops large-scale product lines of supply chain management systems. We studied the development and evolution of a family of warehouse management systems (WMSs). WMSs are used to manage the inventory sent to and from warehouses to fulfill orders received from customers. SCMCo has developed a platform that provides the common functionality for all the customer segments. Component variants are built on top of the platform to deliver customized versions of a WMS. Our study focused on gaining insights on the change management process followed major challenges involved in managing change, and practices used to address them. We conducted semi-structured interviews with six project managers and systems analysts. Data from these interviews was analyzed using open, axial, and selective coding techniques from the grounded theory method [12]. We also performed content analysis of several change requests documented by developers at SCMCo. Further, we examined the change management tools used by developers. Based on these analyses, we identified three important patterns of change. We also identified practices that effectively address the problems associated with these patterns. **C**

¹Although for simplicity, we consider C1 and C2 as simple changes, these are typically very complex.

2. *Increasing the degree of evolving variance:* Variation points in the platform are locations in the design that provide alternative ways of deriving product variants. Table 2 lists a few examples from SCMCo. The degree of variance refers to the number of variants that can be derived using the alternatives available at variation points. The degree of variance supported by a platform can be increased by increasing the number of alternatives supported at each variation point. Figure 2 shows that the

Change	Description
C1	This change is made in the conveyor system used to move products in a warehouse. The time taken for an item to travel from the entrance to a routing station in the warehouse depends on the conveyor system used. Before the item arrives at the routing station, the WMS should determine its destination so that it may be appropriately routed. When the conveyor system is modified with C1, it may be necessary to change the routing process as well so that the routing is determined before the item arrives at the routing station.
C2	This change involves modifications to the interface that supports communication between the warehouse system and the conveyor control system. Due to the changes in the conveyor system, the requirements for the interface were changed.
C3	This change involves a modification to the algorithm used for determining the location of items. This change depends on C1, as this algorithm was significantly changed as part of C1.
C4	This change involves modifications to the location algorithm to accommodate changes in the warehouse structure, and hence the location of various stations.

number of alternatives provided by three variation points increases as the product line evolves. However, a major factor that affects the degree of variance is the impact of changes incorporated in these variation points.

When variation point 3 is changed to introduce an additional alternative (support for a new labeling system), it affects the introduction of a new alternative at variation point 1 (support for a new material handling equipment) because the speed at which labels are prepared depends on the level of automation of material handling. Several constraints such as this that result from changes to variation points are shown in Figure 2. Also, changes made to variation points affect changes made to the product variants down the line. For example, the introduction of the new alternative in variation point 1 affects the way the changes C1 and C4 are implemented in variants 1 and 3 respectively (in Figure 1).

Figure 2. Impact of changes in degree of variance.

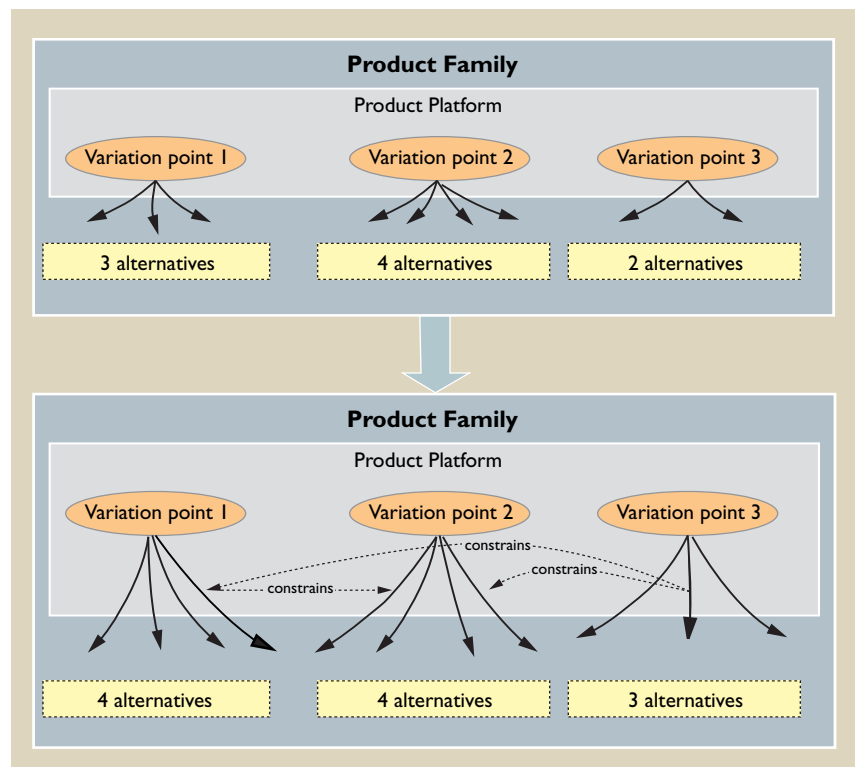
Increasing variance results in numerous constraints across variation points.

3. *Reinvented variations:* In SPLs, the alternatives incorporated in a variant may already exist in another variant. However, due to the boundaries between application engineers who work on different product variants, it is often difficult to leverage such existing implementations. For example, an alternative being implemented in V3 was already implemented in V1. Unfortunately, due to a fragmented application engineering process, the design team was not even aware of this solution, thus resulting in duplicated effort.

RECOMMENDATIONS FOR CHANGE MANAGEMENT

1. *Modularize changes and variation points:* Variation points should be designed to be independent of each other as much as possible and changes made to specific product variants should be isolated from other changes. Industry best practices such as design patterns and refactoring can be used to achieve modularization. Project managers should develop design guidelines in the form of a checklist of such practices that may be followed when implementing new variation points or

Table 1. Examples of changes.



Variation Point	Name	Description
1.	Material handling equipment	This variation point facilitates the use of different types of material handling equipment with the warehouse management system. The alternatives supported in this variation point differ in the level of automation in moving items within the warehouse.
2.	Order management system	This variation point provides support for various types of order management systems that need to communicate with the warehouse management system. Messages are sent to the material handling equipment to collect and store items that are required to fulfill an order.
3.	Labeling system	This variation point facilitates the use of different systems that are used in creating labels for items that arrive at the receiving docks in the warehouse.

Table 2. Examples of variation points.

making changes. This will help developers follow consistent practices across the product line and reduce interdependencies. For instance, ‘protected variations’, an analysis pattern proposed by Larman [7] (adapted for use with product lines) is an effective way to operationalize part of this guideline and reduce undesirable impacts of a variation on other design elements. Also, as a product variant evolves, dependencies between current and past changes may impact changes in other product variants (as described in Figure 1). Since such cases are not addressed by ‘protecting’ variations, each change in each variant that may evolve across both time and space must be isolated as much as possible from other changes. We suggest that modular design, a commonly acknowledged best practice in software engineering, when focused on variation points, will be very beneficial in SPLs.

2. Track the scope and life of variations: Since it is usually not possible to completely modularize variation points, it is essential to track their impact on other parts of the product platform. Similarly, it will be useful to track the changes made to a product variant as it evolves. This can be achieved by establishing traceability across various components of the product line. Traceability (the ability to describe and follow the life of any artifact in a development

process) has been widely used in complex system development as an effective aid for impact analysis. It is also well recognized that traceability practices should be tailored based on project characteristics [11]. Common traceability practices that involve linking requirements, design elements, and source code modules do not specifically recognize variation points and product variants—concepts that are specific to product lines. Therefore, project managers should develop a customized traceability practice that specifically establishes traces to variation points, changes incorporated in product variants and interdependencies among them. Reference traceability models [11] can be specialized to represent this information in the form of traceability matrices or networks. We suggest that variation points should be considered as central to traceability practice in SPLs to effectively aid in impact analysis [9].

3. Facilitate reuse based on knowledge sharing: Development and change management practices should provide easy access to knowledge about variation points in the product line. This will help various stakeholders involved in the process (domain engineers, application engineers, sales, marketing and so forth) easily understand and exploit the existing variations when new requirements for variations are identified. Specifically, access to rationale behind variation points and the alternatives supported at each variation point will help developers identify opportunities for reusing existing designs and implementations. Several models and tools for acquiring and reusing design rationale that have been successfully applied in complex software development [8]

Interdependencies among changes incorporated in different variants
pose a problem in managing product line evolution.

Issue Raised by Pattern	Recommended Practice	Description
Interdependencies among changes made to variants	Modularize changes and variation points	Modularization will help minimize interdependencies and therefore minimize the effect of changes on other members or parts of the product line.
Increasing the degree of evolving variance	Track scope and life of variations	As the number of variation points and alternatives increase, tracking the scope and life of these variations helps developers to understand and exploit the alternatives available.
Reinvented variations	Facilitate reuse based on knowledge sharing	Variations can be reused across variants if developers can readily identify existing implementations of similar variations.

*Although the mapping between the patterns and practices are many to many, we show only the most important mappings.

may be adapted for managing knowledge about variations (see Table 3).

CONCLUSION

If left uncontrolled, changes incorporated in product lines will become very expensive to manage. The change management practices recommended here are fundamental to effectively controlling the evolution of SPLs. In general, a scalable product line management strategy that increases variety while emphasizing effective change management practices is essential to prevent uncontrolled evolution of the product platform into families of platforms. Though the guidelines have been carefully developed based

RELATED RESEARCH



Recent research has developed guidelines and identified best practices for several product line practice areas. The Software Engineering Institute's Product Line Practice Initiative and Fraunhofer Institute of Experimental Software Engineering have developed several useful frameworks (see [3, 10]). Feature-oriented techniques that identify prominent features in a set of products in a particular domain are good candidates for managing change. Feature models can be used to represent dependencies across features in a product line and analyze the impact of changes on features. Feature-Oriented Domain Analysis (FODA) [2, 4], which uses several models such as context models and architectural models besides feature models, can also facilitate impact analysis. Feature-Oriented Reuse Method (FORM), an extension of FODA, addresses design issues from a marketing perspective [5]. However, FODA is limited to managing domain- and feature-related dependencies and requires complementary practices to manage dependencies at the detailed design and implementation phases. Though the feature-oriented approaches are effective in supporting change management in SPLs at the level of features, many changes required during the evolution of product variants are not necessarily at this level of granularity. Examples include cross-cutting changes or implementation-oriented changes (that are not feature-oriented) as illustrated in Tables 1 and 2. In such cases, the recommendations presented here can be operationalized to complement feature-oriented methods. 

Table 3. Mapping practices to patterns.*

on data from a case study, similar studies in other domains will help generalize the findings. Future research on developing specific operationalizations that are unique to product line engineering is ongoing. The development of tools that are embedded in

work practices is essential for successful operationalization of our recommendations. 

REFERENCES

1. Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Upper Saddle River, NJ, 2002.
2. Cohen, S.G., Stanley, Jr., J.L., Peterson, A.S., and Krut, Jr., R.W. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain and Appendices A-I*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1991.
3. *A Framework for Software Product Line Practice: Bibliography*; www.sei.cmu.edu/productlines/frame_report/frame_bib.htm#Kang98.
4. Kang, K.C., Cohon, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S. *Feature-Oriented Domain Analysis (FODA): Feasibility Study*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.
5. Kang, K.C., Lee, J., and Donohoe, P. Feature-oriented product line engineering. *IEEE Software* 19, 4 (2002).
6. Krueger, C.W. Variation management for software product lines. In *Proceedings of the Second Product Line Conference (SPLC 2)*, San Diego, CA, 2002.
7. Larman, C. *Applying UML and Patterns*. Prentice Hall, 2002.
8. Lee, J. Designing rationale systems: Understanding the issues. *IEEE Expert* 12, 3 (1997), 78–85.
9. Mohan, K. Managing variability in software product families: A traceability-based approach. Ph.D. dissertation, Department of Computer Information Systems, Georgia State University, Atlanta, GA, 2003.
10. Publications on product line software engineering from Fraunhofer IESE; www.iese.fraunhofer.de/PuLSE/Publications/.
11. Ramesh, B. and Jarke, M. Towards reference models for requirements traceability. *IEEE Transactions on Software Engineering* 27, 1 (2001), 58–93.
12. Strauss, A. and Corbin, J. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage Publications, Newbury Park, CA, 1990.

KANNAN MOHAN (kannan_mohan@baruch.cuny.edu)

is an assistant professor in the Department of Computer Information Systems at the Zicklin School of Business, Baruch College, New York.

BALASUBRAMANIAM RAMESH (bramesh@cis.gsu.edu) is a professor in the Department of Computer Information Systems at Georgia State University in Atlanta.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.