Managing variability requirements and variation points for Software Product Lines
A Case Study

Jonas Andersson
Lund University
jonas.t.andersson@gmail.com

Samuel Nygren
Lund University
samuel.nygren@gmail.com

Abstract

In this article we are presenting an evaluation of the product derivation process for the product line approach at Sony Ericsson Mobile Communications. The paper examines the weaknesses in the current derivation process and the gaps between the requirements and the configuration of products. We also present improvements to the current process for product configuration. This was done to connect the configuration to the initial requirements and to be able to configure products in a more efficient way, clarifying the responsibilities among the different stakeholders.

1. Introduction

How do you manage to configure over 40 products a year with different hardware and branding from the same software base with about ten million lines of code? For each product the customers have different customization and demands. Sony Ericsson does this today, but they are not quite sure how their products are actually derived from their product line.

In this article we summarize the findings from a case study made at Sony Ericsson, where the current process for configuring products is evaluated and suitable improvements are proposed.

In chapter one we give an introduction to the current problems in the configuration process. Chapter two introduce the reader to the subjects of variability and software product lines. The third chapter contains a short description of Sony Ericsson and their way of managing software product lines today. Chapter four and five describes the actual evaluation containing weaknesses in the derivation process and an improvement proposal intended to solve these. Finally a conclusion is presented together with reflections regarding future work.

1.1. Current problems

In the recent years the number of products being released every year by Sony Ericsson have increased rapidly. Today, they manage to release competitive products suited for the market needs but they lack a clear overview of how the products are derived and who is responsible for performing their configuration. The rise of new products has also resulted in an escalating number of variation points available in the software. These have become difficult for the configuration managers to overview and manage. Partly because they are too many, and partly because the granularity is too detailed which makes it difficult to assign the correct values to each variation point.

Since the configuration process is incomplete there is no clear connection between the requirements and the configuration parameters. This aggravates the possibilities to verify the configuration.

2. Theory

Today, it is common that software is produced as a family in which several variant are available. For example, embedded systems might exist in different variants for different CPU’s or a program calculating salary may have different variants for different laws [1]. In other words, software variants are often used to meet the needs for new markets to be able to sell more copies of a system [7].

Managing these variants is not easy and requires that commonalities and differences between different variants is being captured. If the variants can be managed and controlled, it has been proved to increase the productivity greatly [1]. According to Babich this addresses the need to store the parts of the software that are common or differ in a way that avoids double maintenance problems. This can be accomplished through software reuse between different variants and by controlling this reuse through configuration management.
2.1. Variability

To manage different variants is one of the most complex tasks in software configuration management. Variants can be seen as parallel versions, adding an extra dimension to the version tree of a version controlled item. Managing parallel versions are supported by most version control systems by the use of branches, but this concept is most often too simple to efficiently manage variants. [7]

To be able to achieve variability it is important to identify possible parts of the software where variability can occur. One way to do this is by using a feature graph. A feature graph shows the different features between the products which is helpful when trying to identify variations [9]. Each point in the software where different variation implementations can be used is called a variation point (see figure 1).

![Figure 1. Example of a variation point. The door lock exists in two implementations, Keypad and Fingerprint scanner. The door lock is a variation point and the Keypad is a variant implementation. [8]](image)

There are many ways of implementing a variation point. Depending on which technique that is used, the time when the variant implementation is chosen can vary. This is called the binding time and refers to when the implementation is bound to the product, replacing the variation point. After the variation point is bound it can no longer vary and when all variation points are bound the product is completely derived. The binding of a variation point can be divided into external binding and internal binding. External binding means that the system is not aware of the variation point and does not need to consider the variability. Internal binding means that the system is aware of the variation point and choose which variant feature to populate with. This means that external binding is done outside of the system, by a software architect or developer. Typically this is done before the system is compiled. Internal binding is however done by the system, by e.g. analyzing the value of a configuration variable and choosing the correct functionality using a simple if-statement. [9]

Introducing a variant feature at an early stage and binding it at a late stage results in a life cycle where the software can vary over a long time. This means that designers, developers and testers need to manage all variants, making the process very costly. This is visualized in figure 2.

![Figure 2. Number of possible systems that needs to be managed, depending on when the variation point is bound. [6]](image)

2.2. Software Product Lines

A common way of organizing the variability within product families is to use a Software Product Line approach. A product line allows for common functionality to be shared between products by the reuse of artifacts [9]. To be able to do this efficiently, the organization need to plan and be proactive [2].

2.2.1. Scoping

When the organization has decided to develop a product line, the first step is to determine the scope of products and features. In this activity the products that are suitable for a product-line are selected. This may not include all products, since some products may have conflicts between features that makes them unsuitable for the same product-line. The second step is to determine which of the features in the selected products that are candidates for the product-line. By identifying the features available in each product and creating a matrix showing the products in one dimension and the features in one dimension, the candidate features can be identified as those present in most products (see figure 3). [2]
Deelstra et al. [4] divides the scope of reuse into four levels. At the Standardized infrastructure-level, only the way products are built is reused. This can be aspects like operating system, database system and development tools. The Platform-level denotes that a set of artifacts that capture the commonalities between the products are reused. When the reuse is extended to functionality that is shared by a sufficiently large subset of products the Software Product Line-level is reached. At this level variation points are added to support the differentiation. The highest level of reuse is called Configurable product family which means that all common and different characteristics are captured in the same configurable product family. This means that products are constructed from a subset of the artifacts and that most development effort is focused on the product line development rather than product development. Product derivation is often automated and applications are created from a configuration specification. This level can also be seen as a maximalist approach of a Software Product Line. The effort allocation for product line engineering and product engineering for the different approaches can be seen in figure 4.

In a product line approach requirements elicitation for a product mainly consists of deriving them from the platform requirements rather than defining new requirements. Nevertheless, differences between the platform requirements and the product requirements and their dependencies must be carefully analyzed. When doing this, several problems can occur. Product features can overlap existing platform features and needs to be broken out or redefined. A platform feature that other features depends on can be removed, or a product feature can be in conflict with a platform feature. [8]

### 2.2.3. Benefits

Using a product line approach can result in many benefits in an organization. From an organizational point of view it can decrease time-to-market, help keeping up with unpredicted growth of software, improve the quality of the product [5] and fulfill reuse goals in the organization [3]. The more complicated a system gets, the more the cost can be reduced if performance, distribution and reliability issues can be solved in a satisfying way. A product line approach also leads to benefits such as avoiding integration phases, fewer delays and finally it will liberate more time to explore new technology. [3]

### 3. Presentation of Sony Ericsson

Sony Ericsson Mobile Communication (SEMC) is one of the leading companies within the mobile phone industry, providing mobile multimedia devices including feature rich phones, accessories and PC cards. The company was founded in 2001 by Ericsson and Sony.

By using a product line approach the number of products have increased lately and are being released in many different variants sharing major parts of the software. The product line actually consists of several product lines where each of them cover different technologies and markets. There is e.g. one low-end, one mid-range and one high-end product line within the company. The mid-range product line

---

### Figure 3. Feature/Product matrix for selection of features for the Software Product Line

| Feature/Product | P1 | P2 | P3 |...
|-----------------|----|----|----|------
| F1              |    | X  |    |      |
| F2              |    | X  | X  |      |
| F3              | X  | X  |    |      |
| F4              | X  |    | X  |      |

---

### Figure 4. Effort allocation to product and product line development for different levels of reuse. [4]
is based on an in-house developed software platform supporting a range of different product variants in the segment. This product line is the one we have been focusing on in this article.

The product lines are quite separated in the implementation and test phases, but they still share the same requirements to some extent and therefore share the same requirement base and requirements management system.

These product lines allow SEMC to produce many product variants with short lead times by using the same software artifacts, specially configured for each produced product. The configuration is done both for differentiation purposes and for hardware specifics that vary between products. Differentiation imply advanced feature sets and branding that is specific to a product or product type. Most of the configuration though, depends on hardware characteristics of the products, like available hardware features including e.g. WLAN and GPS but also variants of common hardware like screen resolution and camera properties.

4. Evaluation of current process

4.1. Method

The evaluation of the current derivation process was performed by studying documents and performing interviews with key personnel at the company. The documentation was acquired from the internal document management system at the company and mainly consisted of process descriptions, specifications and design documents. This system was also explored to understand how variability was defined at requirement level.

4.2. Results

![Figure 5. Project organization. The platform project creates the base for the different product projects.](image)

The development process is organized in two kinds of sub processes where one supplies the platform and the other creates a product. There is one platform project, one main product project and several sub product projects, where the platform project uses the full development process and the product projects uses a scaled down version (figure 5). The effort allocation is on the level of Configurable product family, as defined by Deelstra et al. [4]. This means that most of the effort lies in the platform project or product line engineering.

The requirements process at SEMC is mainly focusing on software requirements, and there are several areas affecting it. In this case aspects mostly from customers, standards, legal issues and suppliers need to be taken into account. The products must be differentiated and customers must be able to differentiate the same product. The requirements engineers are responsible for a so called technology roadmap, which serve as a source to the requirements in the platform project. This roadmap includes requirements known from product planning, associated areas, previous projects and customers. The requirements engineering process has lately been extended with new tools to manage variability more easily. The requirements are grouped into sets of requirements called Configuration Packages (CP) (similar to 2.2.2). These are grouped to form a logical unit of functionality, as described in [2]. Every requirement is associated with at least one CP, which are supposed to be the atomic means for differentiating products.

![Configuration Package Module](image)

![Requirements module](image)

![Figure 6. Organization of Configuration Packages and requirements. Each requirement is connected to a Configuration Package.](image)

A CP could be seen as a module, which together with other modules in the platform can be used to create a product. Naturally these modules have dependencies between each other, as described by Bosch [2], and some of them are mutually exclusive. For example, "FM Radio Advanced" might have dependencies to "FM Radio Basic", requiring the later CP to be chosen if the first CP is used (see figure 6). There may also be two different variants of a music player, which are equal and mutually exclusive, meaning that a product cannot have both CPs.
By having this information separated into Configuration Packages but still connected to each requirement through links, the model is very similar to the Orthogonal Variability Model proposed by Pohl et al [8]. With the capabilities of defining dependencies and constraints between Configuration Packages and the traceability links to the requirements, the resemblances are remarkable. The major difference between Pohl’s proposal and SEMC’s model is that the Configuration Package model is not yet used outside of the requirements management process.

Figure 7. The current product configuration is managed through an iterative cycle.

During the investigation different stakeholders were interviewed about their experiences regarding the current way of managing the configuration. This showed a big lack of knowledge within the company of how products are configured and who is responsible for doing it. The response was overall unanimous and everybody agreed that a consistent way of following the process all the way from requirements to the product derivation phase was missing. When the data collected from these interviews were gathered it came out that a product was configured through a cyclic iteration. The configuration managers delivered builds to testers who discovered faulty configuration. Testers, in their turn, deliver bug reports for these errors to developers who correct the errors and deliver the changes back to CM. This is visualized in figure 7.

During the investigation different stakeholders were interviewed about their experiences regarding the current way of managing the configuration. This showed a big lack of knowledge within the company of how products are configured and who is responsible for doing it. The response was overall unanimous and everybody agreed that a consistent way of following the process all the way from requirements to the product derivation phase was missing. When the data collected from these interviews were gathered it came out that a product was configured through a cyclic iteration. The configuration managers delivered builds to testers who discovered faulty configuration. Testers, in their turn, deliver bug reports for these errors to developers who correct the errors and deliver the changes back to CM. This is visualized in figure 7.

The commonalities in the different stakeholders perspectives on how products are created is visualized in figure 8. In the Provide Platform process, the roadmap and customer requirements are transformed into platform requirements and a scope, listing which requirements that are to be implemented in the project. The developers refine the requirements and create design documents which are used in the development of the actual software. During the implementation variation points are created by the developers. When new products are created CM copies the configuration of the most similar previous product. The variation points are then changed by the developers in the Provide Product process according to input from the product project and testers.

Figure 8. Process with focus on variability and product derivation

The architecture consists of variation points with both external and internal binding time. The external binding is realized as conditional compilation constructs, while the internal binding is resolved at runtime using variables read from a file at startup.

The interviews showed that variability is today most often not communicated to the developers through the requirements. This information is instead communicated verbally and is not documented. Because the platform is created before the products are derived, this information is often not available during platform implementation, resulting in the proactive creation of variation points based on assumptions. This has lead to a great amount of variation points and fragmented source code (collected metrics showed that there are approximately 30 variation points per variation parameter). From the interviews information about the reasons behind the great amount of variation points was captured:

**Assumptions** Developers anticipate future requests from customers and therefore create variation points proactively, based on assumptions.

**Ease of creation** Variation points are created because it requires less effort than the use of internal binding, which requires communication and approval from other teams.

**Version Control** Sometimes variation points are created to be able to remove the functionality for maintenance releases of previous products, instead of using branches.

**Integration** When different development groups have dependencies between their modules, variation points are created to remove the functionality until the other group have delivered their work.

**Debuging** Variation points are declared for debugging purposes in the configuration. These variation points are never used for the configuration of products, but is still present in the configuration file.
License costs Some parts of the software are related to license costs, such as e.g. codecs. It is therefore desirable to be able to exclude these parts.

Customer specifics Customers sometimes have special variants of each product which contains customer specific functionality. This functionality is to some extent controlled using conditional compilation. As mentioned before, this is mainly done because it is easier to achieve than to do it through internal binding.

Product specifics Sometimes the requirements states that the feature should be configurable, which makes it’s obvious that a variation point is needed.

The variation points are controlled using a configuration file which states the different values of the parameters for all products. The changes to the configuration were analyzed and distributed over the milestones of the project. This is visualized in figure 9. Over 60% of the changes to the configuration were made after the software was released to the product testers. This supports the conclusion of the iterative process described earlier.

- Unique and undocumented knowledge possessed by individuals.
- Inability to verify the configuration of a product.

Currently these problems are tackled by the use of unofficial documents specifying the product characteristics for both hardware and software. The document is informal and neither reviewed nor approved. Yet, it is used throughout the organization to fill the hole of the nonexistent product specifications. The fact that the document is not reviewed makes the content of it very unreliable. Because it is not part of any process, it is maybe not updated regularly as decisions are made. When the information is used by developers and testers this may result in to some extent different products than was originally planned.

4.3. Updated problem perspective

In the presentation of the current problems a clear overview of the product derivation and configuration process was missing. The number of variation points were very high and it was very hard to configure new products, especially since there were no clear connection between the requirements and the configuration. As a result from this case study, we now know that products are created in an iterative manner, with unclear responsibilities. There are no ultimate product configuration specification and no defined process for product derivation. The lack of definition and ownership for the product derivation process results in absence of strategic goals and long term improvements. Because the variation points cannot be traced back to the requirements, a product configuration cannot be verified against them.

5. Proposed improvements

There are two major issues with the current process discussed in the previous chapter. The first one concerns the lack of exhaustive specifications. The second issue concerns the granularity of the configuration parameters, resulting in the configuration managers configuring low level features rather than products and developers configuring products rather than low level features. To cope with these problems we propose to introduce an abstraction layer in the configuration interface allowing a clear separation of product configuration and feature configuration. This will result in a clear and unambiguous responsibility distribution. We also propose that the new product specifications based on Configuration Packages will be the one source for product differentiation, which will be used by all stakeholders in the development process, from product planners to developers and testers.
5.1. Abstraction layer

In figure 10 the current structure and the proposed abstraction layer structure are visualized. In the current structure the configuration file contains all detailed feature configuration on a very low level for every defined product. The configuration parameters are read in the software modules which are also specified in the configuration file. This file is edited by both CM and developers and because of its size and granularity it is vulnerable and subject to merge conflicts. In the proposed structure a new abstraction layer, CP-Conf (figure 10), is introduced between the product configuration interface and the software modules. The low level configuration is moved into this layer and a high level product configuration based on Configuration Packages are used on the product configuration level. This clearly separates the responsibilities between the developers and Product CM. The developers naturally becomes responsible for the CP-Conf-layer and it’s associated modules. CM is only responsible for the high level product configuration.

5.2. Distribution of variability specification

The product specification is associated with the extension of Configuration Packages for the requirements engineering process. This specification is managed in the requirement management system and specifies the products based on Configuration Packages.

As mentioned earlier, many stakeholders are in need of a clear product specification in their work. By using these product specifications throughout the organization, all stakeholders can refer to the same specification in their work and communication with others. By having these documents controlled and managed, their content will be reviewed and updated in a correct way. Because the specifications are based on Configuration Packages, they are very lightweight and consists only of a “Yes” or “No” for each package, plus a list of hardware characteristic values.

5.3. Benefits

By grouping the requirements into Configuration Packages traceability can be established between the requirements and the parameters in the configuration files. This facilitate the possibilities to verify the configuration against the requirements. The new division into atomic Configuration Packages also leads to a clearer overview of the available variability which ease configuration of new products for Product Planners. This grouping also enables the possibilities for automatic generation of the high level configuration from product specifications based on Configuration Packages in the requirements management system.

We propose that the responsibilities for the different roles and actions are stated in a more clear way. Division of the responsibilities can be made in the following way:

- Configuration Packages are defined and owned by the Requirements Engineers.
- A development group is responsible for one or several Configuration Packages and to handle the detailed configuration inside them.
- Product Planners are responsible for specifying new products using available Configuration Packages.
- Configuration Managers are responsible for managing the configuration of a product according to the Product Specification.

As mentioned earlier, time can be saved by enable access to an ultimate variability specification that lists the available configuration possibilities. The use of such a document can ease the work for most stakeholders in their daily work and avoid effort spent on redundant work and on detective work within the company.

5.4. Evaluation of proposal

The proposal was evaluated with two different development groups at the company. The current configuration was transformed into the proposed structure using the defined Configuration Packages. This was done without any problems and the needed changes in these two cases consisted of adding an extra CP for near future needs and some dependencies between CPs that were missing.

An evaluation form was sent out to the members of the teams. The result of this evaluation showed that the proposal is likely to improve the efficiency (3.9/5), will not affect the stakeholders work negatively (4.0/5) and is overall a better solution than the current way of working (4.0/5).

In interviews with CM-strategists the proposal received very positive responses. By connecting the configuration to the requirements the process will be more manageable and
clear and the product derivation process can be defined. It allows for verification of the configuration and possibilities to automate much of the product derivation process. The considerations regarded the need for a streamlined process for managing Configuration Packages, covering the possible exceptions and late changes to the configuration.

### 5.5. Risks and costs

The greatest risk with the proposal is that the Configuration Packages defined in the requirements engineering process will not match the variability needed in the software. This is tackled by having clearly documented processes for the creation and management of Configuration Packages. Such processes are today defined in the requirements engineering process.

There is also a risk that the configuration needs to be changed very quickly, in a way that is not supported by the defined Configuration Packages. By ensuring that the process for creating new Configuration Packages is streamlined, this can be handled. Late and immediate changes to the configuration will of course be associated with great risks since there may not be time to retest the product as needed. From this perspective, it is healthy to restrict these kinds of changes to the configuration.

Another potential risk is the reluctance from different stakeholders. The developers are the largest group affected by the proposal, but the evaluation showed that they are overall positive to it.

The costs of introducing the proposal are mostly associated with education of the stakeholders. From the evaluation we see that the question regarding how easy the proposal is to understand scores 3.8 in average, which is good considering that the participants got a half page of text and a figure as description.

### 6. Conclusion

In the investigation it was found that new products evolve by copying the configuration of the most similar previous product and by iteratively configuring the product in a cycle between developers, CM and test. Currently, the variability is not clearly documented, the responsibilities are unclear and there is no traceability from requirements to the configuration possibilities in the product line. These aspects affect the possibilities to verify the configuration and the effort spent on product configuration.

These conclusions have resulted in an improvement proposal consisting of an abstraction layer in the configuration interface. We have also proposed Configuration Packages to lie as a base for the Product Specification to clarify the configuration and to ensure that everyone within the organization can work in a consistent way. We expect this proposal to reduce the effort spent on iterative configuration and believe that it will ensure a higher product quality by the ease of verification, by stating clear stakeholder responsibilities and by making the information concerning variability more accessible within the organization.

It has been stated by several previous investigations that the use of a software product lines approach helps to improve product quality and reduces the time spent developing them [2]. Nevertheless it is critical to manage a product line and its variation points in a consistent and well documented way. To be able to do this, the Configuration Packages must only reflect the current needs for variability and never be created for proactive purposes that deviant from the initial requirements.

The evaluation of the change simulations performed together with development teams showed no major obstacles and the developers were coherently positive to the proposal. However, education is of high importance to convince stakeholders of the gains achieved following the new way of working in a consistent way and make them aware of the responsibilities for each stakeholder.

The contributions of this thesis includes a detailed investigation on product derivation from a large software product line, the problems associated with managing it and finally an improvement proposal which has been evaluated.

### 7 Future work and research

#### 7.1 Future work at SEMC

The thesis suggests how the variability can be managed and connected to the requirements, but it does not fully specify a process for product derivation since this would be of less general interest. Therefore SEMC need to define a process for managing Configuration Packages at the configuration level and deriving products from the specifications available.

When the changes suggested in this thesis are further established and a process is followed consistently, tools can be introduced to ease the management of the process. As we see things, these possibilities concerns tools used to generate configuration directly from the requirements management system and an interface connected to the requirements management system used to ease the choice of Configuration Packages for different products.

Other improvement areas of the overall configuration is to go from static configuration to more dynamic configuration (binding at an later stage), as far as possible. This transition already constitutes one of the ambitions by the software architects at SEMC.
7.2 Future research

There is already a large amount of literature concerning Software Product Lines and since the companies using a product line approach are relatively large, more detailed research may not be of general interest. Case studies like this one can however help companies to identify problems similar to their own and get inspiration from the solutions presented. Therefore it is important that the case studies are detailed enough to cover all problems that might be of interest to other companies.

The lack of tool support for product derivation was mentioned in some case studies, but again it is hard to use off-the-shelf products for this kind of activity in a large organization with an existing product line that is not adapted to the structure of the tool.

Future research in this area, possibly based on case studies like this, will probably be of general interest.

References