A Method for Software Product Platform Design Based on Features

Hamad Alsawalqah
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
ellissar@kaist.ac.kr

Sungwon Kang
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
sungwon.kang@kaist.ac.kr

Danhyung Lee
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
danlee@kaist.ac.kr

ABSTRACT
Due to the increased competition and the advent of mass customization, software firms are applying the Software Product Line Engineering (SPLE) approach to provide product variety in a cost-effective manner. Although the key to designing a successful software product family is the product platform, yet there is lack of measures and methods that are useful to optimize the product platform design. This paper proposes a method to provide decision support to determine the optimized product platform design. The method targets at identifying the optimized product platform design in order to maximize the cost savings and the amount of commonality while meeting the goals and needs of the envisioned customers' segments. It generates, validates, and evaluates alternative product platform designs while considering market concerns (e.g., customer preferences) and technical product platform concerns (e.g., decisions regarding shared features, economic benefit). We demonstrate its applicability with an example of platform design problem in smart phones domain.

Categories and Subject Descriptors
D.3.3 [**Reusable Software**]: Domain engineering

General Terms
Management, Measurement, Economics

Keywords
Software product line; product platform design; commonality index; Kano scheme.

1 INTRODUCTION
In Software Product line Engineering (SPLE), systems are built from reusable platform common to the whole family and of specific parts extending it [1]. The key to designing a successful product family is the product platform [2]. The product platform is the common basis of all individual products within a product family from which these products can be derived. Implementing platform commonality has been well recognized as an effective means to provide high-quality products at low costs and short time to market [1, 3, 4, 5]. Increasing the amount of commonality can decreases the effort required in design for flexibility [3]. While commonality can offer many advantages for a company, too much commonality within a product family can have major drawbacks. For example, commonality can diminish the ability to match varying customer requirements and hides the differences among different software products. Without considering the benefits of the variability, one easily overestimates the need for commonality and, thus, ignores the essence of the product line approach. Moreover, too much variability minimizes the benefits of the product line by increasing cost and complexity [3, 4, 5]. Hence, the commonality decision, which is the extent to which commonality should be used within product family, is fundamentally important for Software Product Line (SPL) success. The commonality decision consists in identifying the set of features that should be implemented either as full commonality (common among all the products in the family) or partial commonality (common among some products), the exact number of features' variants to use, the assignment of variants to the individual products, and the set of features that should be uniquely developed for each product in a SPL. As a note on terminology, in this paper we use the term product platform design to refer to a commonality decision.

Most frameworks and methodologies for SPL development consider product line scoping as a critical activity. This is because during scoping a number of crucial decisions for SPL success take place. Among these decisions is the commonality decision. Although a number of methods and techniques for SPL scoping have been developed during the past decade [4, 6, 7, 8, 9, 10, 11, 12, 13], yet there is no explicit use of automated (computer-based) optimization approaches that connect business and technical product platform concerns for solving the commonality decision problem. In addition, although some of these approaches perform economic benefit analysis to define products in the SPL [4], they still can't produce quantifiable and measurable results about the amount of commonality which should be used within product family. One reason is the lack of useful metrics to assess a software product family based on commonality. Therefore, until now, it is very hard to solve commonality decision problem optimally.

This research introduces the first systematic method to provide decision support during product platform design based on end-users' features. The proposed method aims at identifying the optimal product platform design in order to maximize the cost savings and the amount of commonality while meeting the goals and needs of the envisioned customers' segments. It generates, validates, and evaluates alternative product platform designs by adapting commonality and variability analysis rules presented in [3] and integrating market considerations with technical product platform concerns. Note that we are not proposing a new scoping

SPLC 2013 workshops August 26 - 30 2013, Tokyo, Japan

approach; the aim is to improve and expand upon the common scoping approaches that help in deciding on commonality.

The remainder of this paper is organized as follows: Section 2 presents the research work related to the solution approach. Section 3 presents background details of the techniques applied in the proposed method. Section 4 presents the method. Section 5 presents an illustrative application example of the method on a hypothetical SPL. Section 6 presents the conclusion with an outlook on future research direction.

2 RELATED WORK

There are three main research areas that are related to the proposed method. These areas are commonality and variability analysis, commonality measurements, and software product line cost models.

2.1 Commonality and Variability Analysis

In SPL literature, the process of identifying the commonality and variability is a basic part of the product line scoping [14, 15]. Most frameworks and methodologies for SPL development include scoping as a distinct activity. During the last decade a number methods and techniques for software product line scoping have been developed (e.g., [4, 6, 7, 8, 9, 10, 11, 12, 13]). Among them, some are relatively informal which rely on market analysis techniques (e.g. surveys) to elicit the scope of the product line (i.e. the Product Line Practice (PLP) Framework) [16]. On the other hand, there are very technical scoping techniques. PULSE-Eco v2.0 which is one of the Technical Components of PuLSETM is one such approach [4]. PULSE-Eco v2.0 does not, however, explicitly use automated optimization technique to come up with the optimized product platform design. Recently some work has been suggested to address the optimization of product portfolio scoping with regard to profit maximization [17, 18]. Yet, none of these works explicitly address commonality and variability analysis or the optimization of the platform design in their models. Instead of that, the suggested mathematical programs in [17, 18] allow determining whether feature is developed as core asset base or exclusively for the product(s) without deciding which can be made common or remain variable. Hence, their works have limits in deciding about commonality and evaluating the implications of different product platform designs. Moreover, they have dealt with only one important contributor to the overall cost of the system, namely development cost. Considering that evolution and maintenance usually cost more than the development, the restriction to development costs might not be realistic.

2.2 Commonality Measurements

In the literature, to the extent of our knowledge, only very few preliminary studies for defining the suitable commonality metrics have been conducted [19, 20, 21, 22]. However, none of these attempts consider the differences among components while measuring the commonality, for example, some components cost much more than other components. The impact of such difference on the commonality is not considered. Furthermore, the work presented in [20] attempts to measure the degree of commonality at the component level. Where only 'fully common components' (common among all the products) contribute to the amount of commonality while 'partially common components' (common among some products) do not contribute at all. The other works which have been proposed in software reuse [23, 24], have limitation in evaluating the concept of commonality in SPL. These metrics are rather oriented towards traditional reuse.

2.3 Software Product Line Cost Models

The Software product line cost models exist at various levels of detail. For instance, The Constructive Product Line Investment model (COPLIMO) [25] is used to make detailed cost estimations considering the cost of initial development of the SPL in addition to the costs and revenues from the future extensions of the product line, but is time consuming. As opposed to COPLIMO, Other cost models [26, 27] work on an abstract level with less accuracy, but higher speed than more detailed models. Structured Intuitive Model for Product Line Economics (SIMPLE) [26] is one of the most popular cost models that determine the cost at an abstract level. SIMPLE provides four cost functions ($C_{org}()$, $C_{cab}()$, $C_{reuse}()$, and $C_{unique}()$) that can be combined in a number of scenarios covering several possibilities of evolving and initiating a SPL [27]. However, those cost functions return the costs considering the whole product line. For example, $C_{cab}()$ returns how much it costs to develop a core asset base suited to satisfy a particular scope not on individual features level. In this work, the products are described in terms of features. Therefore, assessing SIMPLE's cost functions in our framework will require that we evaluate features' cost. The interested reader is referred to [28] for detailed survey of economic models for Software Product Lines.

3 BACKGROUND

This paper aims at optimizing the commonality decision on the bases of its effectiveness on the projected cost savings and the overall commonality. To estimate cost savings resultant of adopting a SPL, we adopt SIMPLE model [26]. To measure the amount of feature commonality, we propose an analytical tool (i.e. Commonality Index (CI)). In the following subsections, we present an overview of these techniques.

3.1 Cost Savings

As mentioned earlier, our cost savings model is based on SIMPLE model. Out of the four cost functions of SIMPLE, we assume that the function $C_{org}()$, which is used to calculate organizational and process related costs, does not depend on the product platform design. Therefore, we do not include it in our model. In addition to the development cost, by adopting SPL approach, organizations can exploit further opportunities to benefit from not having to duplicate corrective and evolutionary maintenance activities, though they do have to incur some cost to incorporate upgraded features(s) into their products. The total cost saving is the cost saved in the development cycle, plus the cost saved in the evolution and maintenance cycle. Thus, we propose a model for estimating the projected life cycle cost savings as follows:

$$Costs\ savings = \Delta Cost_{Dev} + \Delta Cost_{Evo} \quad (1)$$

with

$$\Delta Cost_{Dev} = \left[\sum_{j=1}^{J}\sum_{i=1}^{I}[req(j,i) + post(j,i)] \times C_{u,j}\right] - \left[\sum_{j\in CAB}[C_{cab,j} + C_{r,j} \times NP_j] + \sum_{j\notin CAB} C_{u,j} \times NP_j\right] \quad (2)$$

$$\Delta Cost_{Evo} = \left[\sum_{j=1}^{J}\sum_{i=1}^{I}[req(j,i) + post(j,i)] \times CE_{u,j}\right] - \left[\sum_{j\in CAB}[CE_{cab,j} + CE_{r,j} \times NP_j] + \sum_{j\notin CAB} CE_{u,j} \times NP_j\right] \quad (3)$$

where $req(j,i)$ is equal to 1 if feature j is basic, high priority, or differentiated feature for product i, it is 0 otherwise. $post(j,i)$ is equal to 1 if feature j can be introduced for product i (feature j is delighter, satisfier, high priority Indifferent, or identified as strategic feature for product i), it is 0 otherwise. These terms are explained in Section 4. We introduced the functions $req(j,i)$ and $post(j,i)$ similar to [4]. The reason for this is twofold: first, using $req(j,i)$ allows us to compare the cost of the initial assignment of the feature to the products with the cost of making that feature as common one. Second, using $post(j,i)$ allows us to decide in which product(s) the feature can be inserted into. If the feature cannot be inserted into some products (i.e. customers of these products do not care about it and are not willing to pay for this extra feature), then this feature represents extra cost on the product line. The remaining parameters are defined as follow: J is the number of features, I is the number of products in the family, NP_j is the number of products using feature j, CAB is the set of features in the core asset base, $\Delta\text{Cost}_{\text{dev}}$ is the development cost savings, $\Delta\text{Cost}_{\text{evo}}$ is the evolution cost savings, $C_{u,j}$ is the cost of developing feature j the traditional way, $CE_{u,j}$ is the cost of evolving feature j the traditional way, $C_{cab,j}$ is the cost of developing feature j as a reusable asset, $CE_{cab,j}$ is the cost of evolving feature j as a reusable asset, $C_{r,j}$ is the cost of reusing the shared feature j from the core asset base, and $CE_{r,j}$ is the cost of reusing the evolved feature j from the updated core asset. For simplicity, we assume the evolution cost estimates represent the whole evolution and maintenance cycle (i.e., the total number of upgrades).

3.2 Commonality Index

In this paper, we define feature commonality as the amount of feature sharing among family of software products. To measure the amount of feature commonality, we propose an analytical tool called Software Commonality Index (SCI) to measure the degree of commonality and evaluate the design of a product family on 0-1 scale similar to those used in manufacturing domain [29]. The index is proposed as the ratio between the number of features shared and the number of features that could be shared in a given product family. This ratio is calculated for each distinct feature in all products in the family being analyzed. These quantities are summed and then divided by the total number of distinct features. This index is defined as:

$$SCI = \frac{\sum_{j=1}^{d} \left[\frac{S_j - 1}{I - 1}\right]}{d} \tag{4}$$

where S_j is the number of products sharing feature j (number of repetitions), $I - 1$ is the maximum number of products that could share feature j, and d is the number of distinct features in the family. Each feature developed exclusively for a product or as a reusable asset is considered a distinct feature. The concept of distinct feature is explained as follows:

Example 1: Assume that we have two features X and Y. X is developed as a common feature and shared among four products (A, B, C, and D) while Y developed exclusively (stand-alone fashion) for products A and B, denoted as Y_A and Y_B, respectively. Consequently, we have three distinct features: X with four repetitions in the family, Y_A with one repetition (used by one product only), and Y_B also with one repetition. However, if for product D we have to develop another implementation for feature X, X_D, then we will have four distinct features in the family.

To reflect the differences among features while measuring the commonality, we incorporate feature's development cost as weighting parameter to SCI. Thus SCI can be redefined as:

$$SCI_{cost} = \frac{\sum_{j=1}^{d} C_j \left[\frac{S_j - 1}{I - 1}\right]}{\sum_{j=1}^{d} C_j} \tag{5}$$

where

$$C_j = \begin{cases} C_{cab,j} + [C_{r,j} \times NP_j] & if\ j \in CAB \\ C_{u,j} & if\ j \notin CAB \end{cases}; (j = 1,2,\dots,J) \tag{6}$$

SCI_{cost} provides the capability to compare different platform designs using the degree of commonality in addition to the projected cost savings.

4 METHOD

In this section, the proposed method is introduced, and its phases are explained.

4.1 Overview

The proposed method has three phases. The method requires three types of inputs. The first input is SPL portfolio related information. This input includes the segments of customers and their corresponding products, the list of features to be offered by the product line, and the list of differentiated features among the products. The product related information can be generated by using existing portfolio scoping approaches.

The second input is the features' costs estimates. This input includes $C_{u,j}$, $C_{cab,j}$, $C_{r,j}$, $CE_{u,j}$, $CE_{cab,j}$, $CE_{r,j}$. Estimating these costs is not an easy task, but considering that the purpose of assets scoping is to estimate the Return on Investments (ROI) for a product line using defined cost and benefits function [30], these estimates have to be available in order to do that. Several techniques can be used for generating these estimates (i.e. organization's own historical data, domain experts Delphi measures). Hence, the proposed method utilizes the typically available information during scoping phase.

The last input is Kano's survey data for each market segment. This input is generated by Kano surveys. Kano surveys are conducted to acquire the customers' assessment of the features according to the functional and dysfunctional forms of questions. In which, each respondent is required to answer the Kano questions with respect to each and every feature in the list. Interested reader is referred to [31] for the construction of the Kano questionnaire and evaluation table.

The first phase in this method is to analyze and interpret the surveys inputs. For each customer segment, survey inputs are evaluated and interpreted according to the frequency of answers. The second phase is to analyze features with the inputs of human experts. The third phase is to optimize the product platform design using the outputs of the first two phases. This phase helps decision maker to perform trade-off between conflicting objectives (if exist). Each phase is described next.

4.2 Phase 1: Analyze and Process Kano Surveys Inputs

4.2.1 Identify Kano categories for features

Kano's model method [32] allows choosing a set of product features that yield high customer satisfaction. Essentially, Kano's model categorizes customer needs into five major attributes, namely, Basic (B), Satisfier (S), Delighter (D), Indifferent (I), and Reverse (R). These categories are described in the following paragraphs.

Basic requirements: these needs that are not mentioned explicitly and taken for granted by the customer; absence of these features leads to high customer dissatisfaction, whereas their presence or further enhancement does not contribute to customer satisfaction.

Satisfiers: these features result in customer satisfaction when fulfilled, and dissatisfaction when not fulfilled. The more it is fulfilled, the more the customer becomes satisfied in proportional way to the degree of fulfillment.

Delighters: customers may not expect these features, however, fulfilling those leads to more than proportional satisfaction. On the other hand, if they are not met, there is no dissatisfaction.

Indifferent: these features that customer is indifferent to and is not very interested in whether it is present or not.

Reverse: these features that the customers do not desire.

After having combined the answers to the functional and dysfunctional question in the evaluation table (which shows the overall distribution of the feature categories), the most frequent observation approach is used to identify Kano categories for features. Basically, the most frequently observed Kano category is assigned to each feature. For example, If 30 survey inputs show that a particular feature is a basic feature, 22 inputs result in a satisfier feature, and 15 inputs indicate a delighter feature, that feature is assumed to be a basic feature since that is the most frequent observation among all. This step is applied for each market segment. If the individual product features cannot be unambiguously assigned to the various categories, the evaluation rule "B>S>D>I" is used. Kano's method provides valuable help in trade-off situations in the product platform design stage. If two product features cannot be met simultaneously due to technical or financial reasons, the feature which has the greatest influence on customer satisfaction can be identified [31]. Measuring the impact of individual features on customer satisfaction is performed in the next step.

4.2.2 Calculate the absolute importance values of features

Once Kano categories are identified, the priorities of features are identified using the absolute importance values of features. The absolute importance values of features are calculated by using two terms: impact on customer satisfaction (SAT_j) and impact on customer dissatisfaction (DIS_j) [33]. SAT_j is indicative of how strongly the presence of a product feature, j, may influence customer satisfaction, and DIS_j is indicative of how strongly its absence may influence customer dissatisfaction. SAT_j and DIS_j are calculated using:

$$SAT_j = \frac{D_j + S_j}{D_j + S_j + B_j + I_j} \quad (7)$$

$$DIS_j = \frac{B_j + S_j}{D_j + S_j + B_j + I_j} \quad (8)$$

where D_j , S_j, B_j , I_j represent the percentage of responses for feature j. Similar to [34], we assume that achieving customer satisfaction is as equally important as avoiding customer dissatisfaction. For this reason, we identify the absolute importance of each feature as the highest of either SAT_j or DIS_j using:

$$w_j = Max(Sat_j, Dis_j) \quad (9)$$

This step is applied for each market segment.

4.3 Phase 2: Features Analysis

After analyzing Kano survey inputs, we analyze features for further indicators of potential commonality or variability. To do that, we adapt the priority based commonality and variability analysis process presented in [3]. Human experts are involved in this process. The outputs of this phase are used as inputs for controlling the optimization process.

4.3.1 Identify high priority features

In addition to the basic features, high-priority features have to be identified. Since our focus is on the end customers' needs, we use the calculated absolute importance values of features (w_j) to order the features within each category. For example, if there were several customer features whose mode was delighter, we can rank the delighter features in descending order of importance. Using that order, we select the top delighter feature(s) as high priority ones. A threshold can be used on the number of features that can be considered as high priority within each category. This applies to satisfiers and delighters categories. For example the top 50% satisfiers and the top two delighters for a product can be selected as high priority features for this product. Prioritizing features within each category for each market segment provides decision capabilities to make trade-off between features when it necessary. However, this can be only one of several factors (i.e. competitor's products, costs) that will dictate what should be identified as high priority features.

4.3.2 Analyze features with regard to strategic commonality

Based on empirical evidences, [35] provided an interesting theory of Kano categories dynamics that follows a life cycle such as the following: indifferent → delighter → satisfier → basic. In other words, in the passage of time, what now perceived as delighter, or even perceived as indifferent, will become a basic thing in the near future because it will have become a common thing. [3] Defined the foreseeable basic needs that will appear in the product line's life time as strategic commonality. Thus, features identified as strategic ones are candidate to be implemented as commonalities to attain a stable set of common artifacts and to differentiate from competitors products [3]. The decision is whether to make it common now or in the future (variable now).

4.3.3 Analyze features with regard to delta

Another category of features that should be considered as candidates for common features is the features that only differ marginally (we refer to it as delta). Delta is identified by domain experts and can be in term of functionality (superior vs. inferior), or in term of implementation of functionality (i.e. performance issues). Furthermore, the impact of resolving the delta (i.e. rule 7 in the optimization phase) on other features within the product should be analyzed by the domain experts.

4.3.4 Analyze features with regard to technical conflicts and dependencies

In SPL, not all features can be implemented independently. Hence, mutual dependencies and interactions of features are important constraints when developing a SPL. Features with high ratings, possibly from different customer groups, which cannot be realized within the same product as they are in conflict with each other (i.e. technical incompatibility) can lead to the introduction of variability [3]. This information is used to control the optimization process to ensure the validity of the generated platform designs.

4.4 Phase 3: Optimization

The purpose of this phase is to generate, validate, and evaluate platform designs. The designs are generated based on a set of rules and the outputs of the previous phases. Each rule is explained next.

Rule 1: Each product has to fulfill its requirement. We refer to the sets of basic, high priority and differentiated features for product *i* as "*requirement i*". For instance, "*requirement i*" can be fulfilled by product *i* if and only if product *i* contains at least the set of basic features, the set of high priority features or valid replacement for those features (in case the feature is grouped to one delta), and the set of differentiated features. This is to ensure customer satisfaction and the delivery of competing products in the market.

Rule 2: feature categorized as basic by all the products in the portfolio will be implemented as a common feature if the cost of developing and evolving that feature as a common feature among all the products is less than the cost of developing and evolving it the traditional way:

$$Cost\_Initial_j - Cost\_common_j \geq 0 \quad (10)$$

Rule 3: feature categorized as strategic feature to all the products in the portfolio can be inserted into the products and implemented as a common feature if cost savings constraint is satisfied (10).

Rule 4: feature rated as high priority by more than or equal to θ of the products in the portfolio and the other products do not reject it (categorize it as a reverse feature) can be implemented as common feature if (10) satisfied. θ represents a threshold on the number of products in which the feature is identified as high priority feature. If the number of products is less than the value of θ, then that feature is considered as a differentiated feature to the products in which it is identified as high priority feature. Here, $Cost\_Initial_j$ represents the cost of developing and evolving feature j the traditional way for the products which rated this feature as high priority.

Rule 5: feature which is neither basic nor strategic nor high priority to all products and not rejected by other products still can be implemented as common if (10) is satisfied and:

$$\theta \leq \sum_{i=1}^{I} \left[h^{j(i)} + B^{j(i)} + S^{j(i)} \right] \quad (11)$$

where $h^{j(i)}$ is equal is equal to 1 if feature j is a high priority feature for product i, it is 0 otherwise. $B^{j(i)}$ is equal to 1 if feature j is categorized as a basic feature for product i. $S^{j(i)}$ is equal to 1 if feature j is a strategic feature for product i, it is 0 otherwise. Here $Cost\_Initial_j$ considers the products in which feature j is either rated high or basic or strategic. We assume no overlapping between these categories.

Rule 6: feature which cannot be made common among all the products (i.e. rejected by some products or differentiated) still have the potential to be implemented as common among the products that use it if (10) is satisfied. Here $Cost\_Initial_j$ and $Cost\_common_j$ consider the products that use feature j only.

Rule 7: feature grouped with other feature(s) as one delta, upgrade inferior feature j in product i to the superior feature (j+1), upgrade $(j \rightarrow j+1, i)$. However this must satisfy the following constraints:

$$Cost\_Initial_j - Cost\_common_{j+1} \geq 0 \quad (12)$$

$$Satisfaction_{(j+1,i)} \geq Satisfaction_{(j,i)} \quad (13)$$

Constraint (12) states that the cost of resolving the delta by upgrading features is less than the cost of the initial state for that delta (before upgrading). Constraint (13) states that customers of product i must be satisfied with the upgrade based on the rule: D > S > B. For example, a delighter (D) will generate greater customer satisfaction rather than a basic (B) one. In addition, other constraints have to be considered while deciding about upgrading features, such as hardware compatibility and resource constraints for individual products**.** In Section 5, we illustrate the delta model using the example SPL.

The generated design must be valid against the identified mutual dependencies and interactions of features in Phase 2. Hence, in order to ensure the generation of feasible solutions, the following constraints are introduced:

Market Constraint (differentiated features): those are the constraints imposed by the market needs to maintain the differentiation among segments where the differentiated feature for a product(s) cannot be inserted into other products.

Require/Exclude constraint: a constraint that requires that a feature and another feature(s) must (in the case of require) or must not (in the case of exclude) simultaneously exist in the same product. This type of constraint is already described in literature and used in modeling the relationships between features.

Other constraints also can be imposed by the specifications of the hardware environment in which products will operate and by the available resources for operating the products. For instance, in the embedded database management system domain, the energy consumption and footprint of the products are considered as the design constraints that each product must satisfy.

Evaluating and finding the optimized platform design in the feasible solution space is another issue. In this research we use two functions as possible measures of the platform design optimality. The first measure is the commonality index (5). [15] Stated that it is important to have as much commonality as possible, and thereby to reduce the amount of variability to the required minimum. However, this can be considered as a valid objective if and only if we maintain the minimum required variability that satisfy the goals and needs of the envisioned customers and/or market segments [3]. This is ensured by applying the rules and constraints discussed above. The second measure is the amount of cost savings (1) the company can achieve by using the candidate design of the platform. Similar to [4] we consider this as a valid objective of SPL approach. However, our cost savings function considers the evolution and maintenance cost savings in addition to the development cost

savings. We defined a deterministic algorithm by scanning all the possible platform designs (brute force). The algorithm start with the initial set of features (requirements of each product in the portfolio) and vary the level of commonality (by adding, substituting or removing) features of the product variants from the reusable platform. The algorithm controls the amount of commonality by applying commonality and variability rules considering the above inputs and constraints. The algorithm identifies which product platform design(s) maximizes the amount of commonality and cost savings among the generated ones. If competing designs exist, we perform offline analysis to find the Pareto set. The Pareto set platform designs are identified using the commonality and the projected cost savings. Next is to select a proper solution from the found Pareto solution set based on decision makers' preferences.

5 DEMONSTRATION OF THE PROPOSED METHOD ON AN APPLICATION EXAMPLE

In order to demonstrate how our method is used to optimize the product platform design, we designed a product line for smart phones as an illustrative application example. Our hypothetical product line consists of three segments (price-segment): Low-end, Mid-range and High-end segment. Furthermore within the High-end segment, two different user-categories are distinguished: Active end users and Business end user. Table 1 shows the variety of products and the features planned to be offered by the product line with their differentiated features. Features' costs estimates are shown in Table 2. With respect to the identified feature, the Kano questionnaire is fabricated. For each segment, surveys have been conducted to acquire the assessments of ten hypothetical customers of the features according to the functional and dysfunctional forms of questions. In which, each respondent was required to answer the Kano questions with respect to each and every feature. Once having combined the answers to the functional and dysfunctional question in the evaluation table, customer expectations were categorized as B, S, D, I, or R.

Next, the impact on satisfaction and the impact on dissatisfaction for each feature by using (7) and (8), and select the highest one as the absolute importance value using (9). Again, this procedure is repeated for each product. In the second phase, the features have been analyzed for further indicators of potential commonality and variability. For instance, f5 and f6 have slight differences among them, thus they grouped as one delta. Customers of P1 and P2 perceive f6 as D and S respectively. Thus, (13) is satisfied. Now, if (12) is also satisfied, then it is valid for the algorithm to explore the impact of upgrading f5 to f6 in P1 and P2 on the commonality and cost savings. The algorithm will also check if this upgrade requires or excludes other features in P1 and P2. The output of phase 2 is also tabulated in Table 1.

The result of the optimization presents the valid product platform designs with their commonality values and projected cost savings. Using this information, the non-dominated solutions are identified. For our example, the product platform designs PPD 8 and PPD 9 perform better than all the other candidate designs in the solution as identified in Fig.1. Hence, they represent the non-dominated solutions. Platform design PPD 8 is the optimized design on cost savings with the highest cost savings among all the candidate product platform designs. Platform design PPD 9 is optimized on commonality with the highest possible commonality (based on the rules). In PPD 9, f8 is to be implemented as a fully common feature (common among all the products in the family) compared to PPD 8. However, Product 1 customers perceive this feature as a "low priority indifferent" feature. Thus, they are not willing to pay for this extra feature. Consequently, and based on our cost savings model, inserting this feature into Product 1 will add extra costs on the company. With such differences in customers' preferences, it's possible that introducing commonality at some decision points could decrease costs while decreasing commonality at some other decision points could decrease costs. Furthermore, the offline analysis of the results highlights that some commonality decisions decreased the development cost, but increased evolution cost. For instance, implementing f5 as a common feature among Products 1 and 2 (partially common) minimizes development cost by $200 but it increases evolution cost by $100. Hence, it's necessary to assess both types of cost while identifying the optimized decision.

The decision maker analyzes the non-dominated solutions and selects the one of them based on his perception of the perceived benefit of increasing commonality. This perception differs widely among decision makers.

Table 1 Portfolio related information and the outputs of Phases 1 and 2

Smart Phones Product Line									
Feature		**Low-end**		**Mid-range**		**High-end**			
		P1		***P2***		***P3(Active)***		***P4(Business)***	
		Category	W_i	***Category***	W_i	***Category***	W_i	***Category***	W_i
f1	MS Outlook-Synchronization	R	0.0%	I	39.1%	I	42.0%	D √	67.8%
f2	HTML-Browser	B	66.1%	B	72.2%	B	74.5%	B	86.4%
f3	MS Word and Excel compatibility	I	42.0%	D	58.0%	S	65.2%	**S**	71.2%
f4	Animated Backgrounds	D √	66.8%	I	35.8%	I	42.0%	R	0.0%
f5	Wi-Fi (basic standards)	**S**	59.1%	B	60.4%	I	36.6%	I	28.5%
f6	Wi-Fi (advanced standards) f6(Δ f5)	D	60.0%	S	58.2%	B	68.3%	B	81.0%
f7	Making Call Using Head Phones	B	53.4%	B	71.0%	B	81.2%	B	86.4%
f8	GPS Assisted Navigating	I	31.3%	**S**	63.5%	**S**	68.4%	**S**	73.2%
f9	Tolerance For Vibrations And Humidity	I	24.0%	I	53.6%	D √	65.2%	I	36.8%
f10	Video Playback	I	56.0%	D √	68.5%	I	35.8%	I	26.6%

√ : Differentiated feature B: Basic S: Satisfier D: Delighter I: Indifferent
R: Reverse (Rejected) High priority features are bolded Strategic features are underlined

Table 2 Features' cost estimates (in hundred dollars)

Feature	*Traditional Development fashion*		*Product line development fashion*			
	C_u	CE_u	C_{cab}	C_r	CE_{cab}	CE_r
f1	4	3	6	0.8	5	0.8
f2	6	4	9	1.2	6	1.2
f3	5	3.2	7.5	1	5	1
f4	2	1	3.6	0.4	2	0.4
f5	6	4	8	1	7	1
f6	9	6	13	1.45	10	1.45
f7	4	2	6	0.8	2	0.8
f8	5	2	7.5	1	3	1
f9	8	7	12	1.6	6	1.6
f10	3	4	4.5	0.6	6	0.6

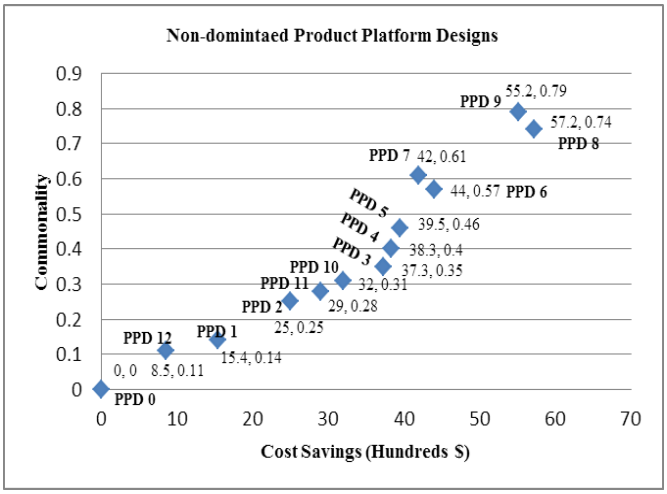

Figure 1 Sample of valid generated platform designs and the non-dominated platform designs

Table 3 Product platform designs: PPD 0, PPD 8 and PPD 9

Feature	PPD 0				PPD 8				PPD 9			
	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4
f1				U				U				U
f2	U	U	U	U	C	C	C	C	C	C	C	C
f3				U	C	C	C	C	C	C	C	C
f4	U				U				U			
f5	U	U										
f6			U	U	C	C	C	C	C	C	C	C
f7	U	U	U	U	C	C	C	C	C	C	C	C
f8		U	U	U		C	C	C	**C**	C	C	C
f9			U				U				U	
f10		U				U				U		

U: Uniquely developed C: Common feature

One decision maker is extremely sensitive to shorten the time for reaching the market for the new products and releases; therefore, he is willing to ignore some extra costs for achieving higher amount of commonality. On the other hand, another decision maker is extremely sensitive to reduce costs; thus, ignores some extra commonality which can increase costs. While the first two decision makers considered one objective at a time, another decision maker analyze the non-dominated solutions looking for candidate design that performs better than all the candidate product platform designs when both the objectives are considered together. However, such design does not exist in our example. The selected solution from the found Pareto solution set identifies the list of features that should be implemented either as full commonality (common among all the products) or partial commonality (common among some products) in addition to the list of features that should be uniquely developed for each product. For instance, Table 3 gives an overview of the platform designs PPD 8 and PPD 9 compared with the initial null platform design PPD 0 (design with 0 commonality).

6 CONCLUSION

In this paper, we demonstrated that solving the platform design problem optimally is difficult. It requires an assessment of the total costs of reuse and the benefits of using the platform (e.g. projected cost savings) associated with different platform designs while meeting the goals and needs of the envisioned customers' segments. We have proposed a method to provide decision support for optimizing the product platform design. The proposed method brings together the concepts and techniques from product development and customer satisfaction, and software engineering to generate and evaluate software product platform designs. We proposed an analytical tool to measure feature commonality and demonstrated how it can provide decision making capabilities in addition to the cost savings model. For realistic PLs, which may comprise hundreds or thousands of features, exact techniques which explore all the possible configurations suffer from exponential algorithmic complexity and typically require days or more, to identify the optimal platform design. However, our method is just the beginning of a process to build a tool based approach for optimizing the platform design. Hence, additional work must be done for creating complete mathematical model and building different heuristic optimization algorithms to make it possible to automate the method. Future work will focus on adding further techniques to perform trade-off analysis such as risk estimation and pricing models. Another promising area for

future research is to consider additional criteria for making commonality decisions (e.g., quality, time to market).

7 REFERENCES

[1] Riebisch, M.; Streitferdt, D, and Philippow, I. (2001), Feature Scoping for Product Lines, in 'Workshop on Product Line Engineering – The Early Steps: Planning, Managing and Modeling'. IESE Report No: 050.01/E, Fraunhofer IESE. 2001.

[2] Thevenot, H. J. *A Comparison of Commonality Indices for Product Family Design*, M.S. Thesis, Department of Industrial and Manufacturing Engineering, The Pennsylvania State University, University Park, PA, 2003

[3] Pohl, K., Böckle, G., and van der Linden, F (2005).: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Heidelberg.

[4] Schmid, K. A comprehensive product line scoping approach and its validation. ICSE 2002: 593-603.

[5] Van der Linden, F., Schmid, K., and Rommes, E (2007): *Software Product Lines in Action -The Best Industrial Practice in Product Line Engineering*. Springer, Heidelberg.

[6] Noor, M.A, Rabiser, R.; Grünbacher, P. Agile product line planning: A collaborative approach and a case study. Journal of Systems and Software Volume 81 , Issue 6 .2008.

[7] Inoki, M., and Fukazawa, Y. Core asset scoping method: product line positioning based on levels of coverage and consistency , *MESPUL 07* , 2007.

[8] John I., Knodel J., Lehner T. 'A Practical Guide to Product Line Scoping', *Proceedings of. SPLC'06*, pp. 3-12, IEEE CS,2006.

[9] Bandinelli, S., and Mendieta, G.. Domain Potential Analysis: Calling the Attention on Business Issues of Product Lines, *LNCS* 1951, Springer, 2000.

[10] Geppert, B., and Weiss, D. Goal-Oriented Assessment of Product- Line Domains. *(METRICS'03)*.pp. 180- 188. IEEE Computer Society, 2003.

[11] Park, S.Y., and Kim, S.D. A Systematic Method for Scoping Core Assets. *Proceedings of the 12th Asia-Pacific Software Engineering Conference APSEC '05*. 2005.

[12] Rommes, E. A People Oriented Approach to Product Line Scoping. *'International Workshop on Product Line Engineering The Early Steps'*, IESE Report 139.03/E, Fraunhofer IESE. 2003.

[13] Taborda L. Generalized Release Planning for Product Line Architectures. *Proc. of the 3rd Software Product Line Conference: SPLC 2004*. LNCS 3154.

[14] Coplien, J., Hoffman, D., and Weiss, D. 'Commonality and Variability in Software Engineering,' *IEEE Software*, vol. 15, no. 6, pp. 37-45, 1998.

[15] Ardis, M.A., and Weiss, D.M. 'Defining Families: The Commonality Analysis', *In:Proceedings of the 19th International Conference on Software Engineering (ICSE '97), Boston, Massachusetts,* 7–23, 1997.

[16] Clements, P., Northrop L, et.al., *A Framework for Software Product Line Practice – Version 5.0*, SEI, 2001.

[17] Muller, J. 'Value-Based Portfolio Optimization for Software Product Lines,' *Software Product Line Conference (SPLC), 2011 15th International* , vol., no., pp.15-24, 22-26 Aug. 2011.

[18] Gillain, J., Faulkner, S., Heymans, P., Jureta, I., and Snoeck, M. 2012. Product portfolio scope optimization based on features and goals. *In Proceedings of* (SPLC '12), Vol. 1.

[19] Her, J. S., Kim, J. H., Oh, S. H., Rhew, S. Y.,and Kim, S. D. A framework for evaluating reusability of core asset in product line engineering. Information and Software Technology, 2007, 49(7): 740-760.

[20] Berger, C., Rendel, H., and Rumpe, B. Measuring the Ability to Form a Product Line for a Set of Existing Products. In Proc. 4th Int. *VAMOS*, 2010.

[21] Capra, E., and Francalanci, C. Cost implications of Software Commonality and Reuse. In Proc. 3rd Int. Information Technology Conf., 2006, pp. 40-45.

[22] Peterson, D R. Economics of Software Product Lines. In *5th Int. Product Family Engineering PFE*, 2004, pp. 381-402.

[23] Jeffery Poulin. Measuring Software Reuse. Addison Wesley, 1997.

[24] Poulin, J., and Caruso, J. A Reuse Metrics and Return on Investment Model. In Proc. *2nd Workshop on Software Reuse: Advances in Software Reusability*, IEEE, 1993.

[25] Boehm, B., Brown, A. W., Madachy, R., and Yang, Y. A Software Product Line Life Cycle Cost Estimation Model. In Proc. Int. Empirical Software Engineering Symp., 2004, pp. 156-164.

[26] Boeckle, G., Clements, P., McGregor, J., Muthig, D., and Schmid, K. A cost model for software product lines. *Lecture Notes in Computer Science*, vol. 3014, 2004, pp. 310-316.

[27] Boeckle, G., Clements, P., McGregor, J. D., Muthig, D., and Schmid, K. Calculating ROI for Software Product Lines. *IEEE Software*, 2004, 21(3).

[28] Ali, M. S., Babar, M. A., and Schmid, K. A Comparative Survey of Economic Models for Software Product Lines. In *Proc. 35th Euromicro Conference on Software Engineering and Advanced Applications*, 2009.

[29] Johnson, M. D., and Kirchain, R. Developing and Assessing Commonality Metrics for Product Families: A Process-Based Cost-Modeling Approach. *IEEE Trans. Engineering Management*, 2010, 57(4):634-648.

[30] Lee, J., Kang, S., and Lee, D. A Comparison of software product line scoping approaches. *Int. J. Soft. Eng. Knowl. Eng.* (2010).

[31] Matzler ,K., and Hinterhuber, H. H. "How to make product development projects more successful by integrating Kano's model of customer satisfaction into quality function deployment," *Technovation*, vol. 18, no. 1, pp. 25–38, 1998.

[32] Kano, N., Seraku, N., Takahashi, F., and Tsuji, S. Attractive quality and must-be quality. The Journal of the Japanese Society for Quality Control 1984; 14(2):39--48.

[33] Berger, C., Blauth, R., Boger, D., and Bolster, C. Kano's methods for understanding customer-defined quality. *Cent Qual Manage J* 1993, 2(4):2–36.

[34] Sireli, Y., Kauffmann, P., and Ozan, E.; , "Integration of Kano's Model Into QFD for Multiple Product Design," *Engineering Management, IEEE Transactions on* , vol.54, no.2, pp.380-390, May 2007.

[35] Witell, L.N., and Fundin, A.. Dynamics of service attributes: A test of Kano's theory of attractive quality. *International Journal of Service Industry Management* 2005; 16(2):152—168.