A Proposed Index for Evaluating Component Commonality for Software Product Family

Hamad Alsawalqah[1], Bashar Al-Shboul[1], Yazan Alshamaileh[1], Hossam Faris[1], Ibrahim Aljarah[1], Ahmad Abadleh[2]
[1]The University of Jordan
Amman, Jordan
[2]Mutah University
Al-Karak, Jordan
{h.sawalqah, b.shboul, y.shamaileh, hossam.faris, i.aljarah} @ju.edu.jo, ahmad_a@mutah.edu.jo

Abstract

During the last two decades, Software Product Line approach has been applied by many companies due to its concepts of commonality and variability to provide product variety in a cost-effective manner. Yet, the effect of different amounts of component commonality on the perceived benefits from adopting SPL approach is not well understood. One reason is the absence of appropriate methods and useful analytical measures (i.e. indices) to assess the software product family based on commonality concept. This paper proposes an analytical tool, i.e. Software Component Commonality Index, to measure the amount of component commonality among a family of software products. In principle, it measures the amount of component sharing in the software product family based on the components of each product, their implementation, and connections. A software product line example from digital watch embedded software domain is used to demonstrate the application of software component commonality index.

Keywords: component; software product line; commonality

1. INTRODUCTION

An increasing number of companies are adopting the Software Product Line (SPL) approach to improve customization while shortening time to market and reducing costs. SPL is a current software development paradigm that applies the concept of product families to the development of software products and software–intensive systems [1]. SPL approach applies systematic reuse by exploiting commonalities within a set of products while maintaining the distinctiveness among those products.

The benefits of commonality, e.g. reduction of development costs, reduction of time to market, and enhancement of quality, are widely known by software companies; however, commonality's quantification methods, and correlation to commonality benefits, are yet not fully understood. Therefore, when developing new products poor understanding of commonality will produce poor implementation. Consequently, companies lose some of commonality benefits rather than taking full advantage of it. A reason is the absence of solid indices tailored to assess the software product family based on commonality concept. As a result, no real attempts are made to correlate component commonality to some of related quantifiable benefits (e.g. the reduction of development cost). The development of such indices is a prerequisite to understand the relationships between commonality and its benefits.

Although a number of indices to measure component commonality have been proposed during the past decade [2, 3, 4, 5, 6, 7], none of these indices consider the component's connections (i.e. Interfaces) with other components while measuring component commonality. Component's interfaces can result in very significant and painful differences due to the complexity and cost of reusing that component in other products. Such a factor should be considered when assessing the commonality of the component.

Another limitation of these indices is that they do not consider the desired level of variety in a product family, penalizing it most of the time. This means that such indices can reach their perfect value (the maximum commonality) just when all the parameters are common between all the components in all the products in the family regardless of whether these components are adding desired variety (required by different market needs) to the product family or not.

This paper is an attempt to address these shortcomings in the existing software commonality indices. One purpose of this study is to introduce the first metric to assess the impact of each component on the overall level of commonality and diversity in a software product family on a 0-1 scale, based on the components in each product, their

implementation (i.e. level of functionality and/or quality), connections, cost, and the allowed diversity in the family. The proposed metric, Software Component Commonality Index (SCCI), is a modified version of the Comprehensive Metric for Commonality (CMC) [11]. The second purpose of this study is to demonstrate how commonality indices can provide useful information that they can provide for software product family design and redesign.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the proposed index. A demonstration of the computation and usage of the proposed index on an illustrative case study is presented in Section 4, while Section 5 concludes the paper with a summary and an outlook on future research direction.

2. RELATED WORKS

In reviewing SPL's literature, it is noticeable that only few preliminary studies defining suitable commonality indices have been conducted [2, 3, 4, 5, 6, 13]. Some of them measure commonality at the feature level [6, 8, 13] while others measure commonality at the component level [9]. In [3], a research plan addressing the implications of commonality and reuse on the cost of software maintenance was proposed.

In another attempt, Peterson [5], three commonality parameters have been introduced with the required theoretical rigor for their ideas. However, none of these attempts consider the differences among features or components while measuring commonality. The other works which have been proposed in software reuse [8, 9], are limited in evaluating the concept of commonality in SPL. These indices are rather oriented towards traditional reuse. Consequently, understanding and evaluating the implications of component commonality decision become very difficult.

As an attempt to address the shortcomings of the aforementioned indices, the authors in [7] analyzed and adopted some analytical tools developed in manufacturing domain in order to measure the commonality in SPL. They stated that the main limitation of the analyzed indices is that they do not fully consider the desired variability in the product family. In other words, these indices promote commonality among all components, including the ones that should remain product specific or variant to differentiate products in the family, because those components add desired variety to the product family. Their study suggested that reusing commonality indices from the manufacturing domain to assess the commonality of products in SPLs is sensible and for some indices, modifications on these indices and their computation process are required in order to use them is SPLs

3. SOFTWARE COMPONENT COMONALITY INDEX (SCCI)

The proposed index is a modified version of the CMC which was proposed by Thevenot and Simpson [11] to assess the impact of each component on the overall level of commonality and diversity in the product family on a 0-1 scale. CMC evaluates the commonality based on the components in each product, their size, geometry, material, manufacturing process, assembly, cost, and the allowed diversity in the family. The CMC is defined as following:

$$CMC = \frac{\sum_{i=1}^{P} n_i \times (C_i^{max} - C_i) \times \prod_{x=1}^{4} f_{xi}}{\sum_{i=1}^{P} n_i \times (C_i^{max} - C_i^{min}) \times \prod_{x=1}^{4} f_{xi}^{max}}$$

Table 1 describes the parameters used in the CMC. A detailed process for calculating this index is shown in [11]. The main advantage of CMC is that it penalizes only the components that should ideally be common in a product family such that the desired variety added by differentiating components is not penalized.

Software components have different characteristics from manufactured components. For example, we do not need to consider the quantity of software components since we develop a shared component once and we reuse it. Moreover, each manufactured component, even the shared component, must be assembled again or purchased. Accordingly, there is no difference in cost if that component is common or unique one, except for some cost reduction due to factors like quantity discount and process cost (commonality in supply and process). Due to such differences, the computation of commonality among family of software product is unlike the manufacturing domain. Furthermore, as we explained above, CMC considers size/ geometry, process, material, fastening and assembly factors to assess the impact of the component on the family commonality, such factors do not exist in the case of software components. Based on the characteristics of software components analyzed in [12], this study defines two factors which can assess the degree of commonality of a software component:

Table 1: Parameters used in the CMC

Parameter	Description
P	The total number of components.
n_i	the number of products in the product family that have component *i*.
f_{1i}	The ratio of the greatest number of products that share component *i* with identical size and shape to the number products that have component *i* (n_i).
f_{2i}	The ratio of the greatest number of products that share component *i* with identical material to the number products that have component *i* (n_i).
f_{3i}	The ratio of the greatest number of products that share component *i* with identical manufacturing process to the number products that have component *i* (n_i)
f_{4i}	The ratio of the greatest number of products that share component *i* with identical assembly and fastening schemes to the number products that have component *i* (n_i)
f_{1i}^{max}	The ratio of the greatest number of products that share component *i* with identical size and shape to the greatest possible products that could have shared component *i* with identical size and shape schemes
f_{2i}^{max}	The ratio of the greatest number of products that share component *i* with identical material to the greatest possible products that could have shared component *i* with identical materials
f_{3i}^{max}	The ratio of the greatest number of products that share component *i* with identical manufacturing process to the greatest possible products that could have shared component *i* with identical manufacturing process.
f_{4i}^{max}	The ratio of the greatest number of products that share component *i* with identical assembly and fastening schemes to the greatest possible products that could have shared component *i* with identical assembly and fastening schemes.
C_i	the current total cost for component *i*: $C_i = \sum_{j=1}^{n_i} C_{ij}$
C_i^{min}	The minimum total cost for component *i* (obtained when component *i* is common between all the products having component *i*): $C_i^{min} = \sum_{j=1}^{n_i} C_{ij}^{min}$
C_i^{max}	The maximum total component cost (obtained when the component is variant in each of the products having component *i*): $C_i^{max} = \sum_{j=1}^{n_i} C_{ij}^{max}$

A. Function / Quality variation

Component with the same functionality can be implemented in different ways to provide different level of that functionality or to provide same functionality with different quality levels. For example, a component A can have a low resource consumption implementation which can process 10 Transactions per Second as required by market segment. A high resource consumption implementation which can process 100 Transaction per Second for high end market segment B.

B. Interaction-Based Variation

In contrast with the manufacturing components, where the compatibility of a shared component with the rest of a design is usually obvious and the effort for integrating such a component with the overall design is usually small. Component connections with other component can result in very significant and painful differences due to the complexity and cost of reusing that component in other products, such factor should be considered when we assess the commonality of the component.

According to this analysis, we define the SCCI as follows:

$$SCCI = \frac{\sum_{i=1}^{P} n_i \times (C_i^{max} - C_i) \times \prod_{x=1}^{2} f_{xi}}{\sum_{i=1}^{P} n_i \times (C_i^{max} - C_i^{min}) \times \prod_{x=1}^{2} f_{xi}^{max}}$$

where f_{1i} is the ratio of the greatest number of products that share component *i* with identical level of functionality/quality to the number products that have component *i*, (n_i). f_{2i} is the ratio of the greatest number of products that share component *i* with identical connection with the number of the products that have component *i*, (n_i). f_{1i}^{max} is the ratio of the greatest number of products that share component *i* with identical level of functionality/quality to the greatest possible products that could have shared component *i* with identical level of functionality/quality. f_{2i}^{max} is the ratio of the greatest number of products that share component *i* with identical connection to the greatest possible products that could have shared component *i* with identical connection. Other parameters will remain the same as CMC. For the cost factor, this represent the initial cost and total cost of ownership (including integration cost) and it can be decomposed onto details cost factor based on the available level of information.

For a successful product line portfolio, each product within a product line should be different from the other products in ways that are meaningful to the customers in each relevant market segment. Thus, and similar to CMC, the mean advantage of SCCI is

it penalizes only the components that should ideally be common in a product family such that the desired variety added by differentiating components is not penalized. An example on the computation and usage of SCCI is presented in the next section.

4. DEMONESTRATION ON AN ILLUSTRATIVE CASE STUDY

This section presents an overview of our defined software product line example from digital watch embedded software domain to show how to compute the value of the SCCI for a given software product family. This example focuses at the level of architecture and related components. Figure 1 illustrates the reference architecture for digital watch family and its sub components for time and alarm components. Table 2 shows an example product family derived from the reference architecture and consists of three products (P1, P2 and P3). The table also describes the products in terms of components. For cost estimates of components, we made simple cost estimation. Specifically, we assumed component development cost (in the context of P1 or in the family context) as a scoring function on a 10-100 scale based on the functionality and interfaces of each component, and the cost for P2 and P3 can be reuse cost if the component is common (i.e., organizing and retrieving components cost) or cost to adapt the component from the context of P1 (redevelop it for the intended product context, purchase it, and so on).

Table 3 shows sample data required for the computation of the SCCI index. It is important to know that SCCI index classifies components based on their cost factor value and the f_{xi} factors similar to CMC index. The total cost to produce a component i, C_i, ranges from C_i^{min} to C_i^{max} with C_i^{min} being the lowest cost achievable (best commonality) and C_i^{max} being the most expensive cost possible (worst commonality). For example, Time-Date Manager is common among the three products, thus we have cost to develop it as a common component (52), cost to reuse it in P2 (5) and cost to reuse it in P3 (5). Thus C_i for this component is 62 which is equal to C_i^{min} as it deems common between the three products. The C_i^{max} is 156 which is the case when Time-Date Manager is variant in each of the three products. The type of each component (differentiating component, non-differentiating component) influences the value of SCCI. For example, display controller is common between P1 and P2 but vary in P3, we consider it as non-differentiating component since it does not provide unique functionality nor seen by the customers. According to the sample data presented in Table 3, the value of SCCI would be 0.48.

The benefits of commonality indices come from the information that they can provide during product family benchmarking and product family design and redesign. This can assist the designers to make a better decision regarding which design strategy to use among the available alternatives, and focus on components that derive the most the commonality. For example, we can assess the impact of two different modifications on the current commonality level for family design shown in Table 2 in order to maximize the SCCI value:

First modification: make notification common between P1 and P2:C_i= 65+ 24+ 5 = 94, f_{1i}= 2/3, therefore, the new SCCI value is **0.5.**

Second modification: make notification common between P2 and P3: C_i= 20+ 24+ 5 = 49, f_{1i}= 2/3, f_{2i}= 3/3, therefore, the new SCCI value is **0.69.** This can be explained by the big reduction in the value of C_i.

Table 2: Digital Watch product line example

	Product Family					
Component	**P1**	**Cost**	**P2**	**Cost**	**P3**	**Cost**
Display controller	**V**	67	**V**	43	**V**	38
Time-Date Manager	**C**	52	**C**	5	**C**	5
Notification	**V**	65	**V**	24	**V**	20
Sound controller	**-**	-	**-**	-	**U**	51
Light controller	**V**	28	**V**	16	**-**	-
Note: C: common component; **V**: variant component; **U**: unique component, **-**: doesn't exist in the product						

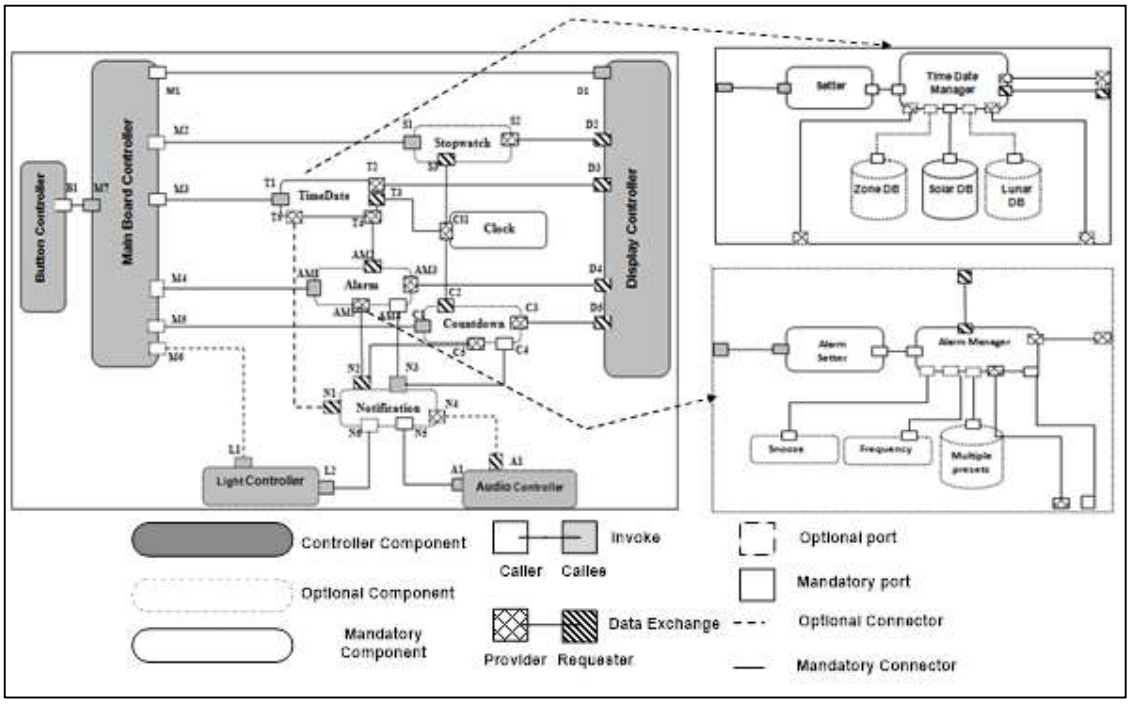

Fig.1: Digital Watch family reference architecture

Table 3: Input data for the computation of SCCI

Component	Product	Differentiating	Functionality level	Connection	n_i	f_{1i}	f_{2i}	f_{1i}^{max}	f_{2i}^{max}	C_i^{max}	C_i^{min}	C_i
Display Controller	P1	None	1	4	3	2/3	2/3	3/3	3/3	148	70	110
	P2		1	4								
	P3		2	7								
Time Date Manager	P1	None	1	8	3	3/3	3/3	3/3	3/3	156	62	62
	P2		1	8								
	P3		1	8								
Audio Controller	P3	Yes	1	2	1	1/1	1/1	1/1	1/1	51	51	51
Notification	P1	None	1	5	3	1/3	2/3	3/3	3/3	109	75	109
	P2		2	5								
	P3		3	6								
Light Controller	P1	Yes	1	2	2	1/2	2/2	1/2	2/2	44	26	33
	P2		2	2								

As we can see, the SCCI offer useful information about the impact of the two modifications on the overall commonality, thus, the designer can choose the second modification as long as it deems consistent with the design constraints, such as the interoperability among components.

In general, the usage of the commonality indices varies based on the strategy of the company and the level of available information on the components [7, 11]. Depending on that, the most relevant index or indices can be chosen. As a consequence of proposing the SCCI, the recommendations for the usage of the commonality indices presented in [7, 11] can be supplemented as shown in Table 4.

Table 4: Target commonality indices based on company strategy

	Criteria	DCI	TCCI	CI	C_{piece}	$SCI^{(c)}$	C_{cost}	SCCI
Strategy	• Focus on the number of common components	√	√	√	√			
	• Focus on the cost of component					√	√	√
	• Focus on non-differentiating components							√
	• Focus on in the variation of components							√
Available information	• Number of products	√	√	√	√	√	√	√
	• List of components for each product	√	√	√	√	√	√	√
	• Components costs					√	√	√
	• Variation parameters (function/quality-based, Interaction-based)							√

5. CONCLUSION

This paper proposed an analytical tool to assess the effectiveness of a given software product family design by measuring the amount of component commonality among a family's products. The proposed index, SCCI, is a modified version of the comprehensive metric for commonality to evaluate the design of a product family on 0-1 scale based on the components of each product, their implementation and connections. The SCCI was proposed due to the differences between manufacturing and software components, such as cost of developing common component and the cost for reusing that component, different component implementations, and connections. This research shows how the SCCI can be used for product family redesign through an example application. The use of the SCCI provides useful information on the redesign of a product family (assessment and comparison of the different design strategies of a product family, which components to redesign, and how to redesign them). This information would help in fast identification of those components that influence the commonality the most. SCCI is analytically feasible against the limited information indicative of early design (even if this information represents rough estimates). Such index can mark the starting point of the design of new families of products and the redesign of existing families. The scale of the problem presented in the paper is rather limited and simple. The main limitation of the SCCI is that it does not take component variability into account; hence it cannot tell how increasing commonality can affect market variability. For future work, we propose further refinement and validation of the SCCI after applying it on real SPLs cases.

6. REFERENCES

[1] Pohl, K., G. Böckle, and F. Van Der Linden, Software product line engineering: foundations, principles, and techniques. 2005: Springer.

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

[3] Capra, E. and C. Francalanci, Cost implications of software commonality and reuse. in Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on. 2006. IEEE.

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

[5] Peterson, D.R., Economics of software product lines, in Software Product-Family Engineering. 2004, Springer. p. 381-402.

[6] Peña, J., M. G. Hinchey, A. Ruiz-Cortés, and P. Trinidad, Building the core architecture of a NASA multiagent system product line, in Agent-Oriented Software Engineering VII. 2007, Springer. p. 208-224.

[7] Hamad, I. A., and Kang, S. W, Al-Shboul, B., Lee, J.H. "Measurement and Development Cost Implications of Component Commonality in Software Product Line Engineering.", International Journal of Computer & Information Science, 2013, 14(2): p. 27-44

[8] John, I., J. Knodel, T. Lehner, and D. Muhing, A practical guide to product line scoping. in Software Product Line Conference, 2006 10th International. 2006. IEEE.

[9] Poulin, J. S., Measuring software reuse: principles, practices, and economic models. 1997.

[10] Poulin, J. S. and J. M. Caruso, A reuse indices and return on investment model. in Software Reusability, 1993.

Proceedings Advances in Software Reuse., Selected Dissertations from the Second International Workshop on. 1993. IEEE.

[11] Thevenot, H.J. and T.W. Simpson, “Acomprehensive metric for evaluating component commonality in a product family,” presented at the ASME 2006 Int. Design Eng. Tech. Conf. Comput. Inf. Eng. Conf., Philadelphia, PA, Dissertation No. DETC-2006-DAC-99268.

[12] Simidchieva, B. I., and Osterweil, L. J. Characterizing process variation (NIER Track). In Proceedings of the 33rd International Conference on Software Engineering (ICSE ’11) (2011), pp. 836–83.

[13] Alsawalqah, Hamad I., Sungwon Kang, and Jihyun Lee. "A method to optimize the scope of a software product platform based on end-user features.", Journal of Systems and Software 98 (2014): 79-106.