Xæææææ Á Á[~c æ^Á} * æ^!æ * Á ææææ { •
Á
ÁÁÁÁÁ^•^*[{ •Áæ åÁ ÁU} [^&

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

Variability in software engineering paradigms

T. Huysegoms, M. Snoeck

Faculty of Business and Economics, Management Information Systems group, KULeuven

Naamsestraat 69, 3000 Leuven, Belgium, (+32)16326882

{Tom.huysegoms, Monique.snoeck}@econ.kuleuven.be

ABSTRACT

The concept of variability is not new in software engineering, but current research mostly remains vague about the overall variability concept when it comes to giving a clear overview of the dimensions of variability. In this paper we evaluate the core variability concept by proposing an overview of the set of definitions concerning variability related concepts and by setting up dimensions of variability. These dimensions represent different possible views on variability for all types of stakeholders. Through an analysis of the gaps in the current state-of-the-art literature, we identify challenges for the current state-of-the-art and we propose an extension to current variability related paradigms to fill in these gaps.

Categories and Subject Descriptors

D.2.11 [Software engineering]: Reusable software – *domain engineering*.

General Terms

Documentation, Design, Standardization, Theory.

Keywords

Variability, SPLE, SOA, software engineering, software architecture.

1. INTRODUCTION

Already for decades, reusability in the context of software applications is a main goal that has been strived for. It is common knowledge that substantial reusability benefits can be obtained through the proper planning of reusability during the early stages of software engineering. Planning for reusability in essence boils down to managing possible variability in software applications. If there is no room for variability, the possible scope of reuse that can be achieved becomes too small, but too much variability makes reuse too difficult. The problem with the concept of variability is that most literature remains vague about its precise definition, and when a definition is given the different dimensions and possible views of variability are not sufficiently covered. In this paper, we evaluate the concept variability within the context of software engineering by presenting an overview of the definitions of variability related concepts and extracting the dimensions of variability out of these definitions. Current state-of-the-art literature is mapped on these dimensional views, and gaps in the body of literature are identified. Based on these gaps we propose an extension of current variability paradigms to fill up uncovered aspects of variability. Section 2 explains the literature review process and gives the results of the review. These results are used to infer definitions for variability related concepts in section 3. Section 4 extracts variability dimensions out of these definitions, and maps them against the results from the literature review. Section 5 identifies the gaps in contemporary literature, section 6 forms conclusions and presents the directions of our

future research.

2. RELATED WORK

In the past literature reviews about software product lines in general have already been published, like [2], and also literature reviews focusing more specifically on variability, like [7], [44] and [46]. From our perspective, a limitation of these literature reviews is that they focus only on the product line paradigm, and omit studies of other paradigms which also address variability. The reviews also mainly focus on approaches to manage variability, or techniques to model or realize variability, but the core of the variability concept is not addressed. A complete overview and literature review on the variability concept itself covering all paradigms is thus still needed.

3. LITERATURE REVIEW

Before trying to set up any definitions of the concepts relating to variability and analyzing the dimensions of variability, we first investigate how these terms are currently used in literature. To obtain a qualitative overview of the current literature on variability, we base ourselves on the systematic literature review method (SLR) proposed by Kitchenham et al. [23]. This technique can be divided into three main phases, namely the planning of the review, the review itself and the analysis of the results, in order to obtain an objective overview. In the following paragraphs, the different phases of the SLR are detailed and applied to our context.

3.1 Review Planning

During the planning of the SLR we formulate a research question for our review which we will use as a guideline through the rest of the process, and as the basis to set up a search strategy. The general goal of our research is to develop a variability management process in a particular organization. More in particular, we are mainly interested in variability as a requirements engineering and design issue (design *for* reusability). Variability implementation is, at this point in time, out of scope of our research. To be able to manage variability, we first need to understand all aspects of variability. We therefore are interested in the definition of variability related terms and the dimensions in which variability is situated. As a result, our research question is twofold: (1) How is the concept of variability perceived and defined in current state-of-the-art literature, and (2) what relevant dimensions of variability are currently addressed? In the first part of the research question we not only look at the term variability itself, but we also take a look at the other terms that are frequently mentioned while talking about variability. By looking at the related terms we can see which dimensions of variability are deemed relevant. In this way, the first part of the research question can be used as a starting point for the second part.

The search strategy we use to conduct our review is based on the use of a search string executed on the abstracts of papers in digital libraries. Table 1 shows the terms of the search string and the reason why they are included in the string. At the same time this search string is also a way to set an initial set of boundaries to the literature review. The choice for these boundaries is also mentioned in the second column of the table.

Table 1. Terms of the search string

Keywords	Reason for inclusion
Variability	This term is the key concept we wish to study, as mentioned in the research question.
Software	The concept of variability is studied in a software context, other sources of variability like hardware variability (e.g. in the context of embedded systems [42]) are considered out of scope.
Engineering	We focus on the engineering aspect of variability. The implementation of variability is out of scope as already mentioned in the general goal.
Modeling	As for the term engineering, modeling is included because we focus on managing variability at design time.

When we combine all the terms mentioned in the table, we become the search string used. Before we were able to form this final string, we constructed and tested many different strings. Due to lack of space, we cannot give an overview of all strings and terms tested. An example of a related, but finally not included term is ‘volatility’. We opted to not include synonyms of variability in our study, because this could introduce a potential bias towards included synonyms versus non included ones. The final search string contains one additional term, namely ‘modelling’ as alternate spelling for ‘modeling’. The final search string is as follows:

variability AND software AND engineering
AND (modeling OR modelling)

The search string is only executed on the abstract section of publications as full text searches would produce many irrelevant hits. The title is not included in the search string scope either, because using the search string on both title and abstract posed practical problems in some digital libraries. An analysis of the search results in several digital libraries where a combined search in title and abstract was possible learned us that no significant papers were added by expanding the scope of the search string to include the title next to the abstract section. Therefore we can say that the decision of only including the abstract does not pose a problem for the validity of the literature review.

A final part of the search strategy consists of defining inclusion/exclusion criteria [23]. Rather than working with explicit criteria, we defined an initial exclusion criterion up front as a list of out of scope subjects. Examples are subjects related to variability implementation or hardware variability in embedded systems. The remaining papers were investigated individually and the researchers decided on a per paper basis whether or not to include the paper in this literature review, hereby gradually extending the exclusion criterion. The final criterion for inclusion was that the paper should mainly focus on the variability aspect of software, not just have variability noted somewhere on the side, and it should be in a software engineering context.

3.2 Review execution

The search string was used on four different digital libraries:

ACM, IEEE Xplore, Science Direct and Springer Link. These libraries cover most of the field of the contemporary literature. When possible, we searched only in the ‘computer science’ part of the library. The searches were executed during the first week of April, and were repeated one week later. The last update of the list of papers was done on the 9th of April 2010. In the table below (Table 2), the amount of papers that were extracted by the search string are specified for each library.

Table 2. Search string results

Name of the library	Number of papers extracted by search string	Number of papers retained after inclusion/exclusion
ACM	14	4
IEEE Xplore	71	20
Science Direct	2	0
Springer Link	26	10
Total	113	34
Extended total	/	38

Once all the papers were extracted, the list was checked for possible doubles. Only one double listing was found. After this check, all papers were scanned for inclusion/exclusion. This was done by both authors separately and the resulting shortlists of to be included papers were brought together. Differences between the lists were discussed by both authors and a consensus was reached for each of the conflicting opinions. The resulting assented shortlist of relevant papers contained 34 entries. In Table 2 also the number of relevant papers per library is given.

As a final step of the search process, after doing an initial reading of all the papers in the shortlist, we extended the scope of the literature review by backward and forward crawling of references. This was done to counter possible gaps in the review by only focusing digital libraries. Key publications will be referenced to a lot and through the crawling they will be included in the review even when they are not mentioned in digital libraries. Also papers falling out of the boat because one key term is not included in the abstract are able to enter the review through this crawling process. For each entry in the shortlist we examined the references made by the paper for additional papers that were to be included in our review. At the same time we used the ISI Web of Science¹ to examine papers which cite the entries of our shortlist. With this two-way reference search we found several new relevant papers not yet included in our shortlist. In case of (almost) identical authors and to prevent having several papers of the same author(s) on the same subject with almost the same content, we decided to only include papers which had a substantially different content from the papers already located in the shortlist. Like the inclusion/exclusion step before, this step was also performed by all authors separately and the results were then put together and matched. As a result of this last step, 4 additional papers were retained, giving an extended total of 38 selected papers.

In the next phase all the papers were analyzed with respect to the research questions formulated earlier. The results of this analysis are reported in the following sections. First we give some remarks concerning the validity of the SLR.

¹ <http://apps.isiknowledge.com/>

3.3 Literature study validity

Although the principles of SLR, gives us a fairly objective and systematic basic work method, several possible threats to validity are present. The first possible threat is that we used our search string only on the abstracts of papers. As mentioned before, in our opinion this poses no real threat because where possible, we did use the search string on both abstract and title and observed no relevant papers being added by including the title in the search scope.

A second possible threat to the validity is the fact that we missed some relevant papers by only focusing on four major digital libraries. We are fully aware that some relevant papers may be missed, but we are pretty confident that these four libraries in combination with the reference search cover most of the field. This confidence is based on the fact that the pre-study of this research used several non-systematic searches with Google Scholar and Citeseer, and all relevant papers found at that time were also brought forward by the literature review conducted.

A third and last possible threat towards the validity of the literature study is the fact that we did not use an explicit inclusion criterion to create the shortlist. This threat to validity is minimized by the fact that the authors of this paper each performed the inclusion analysis separately. When the different lists were put together, conflicting opinions were resolved most of the time in favor of inclusion, so maybe too many papers are included in the literature study. However these false positives are much less dangerous than the alternative of not including relevant papers, so-called true negatives.

4. VARIABILITY DEFINITIONS

This section gives an overview of the definitions inferred on the basis of the SLR. The terms we wish to define all relate to the general concept of variability in a software development context. The purpose of these definitions is not to be consistent with all possible views on each of the terms. They serve to give a general idea about the different variability related terms so we can use these definitions to extract the dimensions of the variability concept in the next section.

4.1 General variability terms

Variability itself is the single most important term, but at the same time this term is the most difficult to define accurately. S. D. Kim et al. [22] give a list of the definitions of variability used in representative works. These definitions however are largely focused on component based development. The definition we suggest is one that better fits all possible software engineering contexts, it is a definition summarizing the different views from the papers of the literature review.

Definition: “*Variability* is the possibility to change, or the possibility to be fully configured at later points in time. Variability only exists in a context where there is some kind of fixed structure present which this variability relates to.”

When we take a closer look at this definition, the first part of it implies that variability can be viewed in two distinct manners. The first view is as an ability to change something which was already specified up front, like e.g. a parameter value in a class definition. The second view on variability is the one where we have the freedom to define something because there is nothing defined up front. An example of this view is the choice of an

algorithm to realize some functionality. It is possible that both views on variability are combined, in a sense that something that is specified but still subject to change can be unspecified, re-specified or extended at later points in time. This also is variability. The second part of the definition implies that you can only speak of variability when it is in contrast to some other parts that do not possess this variability. When everything is variable and nothing is specified, you don’t speak of variability, but of chaos.

Requirements are the demands of the various stakeholders of a software application. These demands can either be of functional nature (functional requirements) or of non-functional nature (non-functional requirements). The requirements for a software application provide the boundaries of the variability. The requirements can create boundaries for something that is already specified but subject to change, or boundaries for a possible set of specifications. Therefore requirements present boundaries for both views on variability, e.g. a security requirement can exclude unsecure network protocols of the possible choices of network protocols.

Software Product Line (SPL) and Service Oriented Computing (SOC) are two software paradigms that systematically use the concept of variability. They both represent more than just a concept, they represent a way of thinking. In the next paragraphs both paradigm’s terms concerning variability are studied in detail, but first a definition of both paradigms is given. *Software product lines* are first mentioned in 1990 by Kang et al. as a part of their FODA specification [20]. A good definition of SPL can be found in the work of P. Clements et al.[8]: “A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”. SPL focuses on a group of related, highly alike software applications, and the concept of variability is used to situate the differences between these software applications. According to Papazoglou [40] *service oriented computing* is “the computing paradigm that utilizes services as fundamental elements for developing applications/solutions”. In this paradigm variability is more dynamic, in the sense that SOC uses services as building blocks to construct the needed functionality at runtime. Software Product Line Architecture (SPLA) and Service Oriented Architecture (SOA) are the architectural models of respectively SPL and SOC. In the following paragraphs, we try to define the variability related concepts inside both paradigms.

4.2 SPL and variability

In the context of the SPL paradigm, there are a few concepts that are really worthwhile defining because these concepts form the variability concept in SPL. It is not our objective of defining all possible concepts in the context of SPL, but just the ones that are relevant to the discussion about the variability concept.

In SPL a distinction is made between domain engineering and application engineering. *Domain engineering* activities all relate to the collection and management of domain knowledge. J. Bosch et al. [3] say that domain engineering is, among others, concerned with identifying the commonality and variability for the products in the product line and implementing the shared artefacts such that the commonality can be exploited while preserving the required variability. So the knowledge created in domain engineering revolves around the variability between products in the product

line. A result of the domain engineering activities is a domain architecture. This architecture represents the differences between applications, modeled with variation points. You can see the results of domain engineering also as the creation of assets, which are then basic reusable parts in application engineering. An example of this kind of assets are feature diagrams.

A *feature diagram* is a visual representation of the feature combinations of an application or an application domain, complemented with the relations between these features. A *feature* of a software application is a logical grouping of some requirements of that application. Features are aspects of an application, while requirements are demands from the stakeholders for the application. Features can consist of subfeatures, which in their turn can have subfeatures themselves. A feature diagram is represented as a graph, and most of the time even as a hierarchical tree. The variability in a product line is represented by the feature diagram, as the diagram shows which features are mandatory and which are optional. Possible alternatives between features are also modeled in feature diagrams. The relationships between features can be very complex, and a lot of study (like e.g. [28]) has already been devoted on the dependencies between features. At the same time feature diagrams are used to model multiple levels of variability. They are used to model both product line (or application domain) and application variability, so also here research is conducted (e.g. [34]) on clarifying things.

Variation points are points at which variation occurs. These points are scattered through the whole architecture of an application domain. Variation points are filled by *variants* which are possible implementations of a variation point. A nice overview related to variation points is given by S. D. Kim et al. [22] A variation point has a lifecycle which consists of different steps. At first when the variation point is not yet identified, it is said to be *implicit*. This means that it does exist, but that is just has not been identified yet. Once a software engineer or architect identifies a variation point it becomes *explicit*. It means that from this point on a variation point can be filled with a variant. The period of time in which variants may be added to the list of possible variants for a variation point can be limited. A variation point for which additional variants may be added is called *open*, when the list of possible variants is fixed, the variation point is called *closed*. As long as no choice is made to fill a variant, the variation point is said to be *unbound*. Finally, once one or more specific variants are chosen to fill in the variation point, the variation is resolved and the variation point becomes *bound*. One should note that in certain cases it is possible to have multiple variants to fill a variation point at the same time. Also, as with features, there can exist certain dependency relations between variants.

The mapping of features to variation points is non-trivial. When one decides to implement or to remove a feature, this will possibly impact several variation points. So one feature can link to several variation points and one variation point can be influenced by multiple features.

Application engineering in general concerns the development of an individual application that meets the requested demands from stakeholders. More specifically in the context of SPL, the assets developed during domain engineering are used to develop the individual applications. Next to the assets that are being reused, it is possible to add application specific assets. Variation points are resolved and features are selected. A possible view on application engineering [3] is seeing it as the inverse of domain engineering,

where domain engineering tries to create more possible variability, application engineering tries to reduce this variability to a point where the resulting application has filled in all variable parts. This view has one flaw, being that it is possible that some variability remains, even when the application is constructed and up and running. Further critical remarks are made in the following sections.

There are a lot of additional terms that are specific to or used in a specific way in the context of SPL. Defining all these terms is out of the scope of this paper. In the next paragraph, we try to define the variability related terms in a SOC context.

4.3 SOC and variability

A *service* is a very loaded concept in current literature, and defining it is therefore not simple. The view we take on services is a software engineering view, so we look at *software services*, also known as application services, and not at the concept of a business service. There is a big difference between both kinds of services, as investigated by R. Haesen [14]. One of the better, general definitions for a service in the software context is given by M. Papazoglou [40]: “Services are selfdescribing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications.” This definition clearly states that services can stand alone on themselves or be combined into larger groupings. A standalone service at the most basic level is often called a simple or *atomic service*. On a higher level these atomic services can be combined into a more complex, *composite service*, which in turn can be used once again in a higher level composition. We remark that the definition of a service here links well with the definition of a feature in the SPL context. Like mentioned there, a feature can consist of subfeatures, which in their turn can have subfeatures themselves. Therefore it is normal that recently more and more research is studying the linking of SOC and SPL concepts, but more about that in following sections, first some related terms are mentioned to give a better feeling of the service concept.

It is a *service provider* who offers some related functionality to a *service consumer*, who uses the functionality offered. This functionality is called upon by the service consumer through the *interface* of the service. This interface elicits the possible actions a service offers and can be published, so every service consumer can look up the functionality of different services. In the context of service composition, it is possible that the service consumer is a higher level composite service which is at the same time a service provider on the higher level.

The variability in the context of SOC comes from two sources. One of these sources is the dynamic aspect of SOC. The fact that service consumers can choose from multiple services to perform the desired actions and that each time a different service can be selected, gives a high variability at runtime. Unlike in SPL, where most of the time a variation point remains fixed once it is bound, in SOC services can be interchanged easily. This strokes with the loose coupling view on SOC. The other main source of variability comes from the fact that services can be used in many different combinations. For services to be usable in different configurations they must be generic enough. This can be achieved by making parts of the services variable.

Obviously, as both paradigms address issues related to variability, the combination of both paradigms offers interesting perspectives. Recently, we witness a growing interest in the combination of SOA and SPL, coined with the terms SOPL and SOAPL.

5. VARIABILITY DIMENSIONS

Software variability as a concept in general has been studied in the previous section. In this section, we combine these definitions with the different views taken from the papers in our literature review to obtain a global overview of the concept of variability. To obtain a useful framework, we classify the aspects in four dimensions that define a related set of aspects. The dimensions are -to the best of our knowledge- fully independent: in principle, each variability approach can combine all dimensions and take an explicit (or implicit) position in each of the dimensions. There are no constraints or dependencies between certain dimensions. The dimensions are also important to get an overview to the different stakeholder visions, because each stakeholder typically has specific demands on (some of) the dimensions. We present the overall space as a table that sets out the paradigm dimension (SOC or SPL) against the three other dimensions. Subsequently, each paper is positioned into one or more cells. Finally the table is analyzed for potential gaps or opportunities in the state-of-the-art literature. These gaps are discussed in the next section.

5.1 The paradigm dimension

This first dimension is the paradigm dimension and is probably the most obvious view you can have on variability. Variability is used in several paradigms, but the two main paradigms where variability has a central role are SOC and SPL. More recent literature also sees a very important place for variability in the paradigm combination of SOC and SPL. These papers are positioned in both columns. When we take a look at the papers from our literature review, we see that variability is researched more in the context of SPL than in the context of SOC. One possible reason for this is that the concept SPL already exists much longer compared to the SOC concept. However in the most recent papers the focus is directed to SOPL, so both paradigms are covered at the same time in that way.

5.2 The space-time dimension

There are two kinds of variability in the lifecycle of software, when we view the variability in terms of its origin. *Variability in space*, the first kind of variability, refers to the fact that it is possible to have different (but related) software applications running concurrently. The variability lies in the fact that the concurrently existing applications are not the same but differ (slightly) from each other. The second kind of variability is variability in time. *Variability in time* is the variation of a single software application over time. This type of variability is however not linked to concepts such as variation points, variants, ...etc. Requirements for an application can change, the scope of the application can change, etc, and so the variability needs to be changed accordingly. Here we see variability in time as the evolution of the variability in space. It is possible for a software application to be subject to both variability in time and space at the same time. When we look at the literature study, we see that the focus is strongly on the variability in space part.

5.3 The requirement dimension

The view on variability from a requirement point of view also distinguishes two kinds of variability, namely variability in the functional requirements (FR) and variability in the non-functional requirements (NFR). This view considers the reason for implementing the variability. Variability in functional requirements provides variability in the things a software application needs to deliver or is expected to do. Variability in non-functional requirements provides variability in the way a

software application does the things it needs to do. Both kinds of requirement variability propagate themselves mainly into variability in space, as different requirement combinations typically lead to different software applications. Non-functional requirements are very seldom addressed in the context of variability. During the literature review, only one paper emerged that focused on the non functional requirements in general and their impact on the complexity of the variability. When we perform searches for specific non-functional requirements, e.g. security, we do find relevant research. The problem is that these papers only cover one aspect of all non-functional aspects in isolation to other non-functional aspects.

5.4 The focus or scope dimension

This dimension of variability we found by adopting a view based on the scope that the variability induces. This dimension may seem a bit awkward at first, but when looking deeper into it the view becomes more interesting. Variability can be seen at different focus levels. Depending on the fact that variability concerns one application, a family of related applications or as much applications as possible, the focus of the applications and their complexity changes. Inside a single applications variability can be seen as e.g. a parameter value that can change. Depending on the value of the parameter, other functionality will be activated. The application will be useful in some contexts thanks to the parameter, but these contexts will be strongly related. The practicality however of this application is big, as just one parameter needs to be set. When variability is viewed on the level of a family of applications, like in SPL, the possible contexts in which an application from the family can be used are much more diverse, but at the same time the process of obtaining a working application becomes more complex. A family of related applications are also called members of the same domain. When the variability tries to scope all possible contexts as in very generic SOC services, the amount of possible contexts is huge, but the complexity of obtaining a working application is also huge compared to the smaller variability scoped projects. In conclusion we can say that the width of the focus of the variability is positively correlated with the amount of possible contexts (the scope), but negatively correlated with the practicality of obtaining a working application, so a tradeoff needs to be made.

5.5 Dimensional overview

Now that we have defined our dimensions, we can combine them with each other to form the total research 'space' on the concept of variability. The papers we obtained through our literature review can be situated inside this multidimensional format so we can define possible gaps in current literature. Table 3 gives an overview of the papers from the literature study.

The table hereunder is an image from the variability 'research space' and must also be seen as such. A gap in the table represents the absence of focus on variability, not the absence of research altogether. Each dimension is mapped against the paradigm dimension in the table, the points on the paradigm dimensions are the two main paradigms (SOC and SPL), their combination (SOPL), and a final point for paradigm agnostic papers or papers that cannot be positioned elsewhere. Other possible mappings between dimensions are not included in the table, but the main gaps can be identified nevertheless. Some papers cover multiple values on certain dimensions, some other do not cover certain dimensions at all. In the next section, we discuss the variability 'research space' more in detail focused on the identified gaps.

Table 3. The variability space

	SPL	SOC	SOPL	Others
Variability in time	[18], [11], [33]	[38], [6]	[27]	[43]
Variability in space	[19], [36], [34], [39], [44], [4], [18], [15], [17], [1], [49], [11], [12], [46], [29], [28], [16], [30], [10], [48], [50], [32], [31], [33], [13]	[38], [6], [25]	[35], [5], [27]	[22], [21], [43], [9]
Variability in FR	[19], [36], [34], [44], [4], [18], [15], [17], [1], [49], [11], [12], [46], [29], [28], [16], [30], [10], [48], [50], [32], [31], [33], [13]	[38], [25]	[35], [5], [27]	[22], [21], [43], [9]
Variability in NFR	[39], [1], [46]	/	[35], [27]	/
Variability on application level	[34]	[38]	[35]	/
Variability on family level	[19], [36], [34], [39], [44], [4], [18], [41], [17], [1], [49], [11], [12], [46], [29], [24], [28], [16], [30], [10], [48], [50], [32], [31], [33], [13]	[6], [25]	[5]	[22], [21], [37], [43], [9]
Variability on highest possible level	[15]	/	[27]	/

6. STATE-OF-THE-ART SITUATION

6.1 The heavy SPL focus

The first thing which is immediately apparent is the heavy focus of contemporary literature on the SPL paradigm when it comes to variability. This is not a gap, but the gap lies in the fact that other paradigms, maybe apart from SOC, do not pay much attention to the variability concept. This is actually strange because variability is one of the main drivers for achieving high reusability, and reusability is something that almost all paradigms focus on. A second remark concerning the heavy SPL focus is the fact that although variability in SOC is studied to a lesser extent in current papers, the movement towards the SOPL combined paradigm ensures that in the future variability will be studied in SOC context. The SOPL paradigm is on itself a great opportunity for further research, as SOC and SPL fit nicely into each other and create much added value in combination with each other. Variability will play a key role in this new combined paradigm.

6.2 Variability and NFR

When we look at the dimension concerning the functional and non-functional requirements, we see that the NFR are much less studied than their functional counterparts. NFR have a major influence on the architecture of the applications, because the architecture of an application determines the level of the NFR. It is necessary to look at the NFR as soon as possible to prevent expensive changes at later stadia of application development. This is all common knowledge in classical application engineering. In a SPL context, a domain architecture is created based on the commonality and variability of the FR. At the same time this domain architecture also partly delimits the value range of the NFR, but these NFR are not considered during the creation of the

domain architecture. The gap lies in the fact that NFR are ignored most of the time in domain engineering, while domain engineering does impact them. It goes even further, it is possible that there exist dependencies between NFR and FR, like there are dependencies between features relating to FR. Investigating interdependencies between FR and NFR variability is another potential research area.

6.3 Variability in time

Variability in time is another part of variability that is not sufficiently investigated, especially in combination with its variability in space counterpart. A typical problem is the question of evolution of family members in terms of shared features: when one of the family members requires a change to a feature shared with other family members, do they all need to change as well? Combining software evolution with variability in space is another potential research domain that is not yet sufficiently covered. When there is a significant variability in space, the variability in time of the different applications becomes more complex. The different applications have their own trajectory over time, but remain linked to each other as they are members of the same family.

6.4 Variability scoping

A last observation concerns the variability scoping dimension. Variability is mostly looked upon at the level of product families. That other levels of scope for variability are studied to a lesser extent is the consequence of the fact that variability needs to be scoped carefully with respect to reusability. Too much variability leads to high reuse potential but low actual reuse due to the amounts of work needed at application engineering level. Too little variability leads to low reuse potential and therefore also low reuse. That most research positions itself at the level of product family is an indication that this is the optimal level for reusability maximization. Here, the research challenge is to explicitly design a tradeoff mechanism between variability and reuse efficiency.

7. CONCLUSION AND FUTURE WORK

Variability remains a concept that is not yet sufficiently addressed by the current state-of-the-art literature. In particular, our systematic literature review demonstrated a number of gaps that need to be addressed further. Due to a lack of good definitions on the variability related concepts in paradigms like software product lines and service oriented computing, parts of the concept remain vague and underexposed. This is very intriguing because variability is an important inductor of reusability. In this literature review we uncovered the dimensions of variability, which are at the same time the possible views from different stakeholders, based on a set of definitions inferred from the reviewed papers. With the literature review performed and the dimensions identified we uncovered the aspects of variability which are ignored in current literature. Based on the previous section, we conclude that the concept of variability still needs further study. A variability concept should start from the SOPL combined paradigm and cover both NFR and FR. Variability needs to be managed in space and time at simultaneously. Only when variability is developed in such a way that it covers the full extent of all aforementioned dimensions, variability will induce the highest possible level of reusability in software engineering. That is why variability in all his aspects is a important topic in our future research.

8. ACKNOWLEDGEMENTS

We gratefully acknowledge KBC Global Services NV (member of the KBC Group²) for funding this research through the KBC research chair "Developing and Managing Business Services as Shared Assets".

9. REFERENCES

- [1] Acher, M., Lahire, P., Moisan, S., and Rigault, JP., Tackling High Variability in Video Surveillance Systems through a Model Transformation Approach. in *ICSE Workshop on Modeling in Software Engineering*, (Vancouver, Canada, 2009), IEEE Computer Society, 44-49.
- [2] Alves, V., Niu, N., Alves, C., and Valença, G., Requirements Engineering for Software Product Lines: A Systematic Literature Review. *Information and Software Technology*, 52 (8), 806-820.
- [3] Bosch, J., Florijn, G., Greefhost, D., Kuusela, J., Obbink, J.H., and Pohl, K., Variability Issues in Software Product Lines. in *International Workshop on Software Product-Family Engineering*, (Bilbao, Spain, 2001), Springer-Verlag, 13-21.
- [4] Botterweck, G., Thiel, S., Cawley, C., Nestor, D., and Preußner, A., Visual Configuration in Automotive Software Product Lines. in *International Computer Software and Applications Conference*, (Turku, Finland, 2008), IEEE Computer Society, 1070-1075.
- [5] Bu-Qing, C., Bing, L., and Qi-Ming, X., A Process-Driven and Ontology Based Software Product Line Variability Modeling Approach. in *International Conference on Grid and Cooperative Computing*, (Lanzhou, Gansu, China, 2009), IEEE Computer Society, 385-390.
- [6] Capilla, R., and Topaloglu, N.Y., Representing Variability Issues in Web Applications: A Pattern Approach. in *International Symposium on Computer and Information Sciences*, (Antalya, Turkey, 2003), Springer-Verlag, 1035-1042.
- [7] Chen, L., Babar, M.A., and Ali, N., Variability Management in Software Product Lines: A systematic Review. in *Software Product Line Conference*, (San Francisco, California, USA, 2009), conference proceedings, 81-90.
- [8] Clements, P., and Northrop, L. *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing, Boston, 2001.
- [9] Dhungana, D., and Groher, I., Genetics as a Role Model for Software Variability Management. in *International Conference on Software Engineering*, (Vancouver, Canada, 2009), IEEE Computer Society, 239-242.
- [10] Gawley, R., Automating the Identification of Variability Realisation Techniques from Feature Models. in *International Conference on Automated Software Engineering*, (Atlanta, Georgia, USA, 2007), ACM, 555-558.
- [11] Gomaa, H., and Webber, D.L., Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model. in *Hawaii International Conference on System Sciences*, (Big Island, Hawaii, USA, 2004), IEEE Computer Society, 1-10.
- [12] Griss, M.L., Favaro, J., and Alessandro, M., Integrating Feature Modeling with the RSEB. in *International Conference on Software Reuse*, (Victoria, BC, Canada, 1998), IEEE Computer Society, 76-85.
- [13] Guelfi, N., and Perrouin, G., A Flexible Requirements Analysis Approach for Software Product Lines. in *Requirements Engineering: Foundation for Software Quality*, (Trondheim, Norway, 2007), Springer-Verlag, 78-92.
- [14] Haesen, R., *Designing Information System Services in Information-Intensive Organizations*. PhD dissertation. KU Leuven, 2009.
- [15] Hartmann, H., and Trew, T., Using Feature Diagrams with Context Variability to Model Product Lines of Software Supply Chains. in *International Conference on Software Product Line* (Limerick, Ireland, 2008), IEEE Computer Society, 12-21.
- [16] Hoyer, C., and Chroust, G., Evolving Standard Process Reference Models of Product Line Development. in *Conference on Software Engineering and Advanced Applications*, (Cavtat, Dubrovnik, Croatia, 2006), IEEE Computer Society, 320-327.
- [17] Huang, S., Shankar, R., and Mangs, J., Towards Strategic Design Reuse by Leveraging Commonality and Managing Variability. in *IEEE Systems Conference*, (Waikiki Beach, Honolulu, Hawaii, USA, 2007), IEEE Computer Society, 1-4.
- [18] Hubaux, A., Heymans, P., and Benavides, D., Variability Modelling Challenges from the Trenches of an Open Source Product Line Re-Engineering Project. in *International Conference on Software Product Line* (Limerick, Ireland, 2008), IEEE Computer Society, 55-64.
- [19] Jaring, M., and Bosch, J., Architecting Product Diversification – Formalizing Variability Dependencies in Software Product Family Engineering. in *International Conference on Quality Software*, (Braunschweig, Germany, 2004), IEEE Computer Society, 154-161.
- [20] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S., Feature-Oriented Domain Analysis (FODA) feasibility study. *Software Engineering Institute, Carnegie Mellon University*.
- [21] Keepence, B., and Mannion, M., Using Patterns to Model Variability in Product Families. *IEEE Software*, 16 (4), 102-108.
- [22] Kim, S.D., Her, J.S., and Chang, S.H., A Theoretical Foundation of Variability in Component-Based Development. *Information and Software Technology*, 47 (10), 663-673.
- [23] Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., and Linkman, S., Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology* 51 (1), 7-15.
- [24] Kurakawa, K., Feature Modeling from Holistic Viewpoints in Product Line Engineering. in *Asia-Pacific Software Engineering Conference*, (Busan, Korea, 2004), IEEE Computer Society, 579-579.
- [25] La, H.J., and Kim, S.D., A Systematic Process for Developing High Quality SaaS Cloud Services. in *International Conference on Cloud Computing* (Beijing, China, 2009), Springer-Verlag, 278-289.
- [26] Laplante, P.A., Zhang, J., Voas, J., What's in a Name? Distinguishing between SaaS and SOA. *IT Professional*, 10 (3), 46-50.
- [27] Lee, J., Kotonya, G., and Robinson, D., A Negotiation Framework for Service-Oriented Product Line Development. in *International Conference on Software Reuse*, (Falls

² <https://www.kbc.com/>

- Church, Virginia, 2009), Springer-Verlag, 269-277.
- [28] Lee, K., and Kang, K.C., Feature Dependency Analysis for Product Line Component Design. in *International Conference on Software Reuse*, (Madrid, Spain, 2004), Springer-Verlag, 69-85.
 - [29] Lee, K., Kang, K.C., Kim, S., and Lee, J., Feature-Oriented Engineering of PBX Software. in *Asia-Pacific Software Engineering Conference*, (Takamatsu, Japan, 1999), IEEE Computer Society, 394-403.
 - [30] Lee, K., Kang, K.C., and Lee, J., Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. in *International Conference on Software Reuse*, (Austin, Texas, USA, 2002), Springer-Verlag, 62-77.
 - [31] Lee, S.B., Kim, J.W., Song, C.Y., and Baik, D.K., An Approach to Analyzing Commonality and Variability of Features using Ontology in a Software Product Line Engineering. in *International Conference on Software Engineering Research, Management & Applications*, (Busan, South Korea, 2007), IEEE Computer Society, 727-734.
 - [32] Lee, Y., Yang, C., Zhu, C., and Zhao, W., An Approach to Managing Feature Dependencies for Product Releasing in Software Product Lines. in *International Conference on Software Reuse*, (Torino, Italy, 2006), Springer-Verlag, 127-141.
 - [33] Lin, Y., Ye, H., and Li, B., A New Parameter for Product Configuration in Software Product Lines. in *International Symposium on Knowledge Acquisition and Modeling*, (Wuhan, China, 2009), IEEE Computer Society, 230-233.
 - [34] Metzger, A., Heymans, P., Pohl, K., Schobbens, P.Y., and Saval, G., Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis. in *International Requirements Engineering Conference*, (Delhi, India, 2007), IEEE Computer Society, 243-253.
 - [35] Mietzner, R., Metzger, A., Leymann, F., and Pohl, K., Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications. in *ICSE Workshop on Principles of Engineering Service Oriented Systems*, (Vancouver, Canada, 2009), IEEE Computer Society, 18-25.
 - [36] Moon, M., Chae H.S., and Yeom, K., A Metamodel Approach to Architecture Variability in a Product Line. in *International Conference on Software Reuse*, (Torino, Italy, 2006), Springer-Verlag, 115-126.
 - [37] Moros, B., Vicente-Chicote, C., and Toval, A., REMM-Studio+: Modeling Variability to Enable Requirements Reuse. in *International Conference on Conceptual Modeling*, (Barcelona, Spain, 2008) Springer-Verlag, 530-531.
 - [38] Narendra, N.C., Ponnalagu, K., Srivastava, B., and Banavar, G.S., Variation-Oriented Engineering (VOE): Enhancing Reusability of SOA-based Solutions. in *International Conference on Services Computing*, (Honolulu, Hawaii, USA, 2008), IEEE Computer Society, 257-264.
 - [39] Nguyen, Q.L., Non-Functional Requirements Analysis Modeling for Software Product Lines. in *ICSE Workshop on Modeling in Software Engineering*, (Vancouver, Canada, 2009), IEEE Computer Society, 56-61.
 - [40] Papazoglou, M.P., Service-Oriented Computing: Concepts, Characteristics and Directions. in *International Conference on Web Information Systems Engineering*, (Rome, Italy, 2003), IEEE Computer Society, 3-12.
 - [41] Pohl, K., and Metzger, A., Variability Management in Software Product Line Engineering. in *International Conference on Software Engineering*, (Shanghai, China, 2006), ACM, 1049-1050.
 - [42] Polzer, A., Kowalewski, S., and Botterweck, G., Applying Software Product Line Techniques in Model-Based Embedded Systems Engineering. in *ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, (Vancouver, Canada, 2009), IEEE Computer Society, 2-10.
 - [43] Reiser, M.O., Kolagari, R.T., and Weber, M., Manifolds of Variability Modeling – Considering the Potential for Further Integration. in *Central and East European Conference on Software Engineering Techniques*, (Poznan, Poland, 2007), Springer-Verlag, 291-303.
 - [44] Sinnema, M., and Deelstra, S., Classifying Variability Modeling Techniques. *Information and Software Technology* 49 (7). 717-739.
 - [45] Stoiber, R., Meier, S., and Glinz, M., Visualizing Product Line Domain Variability by Aspect-Oriented Modeling. in *International Workshop on Requirements Engineering Visualization*, (New Delhi, India, 2007), IEEE Computer Society, 8-13.
 - [46] Svahnberg, M., van Gurp, J., and Bosch, J., A Taxonomy of Variability Realization Techniques. *Software – Practice & Experience* 35 (8). 705-754.
 - [47] Thurimella, A.K., Bruegge, B., and Creighton, O., Identifying and Exploiting the Similarities between Rationale Management and Variability Management. in *International Conference on Software Product Line*, (Limerick, Ireland, 2008), IEEE Computer Society, 99-108.
 - [48] Tian, Q.M., Chen, X.Y., Jin, L., Pan, P., and Ying, C., Asset-Based Requirement Analysis in Telecom Service Delivery Platform Domain. in *Network Operation and Management Symposium*, (Beijing, China, 2008), IEEE Computer Society, 815-818.
 - [49] von der Maßen, T., and Lichter, H., Requiline: A Requirements Engineering Tool for Software Product Lines. in *International Workshop on Product-Family Engineering*, (Siena, Italy, 2004), Springer-Verlag, 168-180.
 - [50] Ye, H., and Liu, H., Approach to Modelling Feature Variability and Dependencies in Software Product Lines. *IEEE Proceedings Software*, 152 (3). 101-109.