International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

A case study on variability management in software product lines: identifying why real-life projects fail

Tom Huysegoms
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/tom.huysegoms@kuleuven.be

Monique Snoeck
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/monique.snoeck@kuleuven.be

Guido Dedene
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/guido.dedene@kuleuven.be

Antoon Goderis
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels, Belgium
www.shortbio.net/antoon.goderis@kuleuven.be

Frank Stumpe
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels, Belgium
www.shortbio.net/frank.stumpe@kuleuven.be

Abstract:
Economies of scale can be seen as some kind of "holy grail" in state of the art literature on the development of sets of related software systems. Software product line methods are often mentioned in this context, due to the variability management aspects they propose, in order to deal with sets of related software systems. They realize the sought-after reusability. Both variability management and software product lines already have a strong presence in theoretical research, but in real-life software product line projects trying to obtain economies of scale still tend to fall short of target. The objective of this paper is to study this gap between theory and reality through a case study in order to see why such gap exists, and to find a way to bridge this gap. Through analysis of the causes of failure identified by the stakeholders in the case study, the underlying problem, which is found to be located in the requirements engineering phase, is crystallized. The identification of a framework describing the problems will provide practitioners with a better focus for future endeavors in the field of software product lines, so that economies of scale can be achieved.

Keywords:
software product lines; variability management; harmonization & variabilization; requirements engineering; case study.

DOI: 10.12821/ijispm010103

Manuscript received: 28 February 2013
Manuscript accepted: 18 March 2013

1. Introduction

Variability between related software systems offers a challenge in an otherwise dull world where each software system would be the same as the previous one. At the same time, such variability can quickly turn into a nightmare for those who need to develop or maintain a set of related software systems. While in theoretical research terms like variability management and software product lines are commonplace, in practice there are still a lot of companies that struggle with variability and with the problems variability brings about. Previous research [1] through an exploratory study in a software intensive company revealed that variability did induce difficulties in practice, so an interesting problem in the domain of software engineering was identified. How is it possible that companies still struggle with variability while in the academic research field there is an abundance of theoretical studies about the problems that can be encountered in practice, and more important, how can companies be offered a way to overcome their problems? The research question that is addressed in this research is the following: "What are possible reasons behind the failures in practice of projects that develop sets of related software systems in order to obtain economies of scale, and how can these problems be resolved?"

Using a case study as the start of our research ensures to a certain extent that the results obtained are usable in real companies. In the rest of this paper, findings on a real life case conducted in a large scale bank and insurance company are analyzed and reported. The second section positions the research against related work. Section 3 discusses the methodological approach opted for. Section 4 describes the context of the empirical case study. The fifth section identifies the different kinds of stakeholders and presents the results of the case study. These results serve as starting point to crystallize the problems impeding successful variability management and achieving economies of scale in practice. Section 6 provides the description of a framework representing the variability decisions that need to be taken during the phase of requirements engineering, which is the phase where the problem is situated according to the case study results. The seventh and last section recaps the research questions addressed and draws the conclusions.

2. Related work

In the past decades substantial research effort has been devoted to solving the issues concerning variability in related software systems. Out of this effort multiple ways to manage variability have been developed. As early as in 1990 variability management was already identified by Kang et al. [2] as part of Feature-Oriented Domain Analysis (FODA). Variability management is defined by Schmid and John [3] as encompassing the activities of explicitly representing variability in software artifacts throughout the lifecycle, managing dependencies among different variabilities, and supporting the instantiation of the variabilities. The need for a clear understanding of variability in the context of requirements engineering is quite obvious, as not only research on the theoretical concept of variability itself is done [4], research on related subjects like e.g. coping with preferences in requirements [5] suggests the same need for a clear understanding of variability. Following the seminal work of FODA, multiple variability management approaches have been developed. Some of these were extensions of the original FODA specification like e.g. FORM [6]. Other approaches were developed more independently like KobrA [7] and COVAMOF [8]. More recent variability management approaches that have received considerable attention can be to some extent retraced to this core strand of research. The Orthogonal Variability Model (OVM) approach by Pohl et al. [9] provides a good example of this. For a more complete overview of the variability management literature we refer the reader to Chen et al. [10], which revises the field of variability management. In summary, it can be stated that the variety and plethora of available approaches indicate that significant attention has been given to the issue by the research community.

The notion of software product lines, is closely related to the issues considered in the strand of research described in the previous paragraph. Software product lines have been introduced in literature at about the same time as variability management has been inducted in research. The much referenced work of Clements et al. [11] defines a software product line as follows: "*A software product line is a set of software-intensive systems that share a common, managed set of features satisfying 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". The Carnegie Mellon Software Engineering Institute defines it as follows on its website [12]: "*A software product line epitomizes strategic, planned reuse. More than a new technology, it is a new way of doing business. Organizations developing a portfolio of products as a software product line are experiencing order-of-magnitude improvements in cost, time to market, and productivity*". Both quotations together stress the major things about a set of related software systems that need to be considered. A software product line is a portfolio or set of relating software systems, also called a product family, which is developed with a mission to improve the way to do business. It therefore affects business and goes beyond its Information and Communication Technology (ICT) context. A software product line has a focus on a specific market segment, in which the software systems are closely related so economies of scale of all sort can be obtained. It can thus be said that software product lines are the sets of software systems that are developed in order to obtain economies of scales.

The most important thing in a software product line is the variability amongst the members of the set, as this is the contrasting factor between the instances of the software product line. The management of this variability is key to make the software product line a success. If the basis, on which the instances are built, the core asset, offers the proper amount of customization through variability, then the product line can reap the benefits attributed to it in literature. The importance of variability in software product lines also explains the relatedness of software product lines research with variability management research.

3. Research methodology

In the domain of software engineering, it was acknowledged by Moody et al. [13] that there exists a problem of knowledge transfer between theory and practice, also known as the technology transfer problem. Kaindl et al. [14] argued that such gap also exists in the domain of requirements engineering, the phase that should primarily deal with variability problems. These studies propose a number of approaches to bridge this technology transfer gap. The objective of all these approaches is to bring theoretical research closer to real life by providing a knowledge transfer channel. The research in this paper starts from a real life case in order to develop and explain theoretical knowledge. By doing so the knowledge transfer channel is present right from the start of the research. Through case study research [15], an empirical research technique, it is largely ensured that any results found are also of value in a practical context. It can be argued that there already have been several case studies in the domain of software product lines, such as those by van der Linden et al. [16], but as stated by Tichy [17], each additional case study is not just a repetition but aims to extend the knowledge previously developed. Our case study proves to be a valuable extension to the body of knowledge already available in that it starts from a '*failed'* project instead of a success story like most of the case studies in this field of research. This failure provided us the perfect opportunity to extract lessons learned valuable to anyone who wishes to undertake a similar project in the future.

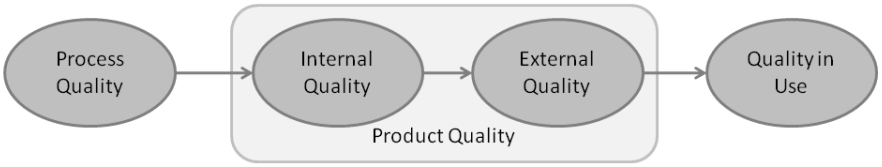

Fig. 1. Categories of Quality in the Product Quality Measurement Reference Model

In order to assess the software product studied in our case, a framework to check the quality of software products was searched for. The ISO/IEC 25010:2011 Software Quality Requirements and Evaluation (SQuaRE) is a series of ISO 9000 compliant standards developed by ISO (www.iso.org), based on the older ISO/IEC 9126 framework and used for evaluating the quality of software products. The Product Quality Measurement Reference Model of the Quality

Measurement Division (2502n) identifies four quality categories of software products related to one another as depicted in Fig. 1. These categories will be used in our case study.

The case study results were processed based on Glaser and Strauss's grounded theory [18]. Starting from a set of open-ended interviews, the transcripts were coded with a qualitative analysis software program to obtain a set of concepts representing the causes of failure of the project according to the interviewees. These concepts were then grouped into categories, on the one hand based on the context of the project itself and on the other hand based on the SQuaRE framework. The project context consisted of a business part, an ICT part and an external (or client) part, and each part was given a separate category in the case specific categorization. The SQuaRE framework categorization corresponds to the quality types defined above. The results of the case study can be found in section five. Before discussing the results, the context of the case is described in the next section.

4. The FinForce case

The case study of this research is based upon a project called FinForce (FF). FF was founded in 2001 as a spin-off company of KBC Group, a European bank and insurance multinational. FF consisted mainly of employees who were transferred from KBC ICT. These employees had, to some extent, experience in the fields of both professional and/or retail payment services. The services FF provided were aimed worldwide towards banks that wanted to outsource their international payment transactions. The business case of FF was based on the fact that international payment transactions were complex, albeit standardized to a certain extent. Before the advent of the Single Euro Payments Area (SEPA) initiative such international transactions were very costly for the banks providing them. Taking advantage of economies of scale, FF would have been able to provide a transaction engine to all banks at a reduced price compared to the in house transaction engines while maintaining a considerable profit margin. The unique selling proposition of FF was being adaptable to any situation possible, hence the tagline in the logo "flexible financial back-office solutions".

The application which was used by FF consisted of four different components, each one with its specific task. There were two main components, called the PE (payment engine) and the PSE (professional service engine), which provided functionalities for respectively the retail and the professional part of the transactions. Besides these main parts, FF was completed with a SIM (smooth interface messaging) component and a TSS (transaction support services) component. The purpose of the SIM component was to provide communication from inside the application towards the external interfaces of the client banks. Thanks to the SIM component client banks could retain their own interfaces. The SIM transformed messages from the FF internal format to the client specific format. The TSS component retained all information needed for the international transactions like e.g. account numbers. Combining these parts provided a complete solution to the client banks for their foreign transactions. The fact that FF consisted of several separate components originated from the fact that although FF was developed from scratch, it was partly based on existing applications inside KBC. Applications like "Pay & Receive" and "Multi-clearing" stood model for respectively the PE and the PSE. An instantiation of the FF application was built for each client with the required adjustments for the interfacing. As such a set of related software systems was created, with only some small differences between the different instantiations.

5. Case study results

To start with, three interviews were conducted with employees who worked for FF, each of them working in a different division, with a different angle on the project. The different angles were carefully selected in order to obtain the most complete overview of FF as possible. To find suitable employees for interviewing, all possible aspects of FF were listed, which resulted in three clearly distinct angles or views, namely the architect or design side view, the business side view and the implementation side view. The setup of the interviews was open ended, as already discussed in the section on research methodology, so the grounded theory approach could be applied afterwards. Some guiding questions were defined, but these merely served the purpose of being some kind of bootstrap. Once the interviews were underway for a couple of minutes, the interviewees continued most of the time out of their own.

During the processing of the three interviews the validity of our choice of angles was enforced as one of the interviewees explained the three organizational pillars of FF, which coincided to a great extent with the views identified a priori. Also some initial observations were made on the causes of failure stated by the interviewees in order to check whether the assumption on the choice of viewing angles could be retained. The main observation was that a significant proportion of the causes of failure were situated in the requirements engineering part of the software engineering process. With this observation the different angles of the interviews were reviewed based on the roles identified in the Volere requirements process [19]. These roles are called supplier (who develops the software), client (who pays for the development), customer (who buys the product after development) and user (who will actually use the software) roles. Besides these, in Volere identified roles a fifth important role also needs to be taken into account. This fifth role is the role of the requirements engineer him/herself. The requirements engineer is responsible for the requirements engineering phase. He has a mediating role between the other roles identified in Volere.

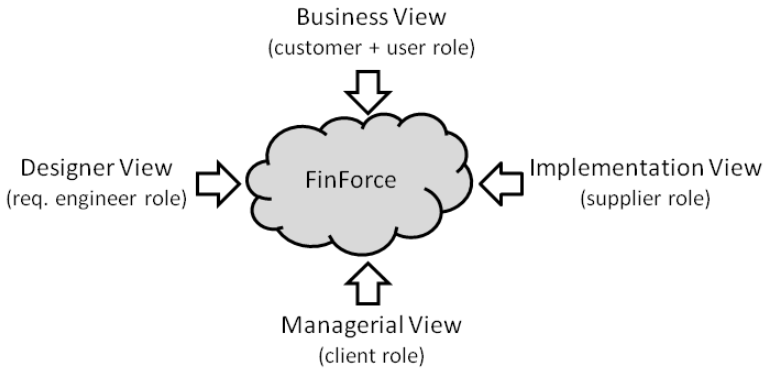

Fig. 2. The different views on FinForce

The five roles as they are identified in the previous paragraph were linked to the viewing angles of the three interviews. The architect's or designer's view in FF can be easily related to the requirements engineer role since it was the designer in FF that was responsible for deciding which requirements to implement. The business side view of FF can be related to the customer role since it was business side that contacted potential customers to relay their requirements to the designers. The business side view can additionally be indirectly related to the user role, because the users are contacted through the customers by whom they are represented. The implementation side view finally can be related to the supplier role as it was the implementation side which was responsible for building the software. Only the client role as identified in Volere cannot be mapped to any of the three views, and subsequently not to any of the three interviews. Therefore it was decided to conduct a fourth interview with someone of FF from a managerial viewpoint as management decided on budget for FF. As such all the roles of Volere are covered, and the assumption that all viewpoints on FF are covered is validated. An overview of the views and the roles can be found in Fig. 2. In the following paragraphs each interviewee's personal opinion will be explored in detail separately before making more general conclusions on the causes of failure for FF.

5.1 The design side view from the technical analyst

The technical analyst consulted did not describe the FF project as a success, but at the same time he stated that "it could not be seen as a complete failure neither". The main problem the technical analyst identified in FF was an issue of scoping. This scoping issue surfaced in several ways throughout the FF case. It started as soon as the requirements engineering was finalized. The requirements were only defined at a very high level of abstraction, without any details. The system needed to be flexible such as to be usable for all possible client banks, but to what extent this flexibility had

to be implemented was unclear. FF was created by several analysts simultaneously, but each analyst implemented the variability as he or she saw fit. Because a lack of central coordination or carefully created delimitations in the requirements, one analyst would build in variability with much ardor by making very small components that could be combined in numerous ways while another analyst would build more coarse grained components only leaving a few variability options. At implementation time, the creation of FF took too much time because of the issue of over-flexibility and the related issue of tiny components, as most of the analysts believed that small components were mandatory to make FF as flexible as possible.

A lot of the flexibility that was built into the FF application by the analysts proved to be useless at implementation time. This was also due the fact that the analysts did not possess profound business knowledge. The business side of FF at the same time defined the requirements of FF solely based on former KBC experiences as KBC was the only customer at the start of the project. There was no knowledge on how other banks (potential customers) processed their payment transactions. The technical analyst suggested that it might have been better that some reference visits had been conducted to these other banks in order to have a better view on the variability issues. The problem with such visits is that no bank was very eager to share its way of doing business. A last point made by the technical analyst was that once FF was built and the business had to sell the application, the vendors were too accommodating towards possible customers as they promised them that each of their wishes would be included in FF. Because of this, every time a customer was to be connected, a great deal of adjustments was needed in the FF application.

5.2 The implementation side view from the system administrator

The system administrator interviewed for this case study was mainly responsible for the SIM component of FF and therefore had a good overview on the interfacing issues. According to the system administrator these interfacing problems were plentiful, and moreover they were the single factor that caused each connection of a new customer bank to take much more time than expected. As a result of the interfacing issues, the time period it actually took to connect a new customer bank was 2.5 times bigger than originally estimated. Moreover, according to the system administrator, the problem of non-matching interfaces was a result of bad communication between the representatives of the customer banks and the people who needed to implement the application. After an agreement with a customer bank had been obtained by the vendors, were the same vendors the responsible for the extraction of user specific requirements. Although these user specific requirements were always discussed and extracted by the vendors, the results of these communications were never clearly passed to the people responsible for the actual implementation. Therefore things were wrongly assumed from both sides, what resulted in problems in the mapping of interfaces.

Another somewhat related issue cited by the system administrator is the issue of staff turnover. People who worked on the instantiation for and the connection of a particular customer bank were rarely involved in projects for other banks. Therefore any knowledge developed in the course of one connection was lost most of the time, as this knowledge was mainly tacit of nature. In combination with the fact that there was some kind of unofficial policy to only address problems if they actually occurred even if they could be envisioned up front, made learning from previous experiences hard. The system administrator told us that nowadays a sort of script is developed that can be followed for future instantiations as a way to transfer experience. This script is however only used to connect KBC Group subsidiaries, as FF is only used for subsidiaries nowadays.

5.3 The business side view from the client acquisition officer

The business side of FF was explained to us by one of the client acquisition officers, who were the vendors of FF. Also on the business side there were several issues which impacted the results of the FF project. First of all there was another communication problem. Unlike the communication problem between implementers and customers described earlier, this problem was situated between the customer banks and the customer acquisition team from FF. Potential customer banks were approached by the customer acquisition team and they proposed to the banks the FF business model as it was seen and implemented by ex-KBC staff. The difficulty herein was that potential customers had to understand the KBC oriented context in which the business model was written, and link this to their own business model. If customers

had to adapt to another context in order to be able to state their own needs, communication got difficult at least or customers even could altogether lose their interest in FF.

Another serious and similar obstacle in obtaining customers was the KBC background of FF. Potential customer banks were not fond of disclosing sensitive information to what they believed a competitor in the bank market. This is best illustrated with a small anecdote from the interviewed client acquisition officer. He told us: "When we approached banks and we wanted to give a presentation, upon booting our laptops the first thing appearing on the screen would be a giant KBC logo, as our equipment came from KBC ICT. Clients would ask straight away what the logo meant, and the first seeds of distrust would be sown".

5.4 The managerial side view from the Chief Executive Officer

The interview with the former Chief Executive Officer (CEO) of FF gave a good overview on how FF came to be, since the interviewee was involved in FF from cradle to grave. He described that while in the end FF was incorporated back into KBC, some benefits still remain present in KBC. Since it was the objective to sell FF towards external clients, it was necessary to document everything as clearly as possible. This extensive documentation still proves to be valuable to KBC. Besides this documentation advantage there were no further major benefits according to the CEO. The issues concerning FF on the other hand were plentiful. One of the main reasons of failure according to the CEO was that standardization was not part of the corporate culture of any bank. Apart from regulations that are imposed by higher authorities, there was no way that all banks would agree that FF was providing the standard way of working. The fact that the focus of FF shifted constantly during the project only added to the confusion and the diversity of the application. For example, at one point in time SEPA-transactions were to be included, while at another point in time they had to be excluded at all costs.

The CEO furthermore mentioned that one of the most characteristic properties of FF was that it provided a very rich and diverse set of functionalities, which seems to be in direct contrast with the will to standardize. The existence of such a contrast was only possible due to an unfocused way of working. Each time some functionality was created it was just added to the complete system without looking at a way to make the new functionality fit to the rest of the system. This led to a system with enormous possibilities which could never be fully used due to the fact that the system was one big clutter.

5.5 Global results of the case study

The previous paragraphs shed some light on interesting particularities for each of the views on the project, but nevertheless the overarching concern relates to the way variability was addressed during requirements engineering. As already mentioned, a big problem was that there was variability on certain places in FF where it should not have been, and there were places where there was no variability but where it was needed. This resulted from a flawed vision on the system, and as much from poor requirements engineering. According to the higher management who decided to create the FF spin-off company, FF was meant to be the application that would set the standard for all international payments transactions. Pre-studies conducted in corporation with renowned consulting firms spoke of worst case scenarios requiring at least 30 client banks within three years. It can be argued that the crisis around 9/11 had an unpredictable impact on the business case, but nevertheless the order of magnitude envisioned was in sharp contrast with the final number of external banks that were connected to FF before the project got drastically clipped in 2007-2008. The overly positive attitude towards FF created the need that FF had to be able to work for everyone and as soon as possible. The broad scope required loads of variability, while the short time span made it impossible to think the project through thoroughly before starting to develop the software system. Apart from some most basic requirements gathering, the requirements engineering phase was thus more or less completely skipped. Variability was implemented without any guidelines. The result was that the application was capable to handle a very broad scope, but at the same time making one instantiation for a particular client took so much time to develop that the whole system's costs were way higher than it could possible gain as benefits by its broad scope.

In order to perform a more systematic analysis of the root causes of failure, all the 51 causes of failure mentioned in the interviews were listed and categorized as explained in section 3. The results from this categorization are to be found in table 1. Some of the failure cause extracted from the interviews could not easily be categorized as can be seen in the "Various" row in table 1. These 15 concepts actually overlap the “Business” and “ICT” categories in the categorization. Looking at the SQuaRE based categorization; we see that 7 concepts do not really categorize themselves into a certain quality. These concepts state facts that do not really have a quality aspect to them, therefore they are labeled “No Quality”.

Table 1. The failure concept categorization

	Process Quality	**Internal Quality**	**External Quality**	**Quality in Use**	**No Quality**	**Total**
Business	9	0	3	0	3	**15**
ICT	10	1	2	0	0	**13**
External	2	0	1	3	2	**8**
Various	5	1	4	3	2	**15**
Total	**26**	**2**	**10**	**6**	**7**	**51**

A lot of the failure concepts are process quality related. Inside the ICT category even around 75% of the failure concepts are categorized as process quality. This leads to the conclusion that the quality of the development process should receive enough attention in order to be more successful. Business and ICT have about the same importance as they account for almost the same amount of failure concepts. This validates that business and ICT should work together as equal partners in order to have success in software projects like the FF project. The need for a focus on process quality along with the need for a good way to deal with variability during requirements engineering led to the development of a theoretical framework describing the issues at hand from the observations made in the FF case. The resulting framework is however usable in any context were (sets of) software systems are made and variability is significantly present. The framework described in the next section provides a focus that is not only useful in the development of such systems, but that should be reasoned about explicitly in order to evade failures like the ones present in FF.

6. The Harmonization and Variabilization framework

The proposed framework consists of two central decisions key to variability management in requirements engineering and is explained with the Volere roles in mind. The first decision concerns the harmonization of requirements. The user, client and customer stakeholders within a certain software system context form the demand side of that software system. Once all the requirements of the demand side have been elicited, an analysis of the (dis)similarity of these requirements can start. Some requirements can be the same for everyone (the common part, or the commonality), while other requirements can differ between demand side members (the variable part, or the variability). It is up to the requirements engineer to find the similarities and differences in the requirements and to confront the clients, customers and users with these (dis)similarities.

As such, harmonization is much more than just requirements gathering and elicitation. At this point a requirements engineer can attempt to convince users, through the customers, to assimilate each other's requirements so that the amount of variability that needs to be dealt with is reduced. One can start from the similarities in the requirements and try to expand these by convincing users that they do not need the differentiating parts of the requirements. The supplier and customer can, for example, attempt to reduce variability by trying to set standards. This balancing is not straightforward as each user wants the to-be-created software system to be tailored as much as possible to his personal

context, while at the same time the client, who is the stakeholder that pays for the development, support and maintenance of the to-be-created software system will rather be in favor of diminishing the variability in order to cut down the costs of development, support and maintenance. The customer will also be affected by the amount of variability. He will be charged a price by the client based on the costs made by the client. Too much variability holds the risk of a too high price for the customer. The amount of variability within the set of all users' requirements should thus not only be identified, but also be managed as well during the harmonization.

The second decision concerns the variabilization of requirements, and can be done once the harmonization is finished. The amount of variability originating from the diverging demand side's viewpoints cannot be altered anymore, but the support of this variability by the to-be-created software system is still to be decided upon. In the case of a supplier developing software for multiple customers, the client may decide that some variability will be taken into account into a project's requirements problem, while some variability (and the corresponding requirements) will be left aside for the customer to deal with in the specific contexts of the users.

Fig. 3 visualizes the harmonization (1) and variabilization (2) as setting boundaries within an octagon representing the union of all users' requirements. The harmonization divides the requirements into a part which is common for all users (commonality) and a part which differs for the users (variability). The variabilization further divides the variability part into a part that is sufficiently shared and will therefore be supported by the client (shared variability), and a part which is too specific to justify investments by the client and will thus be left to the customer to deal with (specific variability). Both harmonization and variabilization should be seen as a careful balancing exercise with the amount of requirements' variability at stake. The octagonal shape is used for the total set of demand side stakeholders' requirements to represent a sense of unlikelihood that the boundaries drawn by harmonization and variabilization are completely located at the edge of the requirements set. This would mean that no division in requirements is made by the harmonization and/or the variabilization, which is in most cases not optimal.

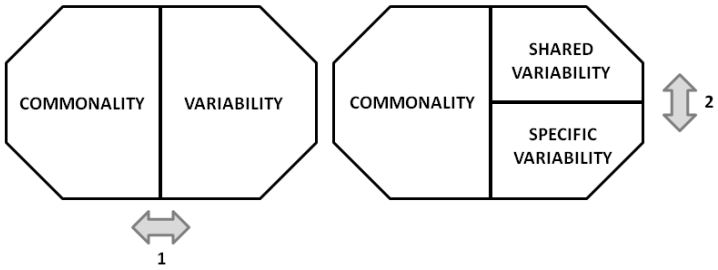

Fig. 3. The harmonization (1) and variabilization (2)

In the FinForce case, no explicit decisions were taken about harmonizing requirements or about variabilization. This led to a situation where there was little commonality and all variability was considered to be shared, or stated in terms of drawn boundaries, they were both drawn at the "edge of the octagon". As a result, the development cost of the software was raising high on the client's side, and moreover the cost for implementing the software at the users' side was raising high as well because of the huge amount of shared variability that needed to be dealt with during each implementation. Should both variability decisions as defined here have been taken consciously during the development of FinForce, the amount of possible causes of failure probably would have been reduced significantly.

7. Conclusion

Variability management is a complex process. The FinForce interviewees all mention this as causes of failure. As such, variability should be considered during the whole development project of a software system, especially when the system's success is critically impacted by variability management, such as in software product lines. The case study showed that it is not only a matter of creating a software product that supports variability, but that every bit of variability should be considered carefully from the requirements engineering phase on. When quality of the requirements engineering process is inferior, the product will not deliver the expected benefits. The decisions concerning the variability should be taken with due attention. Only when all parts of a company, both business and ICT, are ready to deal with variability, this variability management has the chance to succeed. Communication is of critical importance between users, implementers and developers as variability manifests itself in small details. Leaving room for interpretation can be fatal, as literally mentioned by one of the interviewees. Although it is the product that will be used eventually, it is the development process of the product where most problems are situated. This chimes with Deming [20] who states that the most effective way to improve the product quality is to improve the quality of the process that produces the product. For software systems that need to be flexible through variability, this may well be the only way to reach quality.

Based on the observations and knowledge extracted from the case study a framework was developed that describes the variability decisions that need to be made during the requirements engineering phase. The harmonization and the variabilization are the two decisions that should be taken with due attention in order to prevent the software system to come into troubles with the variability. First it must be decided how much variability should be present in the software system. After this, it must be decided whether the needed variability will be supported by the software system that is being created. When these two decisions are taken, a lot of issues, like the ones observed in FinForce, can possibly be avoided as the scope of the project would have been clear. As such the framework can provide the needed knowledge transfer channel towards practice in order to avoid tripping over the pitfall of variability.

References

[1] T. Huysegoms, M. Snoeck and G. Dedene, "Building a Requirements Engineering Methodology for Software Product Lines," in *5th SIKS/BENAIS Conference on Enterprise Information Systems*, Eindhoven, Nederland, 2010, pp. *25-34*.

[2] K. C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak and A.S. Peterson, "Feature-Oriented Domain Analysis (FODA): feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

[3] K. Schmid and I. John, "A customizable approach to full lifecycle variability management," *Science of Computer Programming*, vol. *53*, no. *3*, pp. *259-284*, December, 2004.

[4] S. D. Kim, S. J. Her and S. H. Chang, "A theoretical foundation of variability in component-based development," *Information and Software Technology*, vol. *47*, no. *10*, pp. *663-673*, July, 2005.

[5] S. Liaskos, S. A. McIlraith, S. Sohrabi and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering," *Requirements Engineering*, vol. *16*, no. *3*, pp. *227-249*, September, 2011.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. *5*, no. *1*, pp. *143-168*, January, 1998.

[7] C. Atkinson, J. Bayer and D. Muthig, "Component-Based Product Line Development: The KobrA Approach," in *First Product Line Conference*, Denver, Colorado, 2000, pp. *289-309*.

[8] M. Sinnema, S. Deelstra, J. Nijhuis and J. Bosch, "COVAMOF: A framework for modeling variability in software product families," in *Third Software Product Line Conference*, Boston, Massachusetts, 2004, pp. *197-213*.

[9] K. Pohl, G. Böckle and F.J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, 1st ed., New York, New York: Springer-Verlag, 2005.

[10] L. Chen, N. Ali Babar and N. Ali, "Variability management in software product lines: a systematic review," in *13th International Software Product Line Conference*, San Francisco, California, 2009, pp. *81-90*.

[11] P. Clements and L. Northrop, *Software product lines: practices and patterns*, 3rd ed., Boston, Massachusetts: Addison-Wesley, 2001.

[12] Software Engineering Institute, Carnegie Mellon. (2013, February). [online]. Available: http://www.sei.cmu.edu/productlines/

[13] D. L. Moody, "Using the world wide web to connect research and professional practice: Towards evidence-based practice," *Informing Science Journal*, vol. *6*, no. *1*, pp. *31-48*, January, 2003.

[14] H. Kaindl, S. Brinkkemper, J. A. Bubenko, B. Farbey, S. J. Greenspan, C. L. Heitmeyer, J. Sampaio do Prado Leite, N. R. Mead, J. Mylopoulos and J. Siddiqi, "Requirements engineering and technology transfer: Obstacles, incentives and improvement agenda," *Requirements Engineering*, vol. *7*, no. *3*, pp. *113-123*, September, 2002.

[15] R. K. Yin, *Case study research: design and methods*, 4th ed., Thousand Oaks, California: Sage, 2002.

[16] F. Van der Linden, K. Schmid and E. Rommes, *Software product lines in action: the best industrial practice in product line engineering*, 1st ed. Berlin, Germany: Springer, 2007.

[17] W. F. Tichy, "Hints for Reviewing Empirical Work in Software Engineering," *Empirical Software Engineering*, vol. *4*, no. *4*, pp. *309-312*, December, 2000.

[18] B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*, 1st ed. Chicago, Illinois: Aldine Publishing, 1967.

[19] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 2nd ed., Boston, Massachusetts: Addison-Wesley, 2006.

[20] W. E. Deming, *Out of the crisis*, 1st ed. Cambridge, England: MIT Press, 2000.

Biographical notes

Tom Huysegoms
Tom Huysegoms is a Ph.D. student in computer science at the Katholieke Universiteit Leuven in the Department of Decision Sciences and Information Management of the Faculty of Business and Economics. His research focuses on requirements engineering and variability management and is done in collaboration with KBC, an international bank and insurance company active in East and Central Europe.

www.shortbio.net/tom.huysegoms@kuleuven.be

Monique Snoeck
Monique Snoeck holds a Ph.D. in computer science from the Katholieke Universiteit Leuven. She is full professor in the Department of Decision Sciences and Information Management of the Faculty of Business and Economics of the Katholieke Universiteit Leuven and visiting professor at the Facultés Universitaires Notre Dame de la Paix, Namur. Her research focuses on conceptual modeling, requirements engineering, software architecture, model-driven engineering and business process management.

www.shortbio.net/monique.snoeck@kuleuven.be

Guido Dedene
Guido Dedene is a professor in management information systems in the Department of Applied Economics at the Katholieke Universiteit Leuven, Belgium. His lecturing and research activities concentrate on formal methods for systems development and quantitative systems management techniques. For five years, Prof. Dedene was also director of European services for GUIDE and SHARE Europe, an association of over 2,000 mainframe users.

www.shortbio.net/guido.dedene@kuleuven.be

Antoon Goderis
Antoon Goderis is an enterprise architect at KBC Global Services. He has a PhD in Computer Science from The University of Manchester.

www.shortbio.net/antoon.goderis@kuleuven.be

Frank Stumpe
Frank Stumpe holds a Ph.D. in project management in complex-cybernetic systems. He worked as consultant in project and change management and was lecturer at the RWTH Aachen for Project Management and holds different guest lecturers. The professional career of Frank Stumpe was international and cross industrial. Starting from managing European research projects he works for heavy industry, food industry as well as hotels and transport. He was CEO of IPS Bulgaria Ltd. and member of the Group-ExCo responsible for training and methods. Today working as head of the competence centre project based working at KBC Group.

www.shortbio.net/frank.stumpe@kuleuven.be