From Customizable to Configurable: An Experience Report in Risk Management Software

Gerard Quilty

A thesis submitted in part fulfilment of the degree of MSc Advanced Software Engineering in Computer Science with the supervision of Dr. Mel Ó Cinnéide.

School of Computer Science and Informatics

University College Dublin

28 April 2011

Preamble

The second part of this paper, contained in Appendix A, is the main part for consideration. It contains the paper "From Customizable to Configurable: An Experience Report in Risk Management Software" which has been submitted for publication in the Software Product Line Conference 2011 (SPLC'11). Currently it is conditionally accepted for that conference pending reworking. As the submission dates for submission to UCD and SPLC'11 do not match, the copy submitted here as Appendix A is the original version submitted to SPLC'11 and not the reworked version which is still in progress.

The first part of this paper includes background information on the Appendix and is included to highlight the effort involved in developing the concepts and data within the main submission. This paper is in no way meant to supersede the paper submitted to SPLC'11 and merely forms accompanying details to be considered for the fulfillment of the thesis requirement of MSc Advanced Software Engineering in Computer Science.

As the Appendix forms the main thrust of this submission it is recommended that the reader starts there before continuing on.

Table of Contents

I. INTRODUCTION

Blade is an Operational Risk Management System first developed by Ci3 Consultancy in 2000 and it is still in use and active development. It is the example used in this and the accompanying paper to highlight the differences between configurable systems and customizable systems. It started out as a standalone project for a single client, written in ASP with a Microsoft SQL Server backend. It was originally programed to a tight timeline and a lot of the base ASP code was taken from other projects. Ci3 Consultancy at this time was a bespoke development house. Over time the original customer wished to make changes to the Blade system. For some of these changes .NET technology was used. At the time it was a calculated risk. .Net promised improvements over classic ASP, however the development staff was not familiar with it at the time. The plan was to attempt some new functionality in .Net and if it turned out to be more trouble than benefit the overall project would not be affected. .NET was successfully and has since been used as the framework for all Blade development. Over the years various different parts of the original ASP functionality was redeveloped in .NET until in the last quarter of 2010 all remaining ASP code was removed. By this time Blade was being used by customers around in world in large banks, insurers, reinsurers and asset managers.

The method of managing new customers and their implementations has changed over the lifetime of Blade and this is reflected in the refactoring and development of the Blade codebase. During a customer implementation gaps between the functionality of the software and the customer requirements are identified. These gaps can be large like the requirement for a new scoring methodology to small like the requirement to save a person's date of birth. It is the responsibility of the business analyst in charge of the implementation to identify these gaps and propose solutions for them. These solutions fall into two categories, the first requires code changes and the second is already possible within the system however requires configuration. A lot of the solutions can be a mix of these two approaches.

This paper refers to the process of changing code to match the requirements of a single customer as customization. During the early years of Blade's life customization was used to fill most gaps and development and reuse was ad-hoc. This was due to the fact that there were no configurable assets in the system. When a product is being developed from a customer customizable approach, it is very hard to build in configuration for future customers. As the number of customers grew so too did the customized areas complexity. A paradigm shift happened within Ci3 Consultancy away from customized code to a more maintainable configuration approach. This shift took a long time to get from writing large amounts of code for single customers to a point in which most development work is focused on adding functionality for the entire user base.

It is a commonly held belief among the Ci3 Consultancy development staff that changing the configuration of a system is less time consuming than changing the code. This is because changing the configuration only requires the quality assurance department to test the new configuration while changing the code will require a regression test against all customers to insure that their particular functionality was unchanged. Configuration changes have less chance of breaking the system and as the code is grouped together the breaks are easier to diagnose. This belief forms the basis of the title for the accompanying paper "From Customizable to Configurable". It was the attempt to prove that the belief held through at least in the development of Blade that prompted this work. The same functionality within Blade has been developed from both a customer customization approach and a holistic configurable approach. This means that Blade can be used to directly compare both approaches.

This core question was expanded on by Dr. Mel Ó Cinnéide to incorporate the concepts of Software Product Line Engineering (SPLE) in the hope that the finished paper could be submitted as an Industry Experience paper to the SPLC and consequently be published. SPLC was chosen because the manner in which Ci3 Consultancy had maneuvered from a customizable approach to a configurable approach closely matched the concepts in SPLE.

A software product line is a set of software intensive systems that satisfy the specific needs of a particular market segment developed from a common set of core assets in a prescribed way [1]. While Blade is considered to be just one system each customer's requirements and implementation of that system are markedly different and in some respects can be considered different systems, hence matching the definition of Software Product Lines. The move by Ci3 Consultancy to implement configurable core assets for variation points in the system matches the second part of the definition.

Therefore the hypothesis of the accompanying paper was changed to state that Software Product Lines reduce time to market, improve quality and decrease costs. The focus of the paper was to test this hypothesis by comparing the same functionality in Blade under the two different approaches, i.e. customizable and configurable product line. This comparison is done on three main levels. The first level is the initial development cost of building both versions of the same functionality. The second level deals with the maintenance cost of fixing bugs, implementing new customers and adding changes. The final level considers other effects of the change between a customizable approach and a configurable approach. This last level is more abstract and harder to qualify then the previous levels and deals more with the reaction to the changes within Ci3 Consultancy and the users using Blade.

This paper is structured as follows. Section II covers a review of the literature involved in Software Product Lines as well as in Operational Risk Management. It provides definitions for some of the guidelines as well as expansion on what makes Blade a Software Product Line. It complements the information contained within the accompanying paper.. Section III describes some of the core assets that have been developed in Blade. Section IV covers the gathering of the data used in the comparison in the main paper. Section V gives a brief overview of some of the results of the comparison. Finally, in section VI, conclusions based on this extra information contained in this paper are presented along with comments on the main paper from SPLE experts.

II. Literature Review

To understand the functionality of Blade an understanding of Operation Risk Management is required. As such a review of the high level literature around operational risk was carried out. *Operational risk* is the risk of loss resulting from inadequate or unsuccessful internal processes, people or systems or from external events [3]. Operational Risk is very topical at the moment due to the banking crisis. The first event that triggered the global crisis was due to an operational risk event. The process by which some banks lent money and insured that that money would be repaid failed. The full blow consequences of mortgages not being repaid were not immediately apparent. It caused a credit crisis within the world economy which slowly spilled out of control. Such an event is called a Black Swan [6]. They occur because institutions make assumptions about their environments (all swans are white) that turn out to be incorrect. In the case of the credit crunch the assumption was that enough mortgages would be repaid to cover any losses from unpaid mortgages.

A high level of background detail on Operation Risk Management was included in the submitted paper because it was felt that SPLE practitioners would not have any previous experience with it. This is because the majority of the published work in the Software Product Line area deals with a different area in terms of concepts and software engineering.

The majority of companies involved with software product lines are manufacturing companies or software companies that produce firmware for a product family of electronic components or devices. They are so involved because the principles of manufacturing device product families match the principles of developing Software Product Lines to control those devices. This naturally feed into the development of Software Product Line Engineering. Examples of such industries cover a large percentage of the literature [4][5][6][7]. The concepts however can be taken out of the manufacturing companies and applied to other forms of software development that are not dependent on any accompanying hardware, however the conceptual jump for such industries is higher.

In other to write about Blade in the context of SPLE an understanding of Software Product Lines is required as well as a mapping between the concepts in Blade and the concepts in SPLE. This mapping was the main thrust and goal of the literature review.

Software Product Line Engineering (SPLE) was formalized by Paul Clements and Linda Northrup in their book Software Product Lines: Practices and Patterns [1]. They found that several different companies were using a process of selecting from existing software assets to create new products. This process had been developed independently within the various companies and has allowed them to meet market targets that they would not have been able to meet with a green field project. In essence a SPLE is a reuse strategy used to create new products from the components of an existing product and contains a variation model to control what the different products in the family does.

All of the references for manufacturing companies used above come from the Software Product Line Conferences Hall of Fame. They are held up as examples of what it means to be a Software Product Line. This does not mean that

the field is limited to manufacturing companies and hardware product lines. Clements definition of Product Lines is broader than just the manufacturing industry. In the description of a tutorial given during SPLC'09 Stan Jarzabek descripted a software product line business as a company that deploys multiple product variants to a variety of customers [17]. He goes on to explain that most small to medium sized companies develop SPL's out of a single successful product and that each new product is developed by ad-hoc reuse. As maintenance and complexity increases more formal SPLE processes get introduced. This is certainly the case with Blade, however in the case of Blade the historical data allows us to compare the system from before it was a product line to its more formal product line version.

More evidence for Blade as a product line is available from the story of the vacation home rental application, HomeAway [7]. It is also included in the Hall of Fame. In the case of HomeAway, the driver for development was the acquisition of different companies with web sites in the same business context and the need to ease maintenance of those different websites by putting them under the same roof. In the case of Ci3 Consulting, the driver for developing configurable core assets was the need to support varying customer requirements and to reduce the amount of per customer coding that we were doing for each new implementation. The fact that SPLC's Hall of Fame recognizes a web application proves that web applications and indeed other software disciples can benefit from the Software Product Line approach.

Dialect Solutions experiences with SPL development [8] are also similar to Ci3 Consultancy's experiences. Neither system was intentionally developed as a SPL. There exist variation points at an architectural level however it was not designed original as a SPL or indeed redesigned as one. Instead it has evolved based on customer need. This customer focused method of dealing with variation shares a lot in common with customer-centric one-of-a-kind development in manufacturing [9][10]. In one-of-a-kind manufacturing customers order a product made from available parts. This is very similar to SPL development where new products are developed using existing software assets.

Many success stories exist for the benefits of SPL development and the introduction of SPL methodologies [1] [7][8][11]. As the attached paper questions if these benefits are real in the case of Blade, it is important to understand them. These stories highlight benefits in terms of costs efficiencies, decreases in development time and a quicker time to market. However they also identify a high initial development cost associated with the development of SPLs. Avoidance or postponement of this initial high cost is necessary for small companies like Ci3 Consulting in order for them to move in a SPL direction and realize these benefits. These success stories do not provide numerical backup for their conclusions which the accompanying paper does.

From the initial review of the literature it became clear that how core assets are created and managed is central to the concepts of Software Product Line development. Core assets are the building blocks from which new products are developed. There are also the wrappers in which the variations between customers are managed. Further information was required to map the core assets within Blade and match their development process to the principles by which well known to SPL experts.

Kruger classified three models for developing core assets, proactive, reactive and extractive [12]. Proactive deals with mapping out exactly what the core assets are before commencing their development. It is reminiscent of the waterfall life cycle and is geared towards having a dedicated core development team as it is very employee intensive. In a reactive approach the core assets are built when there is a need for them. The extractive process takes an existing product as the base for reuse. Parts of the product are extracted as core assets and then reused within that product or in new products. It is similar to the process of refactoring.

For Blade a combination of reactive and extractive strategies is used. We react to what is in the roadmap and extract existing functionality from the system in order to make core assets. Such a combination is preferable to proactive strategies for small development businesses as it limits risk and initial investment [11]. Core assets are identified within Blade by finding a need that exists in two or more areas. This can be a need that two or more customers have or it can be a need that two or more modules have. Core assets are not limited to being large areas of code like full modules. The majority of core assets are smaller features that are shared between modules.

Ci3 Consultancy currently employees only one development team of around ten developers. This is set to change as Ci3 Consultancy enters a period of expansion. The single team acts as a business unit as described by Bosch [13]. Such a setup has drawbacks as it does not focus on shared assets and the cost around making decisions involving shared assets is high [14]. Approaches to counter these drawbacks have previously been proposed for example using a champion team who would make the decisions around core assets in place of a core team [15]. In Ci3 Consultancy it

was discovered that we did not encounter these draw backs as we use a combination of strong communication about core assets and encouraging involvement in core asset development.

Variation within core assets is an area that was highlighted by the SPLC reviewers as not detailed enough in the attached paper. Due to Software Product Lines history in small manufactured devices that have a limit on memory the concepts of variation in classic SPLE often refers to the tailoring of code by mechanisms like conditional compilations [18]. This results in the delivered version of the system only having the functionality of a single device. However such compile time mechanisms do not work for Blade customers, for example if a customer requires functionality that another customer is already using it is more efficient to flip a switch in the product than it is to recompile a new version of the product for them to use. A new version would require a complete retest by that customer while enabling the functionality in their current version would not. For Blade all variation needs to be done at run-time and this is why the attached paper overlooked variation control.

As core assets are often used as components within Blade they also contain design-time variation. This is where different modules require the same core asset functionality. In this case an instance of the core asset is created on a new page for the different module and different parameters are passed to configure that instance differently to other instances of the core asset. For complex core assets the parameters take the form of objects that inherit a common interface and it is this object that is used by the core asset to for example save the users personal configuration of that instance of the core asset.

III. CORE ASSETS IN BLADE

Many sections of functionality within Blade have redeveloped from an intensive customizable approach to a configurable framework approach. In the first version customer changes require changes to the code, however in the configurable approach the changes already exist within the code and only need to be turned on or off. Samples of the configurable assets are explained below:

- Generic List: A metadata driven list control that provides inbuilt search, ordering, paging and provides a mechanism for users to select which columns they would like to see in their personalize view of their data.
- Custom Fields: Allows users to add fields to the system in order to record other information that wasn't in the original design of the system. For example if a customer wants to record contract length as part of a person's record then they do not need to go back to Ci3 Consultancy to get a code change, instead they can add it through the system. It supports different control types of dropdown, multi-select, radio button, checkbox, date control and trees. Adding a custom field will also make it selectable within the related Generic List instance as well as within the reporting engine.
- Configurable Trees: Allows the user to define a hierarchy to examine items within their system at runtime and thus means that any user's view of the data is fluid as opposed to being fixed. It allows for different customers representation of the same data to be different without requiring code changes. Custom Fields when created by a user can be used to define a level in the hierarchy.
- Template Items: Customers record different data for the main concepts in operational risk. In order to support this without requiring code changes a template page is used. This page contains a set of controls that can be turned on and off by metadata and hence allows for each customers view of a single entity to be different to the next. This is different to custom fields as it deals with legacy controls that are not dynamically added to the page. These legacy controls are a product of the previous customization approach.
- Scoring: Scoring risk and controls is important in operational risk management and determining if the organization is working under an acceptable level of risk. Each customer has a different scoring model for risks and controls. The Scoring core asset makes the calculation and the factors that go into that calculation for scoring risk and control configurable and it forms the basis of the comparison in Appendix A.

The original concept was to compare the differences between all these core assets with the versions where the system wasn't configurable, however due to time constraints and the difficulty in accurately identifying bugs related to some of the areas this was cut down to focus the comparison on just one configurable core asset.

Determining whether a bug is related to a core asset is a trivial exercise since the process of creating a configurable core asset encapsulates all of the code in a limited set of classes, however determining if a bug is related to the code that core asset replaced is non-trivial. This is due to the spread out nature of that code. For example; the mirror of a generic list would be any list within Blade that the generic list replaced. Each replaced list would have contained the majority of its code on the list page that would also have dealt with other functionality. For each Generic List the footprint on each page is around eight lines of code. Those lines set up the parameters for that instance of a Generic List. A non-generic list would have over four hundred lines of code.

In the case of the generic list, the inbuilt searching and the user selection of columns has no parallel in the non-generic list so the comparison would not be on an equal footing. Likewise the custom fields and the configurable trees have no mirror in Blade before they were implemented other than to implement new fields and trees on a customer need basis. The template items core asset replaced hardcoded if statements across many entities within the system and could have formed the basis for comparison however it was decided that the scoring core asset would make a better comparison. It encapsulates more complex functionality then the template items on fewer pages. This means that categorizing the related bugs would be easier and the variation within the causes of those bugs would be greater.

IV. DATA GATHERING

Ci3 Consultancy has maintained a detailed record of all bugs raised both internally and externally as well as of all change requests raised during the lifetime of the Blade system. It is these records that are used for the basis of the analysis between the configurable version of Blade and the customized version. Over time what is recorded and how it is recorded has changed. At the start of the Blade project bugs were recorded in an in-house bug management system while change requests and specifications were stored in Visual Source Save beside the code. They were managed through the use of excel spreadsheets. Around the end of 2007 an external application called AxoSoft Ontime was brought to manage the entire lifecycle of bugs from raising, recreating, fixing and verification. In early 2009 the management of the change requests and the specification process was moved into Axosoft Ontime as well. The physical specifications are now stored in a Sharepoint website.

The first task in the analysis process for was the bringing together of the two different sets of bug data in a useable format. For this a simple Microsoft SQL Server database was created to hold the combined data. Microsoft SQL Server was used as it is the basis of the Blade system and hence was readily available. The bug data also did not contain fine grain information on where in the system the bug occurred. It was therefore not possible for a machine to identify what the root cause of a bug was related to a particular core asset or to code customized for a particular customer with any sort of accuracy. Therefore each bug needed to be analyzed individually by hand by an experienced Blade developer. This time consuming job was done by the writer of this paper.

This database contained fields for the basic information that would help identify exactly where the bug occurred within the system. The basic information was details like the bug description, recreation steps, developer comments and tester comments. Fields for categorization data was also include like bug priority, raised data, version occurred and status. Fields for the time recorded against the bugs were also included. Extra fields were also added to categorize the bugs according to the relevant core assets and the customizable code that those core assets replaced.

Populating this database in terms of the original in-house bug recording system was an easy task. The original bug management system had been developed in an easy to understand manner without the complications that can arise with systems that are built for two or more competing processes or customers. The only minor complication was the fact that the original bug data was stored in an Access database instead of an SQL Server database. The easy of extracting data from the system was balanced by the fact that that data wasn't as rich as what had been recorded in the AxoSoft application.

The AxoSoft Ontime application however was a lot harder to get information from, but the information was richer and more complete then the in-house application. During the switch between bug applications Ci2 Consultancy took the opportunity to change the amount of detail that was recorded for each bug. The problems from parsing the database stems from the fact that it is a very configurable product in its own right and during the initial implementation Ci3 Consultancy took full advantage of its configurability.

After loading the data from both bug recording systems into the single database there was over 21,000 bugs to analyze. Each of these bugs needed to be individually examined and categorized according to the part of the code that caused them. The amount of time required to fully classify such a number of bugs was deemed to be too high of a hurdle to jump. Therefore the original concept of analyzing all areas within Blade that had at one stage been customizable and had then been rewritten was limited down to one area, Scoring.

To help with the task of classifying all 21,000 bugs into whether or not they were related to scoring and if so if they were related to old customizable scoring or to newer configurable scoring a small application was created. This application simply listed out the details for a single bug from the previously defined database and allowed a user to categorize that bug before quickly moving onto another unclassified bug. An experienced Blade developer could thus categorize a single bug as being related to scoring or not in around five to ten seconds. Categorizing all bugs took in the region of 45 hours. If a fuller classification was used this number would have exponentially increased especially in light of the previous comments on the difficulty of identifying bugs related to the code the generic list replaced.

During the classification of the bugs a hole in the data was identified. For a large proportion of the bugs no time had been recorded. This proportion was larger than would be expected for bugs that didn't require development effort to fix, i.e. bugs that were caused by a user misunderstanding. A considerable amount of the untimed bugs from the individual examination were of the type that would require quite a bit of development effort and in fact the development comments attached to the bugs indicated as much.

To manage this hole in the data it was decided that the untimed bugs would be assigned a time based on the average time across all timed bugs to fix similar bugs. Similar bugs were identified as those that had the same priority as the untimed bugs. The allocated time for critical bugs was 68 minutes, for serious bugs was 65 minutes and for trivial bugs was 35 minutes. The numbers of bugs timed for each category was of a large enough number that these timings are considered to be a fair reflection of the average time to fix a bug of this type. The fact that bugs that were fixed in zero time are now timed at the average does introduce a small inaccuracy into the final numbers, if anything this inaccuracy is most biased against the configurable approach as due to its complexity it is the most prone to user misunderstandings.

V. Comparison Results

Results of the bug comparison can be found in Appendix A. section IV Comparison of Development and Maintenance Effort between Configurable and Customizable approaches. Included there is an analysis of the original development cost. This compares the cost of developing eight customized scoring methodologies with the cost of developing a configurable scoring framework and implementing six scoring methodologies in it. The data for this section was based on the original estimates for the change requests for the different scoring models, the development and implementation time recorded against those specifications and where data was unavailable the estimates of senior developers who would have been involved in the development of the code.

Overall it was found that the initial investment for both approaches worked out evenly for around six different scoring methodologies, however the time to implement new configurable scoring methodologies was much less than the respective time to implement a new customized scoring methodology after the initial investment in building the configurable framework is factored out.

The required and expected maintenance effort for both approaches however was better for the configurable approach. While the initial maintenance cost was high, it was found to not scale with the number of different customer implementations while the maintenance effort for the customizable approach did. This means that future maintenance for the configurable framework will roughly be in line with current maintenance irrelevant of the number of customers and scoring models.

VI. Conclusions

An unexpected benefit of classifying the 21,000 bugs was a clearer understanding of the problem areas within the Blade application. The quick bug categorization tool selected the next bug to categorize in a linear fashion based on date raised so the history of Blade's evolution became apparent. Similar bugs in certain areas of the code had a habit of

reappearing after time and these areas were consistent with areas that the senior developers in Ci3 Consultancy would consider as black spots within the code. Over time these areas were redeveloped into core assets or similar solutions and the bugs that had been reappearing would stop popping up. There is an interesting unit of work to go over the bug data again from this prospective and to map the evolution of Blade from the prospective of maintaining and refactoring troublesome sections of code. Such a task would take far longer than the 45 odd hours required to classify the bugs based on relevance to the scoring module.

It would have been optimal to get more than one person involved in the classification of the bugs. Not only would this have sped up the gathering of the data but it would have compensated for the bias of the individual that no doubt crept into borderline cases. A bug's root cause can vary depending on the individual determining that root cause. Due to the large number of bugs that were determined to be related to Scoring, individual bias on borderline cases should not have too much of an effect on the validity of the data.

As "From Customizable to Configurable: An Experience Report in Risk Management Software" has been reviewed and conditionally accepted for SPLC 2011 it is possible to share some of the reviewers comments, both good and bad. The positive feedback and conditional acceptance is a huge honor and deserves pride of place within this background paper.

"The paper is interesting to read, both for practitioners and researchers. It gives an example of real-world product line and the company's view on PLE. "

"The paper is as success stories from the early product line days; no inspiration from the community but motivated out the industrial organization itself."

Both of these comments indicate that the accompanying paper is bang on target and a worthwhile contribution to the Software Product Line knowledge base.

"I clearly miss information on the scope, on variations and how variations are exactly managed (technically and organizationally)"

This comment highlights the fact that further work is required in order to bring the paper up to publishable standard and to present it at SPLC'11. It is hoped that this work will be completed and acceptable to the conference panel, however it is not possible to include it here as the deadline of this background paper occurs before the deadline for the accompanying paper.

REFERENCES

[1] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison Wesley, 2001.

[2] R.S. Kenett and Y. Raanan, Operational Risk Management: A Practical Approach to Intelligent Data Analysis. John Wiley & Sons Ltd. 2011

[3] N.N. Taleb, The Black Swan: The impact of the highly improbable, Random House, New York. 2007

[4] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, & S. Ferber. Nord, R. (ed.), "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices." *Proceedings SPLC3*, Lecture Notes in Computer Science 0302-9743, vol. 3154. Springer, 2004. Page: 34-50. Boston. ISBN: 3540229183.

[5] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*, Addison Wesley, 1998, Chapter 16.

[6] Peter Toft, Derek Coleman, and Joni Ohta, "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line'" Patrick Donohoe (ed.) *Proceedings SPLC1*, Kluwer Academic Publishers, 2000.

[7] B. J. Pronk. "Medical Product Line Architectures," *Software Architecture. TC2 First Working IFIP Conference on Software Architecture* (WICSA1), 1999, 357-67. ISBN: 0 7923 8453 9.

[8] Krueger, C. Churchett, D. Buhrdorf, R. "HomeAway's Transition to Software Product Line Practise: Engineering and Business Results in 60 Days." In *Proceedings of the 12th International Product Line Conference*, pp297-206, 2008

[9] Staples, M. Hill, D. "Experiences Adopting Software Product Line Development without a Product Line Architecture". Software Engineering Conference, 2004. 11th Asia-Pacific

[10] Tseng, M.M. and Piller, F.T., The Customer Centric Enterprise: Advances in Mass Customization and Personalization. 2003. Springer, New York.

[11] Gang Hong, Deyi Xue, Yiliu Tu "Rapid Indentification of the optimal product configuration and its parameters based on customer-centric product modelling for one-of-a-kind production" Computers in Industry Volume 61 Issue 3, April, 2010 pp. 270-279

[12] Alves, V.; Camara, T.; Alves, C.; , "Experiences with Mobile Games Product Line Development at Meantime," *Software Product Line Conference, 2008. SPLC '08. 12th International* , vol., no., pp.287-296, 8-12 Sept. 2008

[13] Clements, P. and Krueger, C. W. Being Proactive Pays Off/ Eliminating the Adoption Barrier. Point-Counterpoint article in *IEEE Software*, July/August 2002

[14] Bosch, J.; , "Software product lines: organizational alternatives," *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on* , vol., no., pp. 91- 100, 12-19 May 2001

[15] Pohl, K. Böckle G, and van der Linden F.J.. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[16] Takebe, Y. Fukaya, N. Chikahisa, M. Hanawa, T. and Shirai, O. 2009. Experiences with software product line engineering in product development oriented organization. In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, pp. 275-283.

[17] Jarzabek, S 2009, Pragmatic Strategies for Variability Management in Product Lines in Small to Medium-Size Companies. In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, pp. 327.

[18] Anastasopoulos, M. Gacek, C. Implementing product line variabilities. In *Proceedings of the 2001 symposium on Software reusability: putting software reuse in context* (SSR '01). ACM, New York, NY, USA, 109-117

Appendix A

From Customizable to Configurable: An Experience Report in Risk Management Software

Gerard Quilty
Ci3 Consulting
Dublin, Ireland
Gerard.Quilty@gmail.com

Mel Ó Cinnéide
School of Computer Science and Informatics
University College Dublin
Dublin, Ireland.
mel.ocinneide@ucd.ie

***Abstract*— Software Product Lines are supposed to reduce time to market, improve quality and decrease costs. Implementing a Software Product Line also requires a high initial investment. In this paper we examine the transition of a single system to a Software Product Line (SPL) and evaluate if these benefits have occurred in this example. We find that efficiency and quality have improved, while costs have been reduced. The high initial investment associated with evolving to an SPL has been postponed by taking small steps towards an SPL architecture. In addition this approach has given us the ability to expand our product into a wider market and deal with more complex customer requirements without a corresponding increase in staffing and costs.**

Keywords: operational risk management, software product lines, industrial experience

I. INTRODUCTION

Ci3 Consulting is an independent business unit of a large multinational company and provides Operational Risk Management Software to financial intuitions. The product, Blade, has undergone numerous changes in its ten year history. It was originally built as a once-off project for a single customer and now is a recognized leader in the operational risk management software. The processes and coding standards involved in developing and implementing Blade resemble those involved in developing a family of software products or a software product line.

A software product line is a set of software intensive systems that satisfy the specific needs of a particular market segment developed from a common set of core assets in a prescribed way [1]. This paper maps Blade to this definition of software product line and demonstrates some of the concepts applied by Ci3 Consulting to limit the initial development cost associated with software product lines [2]. It also highlights our successes and failures with taking this approach in particular with realizing the proposed benefits of software product line development.

The culture that facilities the development of Blade as a software product line grew within the company without reference to academic literature. This mirrors the first SPLs identified by Clements [1], which grew out of industrial need rather than academic endeavor. This paper aims to present the experiences of Ci3 Consulting developing Blade and maps it to the concepts in software product line engineering. It discusses the benefits of software product lines and illustrates that those benefits are achievable for a small development team. It uses bug and development time data gleamed from the lifetime of Blade to show this.

This paper is structured as follows. In section II gives an overview of the terminology involved in operational risk management. It also covers an overview of the product Blade. Section III describes the processes and policies that Ci3 Consulting has adapted to manage core asset development and eliminate some of the recognized problems associated with SPLs. Section IV deals with a comparison of a section of the Blade code. This section, risk and control scoring, has been implemented initially on a per customer basis but has since been evolved into a configurable core asset. It is an area that experiences lots of variation between different customers. Both these implementations are compared in terms of initial development cost, maintenance cost and the other effects of the change. Finally, in section V, we present our conclusions discuss possible further research and opportunities to improve the efficiency gains made in development by reference to the SPL literature.

II. BACKGROUND

Ci3 Consulting is a bespoke development house specializing in financial software. It currently employs ten developers. They are organized into one development team. In 2003 Ci3 Consulting created a product out of one of its previous bespoke projects and offered it to the market.

This section is split into two sub sections. The first subsection deals with risk management concepts and definitions. It also covers what is involved in managing risk in financial companies. The second subsection deals with how the product helps risk managers manage their risk as well as background on how it became a software product line.

A. Operation Risk Management

Operational risk is the risk of loss resulting from inadequate or unsuccessful internal processes, people or systems or from external events [3]. For example, an earthquake is a *risk* for offices in certain countries. A *risk control* is a protective measure for reducing risks to, or maintaining risks within, specified levels. Earthquake insurance is a control on that risk that mitigates the impact of an earthquake if one occurs. An occurrence of an earthquake is called a *risk event*. The risk is transformed into consequences that may relate to internal or external sources. Without risk events a risk remains as only a potential.

All companies are susceptible to operational risk events and failed processes. Toyota, one of the world's largest car designers, was obliged to recall over 2.3 million US cars due to a failed internal quality process. Toyota lost 21% of its share price and took a huge reputational hit from this event. It could have been avoided if the risk had been identified and proper controls put in place to prevent it [3].

Financial institutions have a regulatory requirement to manage risk within their organization. This requirement covers both internal and external risk. Banking institutions are governed by the Basel II accords [4] while insurers are governed by Solvency II [5]. The Advanced Measurement Approach or AMA is the highest level of operational risk management according to Basel II. It states that the institution has to convince its supervisor or regulator that it is aware of and is managing its risks. To comply with AMA, the company must at a minimum have the board of directors and senior management involved in the process. An operation risk system that is conceptually sound and that is implemented with integrity is required and the companies risk team must have sufficient resources and training.

Early identification of risks and the implementation of controls that lessen the effects of those risks are vital in preventing damaging risk events. Identifying risks is a complicated activity for businesses. It requires a great deal of information from different sources both internal and external to the company. External events are analyzed to determine if the underlying risks that caused the event could exist within the company and if such an event is likely to occur for that company. Companies also record and maintain other information like indicators and scenarios that help them to identify risk and controls and quantify the impact and likelihood of risk events

An *indicator* is a measure of certain activity within or outside of a company over time. It is used to evaluate how likely a risk is to occur. For example if an indicator based on unemployment figures goes up it is more likely people will not be able to repay mortgages. The probability of risks related to unpaid mortgages goes up.

A *scenario* is a collection of related risks that have a very low possibility of occurring together but have a devastating knock-on effect when they do. The knock-on effects of the credit crunch led to the IMF bailouts in Europe. Such unpredictable events are sometimes referred to as black swans as they can occur because institutions make assumptions about their environments (all swans are white) that turn out to be incorrect [6].

Companies will usually maintain a master register of all risks that they are aware of that affect their business and wider industry. Libraries of controls and tests for those controls are also managed. Local risk managers select from this library of companywide risks to create local risk maps for individual departments.

To reassure their regulators and supervisors that they are managing risk within their company they perform risk assessments periodically. A risk assessment can be control-centric or risk-centric depending on the relative importance given to controls and risks within a company. In a risk assessment controls and risks are scored using what can be a very complex scoring model. These complex scoring models serve a duel mandate. Firstly they are an attempt to qualify what can be a quite complex problem like the probability of an earthquake occurring. They are also designed to convince regulators that the organization knows what its risks are and has adequate controls in place to deal with them.

Risk Management is not about avoiding all risk but setting a threshold of acceptable risk and limiting the impact of risk events when they do occur. A risk manager after completing their risk assessments and having signed off on the level of risk within a department needs to report that level of risk up to senior management. Often the risk assessments are rolled together to give a boarder sense of the risk level in the company. This conforms to the first requirement of AMA to insure that senior managers are informed and aware of the level of risk.

Control-centric assessment deals with users signing off on controls and confirming that the control has been performed. A risk with unperformed controls can have a much higher impact than when the controls are performed. This means that senior management can be signing off on a lower level of risk within their company then actually exists. Such action may result in a regulatory breach and fines. Incorrectly signing off on controls as preformed when they have not been is a serious employee infringement and can result in termination.

B. Blade

Blade is a web application that supports many different customer frontends. Web applications have previously been built as SPLs, for example the vacation home rental application, HomeAway [7]. In the case of HomeAway, the driver for development was the acquisition of different companies with web sites in the same business context. In the case of Ci3 Consulting, the driver for developing configurable core assets was the need to support varying customer requirements without an increase in costs. The concepts and fine detail of operational risk management vary considerably between customers.

It was not designed as a product line. The original system was as a standalone project for a single customer. Three years after its initial development it was adapted to fill that a gap in the market. The match between Blade and what the market required was not perfect. It shared a lot of the core concepts of

operational risk management but each customer has a different understanding of the details.

Blade is similar to Dialect Solutions experiences with SPL development [8] as it was not intentionally developed as a SPL and the code cannot be said to contain a SPL Architecture. There exist some variation points at an architectural level however it was not build by design as a SPL. Instead it has evolved based on customer need. This customer focused method of dealing with variation shares a lot in common with customer-centric one-of-a-kind development in manufacturing [9], [10]. In one-of-a-kind manufacturing customers order a product made up of available parts. This is very similar to SPL development where new products are developed using existing software assets.

Many success stories exist for the benefits of SPL development and the introduction of SPL methodologies [1], [7], [8], [11]. These stories highlight the benefits in terms of costs efficiencies, decreases in development time and quicker time to market. However they also identify a high initial development cost for the development of SPLs. Avoidance or postponement of the initial high cost is necessary for small companies like Ci3 Consulting in order for them to move in a SPL direction and realize these benefits.

Blade is targeted at financial institutions such as banks, insurers and asset managers. An operational risk management system needs to be able to conform to the requirements for AMA and help companies identify, categorize and assess their risks. The software provides support for all the varied operational risk management processes. It is a web based application based on the .Net framework and Microsoft SQL Server. It was first developed in 2000 using ASP as the base language. Since then it has been fully redeveloped in .NET with C# being used for backend auto generated data classes and VB.Net used as the code behind on the web pages. In 2008 the model view presenter (MVP) pattern was implemented as the architectural pattern for new pages.

It is split into four solutions. One contains core components like authorization and reporting features that the other three use. Modules for the different processes in operational risk management like risk assessment, control signoff and managing risk events are contained in the main solutions. The two other solutions are specialized modules for the recording of indicators and scenarios. These tasks are outside of the usual remit of an operational risk management team but can hugely benefit the accuracy of risk calculation.

The development of indicators and scenarios led to the refactoring of many core assets out of the main solution into a core asset solution. These assets needed to be available to other solutions. Some core assets like a user definable list and a lazy loading tree were developed specifically for one of the modules to solve a specific requirement. These core assets have been introduced back into the main solution. The user definable list has been so successful that it now has over eighty implementations and is the first port of call for any new lists that are needed in the system.

In terms of the size the total code base is over two million lines of code with one thousand tables and two hundred stored procedures. This is a considerable sized project for a small development team to maintain and upgrade. As a mature product, modules and concepts have changed over time. These changes range from maintenance fixes to complete rewrites to take advantage of developing technology. This leaves areas that have in the past needed to be customized to individual customers but now are configurable instead.

Scoring is one of these areas. It is an area that experiences a huge amount of variation between customers. A risk assessment is where a user will score their risks and controls on a regular basis. The base unit of a score is a single input called a *scoring measure*. This is where the user records a single fact about that risk or control i.e. how damaging the risk is, or the probability of the risk occurring. A user-defined calculation uses these inputs to calculate a *score value*. This score value is then compared against a set of thresholds to evaluate a textual description for that score. These calculations are recursive; one score value can be an input into a further calculation to create a new score.

Inputs to a scoring measure do not necessarily come from the user entering a value on the front end. They can come from thresholds set against the organization that the risk is part of, from external or internal events targeted against that risk, from indicators that the customer records and from external systems. The calculations for a risk score often depends on a special set of inputs called combined control scores. These are calculations using the set of all controls acting on that risk. This makes scoring a complex and challenging problem that varies between customers.

This problem was initially solved for customers by embedding their scoring models and their individual calculations into stored procedures and the code. However in the first quarter of 2009 the underlining framework of scoring was changed to be user configurable. It was originally changed for risks and then after the concept was validated it was applied to controls. The difference between these two approaches forms the basis of the comparison section.

The refactoring of customizable sections like scoring in configurable frameworks has helped us realize the benefits of SPLs. Different sections were refactored slowly over time in order to spread out the initial development cost. This also lead to a spreading out of the benefits, however it was unavoidable as the initial investment required to implement SPLs fully in a short space of time could not be supported by revenue streams.

III. Applying Software Product Line Development to Blade

The organization and processes that Ci3 Consulting employs enables us to utilize the benefits of SPL development. This section covers the details of the organizational approach applied and the processes that it covers. It focuses on the specification and development of core assets and how the focus of development is kept on core asset development instead of on customization of code for individual customers. While these processes have grown out of the culture and experience of the company they do correspond to the experience of other companies when using software product line development

techniques. The literature is references to highlight this correspondence.

The remainder of this section is organized as follows. In section A the development organization within Ci3 Consulting is discussed, while in section B other processes from the broader company perspective are described. The boarder methodology has grown out of the success of core assets within the development team.

A. Core Asset Development and Processes

Creating the core assets is a key activity of SPL development. It is from these core assets that new products are developed. The Ci3 Consultancy development team is small and cannot support a core assets team. Responsibilities for core asset development must be shared amongst the team members on the basis of who is available to work on them.

Kruger classified three models for developing core assets, proactive, reactive and extractive [12]. Proactive deals with mapping out exactly what the core assets are before commencing their development. It is reminiscent of the waterfall life cycle and is geared towards having a dedicated core development team. In a reactive approach the core assets are built when there is a need for them. The extractive process takes an existing product as the base for reuse. Parts of the product are extracted as core assets and then reused within that product or in new products.

Blade development uses a combination of reactive and extractive strategies. Such a combination is preferable to proactive strategies for small development businesses as it limits risk and initial investment [11]. Core assets are identified within Blade by finding a need that exists in two or more areas. This can be a need that two or more customers have or it can be a need that two or more modules have. Core assets are not limited to modules in Blade and the majority of core assets are smaller features that are shared between modules.

The single development team acts as a business unit as described by Bosch [13]. Such a setup has drawbacks as it does not focus on shared assets and the cost around making decisions involving shared assets is high [14] Approaches to counter these drawbacks have previously been proposed for example using a champion team in place of a core team [15].

The development process used deals with these disadvantages in a different way. Over time Ci3 Consulting has developed three main tenets to attempt to insure that the benefits of core asset development are known and utilized throughout the company. These tenets are:

- To always avoid development risk, i.e. insure that the development of the core asset does not impinge on our ability to deliver agreed changes.
- Make core assets that are useful to the development team as well as fulfill a need within the system.
- Communicate changes and new core assets across all teams and rotate the implementations of new instances of the core assets between developers

For the first tenet new core assets must be prototyped in a limited area first and validated by the quality assurance department before it is propagated to other parts of the system. The Generic List control which is a metadata driven list control that allows users to select which columns they wish to see is an example of this. It was first implemented in the Indicators module.

After Indicators was released and the customer reaction to it was judged it was implemented in Scenarios and in the main Blade modules. This recursive rollout allowed the Generic List core asset to be refined and to insure that at each step there was the least amount of impact on the rest of the code base if the control failed. Other companies have also taken a phased approach to extraction of core assets for SPL engineering to limit risk and initial development cost [16][17].

The second tenet means that core asset must make a developer's job easier by removing some time consuming repetitive development task. A standard .NET grid requires the setting up of template columns, bound columns, look and feel standards, sorting and a multitude of other minor tweaks. With the Generic List core asset, a developer gets all these tweaks for free. When the Generic List was applied to Blade, on some pages it removed over four hundred lines of code and replaced it with five lines of code and around twenty lines of metadata. This makes creating list pages a lot easier and quicker when using the core asset.

Making the core assets developer friendly hugely improves developer uptake of those core assets. The Generic List is now so popular amongst developers that it is used as the backbone of all lists in Blade even ones that do not need any of its advanced features.

The third tenet, communication within development teams is a key concern. After completion of the initial prototype which is usually the responsibility of one developer, that developer must inform the rest of the development team about that asset. This is done at the weekly developer update meeting. The communication also goes from the development team out to the business analysts and to the quality assurance team. Without knowledge of the core asset it cannot be reused and the benefits of developing it is lost. Further instances of the core asset are done by developers other than the initial developer to insure that the knowledge of the inner works of the core asset is spread.

Krueger stated that a key to combating entropy and developing core assets is to give someone the responsibility to identify core asset opportunities within the development team [18]. By developing successful core assets like the Generic List that benefit development as well as the product you change the companies culture to be more focused on core asset development as well as delivering on Krueger's point. The fact that it is of benefit to developers means that they kept a lookout for more areas in which a core asset can remove dull repetitive coding tasks. As risk is limited by prototyping only the most successful controls become core assets and these assets deliver real efficiency gains on future development.

B. *Other Process influcenced by Software Product Lines*

Business analysts are kept informed of developed core assets. While the development and identification of core assets is a development task, once they are created the business analysts can make use of them. A hybrid agile waterfall approach is used for historic and commercial reasons. A lot of a quarter's work comes from customers. These changes need to be signed off by the customer before development starts. This requires a specification to be written for that change. Once the specification is signed off it goes to the development team in where the process becomes more agile.

Writing specifications is a time consuming task. In the past, we have experienced bottleneck problems with developers being left idle waiting on complex specifications to be finalized and signed off. Core asset development helped us overcome this bottleneck by speeding up the specification writing process. To do this we applied core assets to the specifications themselves. If a change requires a new list then a business analyst can simply put Generic List in the specification and the developer or Quality Assurance expert reading the specification knows that this is a list that requires user selectable fields, sorting and filtering functionality. The concept of a Generic List has also been explained to customers so that they know what the specification means.

Like design patterns [19], using core assets has given Ci3 Consulting a common lexicon to describe complex pieces of code easily and efficiently [20][19]. Unlike design patterns this lexicon is common across development, business analysts, quality assurance and customers easing communication on all fronts. This does provide a barrier to new employees as they must understand the same concepts that the rest of the team works in. Documentation helps with this. An internal Wiki of all core assets within Blade is maintained and updated. It includes implementation and using guides, feature explanations as well as lists of all implemented instances of the core asset and the features enabled for each.

Having customers and business analysts on board with core assets has led to a lot of focus on improving those core assets. The Generic List since its implementation throughout all of Blade has had numerous feature upgrades like the adding of saved personal filters, quick search and export options. Because of this understanding between customers, business analysts and developer there is no delay on decision making for changes to shared assets. If there is doubt about a change and if it should be available on all instances of a core asset or for all customers then that feature is made switchable and individual customers can enable it if they so wish.

Making core assets switchable and configurable by customers and individual users within that customer removes a lot of the burden of variability management [21][22] from development of Blade. As Blade is a web application instead of lower level software like firmware or drivers it does not have a limit on the size of its install and hence can include the code for different user configurations instead of using conditional compilation to limit out branches of execution [23].

IV. Comparison of Development and Maintainence Effort Between Configurable and Customisable Approaches

In the first quarter of 2009 the risk assessment module of Blade was rewritten. Part of this rewrite was to develop a new configurable scoring framework that could deal with all of the complexities of existing customer scoring as well as future customers' scoring requirements. This new configurable framework replaced the original hardcoded scoring models in Blade. Eight different scoring models were development for customers between the first version of Blade in 2000 and when it was replaced in 2009. These scoring models were also migrated to the new framework.

This comparison covers a time period of nine years from September 2000 to September 2010. The data used in this section comes from two different time recording applications that Ci3 Consulting used during the period covered. This data covering over 21,000 bug reports and 1,000 change requests was combined and all that related to scoring was extracted.

The total number of scoring bugs found was 1,009. This is under five percent of total number of bugs reported. The estimated time for all scoring changes and maintenance over the period is 1,400 hours. This time is calculated off the time recorded for the relevant change requests and bug reports. Where time was not recorded against the bug the average time for bugs of similar type was used. The estimated time to fix all scoring bugs recorded is around twice the initial development time recorded for both configuration and customizable development.

This paper compares the two methods of dealing with customer variation customization and configuration and is performed across three levels. The remainder of this section is as follows. First, in section A the initial time to develop is examined. Next, in section B the bug maintenance effort for each version is calculated. The third section C deals with the other effects of moving from a customizable scoring approach to a configurable scoring approach.

A. *Initial Development Time*

The definition of initial development time used here for comparison is the time recorded against a change request from the start of implementing that change to the time it is first delivered to the customer as completed. It does not include the time taken to specify or negotiate the change request.

At the point in time that the configurable scoring was developed there was eight distinct scoring methodologies that had been implemented in an ad-hoc fashion. The newer configurable scoring framework uses a different data model to the previous implementations. This meant that existing customers wishing to take advantage of the changes in scoring needed to have their historical data migrated. In 2010 six new scoring methodologies have also been implemented using the newer configurable framework.

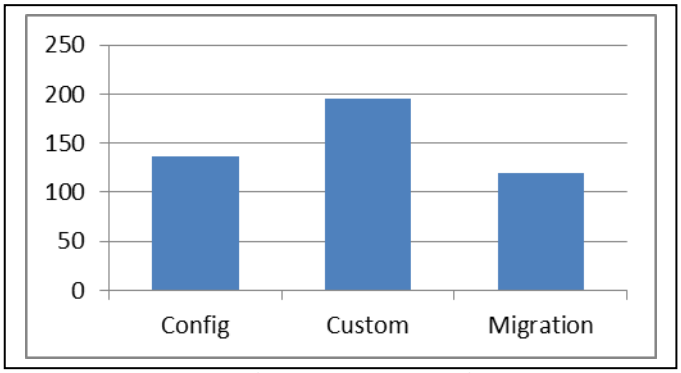


Figure 1. Time recorded for initial development of custom implementions, configurable framework and migration between the two

Fig. 1 shows the breakdown of the time in hours that was spent developing the two different approaches and the time taken to migrate between them. The configuration column corresponds to the time spent implementing the scoring core asset and includes the time spent implementing six new scoring methodologies. The custom column is the sum of all the time recorded for implementing the eight original distinct scoring methodologies.

Replacing the existing customization approach with a configurable approach took a considerable upfront investment as seen from Fig 1. This investment which is the combination of migration and initial development was greater than the initial investment in developing the original customer centric scoring. This level of investment was acceptable as it did not require expansion of the development team while delivering substantial benefits. However when migration between the two approaches is taken out of the equation the average time per configuration of a scoring methodology is 23 hours. For customizing code it is 24 hours. This is a small increase in efficiency for the amount of investment.

The time recorded for the configurable approach includes a once off cost of around 115 hours for building the framework. The customizable approach requires redevelopment for each new implementation and hence a development cost must be paid again. Therefore the average time per implementation of the eight original scoring methodologies of 24 hours is the expected time to implement a new scoring methodology using the previous customizable approach.

The average time to implement a new scoring methodology using the configurable framework when the development of the framework is excluded is four hours. This is a six times increase in efficiency over the customizable approach for when customers require variations from the existing set of scoring methodologies. This is an increasingly common task as the company grows. In the two year period since the creation of the configuration framework six new scoring methodologies have been created.

B. Maintenance Time

The level of maintenance required by a change to a product can often be overlooked. Internal time recording applications have been used to record every defect raised during the lifetime of Blade. Part of the development process requires developers and quality assurance personnel to record time directly against the defects or change requests that they are working on. By recording time in this way we are better able to estimate workloads and plan quarters.

This data can also be used to analyze the maintenance record of any module or component of Blade. The overall count of bugs related to an area and the time recorded against them gives a clear image of the maintenance cost that has been required by a section of code as well as any problem areas there may be within that section.

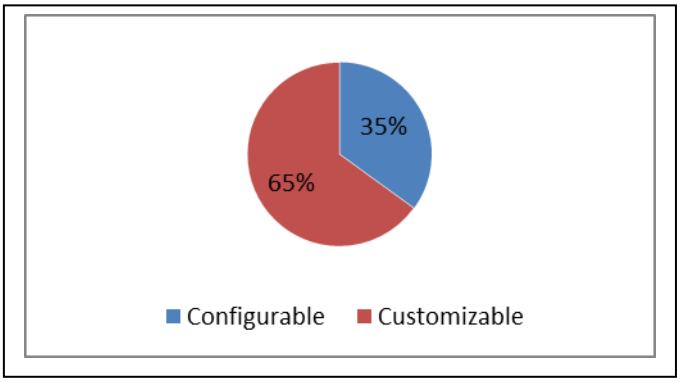


Figure 2. Comparison of maintence effort required by configurable and customizable approaches

Fig. 2 highlights the comparable bug counts between custom scoring and configurable scoring. The configurable scoring count includes bugs raised during the migration of the old scores to the new framework for existing customers. The overall bug count for configurable scoring is significantly smaller than the bug count for the customized code. We can suggest two main reasons for this.

Firstly the custom implementation has a larger code footprint then the configurable implementation. Configurable scoring was designed from the ground up and is limited to a few key classes. The code developed for the custom implementation however is spread out across several pages. For one scoring methodology the same page has been duplicated leading to a requirement to fix the same bug in two different places. The variation on pages using the customizable approach is dealt with by using if statements. Eight different scoring methodologies require eight different conditions. The result of which was a lot of confusion and a lack of readability in the code.

The configurable approach however uses for statements to loop through metadata and build its user controls dynamically. This leads to clearer code. The decentralized nature of the custom code gives rise to the types of bugs that were found. This is shown in Fig. 3. Part of the development goals for configurable scoring was to centralize the code and remove some of the reoccurring bugs that crept into the software with each new scoring methodology when applied using custom code.

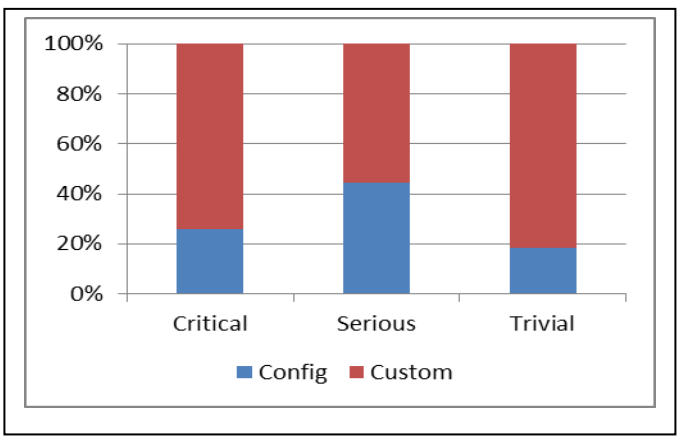


Figure 3. Breakdown of bug count across bug priority

Considerably more trivial and critical bugs were found during the lifetime of the custom code than in the configurable code. These bugs were grouped around several problems areas. The trivial bugs primarily occurred as look and feel bugs on one page of the web application. This page allowed the user to select values for their scoring measures and was the same for all scoring methodologies. Each time a new set of scoring measures were added to the page it would break the look and feel for the other scoring methodologies.

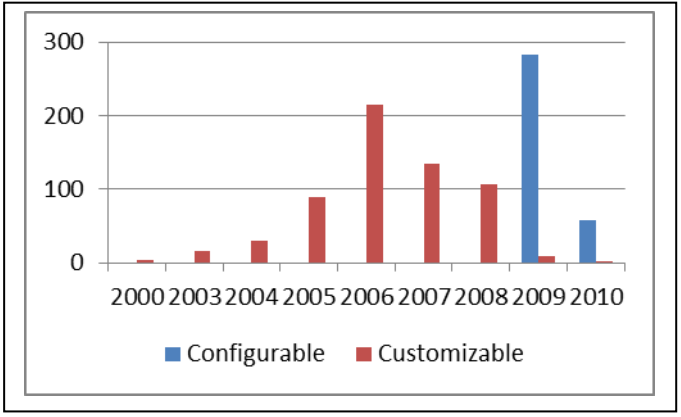
This problem is avoided in configurable scoring as the user interface is dynamically built based on configured metadata. The metadata tells the software what measures to display, the valid values for those measures and in what order to display them. How the controls are added to the page is constant so look and feel cannot be broken for old scoring measures when a new scoring methodology is added. It also insures that the correct measures are shown for the correct methodologies. This dynamic adding of controls forms the basis of the configurable approach taken with scoring.

The majority of critical bugs raised against the custom implementation were grouped around problems in the different calculations. Primarily those calculations were not returning the expected value. The root cause often boiled down to the function doing the calculation getting incorrect values for the measures passed to it or getting the values in the wrong order.

Configurable scoring limits the possibility of these types of bugs by restricting the calculation and entry of scoring measures to one class in Blade. If there is an error in the core code then it is in one place instead of spread across the system as with the customized code. The calculation itself is also configured in metadata. This limits the impact of defects in the calculation as it can be changed without having to change code.

The elimination of the problem areas identified in the custom implementation was designed into the configurable core asset for scoring. The direct result of designing out troublesome areas is that most of the bugs in the configurable approach are related to an area not found in the custom implementations. This is the configuration of the metadata that the scoring runs off and is an area unique to the configurable approach. Metadata bugs tend to be raised as serious and this is why the number of serious bugs is so high compared to the critical and trivial bugs. All the scoring measures and calculations are built up using metadata that is entered by a business analyst or support engineer. The possible scoring configurations are a complex set and this is reflected in the difficulty of setting up a single scoring methodology. This is a known issue with this approach and tools are in development to support the configuration of the metadata. It is hoped that tool support will decrease the number of bugs raised due to metadata issues and improve the 3 hour average time to set up a new scoring methodology.

Another possible reason for the large disparity in bug numbers is due to the fact that the custom implementation has had more time to mature than the configurable scoring implementation. The first scoring methodology was implemented in 2000 at the start of the Blade project. The



majority after that were implemented in 2006. Configurable scoring was implemented at the start of 2009 and the new scoring methodologies in 2010. This variation in the age of the code means that there has been more time to iron out the kinks in the custom implementations and hence more bugs recorded.

Fig. 4 highlights the difference in ages. It shows the estimated hours for both configurable and customizable scoring across the years. The hours were calculated by applying the average time recorded against bugs by the development team for each bug priority to the total number of bugs for that priority relevant to scoring. Between 2000 and 2003 only minor maintenance changes were required. This period corresponds to before Blade was a product and only a simple risk impact by risk probability scoring matrix was available. 2001 and 2002 are missing from the graph because during this timeframe no defects were raised or change requests were implemented. 2003 has data available as the original customer came back with changes that they wished to be made to the system.

Figure 4. Breakdown of estimated hours by year

Fig. 4 also highlights the nature of the maintenance effort over time. The number of hours per year for the customizable approach can be seen growing at an increasing rate up until 2006 and then begins to tamper off a bit. The recorded configurable hours have a different structure per year. They start with a huge amount of effort in the first year and then dramatically decrease. The shape of the graph can be explained with reference to the number of different scoring implementation done in each year.

The amount of maintenance required began to grow in 2004 when Blade was released as a full product to the market place. The basic matrix scoring model from the original implementation was kept however the available values for the different scoring measures were made to be configurable. The calculation and positioning of the measures is left hardcoded. 2005 saw the introduction of two new scoring models. This is reflected in the tripling of the required maintenance for that year. 2006 had four new scoring models and two and a half times as much maintenance effort required as the previous year. The next two years saw a drop off in maintenance effort. Only one new scoring methodology was implemented during this time. New customers were fitted in to the existing scoring models on offer. This was possible due to limited configurability which was available within scoring. Key elements like the calculation used still needed to be hardcoded if a customer needed one that didn't previously exist.

The maintenance effort for configurable scoring in its first year far exceeds the peak maintenance effort for customizable

scoring. As configurable scoring was developed in the first quarter of the year, the next three quarters were spent migrating customers using the old scoring setup. This is where a lot of the maintenance overhead for that year came from. In part it is counted in the initial development cost in Fig. 1. Customers also took the opportunity to revise their scoring methodology during the migration and this led to an inflation of the maintenance cost for that year.

2010 saw the implementation of six new scoring methodologies however it did not see a corresponding increase in the number of maintenance hours as would be expected from the curve for customizable scoring in Fig. 4. The amount of effort required for the maintenance of fourteen different scoring models is below the 2005 levels of maintenance for three scoring models. While the figures for the last three months of 2010 are not included in this paper there was no significant jump in development effort for that time period.

Overall the required maintenance was eight times less than the previous year. This is a remarkable decrease in maintenance effort, even when the inflated nature of the 2009 bug count is considered. This drop in bug numbers was expected. Configurable scoring was designed and developed based on what had been learned during the different custom implementations. Problem areas where designed out of the configurable approach. The decrease in maintenance hours is also in line with the decrease in implementation time for a scoring methodology shown in section A. Overall this equates to a real year on year saving in development effort.

C. Other Effects

The time taken to specify the different change requests for customized scoring and configurable scoring is not included in any of the previous calculations. This is partial because the information is not available for most of the changes. It is also due to the fact that any time taken to specify the previous version of scoring must count towards the time to specify the current scoring core functionality. The configurable scoring model was built upon what was learned over the lifetime of Blade of what the market place need for scoring. This learning came from implementing scoring models and from talking to prospective customers. It was the bringing together of the different strands imbedded in the different customer implementations that allowed the rich tapestry of configurable scoring to exist. The team that designed configurable scoring backs up this point of view in informal discussions. They believe that they could not have designed it without going through six years of listening to customers and designing changes to be made directly to the code.

Existing customer response to the changes has been positive. They have taken the opportunity to make changes to their risk models afforded to them by migrating from their existing custom coded implementation to an implementation configured to their needs. Apart from minor look and feel changes the front end as regards scoring appears to operate the same to end users as it did before. Some customers who use Blade only for the recording of risk events are also considering using risk assessment now that it has the flexibility to match their needs without a large cost attached for customizing the code.

Market response has also been good. Since 2009 the number of customers using Blade has doubled. This increase is not directly contributable to the changes to the scoring model however the ability to configure Blade instead of customize it has helped the company keep up the pace of sales. 2010 saw the record set for most concurrent implementations at nine. This increase in implementations was not met with a corresponding increase in staffing number or workload due in part to the change from a customization approach to a configurable one. Six of these implementations required a new scoring model. One of the new scoring models was very complex and required more than twelve scoring methodologies. Such a scoring model could not have been implemented without considerable investment if a customization approach was taken.

V. CONCLUSIONS

Ci3 Consulting has made great progress in improving both the quality of the code in Blade and the reducing the time it takes to create that code by adopting core asset development. The increased quality is confirmed by the reductions in bug numbers across the implementation relating to risk and control scoring. The increase in efficiency is shown by the comparison of initial development times between a customizable approach and a configurable approach using core assets. Implementations of new scoring methodologies are shown to be considerable faster using a configurable approach than customizing existing code for that methodology. The breakeven point, when the initial outlay of building the configurable core asset is considered, was achieved in the second year of the change after six implementations.

Using three tenets to the development of core assets enables Blade to avoid some of the pitfalls related to SPL development in small development organizations. These tenets, avoid risk, make developer friendly and communicate changes, encourage developers to focus on core assets and increase communicate between the different layers of the organization.

Making customers responsible for change within their own implementation lies at the heart of a lot of the user interface and core asset work that has been done. By giving the customer and individual users within that customer the power to configure the system to their needs at runtime we remove a lot of the burden of variability management from development while delivering added value to the customer. Allowing users within a company to customize their own experience also has unforeseen benefits. It aids collaboration between users [24]. It also allows them to use the product how they want to use it which is a key difference in the market.

To increase efficiency further Ci3 Consulting has componentized the specification so that business analysts can select core assets to implement changes. This is a good match for the concept of SPLs in that new products are developed by selecting and arranging a set of core assets.

It is our recommendation that similar sized companies with large variations between customers should develop configurable core assets that can handle existing customer requirements as well as future requirements. A mature base product can aid the development of core assets as a certain

amount of software entropy is required in order to make great core assets. Mistakes in any industry or practice need to be made before they can be corrected and a better product developed. A good bug recording system is beneficial to all companies in order to identify those bug black spots, however with a mature product the trouble areas can be identified by tribal knowledge and the identification of areas of code that developers hate working in.

Scaling to larger development organization is an unknown. It is unsure how the three tenets that we follow would scale to benefit companies with multiple development teams and projects. There are benefits to all sized companies in avoiding initial risk and encouraging development to build core assets by making development less onerous however communication between teams and departments becomes more complicated.

One area not touched on within this paper is validation of core assets. Ci3 Consulting has not yet developed a method to reduce the amount of time core asset testing requires. In fact changes to core assets can lead to retesting all implementation of that asset which can be a drain on the quality assurance team's resources. This is because we deal with testing core assets as we would deal with testing any other part of the systems. This seems to be an unsolved problem in the SPL community [26]. Applying the concepts of SPL development to the testing process as we have already applied it to the specification process may result in improvements in this area.

Overall Blade's transition to a SPL has been beneficial to Ci3 Consulting. The increases in productivity and implementation time have been key factors in increasing market share. This confirms that, for Blade, the benefits of SPLs are real. It is hoped that even greater benefits can be gained by continuing the focus on developing core assets and applying our experience of developing SPL to other areas of our development cycle, in particular quality assurance.

REFERENCES

[1] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison Wesley, 2001.

[2] Chu-Caroll, M. Separation of Concerns in Software Configuration Management, In *Proceedings of the ICSE 2000 Workshop on Multi-Dimensional Separation of Concerns in Software Engineering*, 2000

[3] R.S. Kenett and Y. Raanan, Operational Risk Management: A Practical Approach to Intelligent Data Analysis. John Wiley & Sons Ltd. 2011

[4] Basel Committee on Banking Supervision, *Consultative Document on Operational Risk*, Basel Committee, January 2001.

[5] European Parliament Directive number 2009/138/EC, *On the taking-up and pursuit of the business of Insurance and Reinsurance (Solvency II)*, Official Journal of the European Union, 25 November 2009.

[6] N.N. Taleb, The Black Swan: The impact of the highly improbable, Random House, New York. 2007

[7] Krueger, C. Churchett, D. Buhrdorf, R. "HomeAway's Transition to Software Product Line Practise: Engineering and Business Results in 60 Days." In *Proceedings of the 12th International Product Line Conference*, pp297-206, 2008

[8] Staples, M. Hill, D. "Experiences Adopting Software Product Line Development without a Product Line Architecture". Software Engineering Conference, 2004. 11th Asia-Pacific

[9] Tseng, M.M. and Piller, F.T., The Customer Centric Enterprise: Advances in Mass Customization and Personalization. 2003. Springer, New York.

[10] Gang Hong, Deyi Xue, Yiliu Tu "Rapid Indentification of the optimal product configuration and its parameters based on customer-centric product modelling for one-of-a-kind production" Computers in Industry Volume 61 Issue 3, April, 2010 pp. 270-279

[11] Alves, V.; Camara, T.; Alves, C.; , "Experiences with Mobile Games Product Line Development at Meantime," *Software Product Line Conference, 2008. SPLC '08. 12th International* , vol., no., pp.287-296, 8-12 Sept. 2008

[12] Clements, P. and Krueger, C. W. Being Proactive Pays Off/ Eliminating the Adoption Barrier. Point-Counterpoint article in *IEEE Software*, July/August 2002

[13] Bosch, J.; , "Software product lines: organizational alternatives," *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on* , vol., no., pp. 91- 100, 12-19 May 2001

[14] Pohl, K. Böckle G, and van der Linden F.J.. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[15] Takebe, Y. Fukaya, N. Chikahisa, M. Hanawa, T. and Shirai, O. 2009. Experiences with software product line engineering in product development oriented organization. In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, pp. 275-283.

[16] Tekinerdogan, B. Tüzün, E. Şaykol, E. Havelsan A. Ş. "Exploring the Business Case for Transitioning from a Framework-based approach to a software Product Line Enginneering approach" *Proceedings of the 14th International Software Product Line Conference* (SPLC '10). pp. 251-254

[17] Iwasaki, T. Uchiba, M. Ohtsuka, J. Hachiya, K. Nakanishiy, T. Hisazumiy, K. and Fukuda A. "An Experience Report of Introducting Product Line Engineering across the board" *Proceedings of the 14th International Software Product Line Conference* (SPLC '10). pp. 255-258

[18] Kruegar, C. Churchett, D. "Eliciting Abstractions from a Software Product Line"

[19] Gamma, E. Helm, R. Johnson, R. Vlissides, J. Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995

[20] Beck, K. Crocker, R. Meszaros, G. Vlissides, J. Coplien, J. O. Dominick, T. and Paulisch, F. 1996. Industrial experience with design patterns. In *Proceedings of the 18th international conference on Software engineering* (ICSE '96). IEEE Computer Society, Washington, DC, USA, pp. 103-114.

[21] O'Leary, P. Rabiser, R. Richardson, I. and Thiel, S 2009. "Important issues and key activities in product derivation: experiences from two independent research projects." In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, pp. 121-130.

[22] Krueger, C. W. Variation Management for Software Product Lines. In *Proceedings of the Second International Software Product Line Conference* (San Diego, CA, U.S.A., August 19-22 2002). Springer LNCS Vol. 2379, 2002, pp. 37-48.

[23] Pech, D. Knodel, J. Carbon, R. Schitter, C. and Hein, D. 2009. "Variability management in small development organizations: experiences and lessons learned from a case study." In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, pp. 285-294.

[24] Bentley, R. and Dourish, P. 1995. Medium versus mechanism: supporting collaboration through customisation. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work* (ECSCW'95), Hans Marmolin, Yngve Sundblad, and Kjeld Schmidt (Eds.). Kluwer Academic Publishers, Norwell, MA, USA, pp. 133-148.

[25] Pohl, K. and Metzger, A.. 2006. Software product line testing. *Commun. ACM* 49, 12 (December 2006), pp. 78-81.

[26] Denger, C. and Kolb, R. "Testing and inspecting reusable product line components: First empirical results" *Intl. Symposium on Empirical Software Engineering*, pp. 184--193,2006.