Research Journey Towards Industrial Application of Reuse Technique

Stan Jarzabek
Department of Computer Science
National University of Singapore
+65 68742863
stan@comp.nus.edu.sg

Ulf Pettersson
Technology Office
ST Electronics (Info-Software Systems) Pte. Ltd.
+65 64131434
ulfp@stee.stengg.com

ABSTRACT

Component-based reuse in mission critical Command and Control system domain was a starting point for a long lasting research collaboration between National University of Singapore (NUS) and ST Electronics Pte. Ltd. (STEE). STEE industrial projects as well as NUS lab studies revealed limitations of conventional architecture-centric, component-based reuse in the area of generic design to unify similarity patterns (e.g., similar classes, components or architectural patterns) commonly found in software. Further research showed that meta-level extensions to conventional techniques could strengthen their generic design capabilities, considerably improving effectiveness of reuse solutions, and increasing productivity gains due to reuse. These experiences led to development of "mixed strategy" approach based on synergistic application of meta-level generative programming technique of XVCL, together with conventional programming techniques. In the paper, we describe university-industry collaboration that proved beneficial for both parties: STEE advanced reuse practice via application of XVCL in several software product line projects. Early inputs from STEE helped NUS team validate and refine XVCL reuse methods, and expand into new research directions. We describe a sequence of projects that led to successful application of XVCL in industrial projects. We describe experiences from those projects and their significance for both industrial practice and understanding principles of flexible software, i.e., software that can be easily changed and adapted to various reuse contexts.

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object-Oriented Programming; D.2.2 [**SOFTWARE ENGINEERING**]: Design Tools and Techniques; D.2.10 [**SOFTWARE ENGINEERING]**: Design – Representations; D.2.13 [**SOFTWARE ENGINEERING]**: Reusable Software - *Domain engineering;*

General Terms

Design, Languages, Experimentation

Keywords

reuse, software product lines, maintenance, generic design

1. INTRODUCTION

In 1998, ST Electronics Pte Ltd (STEE) started a programme to develop a Common Application Platform (CAP) with the objective of providing fast and cost effective customized solutions in the Command and Control domain. CAP was built to form a foundation of reusable components designed to serve as low-level reuse libraries as well as higher-level services designed to facilitate implementation of design patterns. Around the same time, a collaboration agreement was signed between STEE and National University of Singapore (NUS), as a vehicle for joint research. As a result, students from NUS were attached to STEE to help in development of CAP.

Even though the reuse solutions developed for CAP have been used in many projects across STEE, and the programme has been considered as a big success, as STEE was progressing with development of CAP solutions, certain weaknesses of component-based reuse were exposed. In particular, as STEE over time deployed its reusable components to different customers, specific adaptations were often required in areas of business logic and almost always in the area of User Interface. To address such customer specific variations, the underlying component platforms and conventional design techniques proved ineffective in defining generic solutions to avoid explosion of many similar components. Because of these difficulties, the reuse of CAP solutions was limited to functional areas where variations were few and could be easily managed with conventional design techniques, while for other areas cut-paste-modify was applied resulting in explosion of similar components.

We believe problems that we experienced to some extent affect other attempts to implement reuse strategies. Component-based reuse facilitated by modern component platforms can yield significant benefits, but is mostly limited to reuse of common services and middleware. On a wider scale, especially in the areas of business logic and user interfaces, the depth of success of architecture-centric and component-based product line approach, or pattern-driven component reuse, has been rather limited. While there are reuse success stories [5][6], reuse has not become a standard practice, and many problems with realizing reuse strategies with conventional approaches have been reported [8].

At the end of 1999, in an effort to overcome identified limitations of component-based reuse and better address customer expectations, STEE and NUS teamed up with Netron Inc. (Toronto) and University of Waterloo to start a joint Singapore-Ontario research project in the area of "Software Reuse Framework For Reliable Mission-Critical Systems". Our intention was to evolve the frame-based product line techniques [4] used by Netron into a new language-independent and modern platform contexts, to facilitate development of Command and Control (C2) product lines. The project was conducted in the frame of the Singapore-Ontario Joint Research Programme funded by Singapore Agency for Science, Technology and Resaerch (A*STAR) and Canadian Ministry of Energy, Science and

Technology. The result of this project was XVCL reuse technology [16], and also the first pilot application of XVCL by STEE in a C2 Environment.

Results obtained in the first phase of collaboration were encouraging. By June 2002, the Singapore-Ontario project was over, but STEE and NUS continued their collaboration at an increasing intensity level. Collaboration no longer was driven by any specific research project agreement, but rather by specific projects and goals, centered around further development and application of XVCL. Further collaboration resulted in yet other XVCL projects that led to successful design of Web Portal product line at STEE. All our projects already applied good practices of conventional software design, before considering XVCL. Still, the XVCL technique helped us to achieve additional reuse in the range of 60-90% (in % of SLOC reduction), which also led to considerable productivity improvements.

In the rest of the paper, we explain the essence of the engineering problem that we solve with XVCL, with highlights from various projects.

2. DESIGN PROBLEM ADDRESSED WITH XVCL

In our studies of new, well-designed programs, we typically find 50%-90% of code contained in such similar program structures, repeated many times. For example, the extent of the redundant code in Java Buffer library was 68% [10], in parts of STL (C++) - over 50% [2], in J2EE Web Portals – 61% [17], and in certain ASP Web portal modules – up to 90% [14]. Repeated structures ranged from similar code fragments (often called software clones in literature), to large-granularity, design-level patterns of components (termed as structural clones).

In all the above studies, with the exception of [15] we paid attention only to repetitions of *significant engineering importance*, meaning that they created reuse opportunities, induced extra conceptual complexity into a program, and/or were counter-productive for maintenance. Avoiding them with conventional approaches was either impossible or would require developers to compromise other important design goals. However, we could unify all the similarity patterns with XVCL generic structures.

Most of the repetitions are counter-productive for maintenance and large granularity, design-level similarity patterns signify untapped reuse opportunities. Common sense suggests that we should be able to express our design and code without unwanted repetitions. This has been a goal of parameterization and other generic design techniques for decades.

Modern platforms (such as .NET™ or J2EE™, and web frameworks such as Ruby on Rails™) encourage organizing software around standard architectures which allows programmers to reuse common services/components. Pattern-driven development style even further standardizes software structure. Not surprisingly, software developed in that way displays much similarity. For example, we found 61% of code contained in similar program structures replicated many times in variant forms [17]. Despite benefits during development, pattern-driven design may add complexity to future maintenance. It may also complicate reuse of the application domain-specific functionality. This is due to the following reasons: (1) patterns remain implicit in code - we may not know the exact location of pattern instances in a program, and how pattern instances are similar and different from each other, (2) when the pattern-related code is to be changed, it is not clear which of the pattern's instances should be changed and how, and (3) application of patterns scatters code related to application-level functionality across many components (classes), which magnifies well-known problems of tracing requirements to code, and the impact of change, in general.

One important concept and construct is missing to fully exploit the benefit of design standardization. The missing concept is a strong enough mechanism for generic design that would allow us to represent and maintain similar program structures spawning from patterns in a generic, customizable form. Such generic representation should contain formal links to enable one to trace all its instances in a subject program, along with a detailed record of differences of each instance in respect to its generic representation. For complex patterns, one should be able to abstract similarities and structure a generic representation at as many levels of as it is required. It will further boost reuse levels and development/maintenance productivity if the process of injecting pattern instances into a program is automated. The "mixed strategy" approach presented in the tutorial is based on a mechanism for generic design that plays the above role in the context of programming languages and modern software platforms.

XVCL is a mechanism for generic design that plays the above role in the context of programming languages and modern software platforms. Aspect-Oriented Programming (AOP) [13] extends Object-Oriented techniques to express as self-contained meta-level modules computational aspects that crosscut program classes. In a similar way, XVCL provides meta-level decomposition and instrumentation mechanisms to unify any kind of similarity patterns that we wish to represent in a unique, generic and easily adaptable form. In particular, XVCL could effectively unify similarities found in the above mentioned lab studies and industrial projects that in conventional solutions showed as similar program structures repeated many times in variant forms.

An overall solution formed by a conventional technique and XVCL we call a "mixed strategy" solution. We define program logic, user interfaces, etc. using conventional techniques. The usual strategy is to use conventional design techniques, as long as we do not hit some complications related to changeability or genericity. We escape to XVCL when conventional techniques fail to provide a generic solution for the problem in hand. "Mixed strategy" emphasizes this synergistic way of applying conventional technique with XVCL to enhance engineering qualities of an overall solution.

Parameterization is a prime concept for generic design. In XVCL, we use unrestricted parameterization which, unlike conventional generics, is decoupled from the language core. Differences among similar program structures are unified by a generic XVCL structure. Variations among similar program structures are specified as deltas from the generic structure and automatically propagated to the respective instances in a program. From the XVCL window, a designer has a precise picture of program similarities (a generic structure) and differences (instances of a generic structure). An architecture of a program (in terms of its component structure) and code remain an integral part of a "mixed strategy" solution. Any future changes are done via generic structures, so the program proper and its XVCL extensions are always in sync one with another.

We can use "mixed strategy" to simplify a single program or to form a generic representation from which we can derive many similar programs. In the first case, instances of meta-components populate a single program. We gain changeability, as non-redundancy reduces the risk of update anomalies. In the second case, instances of meta-components populate many similar programs that we can build from the generic design. We gain reusability across similar programs.

3. PROJECTS WITH XVCL

We conducted numerous lab studies and industrial pilots with XVCL. In all the experiments, we always started by detailed analysis of a conventional program solution, from the point of view of genericity (reusability) and/or changeability. The key technique that we used was to study similarity patterns, giving most importance to design-level similarities. Most often, repetitions marked program areas where program was getting complicated, and also hinted at reuse opportunities. We used combination of manual analysis, CCFinder/Gemini [12], and Clone Miner [1] to identify similarity patters (Clone Miner is capable of finding design-level similarities, as well as simple clones).

First, we would try to improve the conventional design to fix the problem – that is, to define suitable generic solution unifying a spotted similarity pattern. In case there was no simple way to do so, we would apply XVCL to do the job. In this way, a “mixed strategy” solution was emerging from the conventional program incrementally, each time addressing certain program area and/or certain similarity pattern we wanted to tackle.

In each experiment, we evaluated engineering qualities of both solutions – conventional program and “mixed strategy” solution – using quantitative and qualitative analysis methods, comparing the physical size, conceptual complexity, changeability and reusability.

We focused our first bigger NUS project on internet-enabled Computer Aided Dispatch Systems (CAD for short). STEE has implemented many CAD systems and provided NUS with real-world functional and quality requirements. CAD systems for Police and Fire&Rescue are similar, but specific context of the operation results in many variations. If we ignore commonalities, then each CAD system in a specific context becomes a unique application that must be developed from scratch and maintained as a separate product – an expensive and inefficient solution. In our project, we applied a product line approach that allowed us to exploit commonalities among CAD systems, and engineer CAD systems from a common base of reusable software assets. We expected such a reuse-based approach to radically cut development and maintenance cost. The pilot project confirmed those expectations, and also led to formulating and testing in practice novel methods to realize such benefits [18].

In the first pilot project, we also noted that, at times, simple cases of similar component configurations could not be represented in generic form using conventional technologies (we used Java, EJB™, and DCOM™). The best architectural design and the best use of conventional techniques could not avoid multiplication of similar versions of such components. This problem was contributing to increased complexity of a single CAD system, and also hindered reuse. We saw this as a major bottle-neck for effective reuse and intended to address it with XVCL. Even though our empirical studies of CAD domain were on a small scale, they clearly showed that the idea of “mixed strategy” approach was feasible and could have engineering merits.

Therefore, in the follow up studies, we focused on analyzing software similarity patterns in new, well-designed programs, and on generic design capable of unifying such patterns. We typically found 50%-90% of code contained in similar program structures, repeated many times, most of which were complicating a program and signified untapped reuse opportunities.

A study of the Java Buffer class library gave us an excellent opportunity to observe some important and common sources of similarity patterns, complications they caused, and limitations of OO techniques to deal with the problem [10]. Analysis of similarities revealed seven groups of buffer classes, each containing 7-13 classes similar to each other. Ad hoc differences among similar classes hindered attempts to unify them with inheritance, design patterns or generics. In some cases, when such conventional solution was technically feasible, it would be complicated (which would defeat the very purpose of applying it) or would require compromising other important design goals (such as usability, clarity of the design or performance).

In a “mixed strategy” Java/XVCL solution, we unified each of seven groups of similar buffer classes with unique generic, but adaptable, meta-class, along with information necessary to obtain its instances – specific classes. This unification reduced program complexity as perceived by developers, also reducing the original code size by 68% percent. Non-redundancy reduced the risk of update anomalies which helps in maintenance. Generics could unify 15 among 74 classes, but the solution was subject to further restrictions that were seriously affecting its engineering merits.

Analysis of Standard Template Library (STL) strengthened observations made in the Buffer Library case study as C++ templates are more powerful than Java generics. In certain areas of STL, we found a remarkable amount of similarity and code repetition that could not be avoided with conventional techniques, but could be unified with generic “mixed strategy” solution [2].

Another pilot project by STEE revealed problems similar to those we observed in class libraries, but in the command and control system implemented in C#. In this case, similarities showed as patterns of collaborating components, implementing similar operations for entities such as Task or User. Through application of XVCL, these similarities were unified resulting in more that 68% code reduction, and as much as 89% reduction in new code needed to add a new domain entity.

In the ASP Web Portal (WP) Product Line project STEE applied state-of-the-art design methods to maximize reusability of a Team Collaboration Portal (TCP) in other contexts. Still, a number of problem areas were observed that could be improved by applying XVCL [14]. The benefits of a “mixed strategy” ASP/XVCL solution for TCP were the following:

- Short time (less than 2 weeks) and small effort (2 persons) to transform the TCP into the first version of a “mixed strategy” ASP/XVCL solution.
- Eight-fold reduction of effort to build new portals. New portal modules could be built by writing as little as 10% of unique custom code, while the rest of code could be reused..
- Significant reduction of maintenance effort when enhancing individual portals. The overall managed code lines for nine portals were 22% less than the original single portal.
- Wide range of portals differing in a large number of inter-dependent features supported by the ASP/XVCL solution.

The above results encouraged us to look into reuse on J2EE™ platform. J2EE provides standardized architecture and supports reuse of common services via portlet technology. Unlike ASP, J2EE supports inheritance, generics and other OO features via Java 1.5. We studied similarity patterns in presentation, business logic layers, and Entity Bean components of the data access layer.

Our findings were similar to what we observed in earlier projects [17] (please refer to the discussion of pattern-driven development in Section 2).

4. CONCLUSIONS

We believe software similarities, especially large granularity, design-level similarity patterns, create opportunities for reuse within a given system, and also across similar systems. Unfortunately, conventional methods – component based, architecture-centric approaches as well as language-level features such as generics – often fail to provide effective means to reap benefits offered by software similarities. Many of the modern development platforms encourage design according to pre-defined patterns and architecture standards, without providing generic design mechanisms which are necessary to fully exploit benefits of such standardized design for reuse and during maintenance. We applied "mixed strategy" approach to address the problem and obtained encouraging results. While "mixed strategy" makes generic design easier, flexibility that we gain with XVCL does not come for free. There are important trade-offs to consider. The current form of XVCL can be seen as an assembly language for generic design via parameterization. XVCL's explicit and direct articulation is the source of its expressive power, but it also adds a certain amount of complexity to the solution. It is easier to understand a concrete program than a meta-program. It is also difficult to validate a meta-program as we can derive many concrete programs from it. However, we believe that the benefits of being able to deal with issues of genericity at the meta-level plane outweigh the cost of the added complexity. These benefits include ease of reuse with adaptation, the overall reduction of conceptual complexity and size of the solution, improved traceability and changeability. We are currently collecting empirical and analytical evidence to support the above hypothesis. Feedback from STEE, particularly a Web Portal product line project [14] is encouraging.

Engineering processes play an important role in industrial software development. Currently, we know how XVCL-enabled "mixed strategy" solutions can raise productivity of small teams of highly-skilled expert software developers. We have yet to learn what it takes to inject "mixed strategy" methods into more complex team structures and industrial development processes.

References

[1] Basit, A.H. and Jarzabek, S. "Detecting Higher-level Similarity Patterns in Programs," accepted for *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, September 2005, Lisbon, pp. 156-165.

[2] Basit, H.A., Rajapakse, D.C., and Jarzabek, S. "Beyond Templates: a Study of Clones in the STL and Some General Implications," *Int. Conf. Software Engineering, ICSE'05,* St. Louis, USA, May 2005, pp. 451-459.

[3] Batory, D., Singhai, V., Sirkin, M. and Thomas, J. "Scalable software libraries,", *ACM SIGSOFT'93: Symp. on the Foundations of Software Engineering,* Los Angeles, California, Dec. 1993, pp. 191-199

[4] Bassett, P. Framing software reuse - lessons from real world, Yourdon Press, Prentice Hall, 1997

[5] Bosch, J. *Design and Use of Software Architectures – Adopting and evolving a product-line approach*, Addison-Welsey, 2000

[6] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002

[7] Czarnecki, K. and Eisenecker, U. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000

[8] Deelstra, S., Sinnema, M. and Bosch, J. "Experiences in Software Product Families: Problems and Issues during Product Derivation," Proc. Software Product Lines Conference, *SPLC3*, Boston, Aug. 2004, LNCS 3154, Springer-Verlag, pp. 165-182

[9] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns – Elements of Reusable Object-Oriented Software*, 1995, Addison-Wesley

[10] Jarzabek, S. and Li, S. "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," *Proc. ESEC-FSE'03, European Software Engineering Conf. and ACM SIGSOFT Symp. on the Foundations of Software Engineering*, ACM Press, September 2003, Helsinki, pp. 237-246

[11] Jarzabek, S. and Zhang, H. "XML-based Method and Tool for Handling Variant Requirements in Domain Models", *Proc. 5th Int. Symposium on Requirements Engineering, RE'01*, August 2001, Toronto, Canada, pp. 166-173.

[12] Kamiya, T., Kusumoto, S., and Inoue, K. "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code", *IEEE Trans. Software Engineering,* 2002, 28(7): pp. 654-670

[13] Kiczales, G, et al "Aspect-Oriented Programming," *Europ. Conf. on Object-Oriented Programming,* Springer-Verlag LNCS 1241, 1997, pp. 220-242

[14] Pettersson, U., and Jarzabek, S. "Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach," *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symp. on the Foundations of Software Engineering*, ACM Press, Sept. 2005, Lisbon, pp. 326-335.

[15] Rajapakse, D. and Jarzabek, S. "An Investigation of Cloning in Web Portals," *Int. Conf. on Web Engineering, ICWE'05*, July 2005, Sydney, (also poster at WWW'05).

[16] XVCL (XML-based Variant Configuration Language) method and tool for managing software changes during evolution and reuse, http://fxvcl.sourceforge.net

[17] Yang, J. and Jarzabek, S. "Applying a Generative Technique for Enhanced Reuse on J2EE Platform," *4th Int. Conf. on Generative Programming and Component Engineering, GPCE'05*, Sep 29 - Oct 1, 2005, pp. 237-255.

[18] Zhang, H. and Jarzabek, S. A Mechanism for Handling Variants in Software Product Lines," *Science of Computer Programming*, Volume 53, Issue 3, Dec. 2004, pp. 255-436.

[19] Zhang, W. and Jarzabek, S. "Reuse without Compromising Performance: Experience from RPG Software Product Line for Mobile Devices," *9th Int. Software Product Line Conf., SPLC'05*, Sept. 2005, Rennes, France, pp. 57-69