University-Industry Collaboration Journey towards Product Lines

Stan Jarzabek[1], Ulf Pettersson[2], and Hongyu Zhang[3]

[1] School of Computing, National Univerisity of Singapore, Singapore
stan@comp.nus.edu.sg
[2] Technology Office, ST Electronics (Info-Software Systems) Pte. Ltd.
ulfp@stee.stengg.com
[3] School of Software, Tsinghua University, Beijing 100084, China
hongyu@tsinghua.edu.cn

Abstract. Product Lines for mission critical Command and Control systems was a starting point for a long lasting research collaboration between National University of Singapore (NUS) and ST Electronics (Info-Software Systems) Pte Ltd (STEE-InfoSoft). Collaboration was intensified by a joint research project, also involving University of Waterloo and Netron Inc. that led to development of reuse technology called XVCL. The contribution of this paper is twofold: First, we describe collaboration modes, factors that were critical to sustain collaboration, and benefits for university and industry gained over years. Among the main benefits, STEE-InfoSoft advanced its reuse practice by applying XVCL in several software Product Line projects, while NUS team received early feedback from STEE-InfoSoft which helped refine XVCL reuse methods and keep academic research in sync with industrial realities. Academic findings and industrial pilots have opened new unexpected research directions. Second, we draw lessons learned from many projects, to explain the general nature and significance of problems addressed with the XVCL approach.

Keywords: Software Product Lines, Industry collaboration, Variability management, Generative technique.

1 Introduction

Even though component-based reuse has yielded significant benefits in the past, the depth of success in the industry has been rather limited. While there are reuse success stories, even advanced Product Line reuse approach [6][7] has not penetrated the industrial software development deep enough to become a standard practice. Some problems with realizing reuse strategies have been reported [10].

ST Electronics (Info-Software Systems) Pte Ltd (STEE-InfoSoft) is a Singapore-based company developing turn-key software solutions in a wide range of domains, for local and international markets including defense and home-land security applications. In 1998, STEE-InfoSoft 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. Around the same time, a collaboration agreement was signed between STEE-InfoSoft and National University of Singapore (NUS), as a vehicle for joint research. As a result, students from NUS were attached to the company to help in development of CAP.

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. Even though the reuse solutions developed for CAP have been used in many projects across STEE-InfoSoft, and the programme has been considered as a big success, certain weaknesses of component-based reuse have been also exposed.

In particular, as STEE-InfoSoft 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. For those reasons, without complementary techniques, the component-based approach would not have been able to keep up with the customer evolving expectations of shorter time to market and more cost effective solutions.

We believe challenges that STEE-InfoSoft experienced to some extent affect other attempts to implement reuse strategies. Component-based reuse facilitated by modern component platforms is mostly limited to common services and middleware layers. Reuse potentials on a system-wide scale, especially in the application domain-specific areas of business logic and user interfaces, are more difficult to realize with component-based techniques. Furthermore, in our experience, the benefits of reuse and component platforms are mainly observed during new development, but are less evident in long-term evolution of successful products.

In an effort to overcome identified problems and better address customer expectations, STEE-InfoSoft 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 [3] used by Netron into a new language-independent and modern platform contexts, to facilitate development of Command and Control (C2) Product Lines. The result of this project was the XVCL language [21][12], the XVCL Processor and also the first pilot application of XVCL by STEE-InfoSoft in a C2 Environment.

By June 2002, the Singapore-Ontario project was over, but NUS and STEE-InfoSoft continued their collaboration at an increasing intensity level, even though the collaboration no longer was driven by any specific research project agreement. This collaboration resulted in several interesting XVCL projects that will be described in Section 4.

The contribution of this paper is twofold: *First*, we distil lessons-learned from 10-years of our project collaboration, focusing on collaboration modes, factors that were critical to sustain collaboration, and benefits for university and industry gained over years. Among the main benefits, STEE-InfoSoft advanced its reuse practice by applying XVCL in several software Product Line projects, while NUS team received early feedback from STEE-InfoSoft which helped refine XVCL reuse methods and keep academic research in sync with industrial realities. Academic findings and industrial pilots have opened new unexpected research directions.

Second, we generalize experiences from many projects, most of which we described in earlier papers. Based on that, we explain the general nature, significance and unique contribution of XVCL to the today's toolbox of software methods, and argue about the merits of the XVCL approach on a more general ground than we could do in earlier papers.

In Sections 2, we describe the history and models of our collaboration. In Section 3, we explain the process that led to formulation and application of variability technique of XVCL that was subject of our collaboration. In Section 4, we recap experiences from several projects with XVCL, highlighting the impact of the results and trade-offs involved in application of XVCL. We summarize lessons learned in Section 5. Conclusions end the paper.

2 The History of Collaboration

2.1 First Phase: MOU

Our collaboration started in 1998 when a Memorandum Of Understanding (MOU) was established between the School of Computing at the National University of Singapore (later referred to as NUS) and ST Electronics (Info-Software Systems) Pte Ltd (later referred to as STEE-InfoSoft).

Initial collaboration was facilitated through an Industrial Attachment, where Honours and Master students were attached to STEE-InfoSoft for research projects related to reuse frameworks. First few projects were focused on reliable use of DCOM (http://msdn.microsoft.com/library/) in Mission Critical Command and Control Systems. These first projects delivered concrete and useful benefits to both sides, helping STEE-InfoSoft in technology selection and helping NUS to bring industry-related projects to their students. Positive experiences acted as a booster for the collaborative spirit and brought the two sides closer together looking for more and bigger exploration in the area of Product Lines.

2.2 Second Phase: Singapore-Ontario Project

The governments of Singapore and Ontario, Canada, have established a joint research programme to boost collaboration among universities, and involving industrial partners from both countries. Under this research scheme, in 2000, we started a project to investigate methods for cost-effective, reuse-based development of reliable mission-critical software systems. The project involved four partners, namely NUS, STEE-InfoSoft, Netron Inc. (Toronto), and University of Waterloo. NUS provided software engineering and reuse expertise. University of Waterloo contributed in areas of software reliability – failure detection, fault tolerance and availability. STEE-InfoSoft has been developing command and control mission-critical systems for customers in Singapore (such as Ministry of Defense, police and civil defense) as well as abroad, and had extensive experience in mission-critical system domain.

STEE-InfoSoft was also a potential client for the technology we intended to develop in this project. Our Canadian industrial partner, Netron, Inc. contributed to our project with reuse tools and their rich experience in implementing reuse solutions in companies.

Before formulating the joint project proposal, we had already established a working relationship among project partners, though not all four of them together and not in the exact scope of the proposed project. NUS and University of Waterloo had been planning to pursue joint research on the interplay between reuse and reliability, during sabbatical leave of one of the authors at the University of Waterloo. As mentioned earlier, the link between STEE-InfoSoft and NUS was already established through the MOU and close collaboration was already in place. Finally, the NUS team worked with Frame Technology developed by Netron, Inc. before the joint Singapore-Ontario project, and had accumulated experiences in that area and described them in publications. These existing links helped a lot the four partners in two countries agree on common research goal, and on the approach to working towards the goal.

As our project progressed, the following three focus areas emerged:

- Definition of the XVCL language.
- Implementation of the XVCL Processor [21].
- XVCL-based pilot project [17].

Definition of the XVCL language was facilitated primarily through collaboration between NUS and Netron, where Netron shared their use of frame technologies both through visits to Singapore and through short attachments of NUS students at their Toronto office. The frame-related experiences and feedback provided by Netron (and in particular by Paul Bassett) was essential to the successful definition of a simple yet practical XVCL language.

Implementation of the XVCL Processor was done by students at NUS. As we chose XML as a vehicle for defining and then implementing XVCL, we could benefit from use of open-source components. In 2002, the resulting processor was also made public at SourceForge [21].

Experimentation with XVCL-based Product Lines was done through a pilot project for Computer Aided Dispatch (CAD) in the domain of Police and Fire emergency dispatch. This pilot project started with definition of use cases and study of feature variants both within a police system but also across other areas of the civil defense domain. We worked with software requirements (use cases). Use cases abstracted from real-world projects, contributed by the STEE-InfoSoft, established an understanding of requirements, while NUS students explored how XVCL could be applied to handle feature variants across the Product Line. Implementation was done jointly by NUS students and STEE-InfoSoft staff. The pilot project demonstrated that the XVCL approach was very capable of handling variants in CAD Product Lines, and the result formed an incubator for new experimentation and application of the XVCL technique.

These empirical studies were instrumental in gaining insights into the design of "flexible software", i.e., software that is easy to change and adapt to fit various reuse contexts. We tested the limits of what could be achieved to this end by conventional architecture-centric, OO and component-based programming techniques, and with this understanding it became possible to observe the value of meta-level enhancements implemented into the XVCL method.

In Singapore, overall control and evolution was facilitated through weekly (or bi weekly) research working sessions with participation from both NUS and STEE-InfoSoft. These sessions served as communication channel where:

- ➢ Result, ideas and findings were shared.
- ➢ Feedback was provided.
- ➢ Future work was brainstormed and outlined.

Working sessions served as a vehicle for sharing of experiences and findings well before publications were written, resulting in faster and more agile direction changes. We believe it helped us a lot to accelerate and effectively shape our research.

Initial sessions focused on clarifying requirements and documenting them in a standard way. Subsequent sessions concentrated on discussing novel approaches to modeling “requirements with variants” [15], as it was needed for reuse via Product Line approach, and on novel techniques for designing generic software architectures, capable of handling variant requirements in an effective and simple way.

Working sessions played an increasingly important coordination role as over time, more and more parallel and incremental projects branched out from the second phase.

Finally, working sessions have helped us to strengthen the partnership between NUS and STEE-InfoSoft, and served as a vehicle for continued collaboration beyond the second phase. Through these meetings, the collaboration entered into a third phase, where new projects were initiated, executed and shared without any formal agreement between the parties (apart from the general MOU).

2.3 Third Phase: Continuous Collaboration

In the third phase (still ongoing), we leverage on the XVCL technique to explore Product Lines and reuse in various domains, and we also venture into other research areas inspired by results we were getting on the way. The projects of the third phase are described in later sections, once we have briefly explained motivation and concepts of a XVCL approach to software development.

3 The Development of the XVCL Approach

In this section, we describe the role of our collaboration in development of the XVCL approach to variability management in software Product Lines. The three phases in XVCL development correspond to the three phases of our collaboration described in the previous section.

3.1 Initial Phase

In this phase, the idea of XVCL was initially formulated. XVCL is based on principles of Frame Technology™ by Netron, Inc. [3]. A number of frame-based systems have been implemented in industry and at universities, and we believe any of those systems can handle engineering problems we addressed in our projects. Frames have been extensively applied to maintain multi-million-line COBOL-based information systems. An independent assessment by QSM Associates, Inc. showed that frames could achieve up to 90% reuse, reduce project costs by over 84% and their time-to-market by 70%, when compared to industry norms [3].

The successes of Frame Technology motivated us to explore it further. Designed in 1970s and 1980s, frame commands and tools are very much influenced by the COBOL language and do not address many contemporary design methods and language features. Frame technology should be enhanced to blend into contemporary software development practices (such as architecture-centric, component-based product line development). We thus proposed XVCL to refines frame concepts into a general-purpose language and tool. XVCL can be applied on top of the contemporary programming paradigms to achieve enhanced flexibility and genericity. We also planned to apply XVCL to the practices of STEE-InfoSoft.

3.2 Second Phase

In this phase, the idea of XVCL was developed. In our studies, we found that similarities are omnipresent in software. We repeatedly apply similar design solutions to solve similar problems. In new, well-designed programs, we often find 50%-90% of code contained in similar program structures of various types and granularity, repeated many times (often called clones in the literature). For example, the extent of the redundant code in Java Buffer library was 68% [13], in parts of STL (C++) - over 50% [2], in J2EE Web Portals – 61% [22], and in certain ASP Web portal modules – up to 90% [17]. Similar results have been observed in studies of Open Source web projects [19]. Most of the repetitions that we found represented some important concepts from requirement or design spaces. In our judgment, repetitions were counter-productive for maintenance and signified untapped reuse opportunities.

Software similarities, especially large granularity, design-level similarity patterns, create opportunities for reuse within a given system, or even across similar systems. Unfortunately, at times, conventional methods – component based, architecture-centric approaches as well as language-level features such as generics – fail to provide effective means to reap benefits offered by software similarities. Common sense suggests that we should be able to express our design and code without unwanted repetitions, whenever we wish to do so.

The goal of XVCL is to provide a systematic treatment for the above problems. Developers still use one of the programming languages to define the behavioral core of their program solutions (e.g., user interfaces, business logic or databases). However, when repetitions become evidently counter-productive, and conventional techniques are not sufficient to achieve generic design, rather than using ad hoc solutions, developers can escape to the well thought-out mechanisms to deal with the problem. XVCL defines such mechanisms. XVCL complements conventional OO, component-based and modularization techniques to fully exploit the engineering potential of software similarities.

With XVCL, we represent each group of recurring similar program structures of significant importance with a *generic, adaptable structure*. XVCL representation maintains a complete picture of similarities and differences among specific program structures, instances of the generic structure, as well as their location in a program.

Variations among instances are specified as deltas from the generic structure and automatically propagated to the respective instances. These specifications are both human-readable and executable by the XVCL Processor. Based on the specifications, the Processor adapts generic structures to generate specific program structures in their required variant forms.

3.3 Third Phase

In this phase, we applied XVCL to several projects, including:

- Application of XVCL to strengthen conventional OO techniques in the area of generic design, demonstrated in studies on unifying similarity patterns in Java Buffer library [13] and STL [2].
- Application of XVCL to support a Web Portal Product Line [17] in Active Server Pages (http://msdn.microsoft.com/library/) environment.
- Application of XVCL for reconstruction and reuse within Web Portals in J2EE™ environment [22].
- Application of XVCL to manage variability in a role-playing game Product Line for mobile devices [24], which demonstrated that reuse may go hand-in-hand with improving, rather than compromising, the performance.
- Techniques for detecting design-level similar program structures, so-called structural clones [1], which extended current techniques focusing mainly on detecting similar code fragments.

The above projects already applied good practices of conventional software design, before considering XVCL. Still, the XVCL helped us to raise reuse rates, typically by 60%, which also led to significant simplification of the subject software and productivity improvements. These encouraging results triggered exploration into other research areas such as: tools/techniques to identify, classify and understand design-level similarity patterns in legacy code [1], tools for XVCL development (such as smart editor, static/dynamic analyzer, and debugger), and XVCL language integration into Integrated Development Environments such as Visual Studio .NET™ and JBuilder™.

4 Summary of Typical Projects

To highlight the significance of our results, we briefly discuss representative projects and variability problems we tackled with XVCL. The first project initiated our collaboration; in the second one, we show how variability technique can enhance design of class libraries; in the third project, STEE-InfoSoft's applied XVCL to manage variability in Web Portal Product Line.

4.1 Pilot CAD project

Internet-enabled Computer Aided Dispatch Systems (CAD for short) was our first pilot project. Fig. 1 depicts a basic operational scenario in a CAD system for Police. An Operator receives information about an incident and informs a Dispatcher about the incident. The Dispatcher examines the "Situation Display" that shows a map of the area where the incident happened. Then, the Dispatcher assigns a task of handling the incident to a Police Unit taking into account the distance of a Unit to the place of incident and possibly other criteria. The Police Unit approaches the place of incident and handles the problem. The Police Unit informs the Task Manager about the progress of action. The Task Manager monitors the situation and at the end – closes the case. The information about current and past incidents is stored in the database.

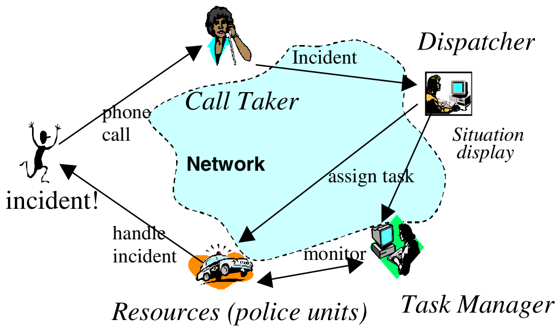

Fig. 1. CAD system for Police

CAD systems are used by police, fire & rescue, and health organizations. At the basic operational level, all CAD systems are similar – basically, they support the dispatch of units to incidents. However, there are also differences across CAD systems. The specific context of the operation (such as police or fire & rescue) results in many variations on the basic operational scheme. For example, CAD systems differ in rules of how resources are assigned to tasks, monitoring, reporting and timing requirements, specific information to be stored in a database, system component deployment strategies, reliability and availability requirements, and so on. 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 [6][7] to exploit commonalities and engineer CAD systems from a common base of reusable software assets, so-called Product Line architecture. We expected such a reuse-based approach to radically cut development and maintenance cost.

CAD systems offer high potential for reuse and, at the same time, pose important challenges for reliability. A typical CAD system must be pretty reliable – for example, 999 call reports should never be lost. So cost reduction must not come at the expense of reliability. CAD project allowed us to understand the interplay between reuse and reliability.

Basing CAD systems on internet, while meeting their real-time and reliability requirements in a manner that would allow high degree of software reuse posed significant research challenges for the project. Given that, and also the fact that we were applying XVCL for the first time, we kept the scope of CAD project simple: Java/XVCL CAD Product Line architecture contained 82 x-frames unifying groups of similar components in user interface and business logic layers. We addressed 24 variants that differentiated CAD systems. In each of the two new CAD systems developed based on the Java/XVCL Product Line architecture we achieved reuse ratios of 84%. Reuse ratio is defined as (Reused LOC) / (Total CAD LOC)*100%), where LOC are physical lines of code without blanks or comments. Further details of this project are described in [23].

The CAD project played two roles: First, it established trust and effective modes of collaboration among parties. We got a sense of benefits that collaboration could bring on both ends. Second, we could observe the strengths of conventional component-based techniques to support reuse, and also their limits in exploiting similarity patterns by means of generic design. We understood how mixed XVCL could help us overcome some of those limits.

4.2 Java Buffer Library and STL

These two projects allowed us to better see the roots of the similarity phenomenon and understand the essence of the problem we were addressing with XVCL in the context of OO techniques such as generics (or templates in C++), inheritance, abstract classes and dynamic binding. As we have described these results in detail in other publications [13][2], here we only recap the main findings.

Classes in the Buffer library JDK 1.5 differ in features such as a memory scheme: Heap or Direct; element type: byte, char, int, double, float, long, or short; access mode: writable or read-only; byte ordering: S – non-native or U – native; B – BigEndian or L – LittleEndian. Each legal combination of features yields a unique buffer class, with much similarity among classes. Analysis of similarity patterns in the Buffer library revealed seven groups of classes, each containing 7-13 classes similar to each other. Most of the variations among similar classes could be traced to feature combinations affecting classes or methods. Many similar classes or methods occurred due to the inability to unify variations in otherwise the same classes or methods.

Furthermore, any attempt to unify similarities would have to be synergistic with other design goals such as usability, conceptual clarity and good performance of buffer classes. In some situations, designers could introduce a new abstract class or a suitable design pattern to avoid repetitions. However, such a solution would compromise these goals, and therefore was not implemented. Java generics proved not effective in unifying similar classes either [13].

In a Java/XVCL solution, we could unify classes in each of the seven groups with generic x-frames. XVCL Processor generated Buffer classes from the Buffer class x-framework. This unification reduced program complexity as perceived by developers, also reducing the original code size by 68% percent. A controlled maintenance experiment revealed higher effort to maintain the original Buffer library as compared to its Java/XVCL representation: For example, the number of modifications to implement a new Complex buffer in Java/XVCL representation was 11, as compared to 91 modifications required for the same purpose in the original Java classes. Non-redundancy achieved by unifying similar classes made modifications easier, enhancing the visibility of ripple effects, and reducing the risk of update anomalies.

The Standard Template Library (STL) strengthened observations made in the Buffer Library [2]. Parameterization mechanism of C++ templates is more powerful than that of Java generics, due to light integration of templates with the C++ language core. STL uses the most advanced template features and design solutions (e.g., iterators), and is widely accepted in the research and industrial communities as a premier example of a generic programming methodology.

Still, we found much repetitions in some STL areas that could not be unified with conventional techniques. For example, four 'sorted' associative containers and four 'hashed' associative containers could be unified with two generic C++/XVCL containers, achieving 57% reduction in the related code. Stack and queue contained 37% of cloned code. Algorithms set union, intersection, difference, and symmetric difference (along with their overloaded versions) formed a set of eight clones that could be unified by a generic XVCL set operation, eliminating 52% of code.

4.3 Industrial Applications of XVCL

Web Portal (WP) Product Line was the first STEE-InfoSoft's application of XVCL in a business product and on a wider scale. A Team Collaboration Portal (TCP) was a starting point for this project. TCP was implemented in ASP. STEE-InfoSoft applied state-of-the-art design methods to maximize reusability of TCP in other contexts. Still, a number of problem areas were observed that could be improved by applying XVCL to increase the genericity of a conventional solution. The benefits of an 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 the ASP/XVCL solution.
- ➢ High productivity in building new portals from the ASP/XVCL solution. Based on the ASP/XVCL solution, STEE-InfoSoft could build new portal modules by writing as little as 10% of unique custom code, while the rest of code could be reused. This code reduction translated into an estimated eight-fold reduction of effort required to build new portals.
- ➢ 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 reader may find full details of this project in [17]. In another industrial project, XVCL was applied to manage variability in a role-playing game Product Line for mobile devices [24]. This project demonstrated that reuse may go hand-in-hand with improving, rather than compromising, the performance.

4.4 Discussion of Project Experiences: Benefits and Trade-Offs

We believe the benefits observed in the above projects are not accidental, but are the result of effective treatment of some problems that are not easily solved with programming language and component platform mechanisms. At the same time, these problems have significant impact on software productivity.

With XVCL, we represent each of the important groups of similar program structures in a unique generic, but adaptable form, along with the information necessary to obtain its instances (i.e., specific program structures). Such a non-redundant program view reduces program complexity as perceived by developers, and reduces the risk of update anomalies which helps in maintenance. At the same time, non-redundancy is difficult to achieve with architecture-centric and component approaches alone.

XVCL program representation contains much information about program design that is useful for maintenance and reuse, in the form that is fully integrated with complete information about the subject program(s) itself. This, for example, includes explicit representation of a program design in terms of its subsystems, architecture, component layers, components and classes, down to every detail of code implementing the above program structures.

As a variation technique in the design of Software Product Line core assets, XVCL allows domain engineers to clearly mark the impact of variant features on product architecture and code components. Product developers can trace and understand this impact, and then use XVCL Processor to automatically generate custom products with required features.

An important add-in value that XVCL brings to component-based reuse is its ability to handle product-specific variants separately from generic, reusable components. This unique feature of XVCL allows developers to evolve the many products we have derived from the generic components according to the specific needs of their customers, without ever disconnecting them from the generic components. Product-specific variants do not pollute generic components, and do not affect other products derived from those components, if this is not required. This feature, along with the ability to represent any group of similar programs structures in a generic, adaptable form allows XVCL to exploit reuse opportunities that are often missed by conventional component-based design techniques.

Conventional component-based reuse is most effective when combined with architecture-centric, pattern-driven development which is now supported by the major platforms (such as .NET™ and J2EE™). Patterns lead to beneficial standardization of program solutions and are basic means to achieve reuse of common service components. Our projects demonstrated that further improvements are possible by applying XVCL on top of these modern development practices. By packaging patterns into XVCL structures we enhance the visibility of pattern application. In particular, XVCL representation shows the exact location of pattern application, as well as the exact differences among pattern instances in a program.

Industrial applications have demonstrated that XVCL technique is easy and fast to learn, and its benefit may outweigh the cost of the added complexity [17]. At the same time, the return on investment may be quick and substantial. Industrial applications have also revealed a number of further problems. Flexibility that we gain with XVCL does not come for free. As we relax the coupling between the parameterization mechanism and the rules (syntax and semantics) of the underlying programming language, the power of the parameterization mechanism increases. We can address genericity concerns without compromising program runtime properties. But as we move towards less restrictive parameterization mechanisms, we also decrease type-safety of parameterized program solutions.

Designing generic, reusable and maintainable solutions is always a challenge which requires more talent and skill than building a concrete program. XVCL is not a substitute for thinking, on contrary, it requires more thinking and up-front investment for future benefits. XVCL targets at long-lived programs that undergo extensive evolutionary changes, or need be tailored to needs of multiple customers.

XVCL software representation is expressed at two inter-mixed levels, in base programming language(s) and XVCL. This creates extra difficulties, especially for debugging. However, we must keep in mind that XVCL representation contains much useful information for maintenance and reuse, in addition to complete information about the subject program(s) itself. There is a great opportunity here for XVCL-support tools to help developers work with XVCL software representation.

The current form of XVCL can be seen as an assembly language for generic design. XVCL's explicit and direct articulation is the source of its expressive power, e.g., we can unify arbitrary types of variations across similar program structures, but it also adds a certain amount of complexity to the problem. Specification, analysis and validation methods for XVCL representations are yet to be discovered.

Engineering processes play an important role in industrial software development. Currently, we know how XVCL-enabled solutions can raise productivity of small teams of highly-skilled expert software developers. We have yet to learn what it takes to inject XVCL into more complex team structures and industrial development processes. Working on those issues is an important direction for our future work, but we realize difficulties involved.

Adopting a new technique always brings some overheads and XVCL is no different. It is essential to understand and evaluate trade-offs involved. In future we will also focus on empirical studies in various application domains and interpretation of the results and comparative studies of XVCL.

Other generative techniques such as AOP [16], MDSC [20] or AHEAD [5], exploit separation of concerns as means to simplify software development and maintenance. Separation of concerns makes software components more generic, reusable in multiple contexts, and easier to maintain. XVCL shares similar engineering goals with other generative techniques, but uses different means to achieve these goals. The emphasis on identifying any kinds of similarity patterns, and capability to unify arbitrary differences among similar program structures is a unique characteristic and contribution of XVCL, and frame principles in general.

5 Lessons Learned

We learned the following lessons from our collaboration:

- **The importance of early feedback on research progress from industry:** From the beginning of the project, we worked with requirements provided by our industry partner. We worked together on technical solutions, and on method formulation. Industry partner shared with research team the knowledge and constraints of industry practices which helped us modify our approach early to stay relevant to practice. We were meeting regularly, getting invaluable feedback. We think these were critical factors that allowed us to progress fast. Everybody involved in the project was benefiting. One PhD thesis, six Master thesis and many undergraduate research projects were completed during the initial three years of our collaboration, and many more sparked from the initial results. Three students involved in project were subsequently employed by ST Electronics (Info-Software Systems). Undergraduate students worked as interns in the company. The project responsibilities did not delay their degrees. On the contrary, helped them faster complete their theses. The fact that we could evaluate the effectiveness of our approach in real industrial environment could appear difficult initially, but was most beneficial in long run. In the course of our collaboration, STEE-InfoSoft advanced its reuse practice by applying XVCL in several software product line projects. NUS team received early feedback from STEE-InfoSoft which helped refine XVCL reuse methods and keep academic research in sync with industrial realities. Academic findings and industrial pilots have guided our research in new unexpected directions.

- **Broad, fundamental research base for collaboration:** Our collaboration evolved around reuse techniques for software Product Lines, a very broad and fundamental research theme, encompassing domain modelling, conventional design methods and variability management, in our case, realized by XVCL. XVCL is to software engineering what a Generic Application Platform is to software development. It is foundation method that can be applied on many technologies and application domains such as Command and Control, Web, or Hand held Applications. This characteristic helped us find a continuous flow of new projects and sustain collaborative research. This characteristic allowed us to progress from one project to another in cycles involving working on a project, assessing the results, extraction of ideas and formulating spin-off-new-project. Each time we learned something, we got ideas about new areas where benefits of XVCL might be significant. We think that in case of narrower, less fundamental (and more vertical) themes for collaborative research it may be difficult to sustain and find new projects time-after-time.
- **Free, problem-oriented, result-driven collaboration style:** It is most important to work on problems of mutual interest and benefit. Our collaboration was fueled by mutual interest of academic and industry partners in project findings. We believe this was a single most important factor to sustain high level of enthusiasm for collaboration in long-term. Often collaboration between university and industry is restricted to a certain topic/theme, with a well-planned schedule. Our collaboration was in a fairly informal, problem-oriented way, focusing on specific projects around XVCL. We made very little long-term plans for our collaborative work. We let the results so far and the current needs drive the selection of projects we embarked on. Such a relaxed and open attitude towards collaboration requires much trust. In return, we can always focus on our strengths, and not miss the best opportunities that current situation has to offer. For example, STEE-InfoSoft applied XVCL to support a Web Portal Product Line under unexpected business pressure, without much prior planning. The results were so good that NUS initiated a number of research studies in the Web domain that revealed unique opportunities for our techniques in this new domain. In a short time, we advanced our understanding of the interplay and synergy between advanced Web technologies and our technique, and learned what it took to turn these findings into further improvements of our approach.
- **The importance of effective communication:** Industrial people tend to describe problems in pragmatic terms, while academic people tend to use “jargons” from recent research conferences or journals. Therefore, it is important to achieve effective communications so that each party can understand the other’s concerns and appreciate the other’s efforts. Furthermore, our project involved people from

multiple countries (Singaporean, Polish-Canadian, Swedish, Chinese, Myanmar, etc.), with different languages, social and cultural backgrounds. The multi-cultural/country aspect collectively gave us a broader brain-base for ideas thus also contributing to the spin-off of more projects. But such diverse backgrounds influence the way people think, feel, and act under different circumstances. This may create communication challenges. Establishing effective communication channels is important for a multi-national collaboration project like ours to succeed. Frequent face-to-face discussions and social events allowing team members interact in relaxed and informal setting helped us build mutual understanding, trust and team spirit.

6 Conclusions

This experience report documents ten years of university-industry collaboration between National university of Singapore and ST Electronics (Info-Software Systems) Pte. Ltd. The project was of mutual interest and brought benefits for both sides. In particular, collaboration led to industrial application of variability technique developed at the university. Numerous graduate (PhD and Master) thesis and undergraduate research projects benefited from collaboration. Three students involved in project were subsequently employed by STEE-InfoSoft. We hope that our experiences and lesson learned described in the paper can be beneficial to others.

Acknowledgments

We thank STEE-InfoSoft staff and NUS students who participated in various phases of our collaborative project. Paul Bassett's advice was invaluable in formalizing XVCL and understanding its applications.

References

1. Basit, A.H., Jarzabek, S.: Detecting Higher-level Similarity Patterns in Programs. In: European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC-FSE 2005, Lisbon, pp. 156–165. ACM Press, New York (2005)
2. Basit, H.A., Rajapakse, D.C., Jarzabek, S.: Beyond Templates: a Study of Clones in the STL and Some General Implications. In: Int. Conf. Software Engineering, ICSE 2005, St. Louis, USA, pp. 451–459 (May 2005)
3. Bassett, P.: Framing software reuse - lessons from real world. Yourdon Press, Prentice Hall (1997)
4. Batory, D., Singhai, V., Sirkin, M., Thomas, J.: Scalable software libraries. In: ACM SIGSOFT 1993: Symp. on the Foundations of Software Engineering, Los Angeles, California, pp. 191–199 (December 1993)
5. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. In: Proc. Int. Conf. on Software Engineering, ICSE 2003, Portland, Oregon, pp. 187–197 (May 2003)
6. Bosch, J.: Design and Use of Software Architectures – Adopting and evolving a product-line approach. Addison-Wesley, Reading (2000)
7. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2002)

8. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Reading (2000)
9. CPG-Nuke home, `http://www.cpgnuke.com/`
10. Deelstra, S., Sinnema, M., Bosch, J.: Experiences in Software Product Families: Problems and Issues During Product Derivation. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 165–182. Springer, Heidelberg (2004)
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
12. Jarzabek, S.: Effective Software Maintenance and Evolution: Reuse-based Approach. CRC Press. Taylor & Francis (2007)
13. Jarzabek, S., Li, S.: Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique. In: Proc. of ESEC-FSE 2003, European Software Engineering Conf. and ACM SIGSOFT Symp. on the Foundations of Software Engineering, Helsinki, pp. 237–246. ACM Press, New York (2003)
14. Jarzabek, S., Seviora, R.: Engineering components for ease of customization and evolution. IEE Proceedings - Software 147(6), 237–248 (2000); a special issue on Component-based Software Engineering
15. Jarzabek, S., Zhang, H.: XML-based Method and Tool for Handling Variant Requirements in Domain Models. In: Proc. 5th International Symposium on Requirements Engineering, RE 2001, Toronto, Canada, pp. 166–173 (August 2001)
16. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-Oriented Programming. In: Liu, Y., Auletta, V. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
17. Pettersson, U., Jarzabek, S.: Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach. In: Europ. Soft. Eng. Conf. and Symp. on the Foundations of Software Engineering, ESEC-FSE 2005, Lisbon, pp. 326–335. ACM Press, New York (2005)
18. Rajapakse, D.C., Jarzabek, S.: Using Server Pages to Unify Clones in Web Applications: A Trade-off Analysis. In: Int. Conf. Software Engineering, ICSE 2007, Minneapolis, USA (May 2007)
19. Rajapakse, D., Jarzabek, S.: An Investigation of Cloning in Web Portals. In: Int. Conf. on Web Engineering, Sydney (July 2005); also poster at WWW 2005
20. Tarr, P., Ossher, H., Harrison, W., Sutton, S.: N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: Proc. International Conference on Software Engineering, ICSE 1999, Los Angeles, pp. 107–119 (1999)
21. XML-based Variant Configuration Language, `http://xvcl.comp.nus.edu.sg`
22. Yang, J., Jarzabek, S.: Applying a Generative Technique for Enhanced Genericity and Maintainability on the J2EE Platform. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 237–255. Springer, Heidelberg (2005)
23. Zhang, H., Jarzabek, S.: A Mechanism for Handling Variants in Software Product Lines. Special Issue on Software Variability Management, Science of Computer Programming 53(3), 255–436 (2004)
24. Zhang, W., Jarzabek, S.: Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 57–69. Springer, Heidelberg (2005)