University-Industry Collaboration Journey Towards Product Lines

An Experience Report

Stan Jarzabek
Department of Computer Science
School of Computing
National University of Singapore
+65 65612863
stan@comp.nus.edu.sg

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

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 Pte Ltd (STEE). 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 the 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 advanced its reuse practice by applying XVCL in several software Product Line projects, while NUS team received early feedback from STEE 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 conclusions from many projects, most of which we described in earlier papers, to explain the nature, significance and unique contribution of XVCL-based software methods on a more general ground that we could do in earlier papers.

1. Introduction

Even though component-based reuse, design patterns, modern platforms (such as .NET or J2EE), and other conventional reuse techniques have yielded significant benefits in the past, the depth of success in the industry has been rather limited. Conventional reuse techniques have somewhat failed to lived up to the expectations initially forecasted. While there are reuse success stories, even advanced Product Line reuse approach [8][9] has not penetrated the industrial software development deep enough to become a standard practice. Many problems with realizing reuse strategies have been reported [12].

ST Electronics (STEE) 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 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 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, and the programme has been considered as a big success, certain weaknesses of component-based reuse have been also exposed, and 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. In particular, as STEE 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 STEE 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 system-wide scale, especially in the application domain-specific 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. Furthermore, in our experience, the benefits of component platforms are mainly observed during new development, with personnel already familiar with platform mechanisms, but are less evident in long-term evolution of successful products.

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 [7] 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 [28], the XVCL processor [27] and also the first pilot application of XVCL by STEE in a C2 Environment.

By June 2002, the Singapore-Ontario project was over, but NUS and STEE 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 including:

- Application of XVCL to unify similarity patterns in Java Buffer library and STL by strengthening conventional OO techniques in the area of generic design [16].
- Application of XVCL for reconstruction and reuse within Web Portals in J2EE™ environment [29].

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 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 [2], tools for XVCL development (such as smart editor, static/dynamic analyzer [19], and debugger), and XVCL language integration into Integrated Development Environments such as Visual Studio .NET™ and JBuilder™.

The contribution of the paper is twofold: First, we distil lessons-learned from eight-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 advanced its reuse practice by applying XVCL in several software Product Line projects, while NUS team received early feedback from STEE 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 nature, significance and unique contribution of XVCL-based software methods and argue about the merits of the approach on a more general ground that we could do in earlier papers.

In this paper, we describe our joint research journey towards a practical and effective reuse technique for Product Lines. In Sections 2, we describe the initial collaboration journey and how it resulted in the development and application of the XVCL technique. In Section 3, we explain the XVCL approach. In Section 4, we recap experiences from several other projects with XVCL. We highlight the significance of the results and discuss trade-offs involved in application of mixed-strategy in Sections 5 and 6. We describe other spin-off projects in Section 7, and then conclude the paper.

2. The history of collaboration

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 Pte. Ltd. (later referred to as STEE).

Initial collaboration was facilitated through an Industrial Attachment, where Honours and Master students were attached to STEE for research projects related to Reuse Frameworks. First few projects were focused on reliable use of DCOM [14] in Mission Critical Command and Control Systems. These first projects delivered concrete and useful benefits to both sides, helping STEE in technology selection and helping NUS to bring industry-related projects to their students. These 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.

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 1999, 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, 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 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 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 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
- XVCL-based pilot project

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 attachment 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.

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, 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 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 meetings with participation from both NUS and STEE. These meetings served as communication channel where:

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

An important aspect of these meetings was to facilitate more communication between the research teams and focus areas. These meetings also served as a vehicle for sharing 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 meetings focused on clarifying requirements and documenting them in a standard way. Subsequent meetings concentrated on discussing novel approaches to modeling “requirements with variants”, 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.

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

Finally, meetings have helped us to strengthen the partnership between NUS and STEE, 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).

During the Singapore-Ontario project, we learned quite a bit about how to (and how not to) do joint research involving geographically distant research teams including both industrial and academic parties.

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 mixed-strategy approach to software development, and its realization with a generative technique of XVCL.
3. Mixed-strategy approach with XVCL

Similarities are inherent in software. We repeatedly apply similar design solutions to solve similar problems. In our studies of new, well-designed programs, we typically 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% [16], in parts of STL (C++) - over 50% [3], in J2EE Web Portals – 61% [29], and in certain ASP Web portal modules – up to 90% [22]. Similar results have been observed in studies of Open Source web projects [25]. 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 mixed-strategy approach 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 there are engineering benefits to avoid repetitions, but conventional techniques are not sufficient to achieve generic design, rather than using ad hoc solutions, developers can escape to the well thought-out generative mechanism to deal with the problem. XVCL defines such a mechanism. XVCL complements conventional OO, component-based and modularization techniques to fully exploit the engineering potential offered by software similarities. We call this synergistic merger of conventional techniques with XVCL a mixed-strategy approach.

A mixed-strategy software solution represents each group of recurring similar program structures of significant importance with a unique, generic, adaptable structure. Mixed-strategy maintains complete and clear 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.

We explain the general idea of a mixed-strategy approach with an example of a recent Web-based PCE (Project Collaboration Environment) project, that is well-suited for explaining essentials of the approach, and has not been described in our earlier papers. PCE was implemented in PHP [23], a scripting language widely-used in Web development. PHP supports a full range of OO features and can be embedded into HTML.

![Figure 1. PCE modules and architecture](image)

PCE supports collaborative work and includes 14 Modules such as Staff, Project and Task. CPG-Nuke Foundation defines a number of standard modules (e.g., History) and common services (e.g., user account management, session and access control or database access) that can be reuse via APIs. PCE Modules are deployed on top of CPG-Nuke Foundation [11].

PCE Modules implement similar Actions such as Add, Delete, Update, Display. Implementation of each Action involves a number of classes from user interface, business logic and database layers.

Having partially implemented PCE in PHP, we found much similarities in: (1) classes involved in the implementation of a single Action, (2) classes implementing different Actions of the same Module, (3) classes implementing the same Action of different Modules, and (4) classes implementing different Actions of different Modules. The nature and degree of intra-Module similarities (that is, (1) and (2)) and inter-Module similarities (that is, (3) and (4)) varied. Both intra- and inter-Module similarities were however important for clarity of the design.

Attempts to unify similarities with PHP were not successful. Despite much similarity, there were many ad hoc variations in implementation of the same Action (e.g., Add or Delete) in the context of different Modules (e.g., Add for Staff, Project or Task). Generic PHP solutions were soon becoming large, with many if-elseif-else constructs catering for these variations. Code related to specific Actions was scattered across many places, in many PHP files. We felt that such a generic solutions was not scaling up.

In a PHP/XVCL mixed-strategy solution, we consider groups of similar program structures. They include small-granularity structures such as similar UI forms, classes implementing business logic or database connection, and large-granularity structures implementing Actions for Modules, and PCE Modules themselves. For each
important group of similar structures, we built a generic PHP/XVCL representation from which we can generate its custom instances, as needed in PCE. The overall PHP/XVCL PCE representation is built as a hierarchical composition of generic structures, whereby higher-level generic structures are built in terms of lower-level ones, as shown on the left-hand-side of Figure 2.

![Figure 2. PCE in PHP/XVCL mixed-strategy representation](image)

Each of the generic structures, building blocks of a mixed-strategy representation, we call an x-frame, and the overall mixed-strategy representation – an x-framework. At the bottom level (L4), we see x-frames, building blocks for Actions. An example is an editable form x-frame parameterized for reuse in different Actions, for different Modules. Generic Actions Add, Delete, Update and Display are defined above (L3), in terms of x-frames below. An arrow between two x-frames: X → Y is read as “X adapts Y”, meaning that X controls adaptation of Y. Custom Actions for specific PCE Modules (e.g., Add for Staff) are obtained by adapting respective generic Actions (e.g., Add.gen). Generic PCE Module (L2) is defined in terms of generic Actions below. The top-most specification x-frame, called SPC, specifies how to adapt x-frames below to produce Staff, Task and other Modules from the generic PCE-Module.gen. For example, customization for Staff Module are specified in Staff.s.

The XVCL Processor interprets an x-framework starting from the SPC, traverses x-frames below, adapting visited x-frames and emitting PHP code for PCE Actions and Modules. By varying specifications, we can instantiate the same x-framework in different ways, deriving different, but similar, programs from it. In that sense, an x-framework forms a generic design structure that enables reuse within a single program, or across programs. In the latter case, an x-framework implements a concept of a Product Line architecture [8][9].

For completeness, we highlight main XVCL mechanisms (Figure 3). XVCL variables and expressions provide a basic parameterization method to make x-frames generic. A <set> command assigns a list of values to an XVCL variable. XVCL variables module (set in SPC) and action (set in PCE-Module.gen) are generic names for PCE Modules and Actions, respectively. XVCL variables have global scope, so that they can coordinate chains of all the customizations related to the same source of variation or change, that spans across multiple x-frames. During processing of x-frames, values of variables propagate from an x-frame where the value of a variable is set, down to lower-level x-frames. Thanks to this scoping rule, x-frames become generic and adaptable, with potential for reuse in many contexts.

In each iteration of <while> command in SPC, we generate a PCE Module, in the order of Module names listed in the respective <set> command. X-frame PCE-Module.gen defines a common structure for PCE Modules. Unique customization required for specific Modules are specified under suitable option of <select> command, as indicated by the current value of variable module. We see a similar solution in x-frame PCE-Module.gen to specify unique customizations required for specific Actions.

![Figure 3. Highlights of XVCL mechanisms](image)

Command <insert> into <break> has a similar effect to weaving aspect code in AOP [20]. Certain functions specific to Module Staff are defined in x-frame For_Staff Only.x. With <insert For_Staff> in SPC, we include Staff-specific functions at designated variation points <break For_Staff> in x-frames PCE-Module.gen and Add.gen, as shown by dashed arrows. This example illustrates how we deal with ad hoc variations related to specific Modules, without affecting other Modules that should not be affected by these variations. Mechanisms for such selective injection of changes allow us to separate variants from common, generic structures (e.g., PCE-Module.gen), keeping generic structures reusable and easily adaptable.

PHP/XVCL representation contains both PHP code needed to produce all the PCE Modules, also information helpful in maintenance/reuse, such as a record of similarities and differences among Actions or Modules. The size of the PHP/XVCL mixed-strategy representation was 60% smaller (in terms of lines of code, without blanks or comments) and conceptually simpler than its PHP counterpart.
4. Summary of project experiences

4.1 Pilot CAD Project

Internet-enabled Computer Aided Dispatch Systems (CAD for short) was our first pilot project. Figure 4 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.

![Figure 4. An operational scenario in CAD system for Police](image)

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 [8][9] 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 presented significant research challenges. A lot is already known about construction of highly reliable internet backend solutions (eBay, for example, has a backend consisting of several hundred of Windows servers and close to a ten of Unix database servers), and there is a rich theoretical base for reliable distributed systems. However, an internet-based mission critical system such as CAD presented a new set of research problems. Meeting its real-time and reliability requirements in a manner that would allow high degree of software reuse multiplied the research challenges of the project.

Given the above, 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 [30][31].

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-strategy approach with XVCL could help us overcome some of those limits.

4.2 Pilot Command-and-Control (C2) Project

In this project, we ported a C2 application written in Visual Basic to C#.NET and also applied the XVCL technique to exploit similarity patterns. This project was implemented by an STEE software developer that had no earlier exposure to XVCL or to the ported application. The developer was given some support by XVCL expert, but we were generally surprised by the ease with which the developer picked-up the XVCL approach.

The C2 application involved many domain entities such as Task, User or Resource. For each entity, the application implemented up to 10 operations, such as Create, Delete, Update, Find or Save. For example, operation Create for domain entity Task involved components in all the boxes shown in Figure 5, collaborating through message passing. The same logical structure and the dynamic structure were repeated for each combination of a domain entity and
operation, resulting in significant similarities throughout the C2 application.

![Figure 5. Logical model of a domain entity](image)

The nature of variations across operations was such that neither inheritance nor C# generics could unify groups of classes in GUI, Service, Entity and Database layers. Therefore, implementation of operations for different entities was replicated many times, with required variations.

Through application of XVCL, these similarities were unified resulting in more than 68% reduction of overall code for implementation of the domain entities, and as much as 89% reduction in new code needed to add a new domain entity.

### 4.3 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 [16][3], 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 [16].

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 [3]. 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.

### 4.4 An industrial application of mixed-strategy

Web Portal (WP) Product Line was the first STEE’s application of XVCL in a business product and on a wider scale. A Team Collaboration Portal (TCP), with basic requirements similar to PCE described in Section 3, was a starting point for this project. TCP was implemented in ASP. STEE 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 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.**
- **High productivity in building new portals from the ASP/XVCL solution.** Based on the ASP/XVCL solution, ST Electronics 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 [22].

5. Significance of findings

In mixed-strategy, we represent each of the important similarity patterns in a unique generic, but adaptable form, along with the information necessary to obtain its instances (i.e., specific program structures, variations of a generic form). Mixed-strategy offers the following benefits: Explication of repeated program structures reduces program complexity as perceived by developers. Mixed-strategy generic solutions exploit reuse opportunities that are often missed by conventional component-based design techniques. Therefore, mixed-strategy often extends the scope and rates of reuse achievable by means of conventional techniques. Finally, non-redundancy of a mixed-strategy representation cuts the risk of update anomalies which, along with smaller code base and reduced conceptual complexity, helps in maintenance. The approach can be applied to any program, independently of an application domain or a programming language.

Conventional component-based reuse is most effective when combined with architecture-centric, pattern-driven development which is now supported by the major platforms. Patterns lead to beneficial standardization of program solutions and are basic means to achieve reuse of common service components. At times IDEs support application of major patterns, or programmers use manual copy-paste-modify to apply yet other patterns. Our projects demonstrated that further improvements are possible by applying mixed-strategy on top of these modern development practices. In particular, we can enhance the visibility of pattern application and enhance reusability in application domain-specific areas.

Looking for deeper roots of the software similarity phenomenon, we observed that feature combinations were one of the major forces triggering spreading similar program structures across programs. In each of our studies, features meant different program characteristics, though. For example, in the Buffer library features meant data type, memory allocation scheme, access mode and byte ordering; in the STL – data types, ordering, key type and uniqueness of elements in a container; in command and control system and Web Portals – domain entities and actions that applied to those entities. Program structures (such as components, classes or functions/methods) implement the net effect of the required feature combination that is necessary for correct execution. As we combine features, variations arise that force us either to find a unifying generic solution, or to create multiple program structures in variant forms. In industrial software Product Line projects, feature combinations may lead thousands of similar component versions [12].

Ad hoc, irregular nature of variations among similar program structures makes it difficult to unify the differences among similar program structures with conventional generic design techniques. This difficulty is further magnified by the fact that programs had to meet other important design goals, and conventional generic design solutions, even if existed, would require designers to compromise these goals, therefore were not used.

6. Mixed-strategy trade-offs

We discussed difficulties to achieve genericity with conventional methods. While mixed-strategy makes generic design easier, 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. Mixed-strategy targets at long-lived programs that undergo extensive evolutionary changes, or need be tailored to needs of multiple customers.

Industrial applications have demonstrated that the XVCL technique is easy and fast to learn, and the benefit of mixed-strategy may outweigh the cost of the added complexity [22]. At the same time, the return on investment may be quick and substantial. Industrial applications have also revealed a number of further problems.

A mixed-strategy solution is expressed at two intermixed levels, in base programming language(s) and XVCL. This creates extra difficulties, especially for debugging. However, we must keep in mind that a mixed-strategy solution 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-specific tools to help in development of mixed-strategy solutions. We are working on the XVCL Workbench that helps in editing, visualizing, debugging and static analysis of XVCL code (some highlights given in Section 7).

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 mixed-strategy representations are yet to be discovered.

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 must yet to learn what it takes to inject mixed-strategy methods 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.

7. Other spin-off projects

Effective project application of XVCL requires tool support. At the NUS Software Engineering Lab, we are implementing an XVCL Workbench that includes:

- Smart Graphical Editor (SGE) – supports developers in creating, maintaining and reusing x-frames. SGE hides the XML syntax of XVCL commands, and allows a developer to browse and query an x-framework.
- Debugger – allows a developer to trace the XVCL processing, and examine intermediate processing stages.
- Backward Propagation Tool (BPT) – allows a developer to propagate code fixes from the generated source code back to an x-framework, in a semi-automatic way.
- XVCL Metrics Tool (XMT) – computes XVCL metrics.

Yet another complementary branch of research that spawned from our collaboration involves identifying design-level similarity patterns in legacy software [2].

8. Conclusions

We described an eight-year long industry-university collaboration that led to formulation and industrial application of mixed-strategy approach to supporting Product Lines. We demonstrated that mixed-strategy often extends the scope and rates of reuse achievable by means of conventional techniques. It can also enhance the visibility of pattern application and reusability in application domain-specific areas when used together with modern component platforms. Our project provided insights into software similarities, and methods to reduce software complexity by unifying similarity patterns with generic structures. We believe mixed-strategy approach provides a practical solution to the problem.

We collaborate in a fairly informal, problem-oriented way, focusing on specific projects around XVCL. We make very little long-term plans for our collaborative work. We let the results so far and the current need drive the selection of projects we embark 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 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.

In the course of collaboration, STEE advanced its reuse practice by applying XVCL in several software product line projects. NUS team received early feedback from STEE 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.

Adopting a new technique always brings some overheads and XVCL is no different. It is essential to understand and evaluate trade-offs involved. Our current and future work focuses on empirical studies in various application domains and interpretation of the results, comparative studies of XVCL and other similar methods, tool support for applying XVCL, and methodological aspects of the mixed-strategy approach involving synergistic application of conventional techniques and XVCL.

9. Acknowledgments

We thank STEE 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

Eliminating Redundancies with Engineering Components for Mobile Devices


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

Bosch, J. “Design and Use of Software Architectures – Adapting and evolving a product-line approach,” Addison-Wesley, 2000


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


Jarzabek, S., Ru, S., Zhang, H. and Sun, Z. “Analysis of meta-programs: a case study,” Proc. 16th Int. Conference on Software Engineering and Knowledge Engineering (SEKE’04), Banff, Canada, June 2004, pp. 68-73; selected one of the best papers for a special issue of Journal of Software and Systems


PHP: http://www.php.net/


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)


