Chapter 13  Mixed-strategy approach in other projects

Summary of this Chapter:

We summarize experiences from other projects in which we applied mixed-strategy. We observed similar benefits in terms of maintainability and reusability as in the Buffer library experiment described in Chapter 8 and 9, and FRS. Also, the structure of x-frameworks were similar in all the project. XVCL addresses issues changeability, evolution and reuse by applying the same underlying principles of genericity and adaptability, with relatively simple mechanisms. Mixed-strategy focuses on unifying software similarity patterns that cannot be unified with conventional techniques alone. In Chapter 9, we hinted at common reasons why similarity patterns arise, and common problems that make it difficult to come up with generic solutions with conventional programming techniques. As the mixed-strategy approach deals with similarities at the root level, the uniform structure of solutions across very different program domains should not be very surprising.

13.1 STL in C++/XVCL

We discussed similarity patterns in STL in Chapter 9.

Having identified clones in STL with automated clone detector CCFinder [8], we analyzed manually parts of STL affected by clones. We analyzed the reasons why similar functions and templates occurred, and then we designed x-frames for each group of similar templates/functions, in a similar way as we designed x-frames for groups of similar classes in the Buffer library. Therefore, a mixed-strategy solution in this case was formed by imposing XVCL on C++ templates/functions that displayed much similarity, but also non-parametric differences that could not be unified by the template mechanism.

In STL, iterators proved very effective in separating algorithmic details from the data structures on which they operated. We did not find significant clones in algorithms. However, containers displayed much similarity in structure and much repeated code. Four ‘sorted’ associative containers and four ‘hashed’ associative containers could be unified into 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 C++/XVCL set operation, eliminating 52% of code.

The feature combinatorics problem is a challenge for many class libraries: As we combine features, classes grow in number and become polluted by redundancies. At times, even the sheer size of a library may become an issue. This would be certainly the case of the STL, and templates helped designers control the situation. Still, as our experiment showed, there was room for improvement.

13.2 DEMS in C#/XVCL

We discussed similarity patterns in DEMS in Chapter 9.
We unified similarities in DEMS with XVCL at two levels: At the detailed level, similar classes in GUI, Service and Entity tiers were unified with suitable x-frames. Then, patterns of collaborating components (such as shown in Figure 17 in Section 9.7.1) were unified with higher-level x-frames. For example, all the operations Create for different entities could be generated from a single x-frame Create[entity-type]. Similar classes participating in operation implementation could also be unified by suitable x-frames.

**Figure 1. Hierarchical unification of similarities in DEMS**

Figure 1 shows an outline of C#/XVCL x-framework for DEMS. Each group formed by the same operation applied to different entities, (e.g., CreateUser, CreateTask, …) has been represented by one generic operation parameterized by domain entities (e.g., Create.gen). Similarities among different operations for the same entity (e.g., CreateUser, UpdateUser, …) have been unified with bottom level x-frames representing generic classes, building blocks for DEMS operations. This is indicated by x-frames referenced from more than one operation (e.g., generic classes labeled with CU are reused in construction of Create and Update for various entities). X-frame Entity.gen defines the common, adaptable structure for DEMS modules. This includes DEMS architecture, that is, the organization of component patterns implementing various operations, plus any other functions supported by DEMS, not discussed in this example. The SPC contains global controls and parameter settings for the overall process of generating DEMS modules for various domain entities and operations acting on them.

This simplified description does not address classes belonging to the four system layers, namely user interface, business logic, database access, and database structure definition.

Figure 2 zooms into some details of a DEMS x-framework.
XVCL variables `oper` (set in SPC) and `entity` (set in x-frame `Entity.gen`) are generic names for DEMS operations and entities, respectively. In each iteration of a `<while>` loop in the SPC, we instantiate patterns of components implementing operations Create, Update, View, and others. The i'th iteration of the loop uses the i’th value of its control variable `oper` listed in the respective `<set>` command. Unique customizations required for specific operations are specified under a suitable option of the `<select>` command. We see a similar solution in the x-frame `Entity.gen` to specify unique customizations required for specific entities.

By varying specifications, we can instantiate the same x-framework in different ways, deriving different 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 [3].

Functions specific to the operation Create are defined in the x-frame `For_Create Only.x`. With `<insert For_Create>` in the SPC, we insert Create-specific functions at designated variation points `<break For_Create>` in x-frames `Entity.gen` and `Create.gen`, as shown by dashed arrows. This example illustrates how we deal with ad hoc variations related to specific operations, without affecting other operations that should not be affected by these variations. Mechanisms for such selective injection of changes allow us to separate variants from common, generic structures, keeping generic structures reusable and easily adaptable.

The C#/XVCL mixed-strategy representation contains C# code needed to produce all the DEMS operations, and also information helpful in maintenance/reuse, such as the record of similarities and differences among operations for different entities. In most programs, their
system structures such as in Figure 1 and Figure 2, are implicit, and must be expressed in the form of an external documentation. Mixed-strategy makes such views explicit and an integral part of a formal program representation. Mixed-strategy contains both code and the knowledge of program design, integrated into one representation.

The C#/XVCL solution compressed the size of the original C# entity management subsystem by 68%, which also led to improved maintainability. For example, it took eight times faster to create a new domain entity in the C#/XVCL solution as compared to C# solution alone. Typical enhancements of the subsystem (such as change of business rules for domain entities or adding/deleting a new entity or operation) required less modifications with smaller impact in the C#/XVCL solution as compared to its C# counterpart. For example, to add a new domain entity in the C#/XVCL solution required writing 133 LOC and took 2 man-hours, while the same enhancement in C# required writing 1440 LOC and took 16 man-hours. Finally, it was quite easy to reuse the C#/XVCL solution in other similar command and control systems.

The reader should notice that adding more operations and entities to DEMS in C# increases the complexity by a constant value, independently of the similarity of new operations/entities to existing ones. In mixed-strategy C#/XVCL, complexity grows proportionally to the level of novelty that a new operation/entity brings. New operations/entities that differ little from existing ones require little new code to be written.

13.3 Web Portal product line in ASP/XVCL

STEE (ST Electronics Pte Ltd) applied mixed-strategy approach to evolve a simple ASP Web Portal (WP) developed for personal use into a family of WP business products [9], so-called a software product line [3]. While sharing a common core, implemented as an x-framework, the individual products differed one from each other in many requirements. From the ASP/XVCL x-framework, STEE produced over 20 WP product variants that now serve customers from a variety of business sectors such as hospitals or e-learning. This has been the largest in scale so far application of mixed-strategy approach in industrial setting. Below, we outline the main story line of this project and discuss the benefits.

It started with a personal ASP WP developed by Ulf Pettersson of STTE. The personal portal facilitated information sharing via management of users, HTML-content, images and video-clips. It also included access statistics, news/announcements, weather animations, and posting/feedback facilities. The personal portal was then extended in Office environment to form a Team Collaboration Portal (TCP) shown in Figure 3. TCP supported a wide range of functionalities and scenarios commonly useful in collaborative environments. For example, TCP had a Staff Module to manage information about staff working on projects, a Project Module to manage information about on projects, and a Task Module so that project tasks could be assigned to staff. It also supported means for staff to provide feedback about the progress of a project, and monitoring facilities for managers to monitor the progress and identify possible sources of problems. Finally, TCP included a range of generally useful features such as News, Forum, Statistic, and Polls.

Initially used in one division of STEE, a number of TCP variants were then developed and released to other STEE divisions.
During the SARS crisis in May 2003, STEE developed a WP to facilitate analysis of people movements in hospitals. This WP was to register people movement within a hospital to facilitate tracing of people who could have been in contact with SARS infected persons. Under great time pressure (5 days), two persons transformed the TCP into a People Tracking Portal and integrated this portal with an external Radio Frequency Identification (RFID) tracking system at two hospitals. Some key Entity Modules in the People Tracking Portal were: zone, visitor, visitor movement (including movement/contact analysis). With this deployment, the portal actually turned into a business product (even though from a business perspective this was more of a community service).

![Figure 3. Team Collaboration Portal (TCP)](image)

Several other WP variants were later developed and released to various customers. At this stage, maintenance of existing WP products was getting very difficult due to the many product variants. Even though development of new portal modules could be done with reasonable ease (typically 500-1500 code lines each), constantly growing need for enhancements to existing portals and development of new portals resulted in increasing difficulty to maintain the WP family.

Although STEE applied state-of-the-art model-based design and component-based reuse techniques and a single team was maintaining most of the WPs, it was not possible to maintain them from a single code base for the following reasons:

1. 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.

2. Most of the enhancements (new WP features) were only needed in some of the WPs, while other already released WPs were not allowed to be affected by these enhancements. However, such selective injection of features (one of the challenges of reuse-based evolution discussed in Chapter 1) was impossible to achieve using conventional...
techniques: Should such enhancements be implemented into reusable components, they would be propagated to all the WPs, which was not a viable solution. The portals were deployed in a turnkey project manner rather than a product manner. As a result, some portals were in a frozen state where only critical defects would be corrected, and feature enhancements were thus not wanted. Therefore, new enhancements had to be implemented into those WP releases that needed them, resulting in increasing cost of maintenance.

3. Even though some entity modules were required in multiple portals, there were specific variations that applied to each individual portal resulting in difficulties to maintain a single code base.

A new business opportunity occurred, and the team was given seven days to develop an eLearning demonstration portal that required about 20 new modules. With the existing design and code volume required for each portal module, it was deemed impossible to achieve both design and implementation within a week.

At this point, mixed-strategy was adopted to take under the control common core of WPs to facilitate efficient evolution of existing portals and creation of new ones. Figure 4 shows the main steps in evolution of WP products. The numbers attached to transition arrows show evolution steps. Mixed-strategy reuse-based evolution occurs on the right-hand-side of the diagram. In Step 4, a common core of the existing WPs was identified and re-designed into an x-framework. Any further maintenance of existing portals and development of new ones was done by customizing the ASP/XVCL x-framework. As new WPs were developed, the x-framework was continuously refined so that other portals could benefit from new features.

Figure 4. Evolution from personal WP to 20 WP business products

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 [9].

**13.4 CAP-WP in J2EE/XVCL**

The J2EE/XVCL mixed-strategy solution unified both inter- and intra-module similarity patterns in CAP-WP discussed in Chapter 9. This solution represented each similarity pattern with a unique generic mixed-strategy structure, as shown in Figure 5.

![Figure 5. J2EE/XVCL CAP-WP x-framework](image)

Not only did such unification reduce the solution size by 61%, but most importantly it increased the clarity of portal’s conceptual structure as perceived by developers. In particular, it reduced the number of conceptual elements a programmer had to deal with and enhanced the visibility of relationships among program elements that mattered during changes: Rather than maintaining multiple variant code structures delocalized across the CAP-WP, in the mixed-strategy J2EE/XVCL solution a programmer dealt with one generic structure, with full visibility of customizations required to produce instances of variant structures, as well as their exact locations. Non-redundancy of the J2EE/XVCL CAP-WP, achieved by generic mixed-strategy structures unifying similarity patterns, reduced the risk of update anomalies.

**13.5 Role-Playing Games (RPG) for mobile phones**

Mobile games have become an important trend in the mobile phone industry. In Role-Playing Games (RPG), the players take the roles of fictional characters and participate in an interactive story. Figure 6 shows four RPGs called DigGem, Hunt< Jump and Feeding.
At some abstraction level, all RPGs share similar conceptual model. However, RPGs also differ one from each other in certain functional requirements. RPGs are further differentiated by the properties of a specific mobile device platform on which they run, and which affect RPG’s design and implementation. This includes mobile devices with high resolution colorful displays and up to 80M memory, low-end devices with smaller mono displays and less than 100kb memory, the new J2ME MIDP2.0 compliant mobile devices, and MIDP1.0 enabled older ones.

The challenge is that RPGs must perform well across all these different devices. A simple-minded solution is to ignore similarities among RPGS and to develop a separate product for each combination of an RPG and a target mobile device. However, such a solution is costly to develop and costly to maintain.

This project involved RPGs on the J2ME platform. RPGs were re-designed into an J2ME/XVCL x-framework that defined a common core for a range of RPGs and mobile devices. Not only did mixed-strategy approach achieve development/maintenance productivity gains, but also enabled sharing common successful optimization strategies across a range of RPGs. This considerably enhanced performance of all the RPGs.

The transition from ad hoc evolution of RPGs to mixed-strategy reuse-based evolution involved steps shown in Figure 7.
Figure 7. Evolution of RPGs with mixed-strategy approach

Having completed analysis of RPG domain (Step 1), a first-cut J2ME/XVCL x-framework was built based on one selected RPG, namely DigGem (Step 2). The first-cut x-framework was then refined with features of other three RPGs as well as with optimization strategies found in all four RPGs (Step 3). For this point, any enhancements of the four RPGs were done via the J2ME/XVCL x-framework. New RPGs, such as Kung Fu, were developed by customization and extensions of the J2ME/XVCL x-framework (Figure 8).

Figure 8. Deriving new releases of RPGs from the J2ME/XVCL x-framework

The following benefits were observed in this project:

1. Improved productivity in building new RPGs: reduced effort to develop a new game from 88 man-days (in J2ME) to 28 man-days (in mixed-strategy J2ME/XVCL).
2. Enhanced conceptual clarity, uniformity and integrity of the design of RPGs.
3. Improved performance, as all the RPGs shared best design solutions and optimizations. For example, the size of RPGs was reduced by 26% on average.
4. Improved productivity in maintenance as much maintenance could be done via J2ME/XVCL x-framework using XVCL’s capability to selectively propagate new features into RPGs that needed them, without affecting other RPGs that did not need new features.

13.6 Conclusions of this Chapter

We described the structure of solutions in a range of projects with mixed-strategy approach. We evaluated the engineering qualities of presented solutions. Here are some preliminary observations and evaluation:

The structure of a mixed-strategy solution follows a similar pattern: Similar program structures are unified with generic structures built with XVCL. Unification occurs at multiple levels, with lower-level generic structures serving as building blocks for higher-level ones.

The main strength of the mixed-strategy approach is its ability to define powerful generic design solutions. Because of its low level mechanisms to deal with changes, XVCL is capable of unifying any kind of similar structures, of any granularity and independently of the nature of differences among them.

We believe mixed-strategy program representation offers some interesting engineering benefits. In particular, it (1) reduces the code size, yet contains 100% of information necessary to produce the original buffer classes, (2) defines program solution using less number of conceptual elements than the number of conceptual elements in the actual program, (3) represents design information that matters during maintenance in a form that is understandable for programmers and amenable for automation by XVCL Processor, and (4) enhances the conceptual integrity of the design, which Brooks calls “the most important consideration in system design” [2].

In this Chapter, we substantiated the above observations with empirical and analytical arguments. The idea behind the mixed-strategy approach is general, can be applied to any program that exhibits much similarity, independently of an application domain or a programming language.

However, applying and adopting techniques like XVCL does not come for free. Adopting a new technique always brings some overheads and XVCL is no different. It is essential to understand and evaluate trade-offs involved. We conduct comprehensive evaluation of strengths and weaknesses of the approach in the next Chapter.

REFERENCES


