ABSTRACT

Web Applications (WA) are developed and maintained under tight schedules. Much similarity across WAs creates opportunities for cutting development cost and easing evolution via reuse. This tutorial shows a practical way to exploit similarity patterns – at architecture and code levels - to simplify the design of WAs, helping to meet the unique challenges of Web engineering.

Categories and Subject Descriptors

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

General Terms

Design, Languages, Experimentation

Keywords

Web engineering, reuse, software product lines, maintenance, generative programming

Web Applications (WA) are developed and maintained under tight schedules. After delivery, they are often changed. Much similarity across WAs creates opportunities for cutting development cost and easing evolution via reuse. This tutorial shows a practical way to exploit similarity patterns – at architecture and code levels - to simplify the design of WAs, helping to meet the unique challenges of Web engineering.

The approach is based on designing structures to represent similarity patterns in generic way. Simplification of the WA design is then achieved by providing a mechanism to reuse such generic structures by instantiating them in variant forms, as required in various reuse contexts. We apply so-called “mixed strategy” approach: Initial design is done using conventional methods (architectural/component design, component reuse, with Web technologies such as ASP, JSP, PHP or J2EE). Some level of reuse can be realized with these conventional methods (e.g., reuse of common services via APIs and patterns of J2EE or .NET). Then, we apply generative technique of XVCL [8] to build generic structures to unify similarity patterns for which conventional techniques fail to provide effective generic solutions. By applying such “mixed strategy” approach, we turn WAs built with conventional approaches, into a generic WA solution that offers substantial productivity gains in WA development and maintenance. In particular, a generic WA forms a WA product line architecture from which we can rapidly develop new WAs and maintain the whole family of WAs in reuse-based, cost-effective way. On average, we reduce conceptual complexity (and maintenance effort) of a program solution by 60%, raising the levels of reuse by similar rates.

The core of the tutorial is based on industrial project experiences, case studies zooming into details of technical solutions, and then explaining the approach at the concept level. Live demonstrations and hands-on examples are used to complement conventional presentation techniques to facilitate active learning. We base the tutorial on projects by ST Electronics Pte Ltd who applied the “mixed strategy” approach to build ASP and J2EE Web Portals, and on studies that explain technical details. We introduce the concepts of “mixed strategy” approach in a sequence of easy to follow examples. We present the conceptual underpins on the ground of already established firm understanding of issues at the experiential level.

Our empirical studies have shown 50%-90% rates of repetitions that deliberately recurred in newly developed, well-designed programs. Our studies of similarity patterns covered a range of application domains (business systems, Web Portals, command and control, class libraries), programming languages (Java, C++, C#, ASP, JSP, PHP) and platforms (J2EE, .NET, Unix, Windows). Similarity patterns ranged from similar class methods, to similar classes and to patterns of collaborating classes/components representing relatively large parts of a program. models [3]. For example, the extent of similarities in Java Buffer library was 68% [4], in parts of STL, a well-known example of powerful generic solution - over 50% [2], in Web Portal (J2EE) – 68% [9], and in certain ASP Web Portal modules – up to 90% [6] (project by ST Electronics). A survey of 17 Web Applications revealed 17-60% of code contained in clones [7].

We measured the percentage of redundancies by comparing the subject program against the XVCL-enabled “mixed strategy” solution from which the subject program could be obtained. In
our studies, we paid attention only to repetitions of significant engineering importance, meaning that they created reuse opportunities, induced extra conceptual complexity into a program, and/or were counter-productive for maintenance. Avoiding them with conventional approaches was either impossible or would require developers to compromise other important design goals. However, we could effectively treat all the similarity by unifying them with generic structures built with XVCL.

Modern platforms (such as .NET™ or J2EE™, and web frameworks such as Ruby on Rails™) encourage organizing software around standard architectures which allows programmers to reuse common services/components. Pattern-driven development style even further standardizes software development by using frameworks such as Ruby on Rails™ which encourages organizing software around standard architectures that allows frameworks such as Ruby on Rails™ to encourage software reuse.

Software developed in that way displays much similarity. For example, we found 61% of code contained in similar program structures replicated many times in variant forms [9]. Despite benefits during development, pattern-driven design may add complexity to future maintenance. It may also complicate reuse of the application domain-specific functionality. This is due to the following reasons: (1) patterns remain implicit in code - we may not know the exact location of pattern instances in a program, and how pattern instances are similar and different from each other, (2) when the pattern-related code is to be changed, it is not clear which of pattern’s instances should be changed and how, and (3) application of patterns scatters code related to application-level functionality across many components (classes), which magnifies well-known problems of tracing requirements to code, and the impact of change, in general.

One important concept and construct is missing to fully exploit the benefits of design standardization. The missing concept is a strong enough mechanism for generic design that would allow us to represent and maintain similar program structures spawning from patterns in a generic, customizable form. The “mixed strategy” approach presented in the tutorial is based on a mechanism for generic design that plays the above role.

In the ASP Web Portal (WP) Product Line project ST Electronics applied state-of-the-art design methods to maximize reusability of a Team Collaboration Portal (TCP) in other contexts. Still, a number of problem areas were observed that could be improved by applying XVCL 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 interdependent features supported by the ASP/XVCL solution.

We have applied XVCL in three types of projects: (1) to unify similarity patterns with generic meta-level structures for ease of maintenance (in business applications, Web Portals and class libraries), (2) to design generic architectures for reuse via product line approach (command-and-control systems and Web Portals in ASP and J2EE), and (3) to manage variants in domain models [5].

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

References


