An Approach to the Software Product Line System for Web Applications

Makoto Yoshida
Department of Information & Computer Engineering
Okayama University of Science
Okayama, Japan
yoshida@ice.ous.ac.jp

Noriyuki Iwane
Department of Information Sciences
Hiroshima City University
Hiroshima, Japan
iwane@its.hiroshima-cu.ac.jp

Abstract

Quality, cost and delivery are the most crucial factors to be managed in developing software systems in the industry. The management of these software systems includes not only resolving the necessarily technical problems but also the organizational problems and the business problems. Companies depending upon their environments decide their own strategies to solve these problems. Software product line system is one of the solutions to these problems. This paper describes the way we experimented in these several years to develop the software systems at the company to meet some software development solutions. The efforts of web-based application developments by the source code generator we developed are evaluated based on COCOMO model. Furthermore, the software development model for software product line systems using the toolkit is examined. This paper describes an approach to the software product line systems for web application systems.

1. Introduction

Software quality, software development cost and software development speed are the requirements that must be always improved in computer companies. Each company puts their own strategies to solve these problems; which gradually adjust and change their software environments.

There are two main strategies to be adopted; software reuse and automatic software generation. Modular design for software is one of the methods to be adopted for the reuse of software, and it might lead to the software product line systems [1,2,3]. A framework for software product line systems is well described in the literature [1,4,5]. It defines the domain engineering and the application engineering, and the interaction between them produces the software product line systems. Another strategy is to develop the toolkit that automatically generates the source code [6]. This paper describes the approach to the software product line systems for web application systems using the automatic program generation toolkit. It describes the approaches we experimented to develop the software product line systems.

The rest of paper is organized as follows. Section 2 describes the software development process and the toolkit applied. The toolkit generates the source code of web application systems. The toolkit was applied to several projects and their experimental results are evaluated. Section 3 describes the software product line model and systems. Section 4 describes the lessons we learned from these projects and concludes with future works.

2. Software Development Process

2.1 Development Process

Traditional software development processes are; analysis, design, implementation, and testing [8,9]. The application program code produced can be categorized into three classes; the common class that is common to almost all applications (X%), the resemble class that looks common but has some differences (Y%), and the quite different class that depends completely on applications (Z%), as shown in Figure 1. Our aim is to reduce the software development cost without changing the existing development processes, the water fall model has to be preserved. Without any tools, only common libraries (X%) are reused by the programmer in the implementation process in his/her own risks to reduce the coding cost. Focusing on web application systems, we developed the toolkit that automatically generates the program source code to reduce the implementation cost as a first step [6]. The code in the resemble class (Y%) in Figure 1 were automatically generated by the toolkit. It can increase the software reuse Y% and can reduce the coding cost, as will be verified later.

1-4244-0220-4/06/$20.00©2006 IEEE

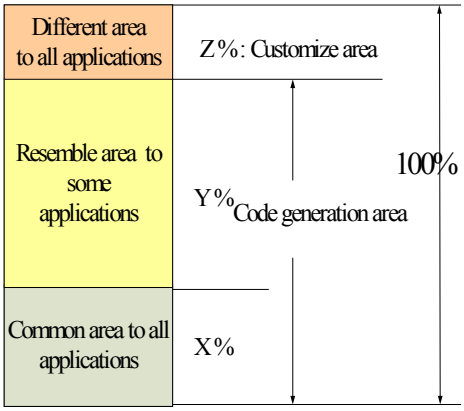

Figure 1. Application Code Classification

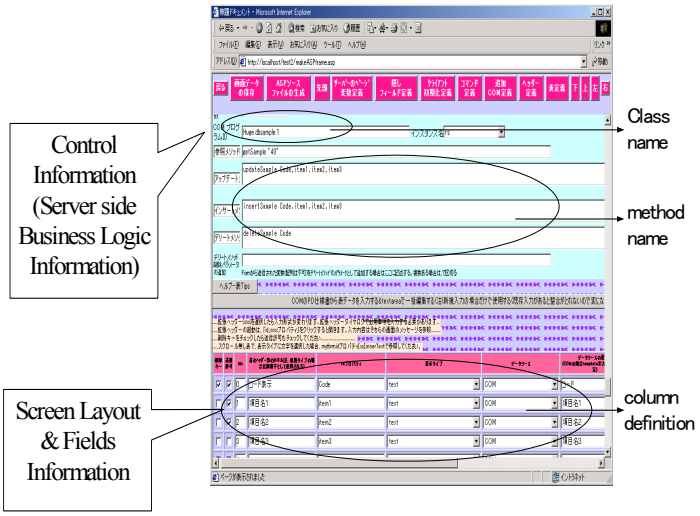

Figure 2. GUI Interface of the Toolkit

2.2 Automatic Program Generator

We developed the toolkit, which automatically generated java source code from program specifications [6]. Presentation code and logic code are automatically generated by using GUI we developed. Figure 2 shows the presentation interface of the toolkit. The screen layout information and some control information, class name and method name, that work on the server side and functions as logic specification of the web applications must be specified. The toolkit automatically picks up functions from specifications and extracts the corresponding design patterns automatically, which in turn call libraries [6]. The libraries are implemented by generic programming [10]. The resemble codes are generalized as design templates of the framework. The functions such as read/write XML files, input/output files, map_file and etc, are defined as general common component classes of the library. The screen layout pattern, the table structure pattern, the transaction control pattern and etc, are defined as templates. These form the design templates of the framework. The input data with the corresponding templates generate the parts of program code, and those parts are merged and adjusted and produce the complete java source codes [6].

2.3 Experimental Development

The toolkit was applied to a wide range of business application systems. The systems for Supply Chain Management System, Custom Relationship Management System, Goods Return System, and Sales Report System are developed using the toolkit. Table 1 shows the experimental results. Ten projects are examined, and the code generated by the toolkit are analyzed. The results in Table 1 show that the total average code generation ratio for web applications is 86 percent, the generation ratio for the presentation is 93.2 percent and for the logic 66.2 percent. It indicates that the logic part has much more variants than the presentation part. In another words, the presentation code have a lot of commonalities. However, three out of ten projects have shown the low code generation ratio. It indicates the strategy adopted to use the toolkit was the poor decision. The use of the toolkit for these applications should not be allowed. The code generation ratio normalized excluding the poor projects indicates that almost 93 percent code generation is possible to various applications. If it is applied to with a good decision, it must be very useful tool for software development. It can be verified by using the cost evaluation model of COCOMOII sizing equations [9].

Adapted code size can be translated into the equivalent new code size by using COCOMOII sizing equations. COCOMOll sizing equation is given in Appendix [9]. The COCOMOll reuse factors are evaluated as follows and the equation is calculated.

AA=0 (average assessment and adaptation effort)
SU=20 (poorly structured and documented)
DM=0 (% design modified)
CM= IM= Actual data obtained
(CM: % code modified, IM: % integration into new system required)

1-4244-0220-4/06/$20.00©2006 IEEE

The results of the evaluation are shown in Table 2. Table 2 shows the equivalent new code size calculated. The results show that the adapted code is equal to 39.2 percent of the new code on the average. It means that 60.8 percent cost reduction must be performed by the toolkit. Similarly, 69 percent cost reduction for the presentation code and 37.7 percent code reduction for the logic must be expected. The normalized data which excluded the poor projects indicates almost 70 percent cost reduction is possible to the general applications. If good decision were taken, it can dramatically reduce the implementation cost.

2.4. Toolkit Extension

The experimental results showed significant improvement compared to the traditional software processes in coding time and cost. In contrast to these advantages, it also has the disadvantages; some applications are not suitable to adopt the toolkit as

Table 1. Experimental Results – Source Code Generation Ratio –

	Source lines of code	Modified lines of code	Code_Generation_Ratio(%)	Reuse Decision
Pj 1-PR	13300	40	99.7	
Pj 2-PR	12617	1036	91.8	
Pj 3-PR	1650	180	89.1	
Pj 4-PR	11728	1313	88.8	
Pj 5-PR	7690	647	91.6	
Pj 1-LG	5800	88	98.5	
Pj 2-LG	3645	1793	50.8	☑ poor
Pj 3-LG	3782	2021	46.6	☑ poor
Pj 4-LG	1605	180	88.8	
Pj 5-LG	2040	1613	20.9	☑ poor
	Total_Source_Code	Souce_Code_Updated	Code_Generation_Ratio(%)	
PR-Average	7831	536	93.2	
LG-Average	2812	949	66.2	
T-Average	5321	743	86.0	

Pj 1: Supply Chain Management System
Pj2: Customer Relationship Management System
Pj3: Goods Return System
Pj4,5: Sales Report System

PR-Average: Average of Presentation Code
LG-Average: Average of Logic Code
T-Average: Average of Total Code

Table 2. Cost Evaluated by COCOMOll Sizing Equations

*IM=1	Adapted SLOC(A)	Equivalent Size Caluculated(B)	B/A (%)	Reuse Decision
Pj 1-PR	13300	2724	20.5	
Pj 2-PR	12617	4181	33.1	
Pj 3-PR	1650	618	37.5	
Pj 4-PR	11728	4446	37.9	
Pj 5-PR	7690	2573	33.5	
Pj 1-LG	5800	1301	22.4	
Pj 2-LG	3645	3598	98.7	☑
Pj 3-LG	3782	2021	53.4	☑
Pj 4-LG	1605	609	37.9	
Pj 5-LG	2040	2989	146.5	☑
	Adapted SLOC(A)	Equivalent Size Caluculated(B)	B/A (%)	reuse Decision
PR-Average	7830.8	2423.7	31.0	
LG-Average	2812.0	1753.0	62.3	☑
T-Average	5321.4	2088.3	39.2	

```
<Screen_Control_Information>                                   Method name
   <Screen_Basic_Information>
        <Screen_Number Param_type="String">000000</Screen_Number>
        <Screen_Title method_name="setPpp" Param_type="String">null</Screen_Title>
        <Screen_Size>
              <width dtype="int">0</width> <height dtype="int">0</height>
        </Screen_Size>
  </Screen_Basic_Information>
        <Screen_Type> ..............</Screen_Type>
        <Condition_For_NextScreen>.....</Condition_For_NextScreen>
        <Input_Information>                                    Pattern name
              .........
          <Mandatory Pattern_Id="MOD" MethodName="aaa0" Param_Type="Int"...
          <Selection Pattern_Id="MOD"  MethodName="aaa1" Param_Type="Inr"...
          <Option Pattern_Id="MOD"  MethodName="aaa2" Para,_Type="Int"...
        </Input_Informaton>
              .........
   <Screen_Extent_Information>.....</Screen_Extent_Information>
</Screen_Control_Information>
```

Figure 3. Example of XML Interface

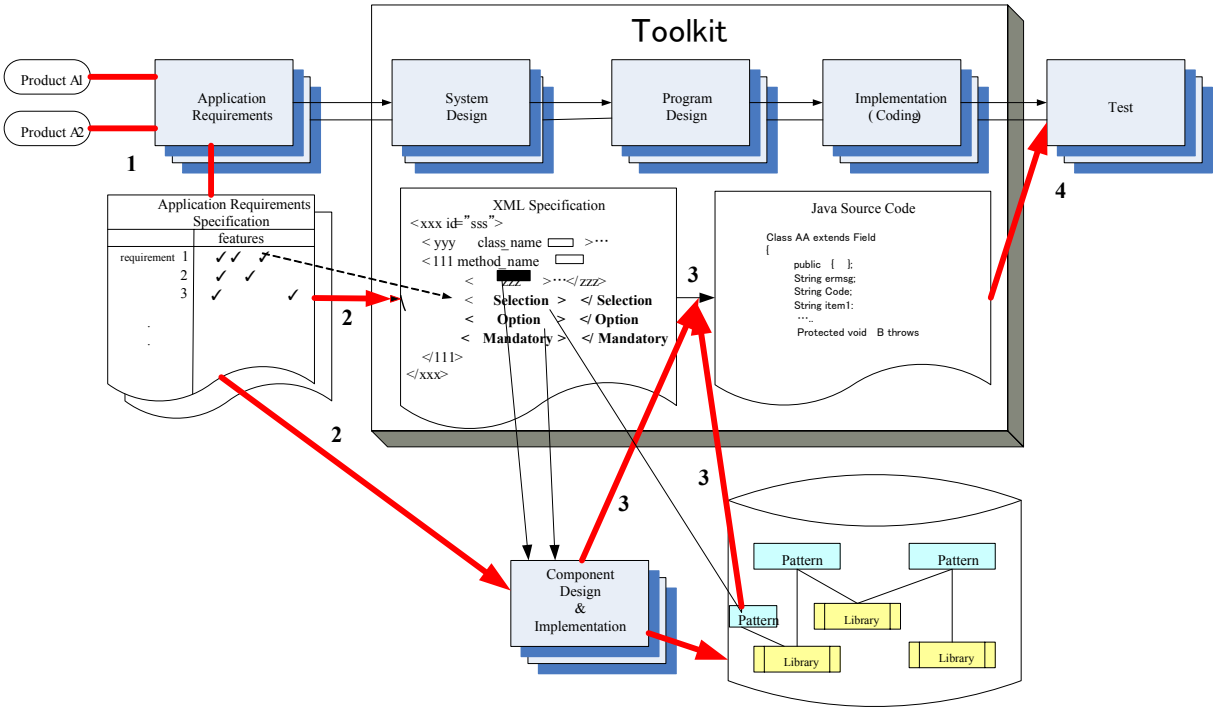

Figure 4. Product Line System Model

shown in Table 1. The projects checked in Table 1 are the projects that were not suitable to use the toolkit, and it showed the low code generation ratio. It is because of the lack of functions provided to the toolkit we developed. It does not have the flexible user interfaces enough to apply to the whole web applications.

We decided to provide more flexible user interface to the toolkit at the next step [7]. The GUI interface was changed to the XML interface without changing the toolkit framework. Class name, field name and method name was specified by using XML instead of using GUI of the toolkit. Several special tags for the toolkit were introduced; the tags for specifying the class names for certain design pattern, the tags for selecting method names, etc were introduced. Figure 3 shows an example of the XML specifications. In addition to changing the interface of the toolkit, another function that translates the user requirements to XML files was developed and integrated into the

toolkit (see Figure 4). The user requirements specified are also shown in Figure 4. The user requirement table which consists of features and requirements are filled out by the stakeholders; on the conversation of system engineers and users. The table is inputted to the toolkit, which automatically translates the requirement specification to the XML files and in turn translates the XML files to the java source code.

The toolkit was applied to the banking system. It could generate about 1,600 screens instantaneously. About 500 KLOC java code for the presentation was automatically generated. It could reduce not only the implementation cost but also the design cost.

When the family of applications has many commonalities and patterns, the approach we took can remove the design phase completely by using the requirement specification tables and the XML specification files as shown in Figure 4.

3. Software Product Line

3.1. Software Product Line Model

Software product line is the family of applications, in which it has many commonalities and some variability [4,5]. The basic software product line model is well described in the literature [1, 4, 5]. Even though most of the references describe the top down approach of the software product line systems, we believe only a few company adopt the top down approach to develop the software product line systems. Most company starts first taking the bottom up approach. This section describes the approach we experimented to the software product line systems.

3.2. Software Product Line Systems

Figure 4 describes our up and down system construction approaches. At the first time, we followed the traditional water fall model of software development processes as shown in the top of Figure 4. The templates of design patterns, analysis patterns and code patterns are gradually formed on accumulating the development know-how of the same kind of applications. When these patterns are integrated into the toolkit, it forms the framework of the application systems. Then, using the toolkit, the requirement specification can be automatically translated into the XML files by using the analysis patterns, and XML file can be translated into Java code by using design and coding patterns.

Thereafter, the style of the software development processes changes from bottom up style to the top down style. Figure 4 shows the top down software development flows of the software product line systems. Application requirement specifications are automatically translated into the XML files by the toolkit. If some new features, those are not previously defined in the requirement specification table, are required, the corresponding components must be designed and implemented, and must be merged when the XML files were translated by the toolkit. The sequences of the software development processes are depicted by bold lines with the sequence number in Figure 4. Encompassing the know-how of the application developments and the user requirements into the toolkit, the family of applications, the product line systems, can work efficiently. Several product line systems can be constructed from common analysis patterns and design patterns. Meta patterns, patterns of patterns, must be recognized in both analysis and design processes.

4. The Lessons Learned

The step-wise improvements of the software development we experienced are described. The following is the lessons we learned from the web-based software developments.

1. More than 86 percent of the java source code can be automatically generated from program design specifications by the toolkit in web based application systems. It makes possible to reduce the coding cost 70 percent if the toolkit were the good choice of decision to use.
2. The Logic part in web applications has much variance than the presentation part, so the decision for the adaptation of the toolkit for the logic part must be seriously managed. On the other hand, the presentation part has a lot of commonalities to be applied to most of the web application systems.
3. Patterns form templates. Program design patterns and coding patterns make templates, which constitute the automatic program generators. These patterns must be consistently combined into the toolkit to produce the product line systems.
4. There are three patterns in the software development processes; requirement patterns, design patterns and coding patterns. These patterns are the know-how of either the system engineers or program designers or coders. Only when all of these three patterns are analyzed and integrated consistently, they can construct the software product line systems effectively. The software product line systems can only be effectively constructed under these circumstances

5. We believe most of the product line systems start from the bottom up construction of the software components. It forms several similar patterns and in turn forms the templates and the frameworks gradually by accumulating the application know-how, experiencing several similar applications. If some forms of the templates and the frameworks are fixed up, the approach for developing the software systems changes from the bottom up approach to the top down approach. It is the way that the product line systems can be developed.

5. Conclusion

The paper described the experimental results of the web based application systems, in which the software product line systems are well organized. The requirement patterns, the design patterns and the coding patterns are combined into the toolkit consistently to generate the program source code automatically. The stepwise evaluations of the automatic program generator are introduced. And the approach to the software product line systems is described. The automatic program generator was extended to the software product line systems and applied to the project, which produced the effective cost reduction to the software development processes.

The next approach we are planning is to combine the toolkit with network based questionnaire systems to be able to capture the user requirements quickly. Capturing the user requirements quickly through the networks and automating the flow of the software development processes by the toolkit will provide the efficient and agile software development. This is our future research.

6. Acknowledgements

The work of this research was made possible when one of the authors was in OKI Software Co., Ltd, in Japan. Thanks to OKI Electric Industry Co., Ltd and OKI Software Co., Ltd.

7. References

[1] Klaus Pohl, et al, Software Product Line Engineering Foundations, Principles, and Techniques, Springer, 2005.

[2] Liliana Dobrica, Eila Niemela, A Survey on Software Architecture Analysis Methods, IEEE Trans. on Software Engineering, Vol.28, No.7, July 2002.

[3] Baldwin Carliss Y, Kim B.Clark, Design rules: The Power of Modularity, MIT press, 2000.

[4] William B.Frakes, Kyo Kang, Software Reuse Research: Status and Future, IEEE Trans. on Software Engineering, Vol.31, No.7, July 2005.

[5] Lenn Bass, Paul Clements, Rick Kazman, Software Architecture in Practice second edition, Addison-Wesley,2003.

[6] Makoto Yoshida, Mitsuhiro Sakamoto, Experimental Knowledge－Based Automatic Program Generator, Networks 2002: Joint International Conference: IEEE ICWHN 2002 and ICN 2002, Atlanta, USA, August 2002.

[7] Makoto Yoshida, Noriyuki Iwane, Knowledge-based Experimental Development of Web Application Systems, The 5th International Conference on Computer and Information Technology, Shanhai, China, September, 2005 .

[8] Robert B.Grady, Successful software process improvement, Prentice-Hall,1997.

[9] Barry W.Boehm, et al, Software Cost Estimation with COCOMOll, Prentice Hall, 2000.

[10] D.R.Musser, et al, Algorithm-Oriented Generic Libraries, HP Laboratories Technical Report, HPL-94-13, 1994.

Appendix COCOMOII Sizing Equations (See the detail the reference [9])

$$\text{Equivalent New code Size} = \text{Adapted code Size} \times \left[1 - \frac{\text{Percentage of Adapted Code Size}}{100}\right] \times \text{Adaptation Adjustment Modifier (AAM)}$$

$$\text{Adaptation Adjustment Modifier (AAM)} = \begin{cases} \dfrac{\text{Percentage of Assessment and Assimilation (AA)} + \text{Adaptation Adjustment Factor (AAF)} \times \left[1 + 0.02 \times \text{Percentage of Software Understanding (SU)} \times \text{Programmer Unfamiliarity with Software (UNFM)}\right]}{100} & \text{For AAF} \leqq 50 \\ \dfrac{\text{Percentage of Assessment and Assimilation (AA)} + \text{Adaptation Adjustment Factor (AAF)} \times \text{Percentage of Software Understanding (SU)} \times \text{Programmer Unfamiliarity with Software (UNFM)}}{100} & \text{For AAF>50} \end{cases}$$

$$\text{Adaptation Adjustment Factor (AAF)} = 0.4 \times \text{Percentage of Design Modified (DM)} + 0.3 \times \text{Percentage of Code Modified (CM)} + 0.3 \times \text{Percentage of Integration Required (IR)}$$