Knowledge-based Experimental Development of Web Application Systems

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 maintained in developing software systems in the industry. Recently, in addition to these factors, customer satisfaction is getting increasing attention. How to acquire customer's needs and reflect them quickly into the software products with least effort becomes important issues. The weight of costs for developing software systems is shifting from coding effort to analysis effort. This paper summarizes the step-wise developments of the web-based application systems that we had experienced in the last several years. Having developed the toolkit which automatically generates application programs from specifications, it was applied to the software product line systems. Furthermore it is integrated with the questionnaire systems. The shift behavior of our application development processes and the lessons we learned to produce the efficient customized software application systems while offering substantial savings in time and cost are described.

1. Introduction

Software products must be delivered to the customers with least time and cost in many enterprises [3]. Several methods for analyzing architectural requirements such as ATAM (Architecture Tradeoff Analysis), CBAM (Cost-Benefit Analysis Method), and ADD (Attribute-Driven Design) are proposed [1,2,3,4]. Previously, we introduced the toolkit that automatically generated the Java source code from system specifications. It could generate more than 80% of the java source code automatically [5]. The experimental results confirmed the dramatic cost reduction in software development in wide range of application domains. It showed the average of 43% cost reduction in the software development [6]. However, there still existed the domains that the toolkit was not applicable to. If these domains were recognized explicitly, it will increase the productivity dramatically. So as a second step, we extended the toolkit to the specific software product line systems [8]. The experimental results showed the 100% automatic code generation by the toolkit having restricted the domains. It could reduce the requirement specification cost in addition to the coding and testing cost [6]. Then as a third step, we tried to reduce the analysis cost by proposing the KITS (**K**nowledge based **I**ntegrated ques**T**ionnaire-**S**oftware product line) model [7, 8]. The questionnaire system that generates questionnaires and the product line system that generates source code are integrated. Knowledge is shared and reused through knowledge-based systems. Figure 1 shows the step-wise approaches of software development in the last several years.

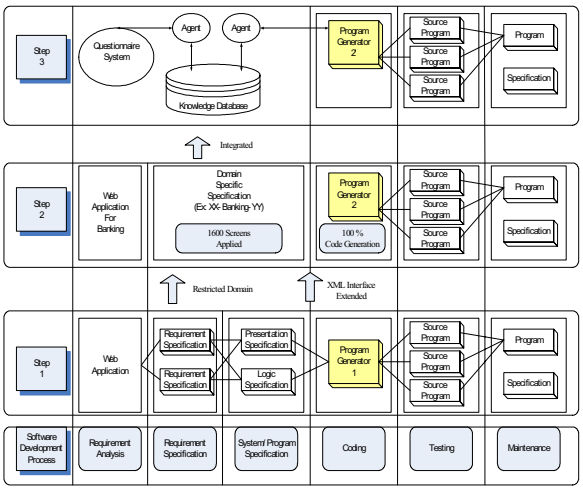


Figure 1. Step-wise Approaches

Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05)
0-7695-2432-X/05 $20.00 © 2005 IEEE

2. Experimental Development

2.1. Step1: Automatic Program Generator

The objective of the first approach was to reduce the coding cost without changing the software development processes which was based on the waterfall model. There has been proposed several approaches: package approach, component-ware approach and so on [10]. However, none of them satisfied the objectives. So, we decided to develop an automatic program generator [5,6]. It was applied to several application systems, and the experiments showed that the toolkit could generate more than 80% java source code automatically; it showed that on the average 90.1% JSP code for the presentation and 72.7% EJB code for the logic [5]. It means that 70% cost reduction in coding process and 43% cost reduction in the software development based on COCOMO model has been accomplished [6, 10, 11].

2.2. Step2: Software Product Line Systems

The first experimental result has shown the significant improvement on development cost as compared to the traditional software development. The next approach experimented was to raise the automatic program generation ratio by the toolkit up more, and to generate system specifications automatically from system requirements. The toolkit was extended to certain application domains and applied to the software product line systems [3].

Two major changes to the toolkit have been accomplished. The one was the toolkit interface, and the other was the toolkit components. Figure 2 compares the system architecture. Figure 2 (a) shows the system architecture of the first toolkit, and figure 2 (b) shows the system architecture of the second toolkit . Some specific code patterns for the software product line were added to the first toolkit, and the user interface was changed to the XML interface. The system architecture consists of three layers: class library layer, code pattern layer and application specific layer. Similar code patterns are generalized as templates and be framed into the code pattern layer. This layer is the core of the toolkit, and all developer's know-how resides in here. ODBC control, thread control, and I/O control are supported in the basic component layer. Transaction control patterns, database control patterns, screen layout patterns, etc., are supported in the code pattern layer.

Figure 3 shows two interfaces of the toolkits. Figure 3-a shows the GUI interface of the first toolkit, and figure 3-b show the CUI interface of the second toolkit. The interface was changed from general GUI interface to the specific CUI interface based on XML to make the system specification flexible. Though both interfaces necessitate class name and method name, the interface was extended to be able to specify several optional patterns in the second toolkit, as shown in figure 3-b. The second toolkit was applied to the specific banking system, and it could generate 100% java presentation codes in the web-based banking applications. It could generate almost 1600 screens automatically. It certified the adaptability of our toolkit to certain software product line systems. Figure 4 shows the software development processes of the second step. The system requirements prescribed on the function/data tables can be automatically translated to the XML files by the specification translator. Then the XML files are inputted to the toolkit, which in turn generates the java source codes automatically. This process greatly reduces the system specification cost in addition to the coding cost.

2.3. Step3: Integrated Software Product Line Systems

Figure 5 shows the KITS (**K**nowledge based **I**ntegrated ques**T**ionnaire-**S**oftware product line) model we proposed [8]. It consists of three sub-systems: the questionnaire sub-system, the software product line sub-system and the knowledge database sub-system. Metadata, that is in the form of function/data tables, are stored in the knowledge database system. As you can see in Figure 6, information stored in the knowledge database is not only system information, but also domain information and questionnaires information. The elements in the trees can be linked by the semantic relations like "same_as" relation, and this makes the independent defined elements as a unified tree and makes possible to access the related information through XML [9].

The KITS model makes the developers (or it may be the manager of the development team) easily capture the user requirements and the needs in advance. It also makes possible to develop the components of the product line system in advance before the system development starts. The more the questionnaire system reflects the user requirements, the closer the software product line system is to the customers. On the other hand, if the resulted analysis of the questionnaires were quite dissimilar to the system specifications, the software product line system must be fine-tuned and the toolkit must be changed to be able to satisfy the customer's needs.

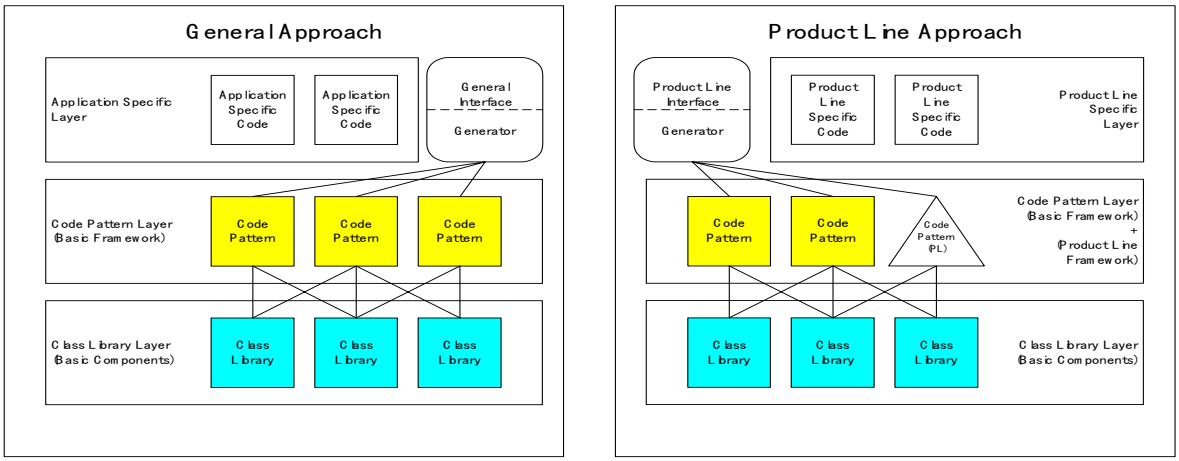
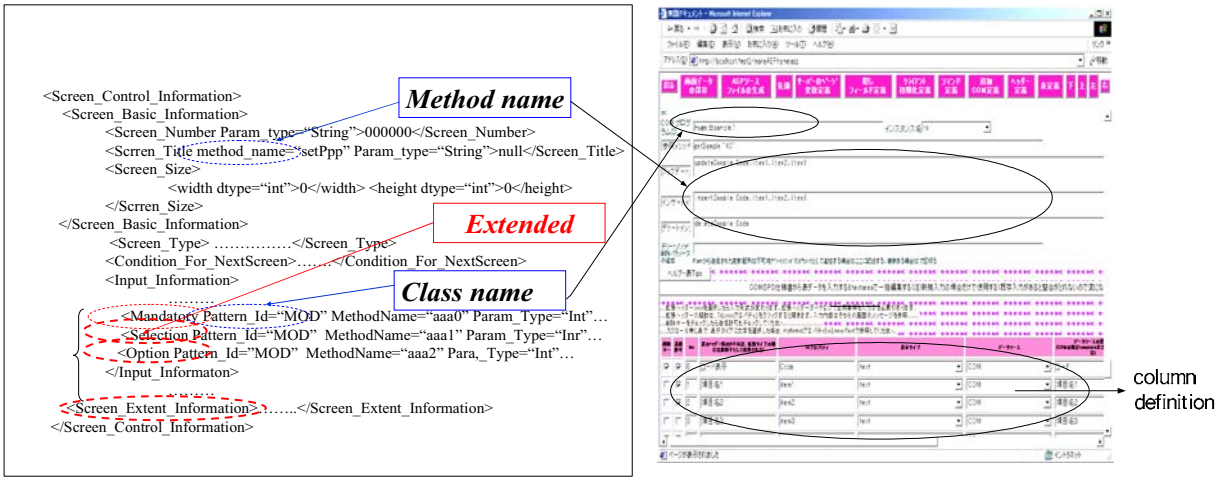


Figure 2. Architecture Compared



(b) (a)

Figure 3. User Interface

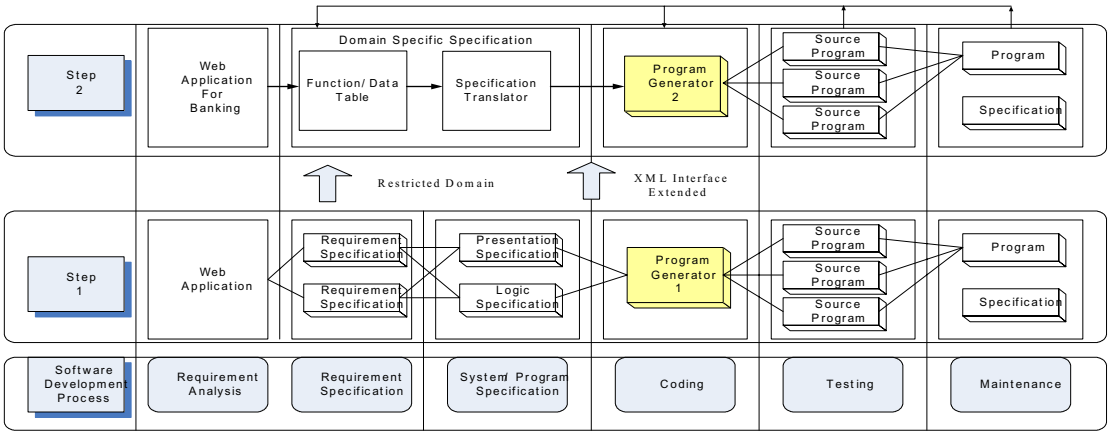


Figure 4. Software Development Process

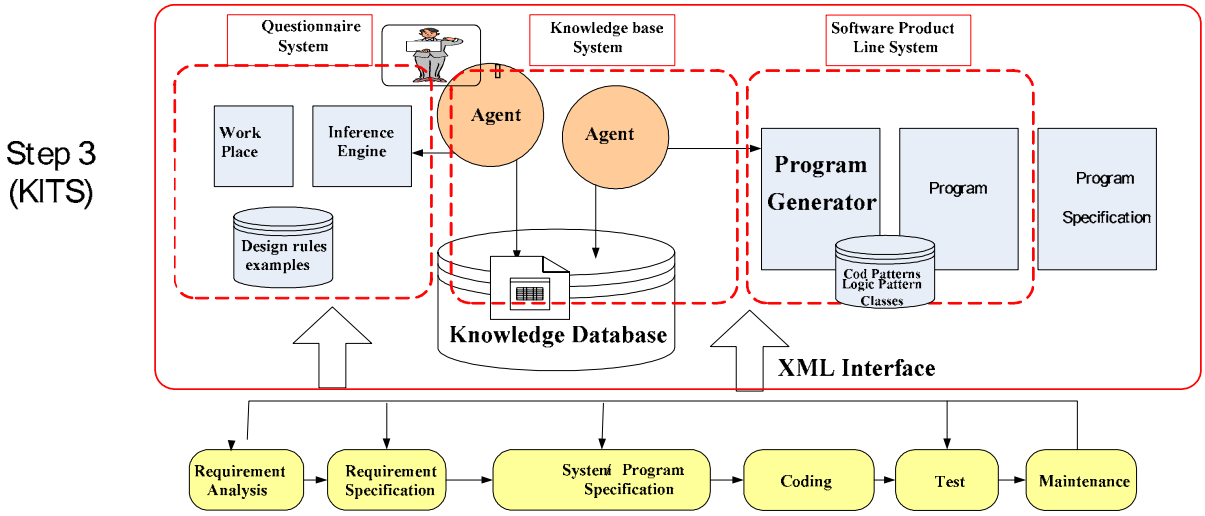


Figure 5. KITS Model

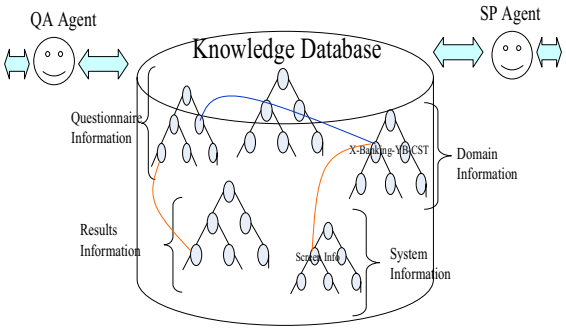


Figure 6. Knowledge Database Structure

3. The Lessons Learned

The step-wise improvements of the software developments we experienced were described. The following is the lessons we learned from these experiences:

1) Step1: Almost 80% of the source code can be automatically generated from the system specification in the web-based application systems. It can reduce the coding cost dramatically. However, the trade-off between the rest 20 software development cost and the cost for developing the toolkit must be seriously considered.

2) Step2: The rest 20% software development cost can be overcome by raising the automatic program generation ratio to 100%. We verified that it was possible to raise the automatic code generation ratio to 100% by restricting application domains in the software product line systems. It is also possible to generate the system specification automatically from use requirements using XML; which in turn can generate program source code automatically. However, it should be noticed that it is not always possible to generate the 100% source code in the software product lines (see Figure 7). It entirely depends on the applications. It means that the automatic system generation ratio in a software product line ranges from 80 % to 100 % depending upon the industry types and circumstances. There still exists trade-offs that we must consider; that is the trade-off between the cost for modifying the toolkit to the software product line and the quantities of applications those are applicable to. It must have a lot of

applications to be paid enough to modify the toolkit.

Step3: The trade-off observed at the second step can be mitigated by integrating the requirement analysis and system specifications. The one approach is the KITS model we proposed (see Figure 5). The meta data, which represent the functions and data that the customer requires, plays an important role to develop the software product line systems. If these meta data can be represented into the knowledge database, we can reduce the analysis cost by automatically generating the system specification and program source code, as is shown in our KITS model. The KITS model we proposed is the way of giving the flexibility to the development of the software product line systems. It models the efficient customized software systems while offering a substantial savings in time and cost in web-based application development.

4. Conclusion

This paper surveyed our step-wise approaches to the integrated software development of the web based software systems. The step-wise approaches to the experimental development of the web-based application systems are described. The consistent view of our software development is to reduce the cost of software development with high quality and flexibility on the time schedule. The architecture of the software population systems, that is the families of the software product lines, is our next research.

Even though we proposed the integrated software product line model, the refinement of the KITS model and its evaluation of the validity still remain. We are developing the KITS model based software population systems. It is necessary to verify the feasibility of concrete web applications as a future work.

Acknowledgements

The work of this research was made possible thanks to OKI Electric Industry Co.,Ltd., and OKI Software CO., Ltd.

References

[1] Rick Kazman, Robert L.Nord, Mark Klein, A life-Cycle View of Architecture Analysis and Design Methods, CMU/SEI-2003-TN-026, September 2003.

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

[3] Paul Clements, Linda Northrop, Software Product Lines: Practice and Patterns, Addison-Wesley, August 2001.

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

[5] 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.

[6] Makoto Yoshida, Mitsunori Sakamoto, Time and Cost Evaluation of Automatic Program Generator in a Web-based Application Environment, 2nd International Conference on Computer and Information Science, Seoul, August, 2002.

[7] Noriyuki Iwane, Yukihiro Matsubara, Makoto Yoshida, Web Application Development Model with Questionnaire System in Software Product Lines,The 7th International Conference on Industrial Management, Okayama, November, 2004.

[8] Makoto Yoshida, Noriyuki Iwane, Yukihiro Matsubara, The Integrated Software Product Line Model, IEEE Conference on Cybernetics and Intelligent Systems, Singapore, December, 2004.

[9] Paolucci. M, Kawamura.T, Payne.T.R, Sycara.K., Semantic Matching of Web Services Capabilities, Int'l Semantic Web Conf, 2002.

[10] Robert B.Grady, Successful software process improvement, Prentice-Hall,1977.

[11] B.W.Boehm, B.K.Clark, An Overview of the COCOMO 2.0 Software Cost Model, http://sunset.usc.edu/research/COCOMOII/Docs/stc.pdf

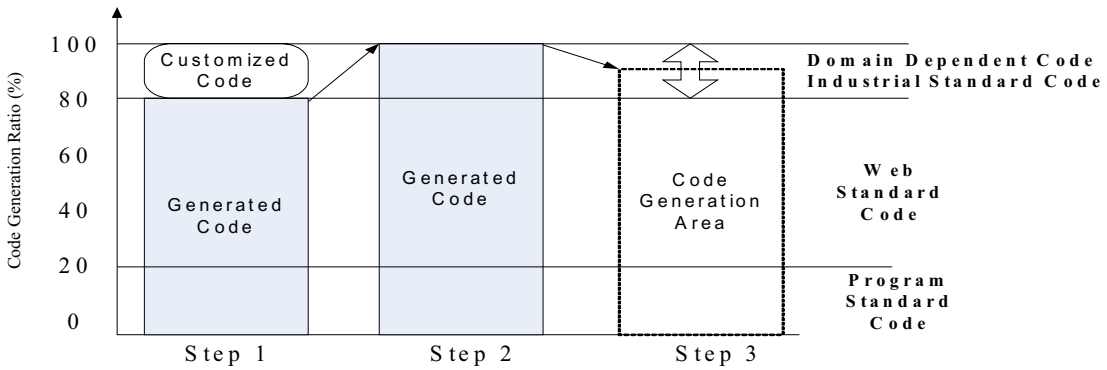


Figure 7. Code Generation Ratio