Developing and Evaluating Web Application Systems Based on Automated Program Generation Toolkit

Makoto Yoshida   Mitsunori Sakamoto

Recent advancements in IT technology are fueling corporate software development efforts to provide high quality, low cost products in an ever-shorter period of time[1]. To help achieve this goal, development techniques such as standard architecture[2], framework [3) 4)], and design pattern [3) 4)] as well as Component-Based Software Engineering (CBSE) [5) 6) 7) 8)] development methods are receiving increasing attention. Through use of these technologies and methods, it is expected that low cost, high quality applications can be achieved.

However, in order to utilize these technologies and methods effectively, well-defined modular approach enabled by superior overall application design is required [9)]. At the same time, it is necessary to address the trade-off between system flexibility and complexity. Flexible solutions increase complexity and raise the cost of implementation. Our aim is to supplement these trade-offs.

We have developed a toolkit for automated program generation, based on design patterns, and have applied it to an actual Web application development project [10) 11) 12) 13)]. Using this toolkit realizes significantly higher productivity through the representation of software development knowledge and know-hows in the form of design patterns.

The toolkit runs on multiple architectures (ASP/COM architecture [10) 13)] and JSP/EJB architecture [11)] and cost evaluations were performed on each architecture [11) 12)]. This article surveys toolkit methodology and cost efficiencies. Table 1 shows the files created by this toolkit.

Table. 1   Table 1 Files created by the toolkit

	Presentation layer	Business logic layer
Java architecture	JSP files	Java files (EJB)
ASP/COM architecture	ASP files	C++ Source code (COM)

Software Development Methodology

A traditional software development process consists of analysis, design, implementation, and testing processes. On the other hand, in a development approach based on components ("componentware"), these processes are modified as follows: analysis, component-oriented design, component composition, and testing processes [5) 14)]. The IBM San Francisco Project is an example of componentware development [4) 6)]. The advantages of component-based development include reduced development costs and high quality (reliability) achieved through component reuse. However, there are disadvantages as well.

- New methodology has not been well-establishied [15)].
- New education for software developers is necessary.
- As the number of components increase, the integration of components becomes more difficult and the maintenance costs increase.

We have developed an automated program generation toolkit, which leverages the advantages of component-based development, while following traditional software development styles.

The following are our guidelines for developing the toolkit. A comparison between a traditional software development process [14)] and the authors' development method is shown in Fig. 1.

Development methodology guidelines:

- Reuse components, but also follow a traditional software developing methodology. Develop a method to integrate the advantages of component development into the traditional development process.
- Encapsulate design patterns that the experienced developers accumulated into the toolkit and provide them to developers with symbolic user interfaces.
- Design each components to be simple and general. The number of components should be minimized.

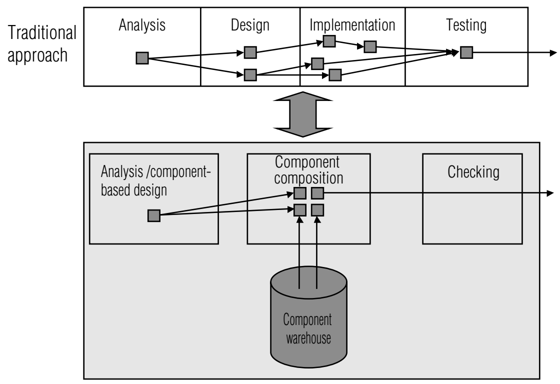


Fig. 1 (a) Componentware development process

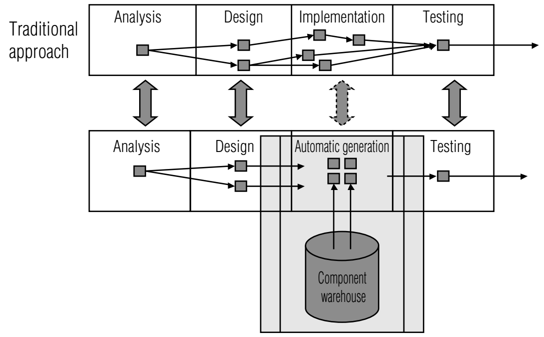


Fig. 1 (b) Software development process using automated program generation tools

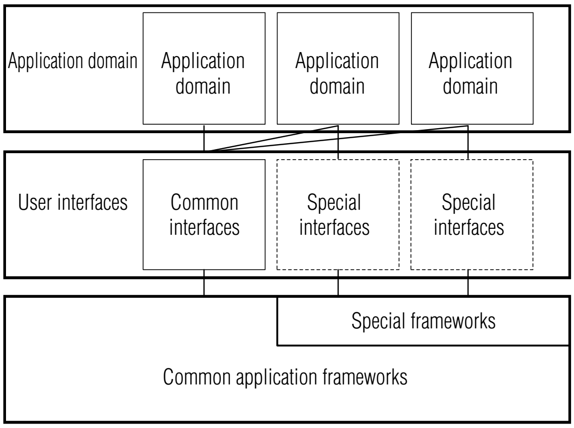


Fig. 2 Software architecture

Software Architecture

Generative programming is a method to automatically select and assemble components as needed. Some consider it a change from the traditional software development paradigm [16]. However, this method is only effective when it is specialized within an application domain. We first built an automated program generation toolkit that is not restricted to the specific application domain and then customized the toolkit as necessary based on effectiveness testing.

The toolkit software architecture is shown in Fig. 2. The software architecture consists of application domain, user interfaces, and application frameworks. User interfaces consist of common interfaces and special interfaces. The common interfaces do not depend upon the application domain. In a typical development, only common interfaces are used. For some development projects, implementation cost may not differ significantly from the case where ordinary manual coding is employed because automated generation results in wasteful code when merely using common interfaces. In the present case, the toolkit is customized for the application domain and special interfaces are established. However as described later, according to our evaluation results, it was verified that even a universal toolkit using only common interfaces is very effective.

We have built common application frameworks for the toolkit by classifying general functions of Web application programs into the following elements:

- Elements that can be generalized as components, and reused as class libraries.
- Elements that cannot be generalized or reused as classes, but which have very similar code patterns (such as repeating code).
- Elements that can be used only for the specific application.

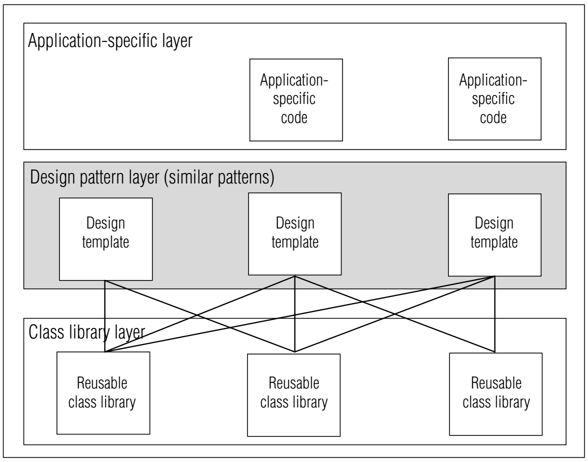


Fig. 3 Application framework

A common application framework, as shown in Figure 3, is based on the three classifications above and consists of three layers: class library layer, design pattern layer and application-specific layer. Similar patterns are generalized as design templates and are integrated into the design pattern layer. This layer is the core of the toolkit, and all developer's know-hows resides in here.

The toolkit represents similar code patterns as simple character strings, and it enables automated code generation by specifying that character strings as an application information into the toolkit. Application-specific sections need to be supplemented manually after generating the programs. However, when the amount of source code modifications is over 25% of the total, it is necessary to customize the toolkit to accommodate to the specific application domain [7) 17)].

Automated Program Generation Toolkit

The toolkit consists of basic libraries and various tools. The tools include a presentation tool to generate JSP/ASP source code, a logic tool to generate EJB/COM source code, and a database definition script generation tool. This toolkit is for building Web applications, so the presentation layer and business logic layer are separated to keep them independent. Steps for generating source code using the toolkit are shown below.

(1) Input the application design specification to the toolkit. Specifications are the presentation definition, the business logic definition, and the database definition; those are the outputs from the design process in the software development processes (detail described in section 3.2 and 3.3).

(2) At the first steps, the toolkit automatically picks up functions from the application specifications.

(3) At the second steps, several design patterns in the code pattern layer are extracted automatically.

(4) At the third steps, according to the design pattern extracted, the corresponding basic libraries are called automatically, and source codes are assembled.

Design of the logic tool interface substantially affects the ratio of automated logic program generation. We have developed the universal logic tool interface shown in Fig. 5. The above information is entered on a spreadsheet and input to the toolkit. Design templates are extracted based on logic names and command names entered through the logic interface. At the same time, a part of the source code is generated and integrated based on parameter and supplemental information. Then source code is automatically generated depending upon the platform archtectures.

Fig. 6 shows an interactive presentation tool interface developed to generate presentation layer code.

The interactive presentation tool interface consists of layout information for screen display and control information to control business logic for the server site. Based on the input information, HTML logic control code, input data checking code, send/receive processing code, logic call code, etc. are generated automatically, as well as JSP/ASP code.

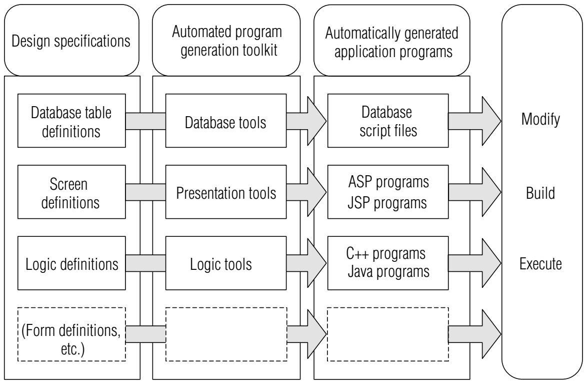
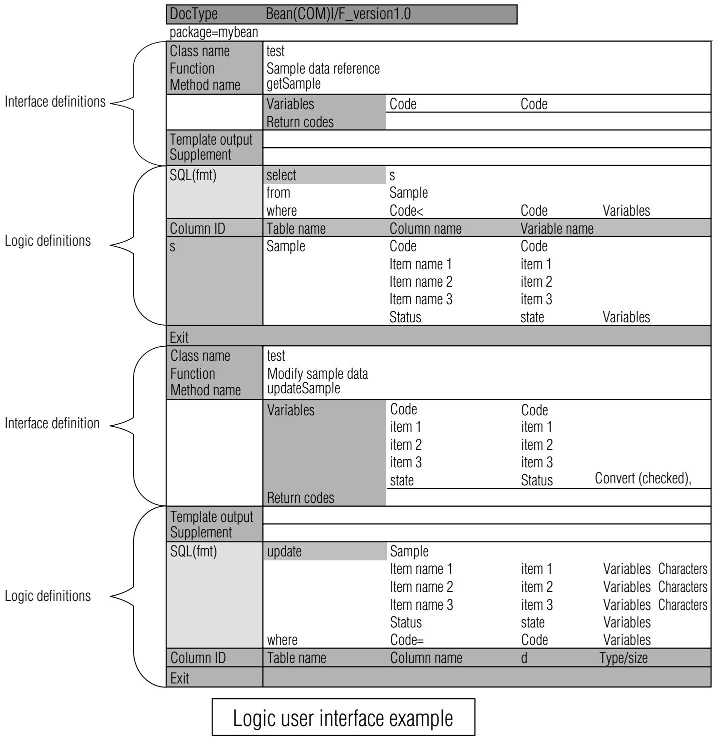


Fig. 4 Software development process


Interface definitions
Logic definitions
Interface definition
Logic definitions
Logic user interface example

DocType	Bean(COM)I/F_version1.0				
package=mybean					
Class name	test				
Function	Sample data reference				
Method name	getSample				
	Variables	Code	Code		
	Return codes				
Template output					
Supplement					
SQL(fmt)	select	s			
	from	Sample			
	where	Code<	Code	Variables	
Column ID	Table name	Column name	Variable name		
s	Sample	Code	Code		
		Item name 1	item 1		
		Item name 2	item 2		
		Item name 3	item 3		
		Status	state	Variables	
Exit					
Class name	test				
Function	Modify sample data				
Method name	updateSample				
	Variables	Code	Code		
		item 1	item 1		
		item 2	item 2		
		item 3	item 3		
		state	Status	Convert (checked),	
	Return codes				
Template output					
Supplement					
SQL(fmt)	update	Sample			
		Item name 1	item 1	Variables	Characters
		Item name 2	item 2	Variables	Characters
		Item name 3	item 3	Variables	Characters
		Status	state	Variables	
	where	Code=	Code	Variables	
Column ID	Table name	Column name	d	Type/size	
Exit					

Fig. 5 Example of logical definitions

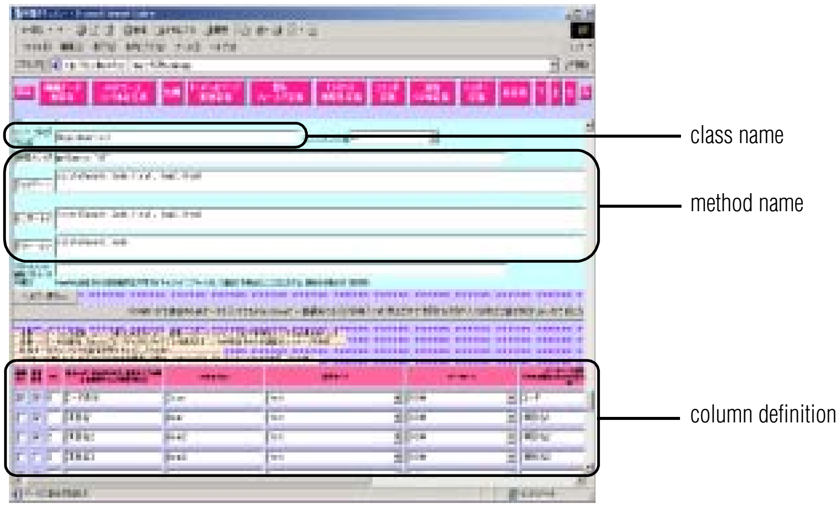


Fig. 6 Presentation user interface

Evaluation of the Automated Program Generation Toolkit

Using the toolkit, we built the same systems for both ASP/COM architecture and JSP/EJB architecture and compared the volume of generated code for each. Table 2 shows a comparison of the resulting number of source code steps. 100% of the source code for the systems was generated automatically by the toolkit. All differences between the architectures are resolved within the toolkit, so there is no difference in terms of input information. From the results shown in Table 2, it was confirmed that there is little difference: any differences between the architectures have been resolved within the toolkit.

Below is presented a cost analysis based on program code results generated on the ASP/COM architecture. Considering that differences between the two resulting systems are minor as discussed above, this analysis could similarly be performed for either architecture.

Table. 2 Code comparison by architecture

	Presentation layer	Logic layer	Total
ASP/COM architecture	993	313	1306
JSP/EJB architecture	1027	284	1311

Table. 3 shows the results of automated program generation when applying this tool to various business applications. Some fields of application are: accepting/placing orders, CRM (Customer Relations Management), merchandise distribution, and sales management. Table. 3 shows that the average ratio of automated generation by the toolkit is 86.2%. This means that the bulk of the programs can be generated automatically. Further, the average ratio of automated generation by the presentation tool is 90.1%, and the average ratio by the logic tool is 72.7%. The average ratio of automated generation by the presentation tool is much higher than the average ratio for the logic tool. These rates may vary depending on characteristics of the business. However, for the selected projects the difference can be accounted for by the fact that the presentation layer (screens) have general properties, while the logic layer consists of more business-specific portions.

In Fig. 7, implementation costs when using the toolkit are shown, based on a Selby curve. The Selby curve is well known to help understand software reuse costs. It is said that reuse requiring 25% modification of existing source code is equivalent to 55% of the cost of rewriting that code from scratch [7) 17)]. Fig. 7 also shows the source code modification ratio, described above, plotted on the Selby curve. It is clear that, for three of the four projects, implementation costs can be reduced by more than 60%.

Table. 3 Ratio of automated program generation

(a) Presentation tool

	Presentation tool		
Application type	Source code steps	Modified code steps	Ratio of automated program generation
Distributed business			
- Ordering system	13,300	40	99.7%
- Sales manag-ment system	19,418	1,960	89.9%
- Returned products processing system	3,690	1,793	51.4%
Financial business			
- CRM system	12,617	1,036	91.8%
Total	49,025	4,829	90.1%

(b) Logic tool

	Logic tool		
Application type	Source code steps	Modified code steps	Ratio of automated program generation
Distributed business			
- Ordering system	5,800	88	98.5%
- Sales manage-ment system	3,645	1,793	50.8%
- Returned products processing system	3,782	2,021	46.6%
Financial business			
- CRM system	1,277	63	95.0%
Total	14,504	3,965	72.7%

(c) Presentation & logic

	Logic and presentation tool		
Application type	Source code steps	Modified code steps	Ratio of automated program generation
Distributed business			
- Ordering system	19,100	128	99.3%
- Sales manage-ment system	23,063	3,753	83.7%
- Returned products processing system	7,472	3,814	49.0%
Financial business			
- CRM system	13,894	1,099	92.1%
Total	63,529	8,794	86.2%

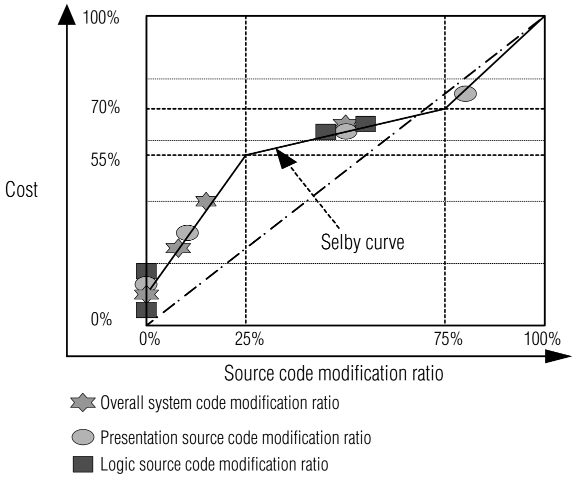


Fig. 7 Software implementation costs

Table. 4 shows calculated software development costs (for the processes of analysis through testing) based on the Selby cost curve. Costs for implementation and testing are calculated using the following formulae.

Implementation cost = Csi x Ci
Testing cost = Cst x Ct
Ct = (10 x Ci + 3) / 13

Where Csi and Cst are standard software development costs (Csi: Costs of standard implementation and Cst: Cost of standard testing), recommended by reference document 1). Ci represents the implementation cost ratio (%) derived from the Selby cost curve based on the data in the former section. Testing costs depend on the number of test items. Tests consist of program tests (unit tests) and system tests (integration tests). Based on our experience, we have set the ratio between unit tests and integration tests at 10:3. Ct is calculated here considering that the number of test items for unit tests is proportional to the ratio of manual to automated coding. Table. 4 shows that using this toolkit it is possible to reduce cost of software development by 43%.

Table. 5 shows software life cycle cost effectiveness using total costs from analysis through maintenance.

Table. 4 Software development costs

Software development costs	Analysis	Design	Implementa-tion*a	Testing*b	Total costs	Reduced costs
Standard (1)	18	19	34	29	100	0
System	18	19	10.2	9.8	57	43
Presentation	18	19	6.8	8.9	52.7	47.3
Logic	18	19	18.7	12.8	68.5	31.5

*a :Calculated from the ratio of automated program generation on the Selby curve
*b :Calculated based on the number of test items

Table. 5 Table 5 Software life cycle costs

Software development & maintenance costs	Development costs*a	Rework costs*b	Knowledge recovery costs	Total costs	Reduced costs
Standard (1)	41	31	28	100	0
System	23.4	15.2	28	66.6	33.4
Presentation	21.7	14.3	28	64	36
Logic	28.1	18	28	74.1	25.9

*a :Development cost = (Standard development cost) x (Ratio of reduction through use of the toolkit)
*b :Rework cost = (Standard rework cost) x (Ratio of reduction through use of the toolkit - 10%) x 0.33

As was done for software development costs, the standard cost models adopted are taken from recommendations in reference 1). Development costs and rework costs in the software life cycle are represented by the following formulae. A standard value is employed for the cost of knowledge recovery.

Development cost = Csd x Ci
Rework cost = Csb x C
C = (Ci -10%) x 0.33

Where Csd and Csb are standard software development costs (Csd: Cost for software development and Csb: Cost for maintenance and rework), which are recommended by reference 1). The cost of rework is 1/3 because of program reuse [1]. Table. 5 shows that using this toolkit can reduce software life cycle costs by 33.4%.

Table. 6 and Fig. 8 summarize the cost reduction effect. The average ratio of automated program generation by the toolkit was 86%. It is thus demonstrated that, using this automated generation method, it is possible to reduce implementation cost by 70%, software development cost by 43%, and software life cycle cost by 33%. Compared to traditional methods, this approach offers the possibility of significant cost savings.

Table. 6 Cost reduction effectiveness

Reduced costs (%)	System	Presentation	Logic
Implementation	70	80	45
Software development	43	47	32
Software life cycle	33	36	26

Conclusion

In this paper, we introduced a toolkit to automatically generate programs for the presentation and logic layers of Web application systems. We described the development methodology, approach, and cost effectiveness of the toolkit. By applying the toolkit to some actual projects, we verified that it is possible to significantly reduce costs compared to traditional methods. Although this article focuses mainly on using the toolkit for implementation, the toolkit also offers substantial effectiveness during analysis and design processes. It has been observed to significantly reduce rework required after implementation. Additionally, it is effective to facilitate the accumulation of expertise using design patterns and programming patterns where that expertise is otherwise difficult to gather and share.

Web services using component distribution still have many problems. However, the authors' approach can be considered an effective approach for Web services.

Thanks to the pervasiveness of J2EE and MVC architecture [2], it won't be long until the component distribution approach becomes widespread. In addition to component distribution through making interfaces more open, integration approaches such as that using this toolkit will also be important. Providing user interfaces and frameworks and achieving Web services with an appropriate level of service granularity are challenges for the future.

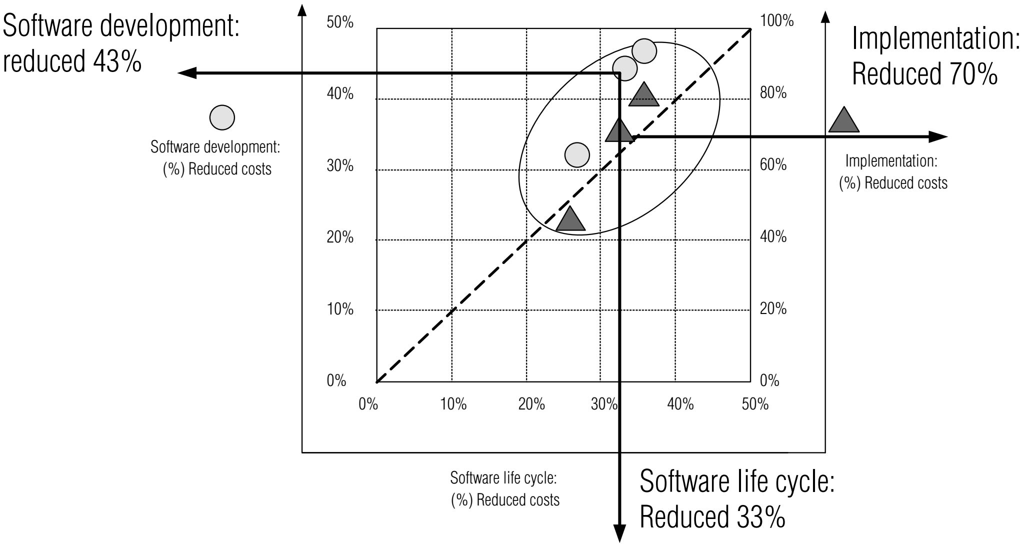


Fig. 8 Cost reduction effect

References

1) Robert B.GradyÅF Successful Software Process Improvement, Prentice-Hall, 1997.

2) Yunoura, et. al.: "Web System Development Techniques using EJB Components", Soft Research Center, March 2002

3) Suzuki, Tanaka, Nagase, Matsuda: "Reviewing Software Patterns - From Origin of Patterns to Future Development", JUSE Press, June 2000

4) R.E. Johnson, Nakamura, Nakayama, Yoshida: "Patterns and Frameworks", Kyoritsu Shuppan, June 1999

5) M.Aoyama: New Age of Software Development How Component-Based Software Engineering Changes the Way of Software Development, 1988 International Workshop on Component-Based Software Engineering, Kyoto, Japan, 4. 1998

6) IBM San FranciscoÅFConcepts and Facilities, IBM Corporation, 1977 Project, Software Development, Vol.6, No.2, 2. 1998

7) R.Selby: Empirically Analyzing Software Reuse in a Production Environment, Software Reuse-Emerging Technology, editor Will Tracz, IEEE Computer Society Press, New York, pp. 176-189, 1988

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

9) Shibata, Gemba, Kodama: "Evolution of Products Architecture", Hakuto Shobo, June 2002

10) M. Yoshida, M. Sakamoto: Experimental Results of Pattern-Based Automatic Program Generator, Proceedings of the IEEE SAINT 2002, Nara, Japan, Jan-Feb. 2002

11) M. Yoshida, M. Sakamoto: Experimental Knowledge-Based Automatic Program Generator, Networks 2002: Joint International Conference: IEEE ICWHN 2002 and ICN 2002, Atlanta, USA, 8. 2002

12) M.Yoshida, M.Sakamoto: Time and Cost Evaluation of Automatic Program Generator in a Web-Based Application Environment, 2nd ACIS Annual International Conference on Computer and Information Science, Seoul, Korea, 8. 2002

13) Sakamoto, Iwane, Yoshida: "Web Application Development using Automated Program Generation Tools", 2002 IEICE General Conference, SA-6-5.

14) Aoyama, Chusho, Kouyama: "Componentware," Kyoritsu Shuppan, August 1998

15) K. Bergner, A. Rausch, M. Sihling: Componentware - The Big Picture, 1998 International Workshop on Component-Based Software Engineering, Kyoto, Japan, 4. 1998

16) K. Czarnecki, U.W. Eisenecker: Components and Generative Programming, ESEC/FSE Åf99,Toulouse, France, 9. 1999

17) D. Batory: Intelligent Components and Software Generators, Technical Report 97-06, Dept. of Computer Sciences, Univ. of Texas at Austin, 2. 1997

Authors

Makoto Yoshida: Oki Software Co., Ltd. Chugoku Regional Office, HDS Planning Office, General Manager

Mitsunori Sakamoto: Oki Software Co., Ltd. Chugoku Regional Office, HDS Planning Office, Development Leader