プログラム自動生成ツールによるWebアプリケーションシステムの開発とその評価

吉田　誠　　坂本　光範

近年のIT技術の急激な進展に伴い，企業のソフトウェア開発では低コストで高品質なソフトウェアを短期間で製造することがますます必要とされている[1)]。このために，標準アーキテクチャ[2)]，フレームワーク[3) 4)]，デザインパターン[3) 4)] 等の開発技術，そしてコンポーネントによる開発方式[5) 6) 7) 8)] が注目されている。これらの技術・方式により，低コスト，高品質のアプリケーションの実現が望まれる。

しかし，これらの技術・方式を有効利用するためには優れた全体設計による優れた分割方法が求められる[9)]。また，システムの柔軟性と複雑性のトレードオフ関係を見極める必要がある。これらは非常に難しい問題である。一般的に，柔軟性を重視するとカスタマイズコストがより発生し，特定領域に限定すると当該ツールまたは方式の適用範囲が限定されることになる。著者らの一つの目的は，これらのトレードオフを緩和することである。

著者らは，デザインパターンを基本としたプログラム自動生成ツールキットを開発し，実際のWebアプリケーション開発に適用した[10) 11) 12) 13)]。これにより従来，熟練開発者に負うところに大きかったソフトウェア開発の知識・ノウハウがデザインパターンとして明示化・表現化され自動化されることにより大幅な生産性向上が可能である。

当該ツールは，各種アーキテクチャ上（ASP/COMアーキテクチャ[10) 13)]，JSP/EJBアーキテクチャ[11)]）で動作しており，各種アーキテクチャ上でのコスト評価も報告されている[11) 12)]。本稿では，当該ツールの方式論およびコスト効果をサーベイする。表1に本ツールによる出力ファイルを示す。

表1　ツールキットの出力するファイル

	プレゼンテーション層	ビジネス･ロジック層
JAVAアーキテクチャ	JSPファイル	JAVAファイル（EJB）
ASP/COMアーキテクチャ	ASPファイル	C++ソースコード（COM）

ソフトウェア開発方法論

従来型のソフトウェア開発プロセスは，分析，設計，実装，試験の各プロセスから構成される。一方，コンポーネントを基本とした開発（コンポーネントウェア）では，これらの各プロセスは，分析，コンポーネントを基本とした設計，コンポーネントの組立，試験に再構成される[5) 14)]。代表的なコンポーネントウェア開発としては，IBMサンフランシスコプロジェクトがある[4) 6)]。コンポーネントを基本とした開発の長所としては，コンポーネント再利用による開発コストの削減と高品質が上げられる一方，以下のような短所がある。

- 粒度，分割方法等，方法論的に十分確立されていない[15)]。
- ソフトウェア開発のための新たな教育を必要とする。
- コンポーネントの数が増えるに従い，統合化および維持管理が複雑化する。

著者らは，従来型ソフトウェア開発を踏襲しつつコンポーネント開発の長所を取り入れた開発方法として，プログラム自動生成ツールキットを開発した。

以下に，著者らの開発方法の指針を示す。また，図1に従来型のソフトウェア開発プロセス[14)] と著者らの開発方法との比較を示す。

開発方法論指針：

- コンポーネントの再利用も図るが，従来型ソフトウェア開発方法論も踏襲する。従来型開発プロセスの中に，コンポーネント開発の良さを埋め込む方法を開発する。
- 熟練開発者のノウハウをデザインテンプレートとしてコンポーネント化し，簡易インタフェースを提供することにより従来型開発プロセスに埋め込むとともに，部品の再利用による効率化を図る。
- 部品設計においては，各部品は小さくかつジェネラルなオブジェクト設計とする。

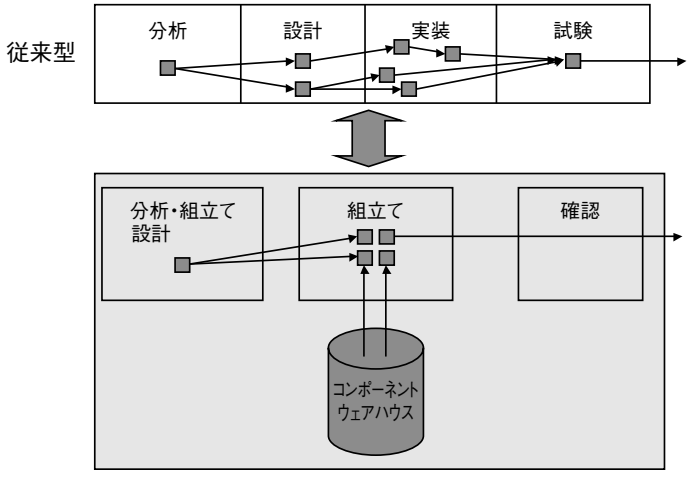

図1　(a) コンポーネントウェア開発プロセス

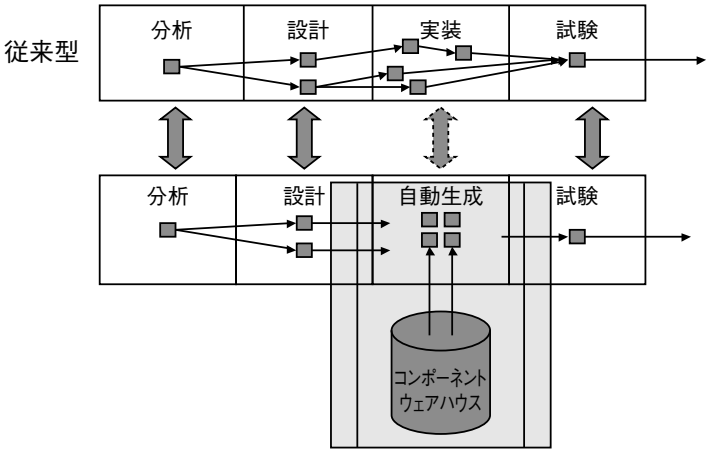

図1　(b) プログラム自動生成ツールによる開発プロセス

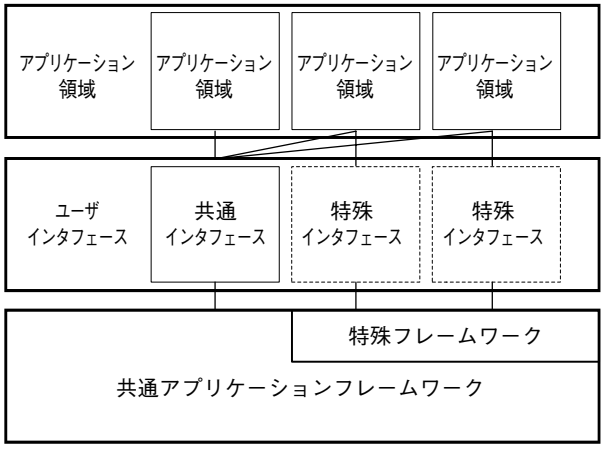

図2　ソフトウェアアーキテクチャ

ソフトウェアアーキテクチャ

生成的プログラミング（Generative Programming）は，要求に応じてコンポーネントを自動的に選択し組み立てる方法であり，従来型ソフトウェア開発パラダイムを変えるものであると言われている[16]。しかし，本手法はアプリケーション領域に特化することにより有効とされている。著者らは，まずアプリケーション領域を限定しない，汎用領域で使用されるプログラム自動生成ツールキットを構築し，当該ツールキットの有効性を確認することにより，必要に応じてツールキットをカスタマイズするアプローチを取った。

図2にツールキットのソフトウェアアーキテクチャを示す。ソフトウェアアーキテクチャは，アプリケーション領域，ユーザインタフェース，アプリケーションフレームワークから構成されている。ユーザインタフェースは，共通インタフェースと特殊インタフェースから構成されており，共通インタフェースはアプリケーション領域に依存しないインタフェースとなっている。通常の開発においては，共通インタフェースのみが使用される。開発するプログラムによっては，共通インタフェースを使用しただけでは自動生成率が悪く，実装コストが通常の手コーディングとさほど変らない場合があり，この場合には当該ツールキットがアプリケーション領域対応にカスタマイズされ，特殊インタフェースが設定されることになる。しかしながら，後述するように，著者らの評価結果においては，共通インタフェースだけを用いた汎用的ツールキットでも十分効果のあることが検証されている。

著者らは，一般的なWebアプリケーションプログラムの機能を以下の構成要素に分類することにより，ツールキットの共通アプリケーションフレームワークを構築した。

- コンポーネントとして汎用化可能な構成要素であり，クラスライブラリーとして再利用される部分。
- クラスとして汎用化再利用することはできないが，非常によく似たコードパターンを持っている部分（例えば，反復したコード，等）。
- アプリケーション特化としてしか使用されない部分。

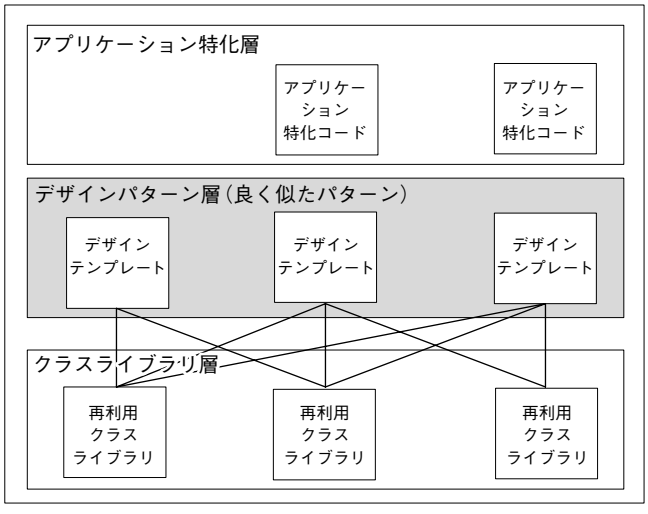

図3　共通アプリケーションフレームワーク

共通アプリケーションフレームワークは，図3に示すように，上記の3つの分類に基づきクラスライブラリー層，デザインパターン層，アプリケーション特化層の3層より構成されている。良く似たパターンは，デザインテンプレートとして一般化され，デザインパターン層に埋め込まれる。当該層はツールキットのコア部分であり，ここに熟練開発者のノウハウが詰め込まれることになる。ツールキットは，アプリケーションで実現する機能を，定義された部品群とユーザインタフェース情報をもとに写像するツールであると言える。ツールキットは，良く似たコード・パターンを簡易な文字列やツールのユーザインタフェースに対応させて，それらをアプリケーション情報として入力することで，コードの自動生成を可能にしている。アプリケーション特化の部分については，プログラム生成後，手コーディングにより補完する必要がある。ただし，ソースコード修正が全体の25%を超える場合を目安に，ツールキットをアプリケーション領域対応にカスタマイズし，アプリケーション特化のユーザインタフェースの設定を検討する必要がある[7) 17)]。

プログラム自動生成ツールキット

ツールキットは，基本ライブラリーと各種ツールから構成されている。各種ツールとしては，JSP／ASPソースコードを生成するプレゼンテーションツール，EJB／COMソースコードを生成するロジックツール，データベース定義スクリプト生成ツール，等がある。本ツールキットは，WEBアプリケーション構築のためのツールキットであり，プレゼンテーション層とビジネスロジック層は，独立性を保つためにそれぞれ分離されている。ツールキットによるソースコード生成手順を以下に示す。

① ソフトウェア開発プロセスでの設計プロセス終了時の各種情報（プレゼンテーション定義（画面定義），ロジック定義，データベース定義）をツールキット仕様フォーマットに従い入力する。
② 入力情報に基づきツールキットから自動的に機能が抽出される。
③ 次に，デザインパターン層から当該機能に基づくデザインテンプレートが自動的に抽出される。
④ 抽出されたデザインテンプレートに応じて，対応するライブラリーが自動的に呼び出され，入力情報を基にソフトウェアアーキテクチャに沿ったソースコードが生成される。

図4にソフトウェア開発過程を示す。

ロジックツールインタフェースの設計は，大きくロジックプログラム自動生成率に影響する。著者らは，図5に示す汎用ロジックツールインタフェースを開発した。これらの情報は，スプレッドシート上に記述され，ツールキットに入力される。ロジックツールインタフェースを通して入力されたロジック名，コマンド名からデザインテンプレートが抽出されるとともに，パラメタ情報，補助情報をもとにソースコードの一部が生成され，統合され，それぞれのプラットフォームアーキテクチャに応じたソースコードが自動的に生成される。

プレゼンテーション層のコードを自動生成するために

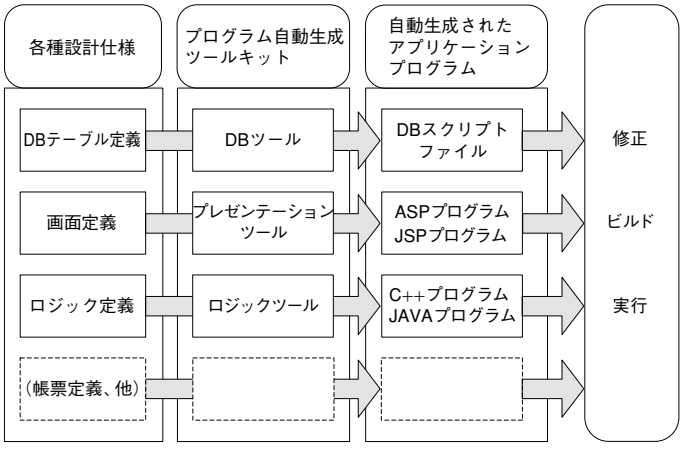

図4　ソフトウェア開発過程

DocType	Bean(COM)I/F_version1.0				
package=mybean					
インターフェース定義	クラス名	test			
	機能	サンプルデータ参照			
	メソッド名	getSample			
		変数	Code	コード	
		リターン値			
	template出力				
	補足				
ロジック定義	SQL(fmt)	select	s		
		from	サンプル		
		where	コード<	Code	変数
	カラムID	テーブル名	カラム名	変数名	
	s	サンプル	コード	Code	
			項目名 1	item1	
			項目名 2	item2	
			項目名 3	item3	
			状態	state	変数
	終了				
インターフェース定義	クラス名	test			
	機能	サンプルデータを修正する			
	メソッド名	updateSample			
		変数	Code	コード	
			item1	項目名 1	
			item2	項目名2	
			item3	項目名3	
			state	状態	変換(checked,
		リターン値			
	template出力				
	補足				
ロジック定義	SQL(fmt)	update	サンプル		
			項目名 1	item1	変数 文字
			項目名 2	item2	変数 文字
			項目名 3	item3	変数 文字
			状態	state	変数
		where	コード=	Code	変数
	カラムID	テーブル名	カラム名	d	型/サイズ
	終了				

ロジックユーザインタフェース記述例

図5　ロジック定義例

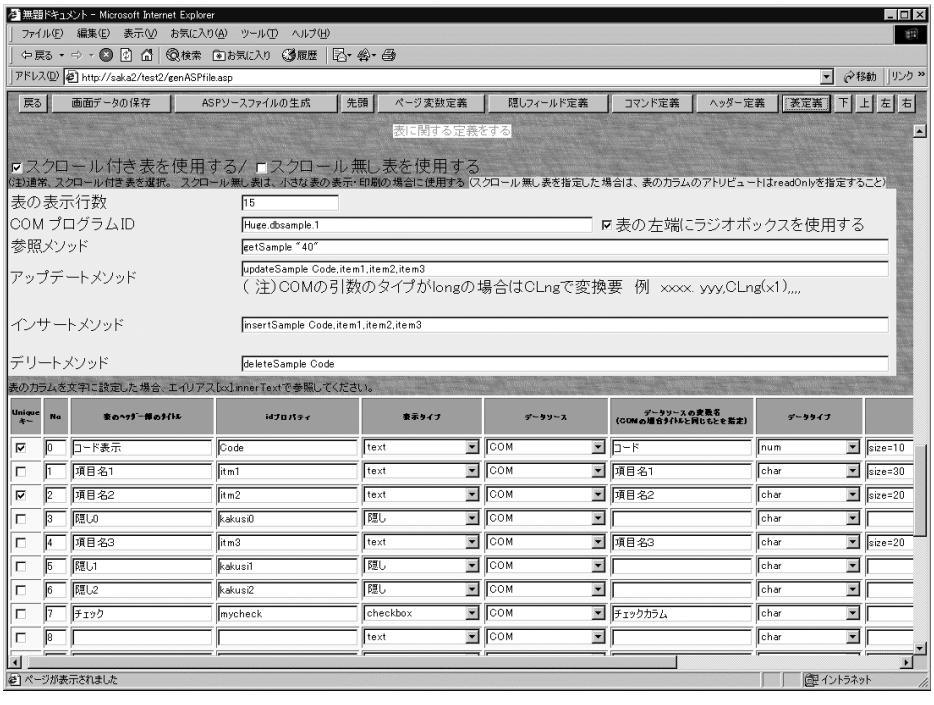

図6　プレゼンテーションユーザインタフェース

開発した会話型プレゼンテーションツールインタフェースを図6に示す。画面上に表示するレイアウト情報と，サーバサイトのビジネスロジックを制御する制御情報から構成されている。入力情報を基に，HTMLロジック制御コード，入力データチェック，送受信処理，ロジック呼出し，等のコードと共にJSP/ASPコードが自動的に生成される。

プログラム自動生成ツールキットの評価

同一のシステムをツールキットを使用して，ASP/COMアーキテクチャおよびJSP/EJBアーキテクチャの両方で作成し，生成コード量の比較を行った。表2にソースコードステップ数の比較を示す。当該システムは，100%ツールキットによりソースコードが自動生成されている。アーキテクチャ上の差異は，全てツールキット内に隠蔽されており，入力情報としての差異は全くない。表2の結果から両アーキテクチャによる差はツールキット内に吸収され，ほとんどないことが確認される。

以下，ASP/COMアーキテクチャ上でのプログラム生成結果データを基にコスト分析をするが，上記結果から以下の分析は，両アーキテクチャに当てはまるものであると言える。

表2　アーキテクチャ別コード比較

	プレゼンテーション層	ロジック層	総計
ASP/COM	993	313	1306
JSP/EJB	1027	284	1311

各種ビジネスアプリケーションに本ツールを適用したプログラム自動生成結果を表3に示す。適用分野は，受発注関連，CRM（Customer Relation Management）関連，商品流通関連，販売管理関連である。表3は，ツールキットによる平均自動生成率が86.2%であることを示しており，ほとんどのプログラムが自動生成可能であることが確認できる。また，プレゼンテーションツールによる平均自動生成率は90.1%，ロジックツールによる平均自動生成率は72.7%であることが確認される。プレゼンテーションツールによる平均自動生成率がロジックツールの平均自動生成率より非常に良くなっており，これらは業務特性に応じて変化すると考えられるが，選択されたプロジェクトにおいてはプレゼンテーション層（画面系）が一般的特質を持ち，ロジック層に業務特化な部分が多かったと考えられる。

図7にSelby曲線に基づくツールキットによる実装時のコストを示す。ソフトウェア再利用のコストに関しては，Selby曲線が知られており，25%のソースコード修正はゼロからのコーディングの55%に相当すると言われている[7) 17)]。前節で記述したソースコード修正率をSelby曲線上にプロットした図が図7である。4つのうち3つのプロジェクトは，実装時において60%以上のコスト削減に繋がることがわかる。

表3　プログラム自動生成率

（a）プレゼンテーションツール

	プレゼンテーションツール		
アプリケーションタイプ	ソースコードステップ	修正コードステップ	プログラム自動生成率
流通業			
－受発注システム	13,300	40	99.7%
－販売管理システム	19,418	1,960	89.9%
－返品処理システム	3,690	1,793	51.4%
金融業			
－CRMシステム	12,617	1,036	91.8%
総計	49,025	4,829	90.1%

（b）ロジックツール

	ロジックツール		
アプリケーションタイプ	ソースコードステップ	修正コードステップ	プログラム自動生成率
流通業			
－受発注システム	5,800	88	98.5%
－販売管理システム	3,645	1,793	50.8%
－返品処理システム	3,782	2,021	46.6%
金融業			
－CRMシステム	1,277	63	95.0%
総計	14,504	3,965	72.7%

（c）プレゼンテーション＆ロジック

	ロジック&プレゼンテーションツール		
アプリケーションタイプ	ソースコードステップ	修正コードステップ	プログラム自動生成率
流通業			
－受発注システム	19,100	128	99.3%
－販売管理システム	23,063	3,753	83.7%
－返品処理システム	7,472	3,814	49.0%
金融業			
－CRMシステム	13,894	1,099	92.1%
総計	63,529	8,794	86.2%

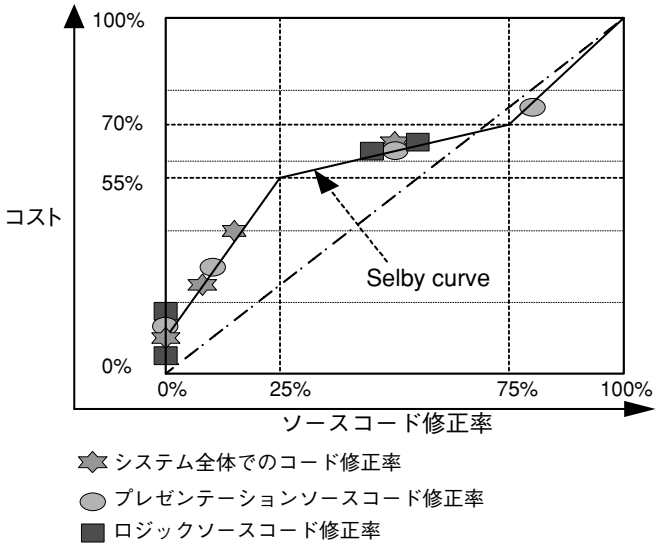

図7　ソフトウェア実装コスト

Selbyコスト曲線を基に，ソフトウェア開発コスト（分析から試験工程までのコスト）を計算した値を表4に示す。実装および試験のコストは，以下の式により計算されている。

実装コスト ＝ Csi × Ci
試験コスト ＝ Cst × Ct
Ct ＝ （10×Ci+3）／13

上記式のCsi，Cst は，参考文献1）で推奨されているソフトウェア標準開発コスト（Csi：標準実装コスト／ Cst：標準試験コスト）である。Ciは，前節のデータに基づきSelbyコスト曲線から導かれた実装時コストの割合（%）を示す。試験コストは，試験項目数に依存するものであり，それらはプログラムテスト（プログラムの部分試験）とシステム試験（システム全体の試験）から構成される。著者らは経験的に当該比率（プログラムテストとシステム試験との比率）を10：3に設定している。ここでは，部分試験の試験項目数は，手コーディングの割合に比例するものとしてCtが計算されている。表4から本ツールキットを使用することによって，ソフトウェア開発において43%のコスト削減が可能であることがわかる。

表5は，ソフトウェアライフサイクルコスト（分析からメンテナンスまでの全てのコスト）における効果を示し

表4　ソフトウェア開発コスト

ソフトウェア開発コスト	分析	設計	実装*1	試験*2	全コスト	削減コスト
標準［1］	18	19	34	29	100	0
システム	18	19	10.2	9.8	57	43
プレゼンテーション	18	19	6.8	8.9	52.7	47.3
ロジック	18	19	18.7	12.8	68.5	31.5

*1：Selby曲線上のプログラム自動生成率から算出
*2：試験項目数から算出

表5　ソフトウェアライフサイクルコスト

ソフトウェア開発＆メンテナンスコスト	開発コスト*1	手戻りコスト*2	知識回復コスト	全コスト	削減コスト
標準［1］	41	31	28	100	0
システム	23.4	15.2	28	66.6	33.4
プレゼンテーション	21.7	14.3	28	64	36
ロジック	28.1	18	28	74.1	25.9

*1：開発コスト＝(標準開発コスト)×(ツールキットによる削減率)
*2：手戻りコスト＝(標準手戻りコスト)×(ツールキットによる削減率－10%)×0.33

ている。対象となる標準コストモデルは，ソフトウェア開発コストと同様に参考文献1）で推奨されているコストモデルを採用している。ソフトウェアサイフサイクルコストにおける開発コストと手戻りコストは以下の式で表わされている。知識回復コストは，標準値をそのまま採用している。

開発コスト ＝ Csd × Ci
手戻りコスト ＝ Csb × C
C =（ Ci-10%）× 0.33

上記式のCsd，Csb は，参考文献1）で推奨されているソフトウェア標準開発コスト（Csd：ソフトウェア開発コスト／ Csb：維持および手戻りコスト）である。手戻りコストは，プログラムの再利用により1/3になるとしている[1)]。表5から本ツールキットを使用することによって，ソフトウェアライフサイクルコストは33.4%のコスト削減が可能であることがわかる。

表6および図8にコスト削減効果のまとめを示す。ツールキットによる平均プログラム自動生成率は86%であった。当該自動生成により，実装時コストの70%，ソフトウェア開発コストとしては43%，ソフトウェアライフサイクルコストとしては33%の削減が可能であることを示した。従来の方法に比べて，大幅なコスト削減が可能であることが実証された。

表6　コスト削減効果

削減コスト(%)	システム	プレゼンテーション	ロジック
実装	70	80	45
ソフトウェア開発	43	47	32
ソフトウェアライフサイクル	33	36	26

ま と め

Webアプリケーション構築において，プレゼンテーション層およびロジック層のプログラムを自動的に生成するツールキットを紹介した。ツールキット構築の方法論，方式およびコスト効果について記述した。実プロジェクトに適用し，従来型方式に比べて劇的なコスト削減が可能なことを検証した。本稿では，主にツールキットの実装時に着目し記述したが，本ツールキットの効果は，実装プロセスだけでなく，分析・設計プロセスへの効果も大きい。実装プロセス後の後工程からの手戻りも極端に減少することが確認されている。また，デザインパターン，プログラミングパターン，等の知識として習得・伝達の難しいノウハウの蓄積・有効活用にも効果がある。

コンポーネント流通によるWebサービスはまだ数々の問題を掲げている。著者らのアプローチは，Webサービスにおける一つの有効なアプローチであると考えられる。

J2EE，MVCアーキテクチャ[2)] の浸透によりコンポーネントの流通が普及する時代も遠くない。しかし，そのためには，アプリケーション群とコンポーネント群のカップリングが必要である。インタフェース公開によるコンポーネント流通に加えて，本ツールキットのような統合的アプローチも重要である。

ユーザインタフェースおよびフレームワークを整備し，適切な粒度によるWebサービスの実現を今後の課題と考えている。◆◆

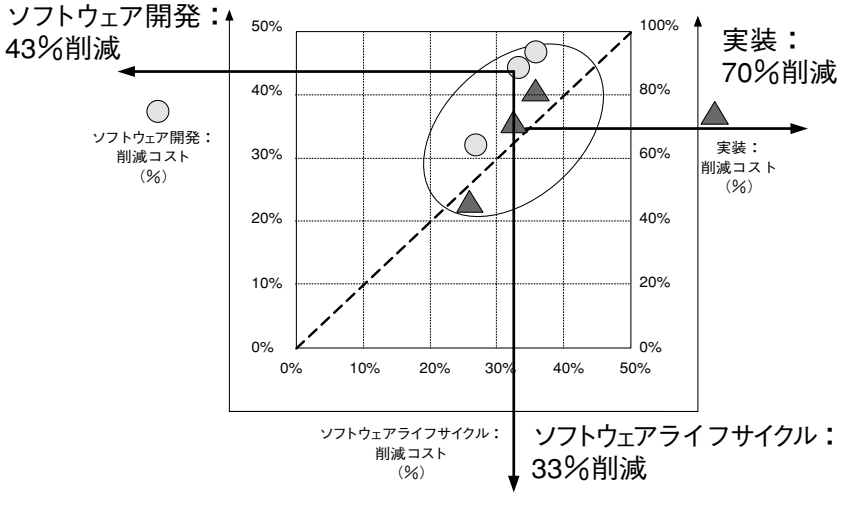

図8　コスト削減効果

■参考文献

1）Robert B.Grady：Successful software process improvement, Prentice-Hall, 1997.
2）湯浦 他：EJBコンポーネントによるWebシステム構築技法，ソフト・リサーチ・センター，2002年3月
3）鈴木，田中，長瀬，松田：ソフトウェアパターン再考ーパターン発祥から今後の展開までー，日科技連出版社，2000年6月
4）R.E.Johnson，中村，中山，吉田：パターンとフレームワーク，共立出版，1999年6月
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 Francisco：Concepts 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, New York, pp. 176–189, 1988
8）D.R.Musser, *et el*, Algorithm-Oriented Generic Libraries, HP Laboratories Technical Report, HPL-94-13, 1994
9）柴田，玄場，児玉：製品アーキテクチャの進化論，白桃書房，2002年6月
10）M.Yoshida,M.Sakamoto：Experimental Results of Pattern-Based Automatic Program Generator, Proc., of the IEEE SAINT 2002, Nara, Japan, Jan–Feb. 2002
11）M.Yoshida, M.Sakamoto：Experimantal 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）坂本，岩根，吉田：自動生成プログラミングツールによるWebアプリケーション開発，2002年電子情報通信学会総合大会，SA-6-5.
14）青山，中所，向山：コンポーネントウェア，共立出版，1998年8月
15）K.Bergner,A.Rausch,M.Sihling：Componentware - The Big Picture, 1988 International Workshop on Component-Based Software Engineering, Kyoto, Japan, 4. 1998
16）K.Czarnecki, U.W.Eisenecker：Components and Generative Programming, ESEC/FSE '99,Toulouse, France, 9. 1999
17）D.Batory：Intelligent Components and Software Generators, Technical Report 97-06, Dep. of Computer Sciences, U. of Texas at Austin, 2. 1997

●筆者紹介

吉田誠：Makoto Yoshida.沖ソフトウェア株式会社 中国支社 HDS推進室 室長
坂本光範：Mitsunori Sakamoto.沖ソフトウェア株式会社 中国支社 HDS推進室 開発リーダ