ソフトウェアコンポーネントの利用性評価

平 山 雅 之[†] 佐 藤 誠[††]

近年のソフトウェア開発では JavaBeans など様々なコンポーネントを利用して開発を進めるケースが増加している．コンポーネントの利用は生産性や品質の向上などの大きなメリットをもたらす．しかし，一方で，数多くのコンポーネントの中で，利用者の利用目的に合致したコンポーネントを選択するという作業が発生している．本論ではソースコードが提供されないブラックボックス・コンポーネントを対象に，その内部特性に関する情報を解析し，コンポーネントの使い勝手を評価する手法を提案する．本論では，コンポーネントの使い勝手の評価方式として，コンポーネントの導入時の学習容易性，実装作業の容易性などを評価する評価観点と評価メトリクスを提案する．また，これらを実際のコンポーネントに試験適用した結果，コンポーネントの使い勝手のおおよその評価を行えるとの結果を得た．さらにこれらの評価視点，評価メトリクスによる評価結果と，実際にコンポーネントを利用したユーザの持つ使い勝手に関する印象についても比較的一致することを確認した．

A Proposal of Software Component Usability Evaluation Method Based on Programmers' Viewpoint

Masayuki Hirayama[†] and Makoto Sato[††]

Recently, software components, such as javaBeans and ActiveX, are widely used in software development. Use of software components can improve the productivity of software development and quality of software systems. On the other hand, users are required for another works like selecting the suitable components from various candidate components. Especially, some components are less usability, and as a result, it is difficult to attain the expected productivity improvement by using these components. In this paper, a method for evaluating and measuring the usability of black box components is proposed. Black box components are the components that are not provided source code or other implement data. The method enables the evaluation of black box component's usability without source code information. The evaluation model in the method consists of the two evaluation viewpoints – easiness of learning for component's function and easiness of implementation. In this paper, we also introduce the experimental application of the method. Results of the experiment shows that the usability of the black box component can be evaluated by adapting the proposed method. In addition, the result from the quantitative evaluation by the metrics is coincidence with the user impression for the target component's usability.

1. はじめに

1.1 背 景

近年，オブジェクト指向の広まりとともに，様々なコンポーネントが利用されるようになってきている．対象製品やドメインの特質に見合った適切なコンポーネントを選択し利用した場合，ソフトウェアの開発効率が向上したり，品質が向上したりするなど大きな効果が期待できる[1)]．しかし，対象製品やドメインの特質にふさわしくない不適切なコンポーネントを利用した場合には，逆に生産性や品質を大きく損なう可能性もある[2)]．

一般にソフトウェアのコンポーネントは，ソースコードなどが公開されているものと，公開されていないものの 2 種類に大別することができる．本論では前者をホワイトボックス・コンポーネント，後者をブラックボックス・コンポーネントと呼ぶ[3)]．このうち，ホワイトボックス・コンポーネントは，公開されているソースコードなどを読むことで，おおよその機能や構造を把握することができ，コンポーネントのユーザは利用可否の判断をつけやすい．一方，ブラックボックス・コンポーネントではソースコードが公開されていないためコンポーネントの機能や構造の把握は難しく，一般にはコンポーネントに付随する README ファイ

† 株式会社東芝ソフトウェア技術センター
Software Engineering Center, TOSHIBA Corporation
†† 株式会社東芝セミコンダクター社
Semiconductor Company, TOSHIBA Corporation

ルやカタログスペックなどの概要情報を参考に利用可否を判断することが多い．

1.2 コンポーネントの使い勝手の評価視点

特に，コンポーネントの利用可否判断では，コンポーネントの使い勝手が重要な判断材料になることが多い．コンポーネントの使い勝手によって，開発効率が飛躍的に高まる場合もあれば，逆に開発の足を引っ張ってしまう場合もある．たとえば，コンポーネントを利用する場合，コンポーネントの利用方法が十分に把握できずに，コンポーネントの動作理解に時間を要したり，それぞれのソフトウェア本体にコンポーネントを組み込む際に工数を要したりといったことが発生すると，開発担当者のフラストレーションを増加させる結果になりかねない．特にオブジェクト指向をベースにしたコンポーネントでは，クラス/メソッドの継承などの扱いによって，その使い勝手が大きく異なってくる．しかし，このようなコンポーネントに関する使い勝手の情報は，多くの場合，利用者の経験に基づくところが多く，使い勝手を評価するための標準的な考え方や手法は整備されていない．

本論では ISO/IEC9126 [17] で定義されたソフトウェアの使用性に関する「習得性」「理解性」「運用性」などの副特性の視点やコンポーネントの利用や実装作業を考慮し，コンポーネントの使い勝手（利用性）について，

- 導入時のコンポーネント利用方法の習得/把握が容易であること，
- 実際にソフトウェアの本体にコンポーネントを組み込む作業が容易であること，

の 2 つの視点に着目する．

1.3 研究の狙い

上記の背景を念頭に，本研究ではブラックボックス・コンポーネントの利用性を簡易的に評価する方式を検討した．具体的にはブラックボックス・コンポーネントとしてコンポーネント利用者に公開される情報をもとに，以下のような特徴を持つ手法を開発する．

① コンポーネント利用者が入手できる情報のみを利用しブラックボックス・コンポーネントを解析評価し，

② コンポーネント利用性に関する内部特性をモデル化し，それを計測して，おおよその使い勝手を評価する．

本論ではこの研究の一環として実施した JavaBeans に関するブラックボックス・コンポーネントの使い勝手を評価する手法と評価のメトリクスを紹介する．また，実際の JavaBeans を対象に，提案手法を適用した事例についても報告する．本論は以下，2 章ではブラックボックス・コンポーネントの評価視点および評価方式について提案する．3 章ではコンポーネントの利用性評価モデルについて紹介し，4 章では提案手法を実際のコンポーネント評価に適用した事例をもとに評価メトリクスの評価基準値の設定について紹介する．5 章では，提案手法の有効性を確認することを目的にした適用評価を紹介し，6 章で考察を加える．7 章では本研究に関連する研究を紹介し，8 章では本研究のまとめとともに，今後の方向性などについて述べる．

2. ブラックボックスコンポーネントの評価

2.1 コンポーネント評価の目的と課題

2.1.1 コンポーネント評価の目的

JavaBeans，ActiveX DLL など昨今，様々なソフトウェアコンポーネントがあふれている [4)~6)]．コンポーネントの多くは，ソースコードを公開しないブラックボックス・コンポーネントとして提供される場合がほとんどである．ブラックボックス・コンポーネントを利用する場合，一般のユーザは，コンポーネントに付随する概要情報などを参考に，利用対象コンポーネントを選択し，利用する．しかしこの場合，記載されている情報の大半は機能的な側面に関する情報であり，その使い勝手などに関する情報は十分には提供されていない．このため，コンポーネントのユーザはコンポーネントを自らのソフトウェアに組み込んで初めて，その使い勝手の悪さなどに気が付く場合がほとんどである．結果として，コンポーネントを利用したことによって逆に開発効率を落としてしまう結果を招くことすらある．

以下，コンポーネントを利用したソフトウェア開発において，導入するコンポーネントの選択の手順を簡単に整理する．一般のコンポーネントユーザの作業を分析しモデル化すると，コンポーネントは**図 1** に示す

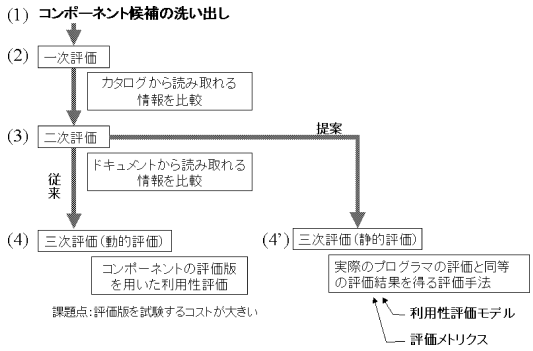

図 1 コンポーネントの選択手順
Fig. 1 Component's selection flow.

ように，コンポーネント候補の洗い出し，一次評価，二次評価，三次評価という 4 段階のステップで選択されることが分かった．各ステップの詳細は以下のとおりである．

Step-1：コンポーネント候補の洗い出し

市場に流通するコンポーネントの中から，最低限必要な機能を持つものを洗い出す．

Step-2：一次評価

候補としてあげられたコンポーネントの中から使えそうなものを絞り込むために，導入コスト，機能仕様，性能，開発環境との相性，過去の採用実績などを比較し，条件が優れているものを選ぶ．

Step-3：二次評価

一次評価で絞り込んだコンポーネント候補に対して，ドキュメントなどから詳細な情報を比較検討し，使い勝手が良さそうなものを選ぶ．

Step-4：三次評価

二次評価では，実際にコンポーネントを開発するシステムに組み込んで，システム開発がしやすくなるかは分からないため，コンポーネントの評価版を入手し，試験実装を行って利用性を評価する．

コンポーネント選択の手順においての問題は，Step-4 の段階でソフトウェア開発者が実際にコンポーネントを試用してみなければ，利用性（使い勝手）は評価できない点にある．一般に Step-3 までの評価で導入対象コンポーネントを 1 つに絞りきれなかった場合，ユーザは複数のコンポーネントに対して Step-4 の評価を行うことが求められ，結果として評価コストが大きくなってしまう．この結果，コンポーネントの評価に時間がかかり開発サイクルが長期化したり，逆にコンポーネントの評価を省いて導入決定したために，導入したコンポーネントを開発者が使いこなすまでに時間がかかりすぎたりするなど，開発効率の向上が実現しないことなどが発生する．

2.1.2 コンポーネント評価の視点

(a) ISO/IEC9126 [17] との関係

一般的なソフトウェアシステムの使い勝手に関しては，ISO/IEC9126 の中で「使用性」として定義されている．ISO/IEC9126 で規程されている「使用性」はソフトウェアシステム全般を対象にしたものであり，本論で扱うコンポーネントの利用性とも深い関係を持つと考えられる．本論ではソフトウェア・コンポーネントの利用性を ISO/IEC9126 におけるソフトウェアの使用性の定義を参考に以下のように定義する．

コンポーネントの利用性

ソフトウェア・コンポーネントの「使いやすさ」「使うことにかかる労力」「使うことによって得られる効果」の良し悪しを評価する．

ISO/IEC9126 では「使用性」に関しては,「習得性」「運用性」「理解性」の 3 つの副特性が提案されている.「習得性」はソフトウェアを使用するうえで参照するマニュアルの使いやすさ，使いこなし・使い方・運用方法などを習得するまでの習得時間などが代表的なメトリクスとなっている．また,「運用性」はソフトウェア導入時の手間や時間（導入性），ソフトウェア操作にかかる手間・時間（操作性）などがメトリクスになっている.「理解性」については，ソフトウェアの利用を検討する際のコンセプトの明快性などを考え，機能の把握のしやすさなどを指標として取り上げている．ISO/IEC9126 で規程されたこれらの観点は，一般のユーザを対象としたソフトウェア製品を念頭においているため，ソフトウェア・コンポーネントを対象とする場合には若干の見直しが必要となる．

(b) 外部特性 vs 内部特性

一般に，ソフトウェアの品質を計測・評価する場合，外部特性と内部特性という 2 つの視点を考えることができる[17]．外部特性はソフトウェアの内部構造などを見ることなく評価する特性であり，内部特性はソフトウェアの内部構造に着目して初めて評価できる特性である．この観点からコンポーネントの利用性を考えると，外部特性としては，実際にコンポーネントを利用する際に確認できる「コンポーネントの利用方法把握にかかる時間（導入・理解性）」や「実際に利用する際に要する時間（運用性）」などがあげられる．一方，内部特性はこうした実際の利用時の実時間データではなく，コンポーネントの設計情報やソースコードなどを解析し，その使い勝手を評価する視点と考えることができる．

コンポーネントの利用性を評価する場合，一般的には実際にコンポーネントを試用し，その際の使い勝手や利用に要した時間など外部特性を観察評価する形をまず考えることができる．しかし，コンポーネントの場合，提供形態は特定のソフトウェア製品として提供されるものではなく，ある特定の機能を実現するオブジェクトとして提供される．これらを実際に動作させる場合にはユーザによって異なった環境（プラットフォーム，システム）に組み込んで動作させることが考えられるが，ある特定の環境下でコンポーネントを動作させ評価した場合に，その評価結果が他の環境下でもそのままあてはまるとは限らない．また，この場合，実際にコンポーネントを試用する必要があり，評価コストがかかるなどの課題もある．

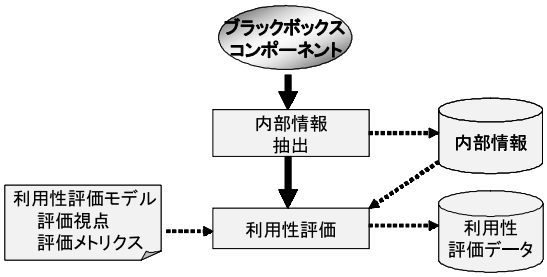

図 2 ブラックボックス・コンポーネントの評価方式
Fig. 2 Evaluation flow of black box component.

これに対し，コンポーネントの利用性を内部特性をもとに評価する方式では，上記のような課題は発生しない．しかし，一方で，コンポーネントの多くは，コンポーネント開発者側が持つソースコードや開発ドキュメントなどが十分には公開されないブラックボックス・コンポーネントであり，内部特性評価のための情報源が限られるといった別の課題が存在する．

2.2 ブラックボックス・コンポーネントの評価方式

ここで我々が提案するブラックボックス・コンポーネントの評価の流れを紹介する．**図 2** に示すようにまず，ブラックボックス・コンポーネントを入力として，内部情報抽出部によってブラックボックスの内部情報の抽出を行う．そして，抽出された内部情報に関して新たに定義したブラックボックス・コンポーネントの利用性評価モデルを適用して評価モデルの各視点ごとの評価ポイントを求め，対象コンポーネントの利用性評価を行う．

(a) 内部情報の抽出

内部情報抽出については，前節で指摘したようにブラックボックス・コンポーネントではソースコードの入手が困難なためソースコード解析などの手法は利用できない．しかし，本研究がモチーフとして選定したJavaBeans の場合，Java のリフレクション機構を用いてメモリ上のオブジェクトの内部情報を抽出することが可能である[7)~9)]．**図 3** は，SampleClass1 というオブジェクトに関して，本手法を用いて抽出したオブジェクトの内部情報の一部を示している．内部情報は図 2 に示すように構造を持った記録様式を用いて記録する．

なお，本論では JavaBeans をモチーフとしているため，リフレクション機構を利用した内部情報抽出が可能である．一方，Java 言語以外で作成されたブラックボックス・コンポーネントの中で比較的利用頻度が高い C/C++のライブラリ（DLL ファイル）や ActiveX コンポーネントなどについては，ユーザアプリケーションに組み込むために付随して提供されるヘッダファイルの解析やそれぞれの言語の開発環境の上でプロパティ情報を確認することによって，クラスやメソッドの情報を取得することが可能である．

```
<ip name="sample.ipml">      ←基礎情報を記録するファイル名
  <object name="SampleClass1" type="class" abstract="FALSE"
          virtual="TRUE" file="ClassA.class">
    <parent name="java.lang.Object" virtual="FALSE" access="public"/>  ←親クラス定義
    <mbval name="mbval1" access="public" type="int"
          const="TRUE" static="FALSE"/>  } メンバ変数の定義
    <mbfunc name="mbfunc1" virtual="TRUE" abstract="FALSE"
          access="public" type="void" static="FALSE">
      <val type="int"/>
      <val type="int"/>  } メソッドI/F のパラメータ定義
    </mbfunc>  } メソッドの定義
  </object>  } オブジェクトの定義
</ip>
```

図 3 オブジェクトの内部情報の抽出例
Fig. 3 Object internal information (example).

(b) 評価モデル

上記の方式で抽出されたブラックボックス・コンポーネントの内部情報について利用性の観点から評価を行う．特にここでは，一般のユーザが利用するコンポーネントを選定する際に，そのコンポーネントの使い勝手や導入・実装の容易さなどの視点を内部特性として評価するモデルを採用する（図 2）．

3. コンポーネント利用性評価モデル

3.1 利用性評価の視点

本研究ではコンポーネントの設計情報などの内部情報を解析し，コンポーネントの利用性を評価する．以下では，このための評価視点としてコンポーネントの利用性に関する内部特性をモデル化する．

ここではコンポーネント利用性に関する内部特性を導出するために，ISO/IEC9126 で定義された使用性に関する「習得性」「理解性」「運用性」などの副特性の視点やコンポーネントの利用や実装作業を考慮し，「導入時のコンポーネント利用方法の習得・把握が容易であること」「実際にソフトウェアの本体にコンポーネントを組み込む作業が容易であること」の 2 つの視点に着目する．本論では，以下これら 2 つの視点をコンポーネント利用性に関する内部特性として，それぞれ「導入時の学習容易性」「実装作業の容易性」と命名する．

3.2 導入時の学習容易性

(a) 視点の意味付け

一般にコンポーネントを導入する場合には，事前にそれぞれのコンポーネントがどのようなコンポーネン

表 1 利用性評価の視点とメトリクス
Table 1 Evaluation viewpoint and metrics for component usability.

導入時の学習容易性	
規模の妥当さ	クラス数
	クラス毎のメンバ数
クラス役割の明確さ	クラスの結合度
	カプセル化の強度
実装作業の容易性	
クラス特定容易性	メンバ数
	ネーミングの規則性
	クラスの結合度
	カプセル化の強度
クラス拡張容易性	継承可能クラスの割合
	継承不可能クラスの割合
クラス利用容易性	継承必要なクラスの割合

トであるかを把握しておく必要があり，そのための事前学習が必要となる.「導入時の学習容易性」では，こうした事前学習がどの程度やりやすいかを評価する狙いがある．そして「導入時の学習容易性」は，主として学習すべき内容の量と複雑さによってその度合いが大きく異なってくると考えることができる．

このうち，学習の量的な側面については，コンポーネントの規模的な側面に大きく依存していると考え「規模の妥当さ」という観点を用意する．また，学習内容の複雑さについては，対象となるコンポーネントを構成する個々のクラスがいかに簡潔で分かりやすいかといった側面から「クラス役割の明確さ」という観点を導入し評価を行うこととした．

(b) 評価メトリクス

「導入時の学習容易性」に関する上記 2 つの視点に関して**表 1** に示すような評価メトリクスを設定する．

規模の妥当さ

一般にコンポーネントはその規模は大きいもの小さいものなど様々である．また，コンポーネントが提供する機能の視点からは単機能コンポーネントもあれば，複数の機能を提供するコンポーネントも存在する．規模の大きなコンポーネントは多くの場合，複数の機能を提供する場合が多く，結果的に複数のクラスから構成される．このため，そのコンポーネントがどのようなコンポーネントであるかを理解しようとすると，クラスやクラスごとのメンバの数だけ，それぞれの役割を確認し学習する必要が生じる．このため,「規模の妥当さ」を評価するメトリクスとして，コンポーネントを構成するクラスの数，ルートクラス数やクラスごとのメンバ数などを計測する．これについては UML などを対象としてプログラムの設計段階での規模や複雑さを評価するためのメトリクスが Genero らによって提案されており[10]，これらも参考にして上記メトリクスを採用した．なお，Java 言語のようにすべてのクラスは Object から派生する場合には，この Object クラスを親クラスとするクラスをルートクラスとして考える．

クラス役割の明確さ

コンポーネントの利用者の目から見ると，複数クラスから構成される規模の大きなコンポーネントは，コンポーネントで実現する機能に関する役割が複数のクラスに分散したり，個々のクラスが複雑に関係しあって 1 つの機能をしたりするといった構成をとる場合が多くなる．このためにコンポーネントの全体構造が把握しにくくなり，実際にコンポーネントを利用する場合どのクラスを利用すればよいかなどが分かりにくいといった問題が生じ，導入時の学習に大きな影響を及ぼす．逆に規模が小さく，個々のクラスの役割などが把握しやすいコンポーネントは，導入時にそのコンポーネントのおおよその利用方法などが比較的短時間で学習できる．

このように「クラスの役割の明確さ」のポイントは，他のクラスとの関係や依存性ができるだけ低く，クラスが単体で適切に設計されていることが重要な要素となる．このため,「クラス役割の明確さ」についてはクラスの結合度やカプセル化の強さなどを計測する．

3.3 実装作業の容易性

(a) 視点の意味付け

一般にコンポーネントは単機能のものもあるが，複数の機能を提供するものも存在する．このため実際にコンポーネントを利用するにあたっては，コンポーネントが提供している機能の中から，真に利用したい機能を特定し，その機能を実装するクラスを特定する必要がある．このため「クラスの特定が容易であるかどうか」が重要な要素となる．また，実際のコンポーネントの利用法としては，コンポーネントに手を加えることなくそのまま利用する場合と，ユーザが継承などを利用しながら一部のクラスを拡張して利用する場合が考えられる．このため,「クラスを拡張することなく利用する場合の使い勝手」と「クラスを拡張して利用する場合の使い勝手」それぞれの観点を評価しなければならない．

(b) 評価メトリクス

実装作業の容易性に関する上記 2 つの視点に関して表 1 に示す以下のような評価メトリクスを採用する．

クラス特定容易性

コンポーネントで提供される個々のクラスがどのようなものであるかが特定しやすいかどうかを判断する．このため,「クラスがいくつのメンバから構成されるか」「クラスのネーミングに規則性があるか」「クラスの結合数」「クラスのカプセル化の強さ」などを計測する．

クラス拡張容易性

一般に Java などで扱われるクラスは private, public, protected などのアクセス指定を持つが，それらとは別にオーバライドを禁止するといった制約を持つものがある．そのようなクラスは拡張性が制限される．このためクラスの拡張容易性に関しては,「継承可能なクラスがどの程度含まれるか」とともに,「継承不可能なクラスがどの程度含まれているか」を計測することが重要である．

クラス利用容易性

ある機能を実行するためのクラス A を利用する場合，特定のクラス B を継承しなければ利用できない場合，クラス A は利用上の制約が課されている形となる．このようなクラスでは，必要なクラスを継承するといった形でコンポーネントに手を加えなければ利用できないためクラスの利用性に制約を受ける．

以上をもとに，表 1 にブラックボックス・コンポーネントの利用性評価に関する視点および評価メトリクスを示す．

4. 評価メトリクス基準値の設定

4.1 評価基準値設定の目的と方法

実際に提案した評価視点・評価メトリクスを用いてコンポーネント評価を行うには，各メトリクスについての評価基準値が必要となる．

ここでは評価基準値の設定方法として，実際に評価の定まったコンポーネントを取り上げ，そのコンポーネントに対し提案メトリクスで特性を計測し，その結果から各メトリクスの評価基準値を求める方式を採用する．今回はコンポーネントとして JavaBeans を対象とする．JavaBeans は Jars.com や jfind.com などのコンポーネント公開サイトで多くのコンポーネントが公開されており[11),12)]，一般ユーザの利用も進んでいる．またこれらのサイトでは，一般ユーザが実際に利用した際の印象などをもとにしたコンポーネント評価情報が公開されており，各コンポーネントに関する評価が定まっている．Jars.com ではコンポーネントは 9 種類のカテゴリに分類され公開されている．今回は特にソフトウェア部品として利用価値の高い Programming カテゴリの中から**表 2** に示す 24 個のコンポーネントを無作為に選び，評価基準値設定に利用した．

表 2 サンプル・コンポーネント
Table 2 Sample component.

No	Beans名	rating	No	Beans名	rating
1	VisualSoft JBProjectGantt	4.0	13	Folder Bean Suite	3.0
2	VisualSoft JBGrid	4.0	14	Masking Bean Suite	3.0
3	VisualSoft JBChart	4.0	15	Drag and Drop Bean Suite	3.0
4	JFCSuite	4.0	16	FtpBean	3.0
5	TextAreaPropertyEditor	4.0	17	BarCode	3.0
6	ExcelAccessor	4.0	18	Gauge Beans	3.0
7	Visual Soft JBColorPicker	4.0	19	ProgressBar Bean	3.0
8	VisualSoft JBCalender	4.0	20	INIFile Bean Suite	3.0
9	Jsuite	4.0	21	AButtonTree	3.0
10	Sock	4.0	22	EditList JavaBean	3.0
11	Scrooge	3.0	23	WiringHelpers Collection	0.5
12	JViewPro	3.0	24	Digit List Bean	0.5

4.2 評価基準値設定の流れ

今回の評価基準値設定では表 1 の評価観点の中で，特にコンポーネントの導入時学習の容易さについて「規模の妥当さ」を，また実装の容易さについて「クラス利用容易性」「クラス拡張容易性」に着目し，① クラス数，② ルートクラス数，③ 継承が必要なクラスの割合，④ 継承が可能なクラスの割合，⑤ 継承不可能なクラスの割合の 5 つのメトリクスの基準値を設定する．基準値の設定では，まず，これら 5 つのメトリクスを利用して，表 2 に示す 24 個のコンポーネントを計測する．そしてこれらの計測値の分布と Jars による評価を対比させることで，メトリクスの基準値を求めた．Jars では実際にコンポーネントを利用したユーザの主観により 0.5 点刻みで最高 4 点の評価値が示されている．今回，対象とした Jars の Programming カテゴリには 100 個のコンポーネントが登録されている．このうち評価点が公表されているコンポーネントは 84 個あった．これら 84 個のコンポーネントの評価点は，4.0 点が 27 個，3.5 点が 22 個，3.0 点が 19 個などとなっており，評価点は高得点側に偏った分布をしており，その平均点は 3.14 点となっていた．このため今回の適用評価では，この平均点を基準として 0.5 点刻みを考慮して，評価点 4.0 を High 評価，評価点 3.5 を Mid 評価，評価点 3.0 以下を Low 評価とした．さらに点数の刻み幅が 0.5 点と小さく優劣の評価が曖昧になるのを防ぐため，今回の評価では Mid 評価のコンポーネントはサンプルから除外し，残りのコンポーネントからランダムに 24 個のコンポーネントをサンプルとして選定した．この結果，表 2 に示すように，対象とした 24 個のコンポーネントのうち 10 個が High 評価のコンポーネント・グループ，14 個が Low 評価のコンポーネント・グループとなった．

4.3 評価基準値

今回，評価基準値の設定を試みた 5 つのメトリクスの中で，コンポーネント規模の適切さを表す ① クラス数，② ルートクラス数などは各コンポーネントごとに計測する．また，コンポーネントの実装時の拡張性を評価する，③ 継承が必要なクラスの割合，④ 継承が可能なクラスの割合，⑤ 継承不可能なクラスの割合については，コンポーネントが提供する各サブコンポーネントごとに拡張性が異なる可能性があるため，それぞれのサブコンポーネントごとに継承利用可否を計測することとした．なお，これらの計測結果は **表 3** (a)，(b) に示す．たとえば表 3 (b) の JFCsuite は 9 個のサブコンポーネントを持つため，それぞれのサブコンポーネントの継承に関するデータを計測している．

以下ではこれらの計測データをボックスプロットにより可視化し，評価基準値の検討を進める．ボックスプロットはヒストグラムを簡略化したもので，データの最大値，最小値を基準に小さい順にデータを並べた際の 1/4 番目に位置する値を第 1 四分位数，3/4 番目に位置する値を第 3 四分位数と呼ぶ（**図 4**）．

ボックスプロットでは，全体のはずれ値などを除外し，第 1 四分位数から第 3 四分位数に入る全体の半数のデータポイントをもとに管理限界値などを検討するのに適している．

(a) クラス数

クラス数が極端に多いコンポーネントは，利用目的に対してどのクラスを利用したらよいかが分かりにくく，導入時の学習が大変になるため，クラス数は極端に多すぎないほうがよいと考えられる．**図 5** (a) のボックスプロットにクラス数の分布を示す．

Score=High のグループでは，平均が 76.9，第 3 四分位数 119.0 となっており，この第 3 四分位数を参考にクラス数の基準値を 120 以下とする．

なお，Score=Low のグループでは平均が 30.9，第 3 四分位数は 42.5 となっており，上述のクラス数の基準値以下になっている．しかしながら，Score=Low のコンポーネントの場合，後述する継承の要否などクラス数以外の要因が大きく影響して利用性が低下していると考えられるため，今回のクラス数の基準値設定ではこの値は参考としていない．

また図 5 (a) の Score=Low のグループに見られるはずれ値に相当するコンポーネントは，多数のサブコンポーネントを内包する特殊なコンポーネントであるため，外れ値として処理している．

表 3 サンプルコンポーネントの評価データ
Table 3 Measured value for each component.
(a) コンポーネント規模に関するデータ

name	Score	クラス数	ルートクラス数
VisualSoft JBProjectGrantt	High	58	0
VisualSoft JBGrid	High	36	1
VisualSoft Jbchart	High	75	2
JFCSuite	High	260	45
TextAreaPropertyEditor	High	4	0
ExcelAccessor	High	82	2
VisualSoft JBColorPicker	High	5	0
VisualSoft JBCalender	High	13	1
Jsuite	High	230	41
Sock	High	6	0
Scrooge	Low	15	8
JviewPro	Low	18	9
FolderBeanSuite	Low	164	16
MaskingBeanSuite	Low	42	0
Drag and Drop Bean Suite	Low	86	14
FTPBean	Low	8	5
BarCode	Low	18	2
Gauge Beans	Low	4	0
ProgressBar Beans	Low	1	0
INIFile Bean Suite	Low	44	0
abuttonTree	Low	3	0
EditList JavaBean	Low	13	0
WiringHelpersCollection	Low	14	0
Digit List Bean	Low	3	0

(b) コンポーネントの実装作業容易性に関するデータ

Name	Score	継承が必要なクラスの比率	継承が可能なクラスの比率	継承不可能なクラスの割合
VisualSoft JBProjectGrantt	High	5.17	93.1	1.72
VisualSoft JBGrid	High	2.78	66.67	30.56
VisualSoft Jbchart	High	4	90.67	5.33
JFCSuite	High	4.12	95.88	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
TextAreaPropertyEditor	High	0	100	0
ExcelAccessor	High	25.64	74.36	0
	High	23.26	76.74	0
VisualSoft JBColorPicker	High	0	100	0
VisualSoft JBCalender	High	7.69	92.31	0
Jsuite	High	1.18	97.64	1.18
	High	0	84.21	15.79
	High	0	94.87	5.13
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
	High	0	100	0
Sock	High	16.68	83.33	0
Scorrge	Low	20	80	0
JviewPro	Low	5.56	94.44	0
FolderBeanSuite	Low	14.47	69.74	15.79
	Low	13.64	72.73	13.64
MaskingBeanSuite	Low	20	80	0
	Low	12.5	87.5	0
	Low	28.57	71.43	0
	Low	22.22	77.78	0
Drag and Drop Bean Suite	Low	15.38	82.05	2.56
	Low	14.89	82.98	2.13
FTPBean	Low	25	75	0
BarCode	Low	11.11	88.89	0
Gauge Beans	Low	0	100	0
ProgressBar Beans	Low	0	100	0
INIFile Bean Suite	Low	22.22	72.22	5.56
	Low	19.23	76.92	3.85
Abutton Tree	Low	33.33	66.67	0
EditList JavaBean	Low	0	100	0
WiringHelpersCollection	Low	14.29	85.71	0
Digit List Bean	Low	0	66.67	33.33

(b) ルートクラス数

コンポーネントの個々の詳細な機能を実現する単位

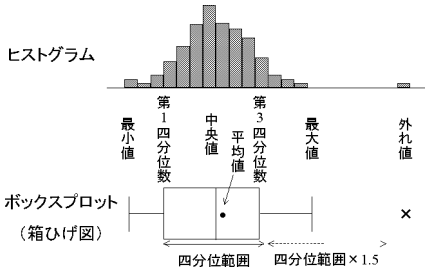

図 4 ヒストグラムとボックスプロット
Fig. 4 Histogram and boxplot.

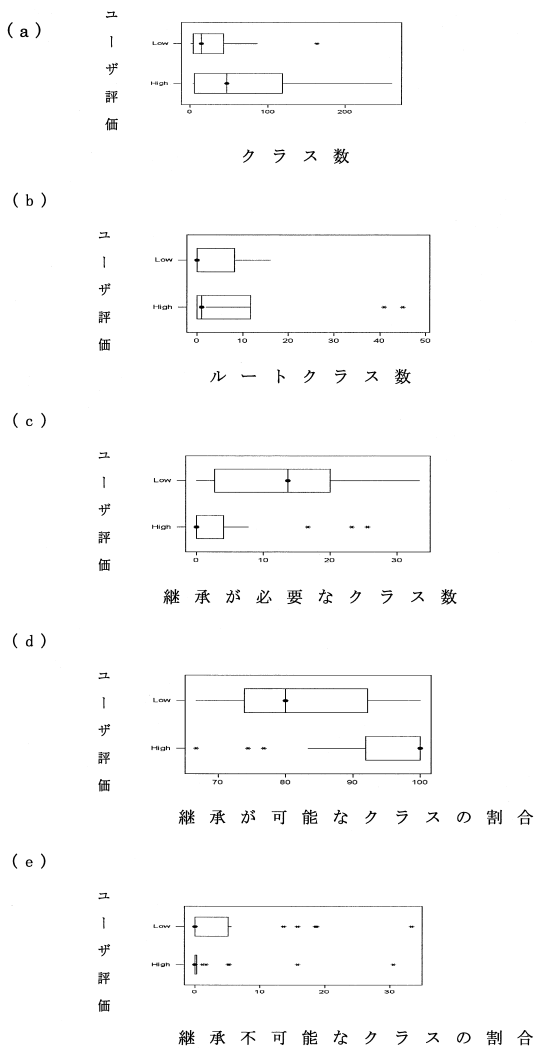

図 5 各メトリクス値とユーザ評価の関係
Fig. 5 Relation between metrics value and user evaluation.

がクラスであると考えると，ルートクラスはこれらをまとめた機能の大分類と解釈できる．ルートクラス数が極端に多いコンポーネントは，機能が必要以上に多すぎて，利用したい機能とクラスの対応関係などが分かりにくくなり，導入時の学習に影響を及ぼす．このため，ルートクラス数に関しても極端に多すぎないほうが良いと考えられる．

図 5 (b) にコンポーネントが持つルートクラス数の分布を示す．Score=High のコンポーネント・グループでは平均が 9.2，第 3 四分位数 11.75 となっており，この第 3 四分位数を参考に，ルートクラス数の基準値は 12 以下と設定する．

また Score=Low のコンポーネント・グループでは，平均が 3.86，第 3 四分位数 8.25 と，上記の基準値以下であったが，継承の要否などルートクラス数以外の要因によって低い評価（Score=Low）となっていると考えられ，今回の基準値設定ではクラス数と同様にこの値は参考としていない．

なお，今回モチーフとしている Java 言語では，すべてのクラスは Object から派生するが，ここでは一般的なオブジェクト指向言語との対応を明確にするために，Object クラスを親クラスとするクラスをルートクラスとして扱い計測している．

(c) 継承が必要なクラスの割合

コンポーネントを利用する際に，他のコンポーネントからの継承が必要とされる場合，そのコンポーネントの利用性は大きく制限をうける．図 5 (c) は継承が必要なクラスの割合を示したものである．Score=High のコンポーネント・グループと Score=Low のコンポーネント・グループの間ではその分布に有意差が認められ，95%信頼区間により，Score=High のコンポーネントでは継承が必要なクラスの割合が 6%以下になっていることが確認された．これより，継承が必要なクラスの割合は 6%以下を基準値として採用する．

(d) 継承が可能なクラスの割合

コンポーネントの利用法として，コンポーネントで定義されているクラスを継承し，新たな機能を付け加えるなどの方法が考えられる．これはコンポーネントの拡張性に関する要因である．図 5 (d) に継承が可能なクラスの割合を示す．Score=High と Score=Low の間に有意差が認められ，分布の 95%信頼区間で，Score=High のコンポーネントは継承可能なクラスを 90%以上持つ傾向が確認できた．一方，Score=Low のコンポーネント・グループは継承可能なクラスの割合が低くコンポーネントの拡張性に問題があるものが多かった．より多くのクラスで継承可能なほうがコンポーネントとしての拡張性が高いことを考え，第 1 四分位数 91.9%を参考に，継承可能なクラスの割合については 90%以上を基準値として採用する．

表 4 評価メトリクスの基準値
Table 4 Evaluation criterion of each metrics.

メトリクス	評価が高いコンポーネントのメトリクス値の適正範囲
クラス数	120個以下
ルートクラス数	12個以下
継承が必要な	コンポーネントをそのまま利用するとき
クラスの割合	6%以下
継承が可能な	コンポーネントを拡張して利用するとき
クラスの割合	90 %以上
継承不可能な	コンポーネントを拡張して利用するとき
クラスの割合	3%以下

(e) 継承不可能なクラスの割合

コンポーネントの拡張性という観点では，継承不可能なクラスの割合は低いほうが良い．図 5 (e) に継承不可能なクラスの割合を示す．Score=High のコンポーネント・グループでは平均値は 2.3%となった．一方，Score=Low のコンポーネント・グループでは平均 4.9%となっており，継承不可能なクラスの割合が高いことが確認できた．これより，Score=high のコンポーネントクラスを参考に，継承不可能なクラスの割合の基準値は 3% 以下とする．

上記，(a)〜(e) をもとに各メトリクスの評価基準値を整理したものを**表 4** に示す．

5. 適 用 評 価

5.1 適用評価の目的

ここでは前章で提案したコンポーネントの利用性評価の方式の有効性を確認することを目的とした適用評価について紹介する．

適用評価では，提案した評価メトリクスを利用して，実際のブラックボックス・コンポーネントの使い勝手の評価を行う．また一方で，これらのコンポーネントを実際のシステム開発で利用し，その際の使い勝手に関する評価をユーザに求め，メトリクスによる評価結果との一致度合いを対比した．これに基づき，提案した評価視点，評価メトリクスの有効性について考察する．

5.2 適用評価の方法

(a) 適用評価の手順

適用評価は Step-1：評価メトリクスを利用した定量評価，Step-2：実際にコンポーネントを利用した際のインプレッション評価による 2 段階で行った．

Step-1：評価メトリクスによる定量評価

まず，対象となる 2 つのブラックボックス・コンポーネント（JB1，JB2）に対し，本論で提案する評価視点/評価メトリクスを利用し，使い勝手を定量的に評価する．

表 5 被験者のスキルプロファイル
Table 5 Skill profile of each engineer.

被験者	ランク	開発経験年数	プログラミングスキル
A	上級者	5年	規模の大きなシステム全体を取りまとめる能力あり Java言語は自由に扱える
B	中級者	3年	中規模のサブシステムの開発を一人でできる Java言語はほぼ自由に扱える
C	初級者	1年	比較的簡単なクラス群の開発ができる Java言語の利用に多少不安がある
D	初級者	1年	比較的簡単なクラス群の開発ができる Java言語の利用に多少不安がある

Step-2：コンポーネント利用時のインプレッション評価

ソフトウェア開発に従事する 4 人のプログラマを被験者として，対象とする 2 つのコンポーネントを実際のアプリケーション開発の中で使ってもらい，その際の使い勝手に関する印象をインタビュー形式で収集した．

インタビューでは，

- コンポーネントが提供する機能の中で，どの機能，どのクラスやメソッドを利用したか
- 利用方法の理解のしやすさや，利用した際の手間

などを中心に 3 段階で定性的な印象を調査した．

なお，4 人の被験者のスキルレベルは被験者 A：上級スキル，被験者 B：中級スキル，被験者 C，D：初級スキルとなっている．これらの被験者のプロファイルを**表 5** に示す．また，開発したアプリケーションは下記の 4 通りである．

アプリ 1 ：JB1 の機能をそのまま利用して実現できる小規模なアプリケーション

アプリ 1' ：JB1 の機能を拡張して利用しなければ実現できない小規模なアプリケーション

アプリ 2 ：JB2 の機能をそのまま利用して実現できる小規模なアプリケーション

アプリ 2' ：JB2 の機能を拡張して利用しなければ実現できない小規模なアプリケーション

(b) 対象コンポーネント

今回の適用では以下のような機能を提供する 2 つのコンポーネントを対象として適用評価を実施した．

JB1：

- 数字，アルファベットによる文字列をバーコードに変換し表示する．

JBarcodeBean (DW22-1)

JBarcodeBean is a JFC Swing-compatible JavaBeans component that lets you barcode-enable your Java 2 enterprise applications. The bean can be used either as a visible client-side component that renders itself graphically, or as a server-side component that can encode a GIF image into an OutputStream. A Servlet could use this feature, for example, to serve barcodes to a web browser.

Submitted by :	Dafydd Walters, dwalters@dragontechnology.com
Resource URL :	http://www.dragontechnology.com/barcode
Source Code :	Not Available
Detailed Review :	Not Available
Type :	Other
Category :	Programming - GUI
Language :	JavaBeans
Marketing :	Commercial - Release
History :	New Submission

JBarcodeBean

Barcode JavaBeans Component

Introduction

Welcome to the homepage of the JBarcodeBean project.

JBarcodeBean is a JFC Swing-compatible JavaBeans component that lets you barcode-enable Java 2 enterprise applications.

The bean can be used either as a visible client-side component that renders itself graphically, or as a server-side component that can encode a GIF image into an **OutputStream**. A **Servlet** could use this feature, for example, to serve barcodes to a web browser. JBarcodeBean is written entirely in pure Java, so it should work under any operating system that supports Java 2 (version 1.2.2 or later).

The following popular barcode formats are currently supported by JBarcodeBean:

- Code 128
- Code 39
- Extended Code 39
- Codabar
- Interleaved Code 25
- MSI

JBarcodeBean is licensed under the terms of the GNU Lesser General Public License (LGPL), version 2.1. See the GNU web site for more details.

Download

JBarcodeBean is hosted at SourceForge. You can download the latest version from SourceForge here.

Code Examples

Click here to see a couple of examples which illustrate how you can integrate JBarcodeBean into your own application.

UIControls Bean Suite (VN4-7)

The UIControls Bean Suite includes a wide variety of beans and classes that extend and improve the behavior of many commonly-used AWT controls. UIControls can be used instead of or in combination with standard AWT components to easily build applications in visual programming environments.

Submitted by :	IBM Corporation, alphbean@us.ibm.com
Resource URL :	http://www.reg.net/product.asp?ID=6169
Source Code :	Not Available
Detailed Review :	Not Available
Type :	Other
Category :	Programming - Widgets
Language :	JavaBeans
Marketing :	Commercial - Release
History :	New Submission

UIControls V 2.00

Installation Instructions

This portion of the document provides the following information:

- *Hardware Requirements*
- *Software Requirements*
- *Installation*
- *Platform Dependencies*

Hardware Requirements

These JavaBeans are designed to be compliant with 100% Pure Java requirements and therefore can serve on any hardware platform that supports the Java Environment.

The *uicontrols.jar* file takes about 615KB of disk space. This includes all customization classes as well.

Extra hard disk space is required to install the Java Development Kit, Bean Development Kit, or any Java Application Builder Tool which are used for building/compiling applications while using these beans.

Software Requirements

The UIControls beans completely correspond to the JavaBeans specification and have been successfully tested by IBM VisualAge for Java. They can be also used in any other builder tools supporting JavaBeans after taking in account their restrictions. For example, it is possible to connect any event only with a non-void method with 0-1 parameters visually (VC feature) in Symantec Visual Cafe.

UIControls Library User's Guide

Table of Contents

- Introduction
  - The Purpose of This Component
  - Who Should Read This Guide
  - How to Use This Guide
  - Copyrights and Trademarks
- Description
  - The UIControls Advantages
    - Common Look
    - More Powerful, Useful, and Stronger than JDK AWT
  - Bean Overview
- Installation Instructions
  - Hardware Requirements
  - Software Requirements
  - Installation
  - Platform Dependencies
- Usage Instructions
  - Deployment Scenario
  - Bean Usage
  - Wiring Scenario
  - Serialization
- User's Interface
  - Dialogs
    - Message Boxes
    - Find Dialogs
      - Simple Style
      - Extended Style 1

Bean Usage

The Jar file of the UIControls Library should be added to the IDE by means of a standard installation procedure for the given IDE. For example, see the *Installation* part of this document.

The wiring scenarios are described below.

Wiring Scenario

To realize a simple graphical editor described in the *Deployment Scenario* part, let the user perform the following steps:

1. Import *uicontrols.jar* into IBM Visual Age for Java.
2. Import special demo DrawCanvas bean and DrawCanvasBeanInfo from the demo\demo2 subdirectory into IBM VisualAge for Java.
3. Create a new DemoFr project/class inheriting BaseFrame, open it visually.
4. Add ToolBar to the north.
5. Add DrawCanvas to the center.
6. Add ColorChoosePanel to the south.
7. Add 3 ToggleToolButtons to ToolBar.
8. Add non-visual beans: ToggleToolButtonGroup, ToolTip, QuitDialogBean, KeyAdapter, PasswordDialogBean, Application

図 6 評価対象コンポーネントのカタログ情報

Fig. 6 Catalog spec. of the target component.

- 自分で用意した形式でバーコードに変換する．

JB2：

- ディレクトリを視覚化し，ディレクトリの追加，消去，変更ができるようにする．

- テキストファイルを指定することで，エディタを開くことができるようにする．

図 6 に JB1，JB2 の Jars 掲載情報およびそれぞれのコンポーネントのホームページ掲載の利用情報を示す．また，これらのコンポーネントに付属して提供される README ファイルなどにも，コンパイル方法と実行方法程度の情報しか掲載されておらず，利用性に関する情報の事前取得は難しい状況となっていた．

コンポーネント・ソフトウェアに関しても，一般のソフトウェアと同様に利用性（使用性）以外にも機能性，信頼性，保守性，移植性，効率性などの側面が考えられる．今回の適用では利用性を評価することを目的とするため，他の側面に関してはそろえておく必要があるが，今回はコンポーネントを組み込む作業の過程で，被験者にコンポーネントの移植や保守などに関する作業は行わせていないため，これら 2 つの特性は評価に影響しないと考えられる．また，効率性については，今回，コンポーネントを利用して開発する 4 つのアプリケーションでは性能その他の効率性に関する要件はきわめてゆるくしており，コンポーネントの効率性についても考慮しなくてもよいように工夫した．また，信頼性については，事前の調査・検討で実装する 4 つのアプリケーションを実現する範囲内で，対象としたコンポーネントの機能については特に不具合などのない範囲で利用すれば実現が可能なことを確認しており，信頼性要因も今回適用評価には影響しないよう配慮している．また，今回提案した「導入時の学習容易性」に関するメトリクスの 1 つとして，コンポーネントの規模を計測しているが，このメトリクスは機能性の評価のメトリクスとしても利用できる．今回設定した評価視点のうち「導入時の学習の容易さ」については，コンポーネントが提供する機能の量と深く関係してくると考えられるため，今回のサンプル JB1，JB2 に関する機能量の差異についてはあえてそろえるといったことはせず，機能量の影響も加味して学習容易性の評価を行った．

5.3 評 価 結 果

5.3.1 定量評価結果

表 6 は提案するメトリクスを利用して，JB1，JB2 それぞれのコンポーネントを計測した結果を示したものである．なお，表中でハッチングをかけた数字は，前述の各メトリクスの評価基準値（表 4）と照らし合わせて，基準値オーバであることを示している．

(1) 導入時の学習容易性

導入時の学習容易性の観点では，クラス数を見ると JB1 は基準値内におさまっているのに対し，JB2 は 170 と基準値を上回っており，JB2 のほうが導入時の学習容易性が劣っていると推定できる．

表 6 コンポーネントの評価結果（定量評価）
Table 6 Component evaluation results.

	JB1	JB2	基準値
クラス数	27	170	＜120
ルートクラス数	5	3	＜12
継承が必要なクラスの割合	14.8%	4.1%	＜6%
継承が可能なクラスの割合	85.2%	88.2%	90%＜
継承が不可能なクラスの割合	0.0%	7.7%	＜3%

表 7 定量評価結果のまとめ
Table 7 Evaluation summary (Quantitative).

		JB1	JB2
導入時のわかりやすさ	クラス数	適	大
実装時の使い勝手			
そのまま利用	継承必要なクラス割合	大	適
拡張利用	継承可能なクラス割合	やや少	やや少
	継承不可能なクラス割合	適	大

(2) 実装時の使い勝手

コンポーネントをそのまま利用する場合，JB1 は継承が必要なクラスの割合が高く利用しにくいと考えられる．

また，コンポーネントを拡張して利用する場合，JB1，JB2 とも継承可能なクラスの割合が 80%台で基準値を若干下回っており，拡張性に難があることが分かる．

また，継承不可能なクラスについても，JB2 は 7.7% となっており，JB2 は拡張性が特に乏しいと判断できる．

これらの結果をもとに定量評価結果をまとめると **表 7** のようになる．

5.3.2 定性評価結果

JB1，JB22 つのコンポーネントを実際に使った際の被験者のインタビューではそれぞれのコンポーネントに関して，次のような評価が得られた．

(a) コンポーネントの導入時の分かりやすさ

- JB1 は適用される機能が適度なため分かりやすい．また機能分割や継承関係も適切で分かりやすい．
- JB2 は提供している機能が多目的で，様々なことができる反面，どの機能をどのクラスで実現しているかが分かりにくい．

(b) 実装の際の使い勝手

コンポーネントを実際に利用したアプリケーション実装作業で得られた印象を**表 8** に示す．この表を見ると，

- JB1，JB2 ともにコンポーネントをそのまま利用する分には比較的使いやすいこと，
- 拡張して利用する場合には，どちらも初心者にとっては利用しにくく，特に JB2 でその傾向が

表 8 コンポーネントの評価結果（定性評価）
Table 8 Evaluation summary (Qualitative).

		被験者A（上級者）	被験者B（中級者）	被験者C（初級者）	被験者D（初級者）
JB1	そのまま利用	◎	○	◎	○
	拡張して利用	○	○	X	X
JB2	そのまま利用	◎	○	○	◎
	拡張して利用	○	△	X	X

◎ 利用しやすかった ○ どちらかというと利用しやすかった
△ どちらかというと利用しにくかった × 利用しにくかった

顕著であること，
などが分かる．

6. 考　　察

ここでは，適用評価の Step-1 で実施した定量評価結果と Step-2 の定性評価結果の対比をもとに，本研究で設定した評価視点ならびに評価メトリクスの有意性について議論する．

6.1 導入時の学習容易性

(1) 評価結果の比較

定性評価では，導入時の学習容易性（分かりやすさ）は JB1 が高く，JB2 は若干劣るという結果になった．これに対し，定量評価におけるクラス数でも，JB2 は基準値を超えていて問題があることが分かり，定性評価の結果を支持する傾向を示していることが分かった．

(2) 評価視点およびメトリクスの適切さ

評価結果の比較より,「導入時の学習容易性」という視点に関して，コンポーネントが提供する機能規模を表現するメトリクスである「クラス数」を利用して評価した結果とコンポーネント利用者が導入時にコンポーネントの理解などを行った際の分かりやすさに関する定性的な評価は一致することが確認できた．この結果からは「導入時の学習容易性」は，コンポーネントが提供する機能の数に大きく影響され，コンポーネントを構成するクラス数によって，評価できると考えられる．

6.2 実装時の使い勝手

(1) 評価結果の比較

コンポーネントをそのまま利用する場合

定性評価では，コンポーネントをそのまま利用する場合には，JB1，JB2 とも比較的使いやすいとの評価を受けている．これに対し，定量評価では JB1 は利用時に継承を必要とするクラスの割合が基準値を超えており，数値のうえでは使いにくいとの評価を受けており，定性評価の結果とは食い違いを見せている．この原因としては，適用評価で実際に利用したクラスは，比較的簡単なものであったために，どのようなクラスを継承すべきかも容易に判断でき，クラスの継承必要性が開発者にそれほどの負担とならなかったと考えられる．

コンポーネントを拡張利用する場合

一方，コンポーネントを拡張利用する場合には，定性評価では JB1，JB2 とも初心者には拡張して利用するのは難しいという評価が与えられた．これは定量評価において，どちらのコンポーネントも継承可能なクラスの割合が若干低いことと符合している．

また，JB2 は中級者にとっても拡張利用がやや難しいとの評価が与えられたが，これは定量評価結果で JB2 は継承不可能なクラスの比率が若干高くなっていることが影響していることも考えられる．

(2) 評価視点および評価メトリクスの適切さ

実際にコンポーネントを利用する場合には，コンポーネントが提供する機能の中の一部を利用することが考えられ，その部分の印象によってユーザの評価が大きくぶれる傾向があると考えられる.「コンポーネントをそのまま利用する場合」で定性評価結果と定量評価結果が食い違った背景にはこのような状況も想定できる．このため，より評価の精度を上げるためには，ユーザが利用するであろうクラスなどに絞って，さらに詳細レベルでの実装容易性の計測を行っていく方式が有効と考えられる．

また「コンポーネントを拡張利用する場合」については,「継承可能なクラスの割合」「継承不可能なクラスの割合」を計測・評価することで，おおよその傾向をつかむことができるとの結果が得られた．

6.3 ユーザスキルの影響

今回の適用評価ではスキルレベルの異なる複数の技術者を被験者とした．この結果，表 8 に見られるように，同じコンポーネントであっても技術者のスキルレベルによって,「拡張して利用しやすい」という評価と「拡張して利用しにくい」という評価に分かれたコンポーネントがあった．これはコンポーネントの利用に関して，特に機能の継承などを行う場合，ユーザのスキルレベルが大きく影響することを示しているといえる．利用性によって評価するコンポーネントの使い勝手は，コンポーネントユーザのスキルレベルとコンポーネント開発者が想定しているユーザスキルレベルとの相対的な差異によって評価が大きく分かれ，その間でギャップがある場合には，ユーザはそのコンポーネントの使い勝手に不満を持つことが考えられる．

これらを考慮するとユーザのスキルレベルを考慮して，初心者の場合の基準値と中上級者に対する基準値を変えて評価するといった方式が有効になると考えられる．**図 7** は今回の利用性に関するメトリクスのう

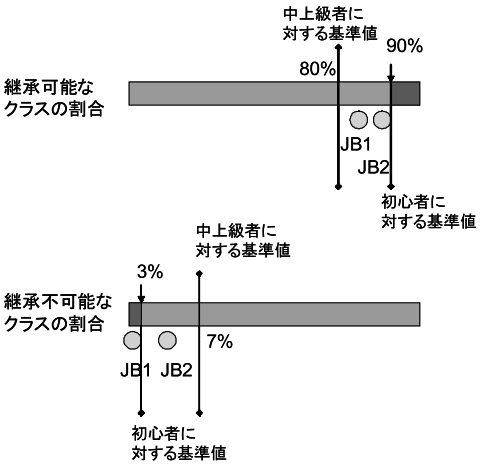

図 7 ユーザスキルレベルに応じた基準値設定方式
Fig. 7 Criterion setting according to the user's skill level.

ち,「コンポーネントを拡張して利用する」という比較的高度なユーザスキルを求められる利用シーンに関わるメトリクスについて，ユーザスキルを考慮した基準値設定を試みたものである．

今回の例では「コンポーネントを拡張して利用」する場合に，継承可能なクラス数の基準値を一律に 90%以上，継承不可能なクラスの割合の基準値を 3%以下と設定した．しかし，上級者の場合は継承可能なクラスの割合は 80%台，継承不可能なクラスの割合が 7%台でも，拡張利用性についてはまずまずの評価が得られている．これを考慮すると，上級者と初心者では評価メトリクスの基準値を図 7 に示したように若干変えて評価することで個々の技術者のレベルを考慮したうえでのそれぞれのコンポーネントの拡張利用性が評価できると考えられる．

7. 関 連 研 究

ここでは本研究に関連する研究を紹介する．従来よりソフトウェアの部品化に関しては様々な研究が進められてきた[13),14)]．ここでは，これらの中から特に本論と関連の深いコンポーネントの品質評価手法を中心に従来手法を紹介する．ソフトウェアの品質については，Boehm らによりソフトウェア品質特性モデルが提案され[15),16)]，後に ISO/IEC9126 として国際標準に採択されて広く認知されている[17)]．ソフトウェア品質評価モデルでは，機能性，信頼性，保守性，移植性，使用性，効率性の 6 つの視点が提唱されている．これらは主としてソフトウェアの製品としての側面に焦点を当てている．

ISO/IEC9126 ではソフトウェア品質を評価するための主特性，副特性を規程しているが，これらの特性を計測する具体的なメトリクスに関してまでは言及していない．ソフトウェアメトリクスは，通常，計測対象物に応じて使い分ける必要があり，一意に定めることは難しい[19)]．

一方，オブジェクト指向に着目したメトリクスとして，Chidamber らが CK メトリクスを提唱している[19),20)]．CK メトリクスは，オブジェクト指向により開発されるソフトウェアの複雑さを計測することを目的としており，クラスのメソッド数，サブクラス数，クラス階層の深さ，クラス間結合度，クラスの反応度，メソッドの凝集度の 6 つのメトリクスが提案されている．CK メトリクスはソフトウェアの複雑さに焦点を当てており，品質特性における信頼性の視点からの評価に適したメトリクスである[21),22)]．また，CK メトリクス以外でも様々なメトリクスが提案されている．Brandi らはオブジェクト指向プログラムの複雑さに着目したメトリクスを利用して保守性の評価を試みており[23)]，Brinkley らはオブジェクト指向の依存性に着目し，保守性などを評価する方法を提案している[24)]．また，Rosenberg らも信頼性に関するメトリクスについて議論している[25)]．また，山崎らはオブジェクト指向プログラムの理解性に関して，クラスの定量化のみではなくインスタンスまで考慮した定量評価が重要との考え方を提示している[26)]．このほかにも，オブジェクト指向のメトリクスを利用した計測については様々な事例などが報告されている[27)~32)]．しかしながら，これらの多くはソースコードを計測するメトリクスであり，ソースコードが公開されていないブラックボックス・コンポーネントの計測には適していない．

一方，山本らはコンポーネントの再利用という側面に着目し，ブラックボックス・コンポーネントの再利用性を計測する手法を提案している[7)]．この研究ではコンポーネントの外部から観察される情報をもとに構造的な再利用のしやすさを計測することを狙っている．さらに鷲崎らは要求に合致するコンポーネントの検索を効率的に行うために，コンポーネントの構造・粒度・振舞いに着目した有効置換性類似度を提案した[33)]．山本，鷲崎らの研究によりコンポーネント利用の本質である再利用という側面からの評価手法は大きな前進をみたものの,「数多くあるコンポーネントの中から使い勝手の良いコンポーネントを選択する」という行為に関しての指針はいまだ十分に与えられていない．

コンポーネントを使う場合には，対象となるコンポーネントの機能や振舞いを正確に理解し，使い勝手の良いコンポーネントを選択することが必須である．

このため，本論ではコンポーネントを選択する際に重要なコンポーネントの利用性（使い勝手）に焦点を当て，その評価方式の提案を行った．

8. まとめ

本論ではソースコードなどが提供されないブラックボックス・コンポーネントの使い勝手を評価するため，下記のような特徴を持つ方法を提案した．

(1) ブラックボックス・コンポーネントの使い勝手を評価するため，JavaBeansを対象にリフレクション機能を利用した評価情報抽出を行う．

(2) コンポーネントの使い勝手を利用前に評価する視点として「導入時の学習容易性」「実装時の使い勝手」の2つの視点を提案した．特に後者は「コンポーネントをそのまま利用する場合」「コンポーネントを拡張利用する場合」に分けて評価する．

(3) 上記の視点を計測評価するために，コンポーネントを実現するクラスに着目し，クラス数や継承の可否などに関する割合を評価メトリクスとして設定し，計測する．

また，実際のコンポーネントにこれらを利用して，評価視点・評価メトリクスによる定量評価結果と実際のコンポーネントの利用者による定性評価結果の比較を行い，提案方法に関する評価・考察を行った．これより，本手法を用いることで，ブラックボックス・コンポーネントを実際に利用する前に，その使い勝手などを定量的に評価できることが確認できた．

また，一部のメトリクスについてはユーザスキルを考慮した段階的な基準値設定が有効になることを提案した．

今回の適用評価では限られた範囲のコンポーネントを対象にしている．このため適用評価の結果は，提案手法の有用性を示唆しているが，本提案手法の精度を向上させるために，今後，引き続き他のカテゴリの多くのコンポーネントを対象に本手法を適用し評価を継続していきたい．

参考文献

1) Port, D. and McArthur, M.: A Study of Productivity and Efficiency for Object-Oriented Methods and Languages, *Proc. APSEC*, pp.128–135 (1999).
2) Aoyama, M.: Component-based software engineering: can it change the way of software development?, *Proc. 20th ICSE*, Vol.2, pp.24–27 (1998).
3) Warford, J.S.: BlackBox: a new object-oriented framewaork for CS1/CS2, *Proc. 13th SIGCSE Technical Symposium on Computer Science Education*, pp.271–275 (1999).
4) Jubin, H.: *JavaBeans by Example*, Prentice Hall Computer books (1998).
5) Mansfield, R.: *DiscoverActivex*, Hungry Minds Inc. (C) (1997).
6) Forest, L.: *Activex and the Internet*, Scott Jones (1998).
7) 山本ほか：静的側面からみたソフトウェアコンポーネントの品質，信学技報，Vol.101, No.98 (2001).
8) Staugaard, A.: *InformationSystems Programming With Java*, Prentice Hall (2003).
9) Nickerson, D.: *Component Programming with JavaBeans*, Ventana Pr (1997).
10) Genero, M., et al.: Empirical Validation of Class Diagram Metrics, *Proc. ISESE'02*, pp.195–203 (2002).
11) http://www.jars.com/
12) http://www.jfind.com/
13) Aoyama, M.: Componentware and composable software architecture, *Proc. 23th Compsac*, p.597 (1998).
14) Pree, W.: Component-based software development — a new paradigm in software engineering?, *Journal of Software*, Vol.18, No.4, pp.169–174 (1997).
15) Boehm, B.W., et al.: Quantitative Evaluation of software Quality, *Proc. 2nd ICSE*, pp.592–605, IEEE (1976).
16) Boehm, B.W.: Software Engineering Economics, Prentice Hall (1981).
17) ISO/IEC9126, Information Technology-Software product evaluation — Quality characteristics and guidelines.
18) Hirayama, M., et al.: Practice of software measurement, *Proc. 11th ICSE* (1989).
19) Chidamber, S. and Kemerer, C.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.* (1994).
20) Chidamber, S.R. and Kemerer, C.F.: Towards a Metrics Suite for Object Oriented Design, *Proc. OOPSLA'91*, Vol.26, No.11, pp.197–211 (1991).
21) Chidamber, S.R., Darcy, D.P. and Kemerer, C.F.: Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis, *IEEE Trans. Softw. Eng.*, Vol.24, pp.629–639 (1998).
22) Subramanyam, R. and Krishnan, M.S.: Emprical Analysis of CK metrics for Object-Oriented Design Complexity: Implicatoions for

Software Defect, *IEEE Trans. Softw. Eng.*, Vol.29, No.4 (2003).
23) Brandi, R.K., et al.: Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics, *IEEE Trans. Softw. Eng.*, Vol.29, No.1, pp.77–87 (2003).
24) Brinkley, A. and Schach, S.: Validation of the Coupling Dependency Metric as a Predictor of Run-time Failures and Maintenance Measures, *Proc. 20th ICSE*, pp.452–455 (1998).
25) Rosenberg, L., Stapko, R. and Gallo, A.: Object-Oriented Metrics for Reliability, *IEEE Int'l Symp. Software Metrics* (1999).
26) 山崎直子，掛下哲郎：性的なオブジェクト指向プログラムに対するインスタンスを考慮した理解コスト計算法，コンピュータソフトウェア，Vol.20, No.4, pp.11–24 (2003).
27) Bansiya, J. and Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE Trans. Softw. Eng.*, Vol.28, No.1, pp.4–17 (2002).
28) Abreu, B.F. and Melo, W.L.: Evaluating the impact of Object Oriented Design on Software Quality, *Proc. 3rd Int'l Software Metrics Symp.* (1996).
29) Bansiya, J. and Davis, C.: Class Cohesion Metric For Object Oriented Designs, *Journal of Object-Oriented Programming*, Vol.11, No.8, pp.47–52 (1999).
30) Basili, V.R., Briand, L. and Melo, W.: A Validation of Object Oriented Design Metrics as Quality Indicators, *IEEE Trans. Softw. Eng.*, Vol.22, pp.751–761 (1996).
31) Briand, L.C., Wuest, J., Daly, J.W. and Porter, D.V.: Exploring the Relationship between Design Measures and Software Quaility in Object Oriented Systems, *Journal of Systems and software*, Vol.51, No.3, pp.245–273 (2000).
32) Briand, L.C., Wuest, J., Ikonomovski, S. and Lounis, H.: Investigation of Quality Factors in Object-oriented Designs: An Industrial Case Study, *Proc. ICSE*, pp.345–354 (1999).
33) 鷲崎弘宣，深沢良彰：有効置換性類似度に基づくコンポーネント検索方式実現と評価，情報処理学会論文誌，Vol.43, No.6, pp.1638–1652 (2002).

(平成 15 年 10 月 14 日受付)
(平成 16 年 4 月 5 日採録)

平山　雅之（正会員）

1960 年生．1986 年早稲田大学大学院理工学研究科博士前期課程修了．2003 年大阪大学大学院基礎工学研究科博士後期課程修了．1986 年（株）東芝入社．ソフトウェア品質技術の研究開発に従事．現在,（株）東芝ソフトウェア技術センター参事，情報処理学会ソフトウェア工学研究会運営委員．工学博士（大阪大学）．

佐藤　　誠（正会員）

1974 年生．1999 年宇都宮大学工学研究科情報工学専攻博士前期課程修了．同年（株）東芝研究開発センター入社．オブジェクト指向言語によるシステム開発に興味を持つ．ソフトウェア IP の品質評価手法についての研究を経て，現在は株式会社東芝セミコンダクター社にて車載用システムプラットフォーム開発に従事．