プログラムソースコードのための実用的な品質評価枠組み

鷲崎弘宜[†] 波木理恵子[††] 福岡呂之[††]
原田陽子[††] 渡辺博之[††]

プログラムソースコードの品質はソフトウェアシステムの開発・保守コストや性能に影響するため，その品質を高い精度で測定/評価する技術が必要である．これまでに種々の品質測定法が提案されているが，網羅性/総合評価能力を欠くといった問題を持ち，十分に活用されていない．我々は，直接にはC言語によるソースコードを対象とし，その静的な解析に基づいて効率良く内部品質を測定し評価する実用的枠組みを提案する．枠組みは，網羅的な品質メトリクススイート，測定値を正規化し全体から部分まで任意のモジュール単位で評価可能とする集計ツール，評価結果の可視化ツール，評定水準導出ツール，および，導出された具体的な評定水準より構成され，従来の取り組みがかかえる問題を解決あるいは改善する．C言語以外のプログラミング言語に基づくソースコードについても，枠組みを構成するメトリクススイートを部分的に再利用できる．現実の組み込みプログラム集合への適用実験により，ソースコードの特に信頼性，保守性，再利用性および移植性の定量的評価について枠組みを有効利用できることを確認した．

A Practical Framework for Quality Evaluation of Program Source Code

Hironori Washizaki,[†] Rieko Namiki,[††] Tomoyuki Fukuoka,[††]
Yoko Harada[††] and Hiroyuki Watanabe[††]

It is necessary to measure the quality of program source code because that affects the final entire system's performance and development/maintenance costs. However, conventional source code metrics have not been well utilized due to the lack of capability of evaluating entire quality and the lack of coverage on quality characteristics. To overcome these issues, we propose a framework for evaluating and measuring internal quality of program source code in mainly C programming language environments. In the framework, all of the measurements are conducted based on static analysis of source code by using existing metrics tools. As a result of evaluation experiments, it is found that our suite can be used to effectively and quantitatively evaluate source code from the viewpoint of reliability, maintainability, reusability and portability.

1. はじめに

組み込みからエンタープライズに至る社会の隅々までソフトウェアシステムによって制御され価値がもたらされる今日において，信頼性に代表される種々の品質特性をできるだけ高い精度と客観性をともなって早期に測定/評価し，該当システムの改善/保守や次のシステム開発へと役立てる技術体系の必要性が増大している．ソフトウェアシステムの品質には，利用時の品質，搭載されるソフトウェア製品の品質（内部/外部品質），ソフトウェアを導出する開発プロセスや人的/物的リソースの品質の大きく3種があり，それぞれ後者が前者に影響を与えることが知られている[1)]．ここで内部品質とは，モデルやソースコードといった中間成果物を含むソフトウェア全般そのものについて，主に静的な解析により測定される品質を指す．一方，外部品質とは，実行コードを搭載したシステム全体をテスト実行（もしくは本番運用）する過程において主に動的に測定される品質を指す．

本稿は，システム搭載前のソフトウェア製品の開発や保守，調達，およびそれらのプロセスの改善に従事する人々を対象とする．さらに本稿は，属人性を排した品質評価の枠組みを提案することを目的とし，上記のうちで主にC言語で記述されたプログラムソースコードについて，静的な解析に基づいた測定によるプログラムの内部品質評価を扱う．ソフトウェアの開発

† 国立情報学研究所
National Institute of Informatics
†† 株式会社オージス総研
Ogis-RI Co., Ltd.

においてプログラムソースコードの品質は，搭載されるシステム全体の性能や開発・保守コストに支配的影響をもたらすため，その品質を高精度で測定/評価し，問題箇所や改善すべき品質特性の特定を実現する実用的技術が必要である．

これまでに種々の品質測定法が提案されているが，網羅性の低さや総合評価能力の欠如といった問題を持ち，結果として測定法や測定結果が十分に活用されていないのが現状である[2]．そこで我々は，直接にはC言語で記述されたソースコードを対象として，内部品質の測定と評価を効率良く実施する枠組みを提案し，従来の取り組みがかかえる問題を解決あるいは改善する．提案する枠組みは，C言語以外のプログラミング言語に基づくソースコードについても，部分的に再利用可能である．

本稿の以降は次のように構成される．2章では，従来の品質測定および評価の取り組みと問題を述べる．3章では，指摘した問題を解決する品質測定/評価の枠組みを提案し，枠組みを構成する個々の要素を詳説する．4章では，枠組みに基づいて複数の実プログラムソースコードの品質を評価し，枠組みの妥当性を評価する．5章では，従来の取り組みを含む関連研究を取り上げる．最後に6章では，本稿の内容をまとめて，今後の展望について述べる．

2. 従来の品質測定の問題

測定法は品質の観点より，情報量の小さいものから順に，品質と関連付けられずに何らかの特徴を測定する測定法（Metric☆），測定値が品質特性と関連付けられて解釈方法が含められた品質測定法（Quality Metric），単一の品質特性について複数の品質測定法がまとめられた品質メトリクス（Quality Metrics），および，複数の品質特性について品質測定法がそれぞれ体系的にまとめられた品質メトリクススイート（Quality Metrics Suite，スイート）の4種に分類できる．

これまで数多くの測定法が提案されながらも，適切な測定法の選別と得られる測定値の解釈は一般に難しく[5]，様々な開発プロジェクトにおいて測定法や測定値を活用できていないのが現状である[2]．特にソースコードに対する品質評価を扱う場合に，従来の取り組みが持つ問題を以下にまとめる．

- 網羅性の低さ：ソフトウェアの設計/実装は複数の品質特性間のトレードオフをともなうため，システムの利用時の品質に影響するあらゆる品質特性を同時に測定/評価できることが望ましい．しかし，ソースコードを対象とした測定法の大多数は，品質と関連付けられていない測定法，もしくは，単一の品質特性と関連付けられた品質測定法/メトリクスである．ソースコードを扱う数少ないスイートとして，REBOOT[6),7)]やQMOOD[8)]，Ortegaらのスイート[9)]，SPCスイート[10)]，EASEプロジェクトの事例[11)]，ISO TR 9126-3参考スイート[12)]などがある．しかし，これらの大多数はソースコード以外の入力（たとえば仕様書）を必要とする．さらに，ソースコード上で測定/評価する品質特性は限られており，ISO9126-1[1)]の品質モデルにおいて規定される品質特性のほぼすべてを網羅するものではない．ISO9126-1品質モデルは，様々な品質モデルを参考とし，代表的な品質特性を網羅するようにまとめられた規格であり，網羅性の判断指標として利用できる．
- 分解性/総合評価能力の欠如：高級プログラミング言語によって記述されるソースコードは，一般に複数の論理的/物理的モジュール間の包含関係によって階層的に構成される．たとえばC言語の場合，一般に関数 ∈ ファイル ∈ ディレクトリ ∈ システムという4階層を構成する．このとき，システム全体の品質を総合評価し異なるシステム間の比較検討に用いる場合や，個々のモジュール単位で品質を評価し問題のある箇所を特定する場合など，目的に応じた単位で品質測定/評価できることが望ましい．しかしながら，ソースコードについて全体から部分まで任意の単位で測定可能なスイートは提案されていない．
- 評定水準導出の非簡便さ：品質を測定した結果を評価するにあたり，個々の測定値の許容度合いを判定するための評定水準（いわゆる閾値）や，その判定結果を品質特性などの単位でまとめて総合的に判定するための評価基準が必要である[13)]．評定水準の一般的な導出法は，一定規模のサンプルを用意し，サンプルを何らかの観点（たとえば被利用状況[10)]や定性的品質評価[14),15)]）に従って優秀な群と劣等な群に分け，両群の測定値の分布傾向を比較したうえで，主に優秀な群中の大多数の測定値が統計的に収まる範囲の上限/下限を閾値として採用するものである．ただし従来の手法では，ソースコードに加えて被利用状況や定性的評価などの情報が必要であり，スイートを構築したい問題領域やプログラミング言語に応じて必ずしも容易

☆ Metricという用語は元来“メートル法の”という意味しか持たない[3)]．そこで混乱を避けるため，測定に関する国際規格ISO/IEC15939[4)]ではMetricに代わりMeasure（測定量）という用語が使われている．ただし本稿では，用語として比較的一般に普及していると考えられる“測定法”（Metric）を用いる．

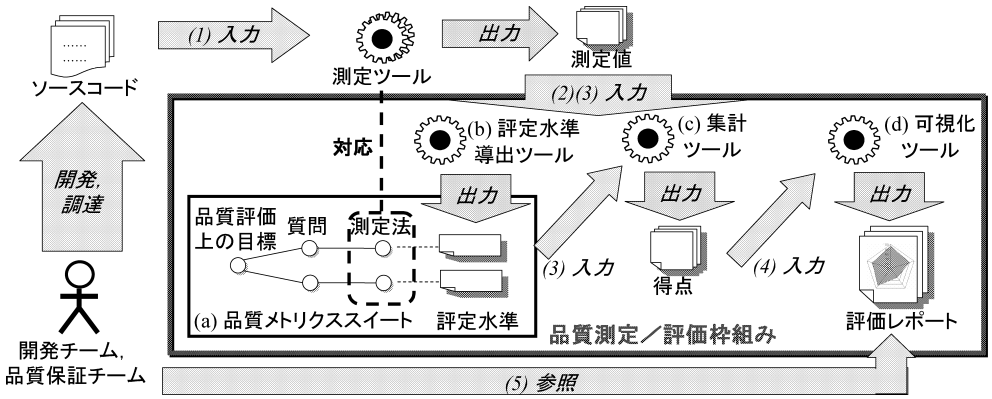

図 1 品質測定/評価枠組みの構成
Fig. 1 Structure of the quality measurement/evaluation framework.

に評定水準を導出できない．

3. 品質測定/評価枠組みの提案

我々は，直接には C 言語で記述されたソースコードを対象として，内部品質を効率良く測定し評価する実用的枠組みを提案する．ただし提案する枠組みは，C 言語以外のプログラミング言語に基づくソースコードについても，部分的に再利用可能である．全体の仕組みと上述の問題解決，および，各構成要素の詳細を以下に述べる．

3.1 全体の仕組みと問題解決

枠組みの構成を**図 1** に示す．提案する枠組みは，スイート，集計ツール，可視化ツール，評定水準導出ツール，および，導出された具体的な評定水準の 5 つの要素より構成され，それぞれ以下のように上述の問題を解決あるいは改善する．なお，枠組みには測定ツールは含まれない．枠組み中のスイートで規定される測定法を扱う既存の C 言語対応測定ツール（QAC[16] や RSM[17] など）を流用する．

- スイート：ISO9126-1 品質モデルに基づいた網羅性の高い品質モデルを作成し，品質モデル上の品質特性と対応付けられた（品質上の解釈方法が定められた）測定法の集合を構築した．これにより「網羅性の低さ」問題について，ソースコードのみを扱うスイートとして，扱う品質特性の網羅性を高めるように改善した．
- 集計ツール，可視化ツール：スイート中のすべての測定法の測定値について評定水準に従って 0〜100 の得点に正規化する方法，および，得点をモジュール単位で段階的に集計する仕組みを実現し，「分解性/総合評価能力の欠如」の問題を解決した．可視化ツールは，集計された得点の集合を直感的に分かりやすい評価レポートに変換する．
- 評定水準導出ツール，評定水準：品質上受け入れ可能なソースコード群，もしくは，品質上の観点から改訂されたソースコード群を用いて統計的に評定水準を導出する仕組みを実現し，「評定水準導出の非簡便さ」の問題を解決した．また，スイートおよび上述の 3 ツールを現実の複数の組み込みソフトウェアに適用して得られた測定結果より導出された具体的な評定水準を，同種の問題領域における参考値として枠組みに組み入れた．

枠組みを利用する流れを以下に示す．なお，ステップ (2) は必須ではなく，同種の問題領域における評定水準を導出後は必ずしも実施しない．

(1) 評価者は，評価対象ソースコードを，スイートが規定する測定法を扱う測定ツールに入力し，評価対象ソースコードの測定値を得る．
(2) 評価者は，当該問題領域における評定水準が未導出であって新たに導出したい場合に，評価対象ソースコードとは独立して，以下のいずれかを用意する．
  - 評価者にとってすべての品質特性が受け入れ可能な度合いに達していると明らかに判断可能な開発終了後，かつ，当該問題領域に属するソースコード群
  - 上記ソースコード群を容易に得られない場合は，何らかの品質上の観点から改訂されており，かつ，当該問題領域に属するソースコード群

  続いて，用意したソースコード群についてステップ (1) の実施により測定値を得て，得られた測定値を評定水準導出ツールに入力し，評定水準を得る．
(3) 評価者は，評価対象ソースコードの測定値を集

計ツールに入力し，得点の集計結果を得る．集計ツールは内部的に，スイートおよび導出済みの評定水準を用いる．

(4) 集計ツールは自動的に集計した得点を可視化ツールに入力し，評価レポートを得る．

(5) 評価者は，評価レポートを参照し，問題箇所や改善すべき品質特性の特定に役立てる．

標準的なソフトウェア品質評価プロセスの規定ISO/IEC14598[13]（以下の (i)〜(iv)）における上述の各手順の位置づけ，および，枠組みの再利用により簡略化可能な作業を以下に示す．

(i) 評価要求の確立（必要な作業：評価目的確立，対象種別識別，品質モデル仕様化[13]）：枠組みのスイートは，特定の品質要求/評価要求に依存しない汎用のものとして作成した．したがって評価要求の違いにかかわらず，スイートの部分もしくは全体を再利用することで品質モデル仕様化の作業を簡略化できる．また評価要求に応じて，評価における品質副特性や測定法単位での重みの変更が可能である．

(ii) 評価の仕様化（測定法選択，評定水準確立，評価基準確立[13]）：スイートでは品質特性と測定法が対応付けられているため，測定法選択の作業を省略もしくは簡略化できる．評定水準確立の作業は，上述のステップ (1)+(2) に相当する．なお評価基準は，別途定める必要がある☆．

(iii) 評価の設計（評価計画作成[13]）：枠組みで直接には扱わないが，たとえば，ISO/IEC14598 シリーズより評価者の立場に応じた作業およびスケジュールを選択し，具体化する．

(iv) 評価の実施（測定，評価基準との比較，総合評価[13]）：測定の作業は，上述のステップ (1)+(3) に相当する．また枠組みの可視化ツールは事前に設定された評価基準としての得点との比較に基づく可視化を実現するため，評価基準との比較作業を支援する．つまり上述のステップ (4)+(5) は，評価基準との比較および総合評価の作業に相当もしくは部分を構成する．

3.2 構成要素の詳細

(a) 品質メトリクススイート

ISO9126-1 の品質モデルを出発点とし，複数の実務者へのインタビューの繰り返しを通じて，特定のプログラミング言語に依存しないプログラムソースコード

☆ 汎用的な評価基準の例として「枠組みによる得点 75 点以上」があげられることを 3.3 節で示す．

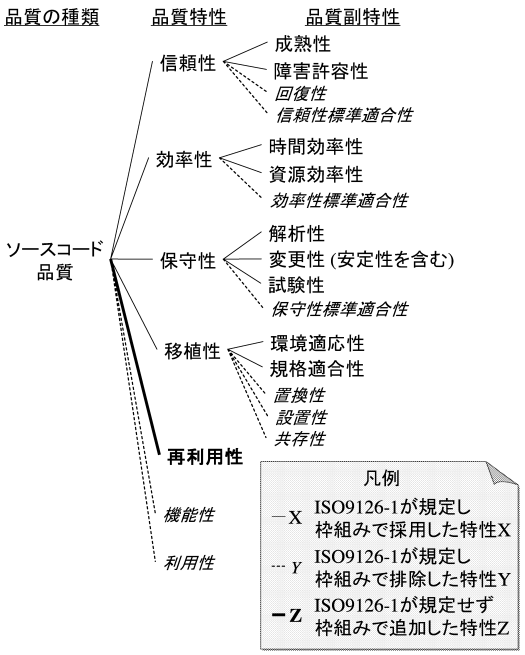

図 2 構築した品質モデル
Fig. 2 Quality model.

一般より測定/評価可能なソフトウェア内部品質特性（典型的には動作させずに測定可能なソフトウェア内部の品質上の特徴集合[1]）を洗い出し，**図 2** に示す品質モデルを構築した．なお，機能性および使用性については，ソースコードのみからの測定/評価は困難であり，実装以前の上流工程で作り込むべき品質特性であるため[18]，除外した．

- 信頼性：指定された条件下で利用するときに指定された達成水準を維持する能力[1]．副特性として，成熟性および障害許容性を採用し，ソースコードのみからの測定/評価が困難な回復性および信頼性標準適合性を除外した．
- 効率性：明示的な条件下で使用する資源量に対比して適切な性能を提供する能力[1]．副特性として，時間効率性および資源効率性を採用し，効率性標準適合性を排除した．
- 保守性：修正のしやすさに関する能力[1]．副特性として，解析性，変更性および試験性を採用し，保守性標準適合性を排除した．また，ISO9126-1 における安定性について，特徴として重複するため変更性に含めた．
- 移植性：ある環境から他の環境に移すための能力[1]．副特性として，環境適応性および規格適合性（移植性標準適合性）を採用し，ソースコードのみからの測定/評価が困難な置換性，設置性お

表 1 構築した品質メトリクススイート（抜粋）
Table 1 Quality metrics suite (excerpt).

言語依存性：プログラミング言語に独立				プログラミング言語に依存	
特性	副特性	目標	質問	副質問	測定法
信頼性	成熟性	対象：ソースコードを測定し 論点：障害の発生頻度を 視点：エンドユーザの視点から 目的：予測する	Q0100 障害が起こりやすいコーディングになっていないか	Q0101 領域を正しく初期化しているか	MFl134, 未初期化 const オブジェクトの数 MFl107 初期化子が配列のサイズよりも少ない配列の数 MFl133 終端のナル文字を保持していない文字列配列の数 MFl169 列挙型の初期化が不十分である数
				…	…
			…	…	…
			Q0400 使用するリソースサイズは予測可能か	Q0401 再帰呼び出しを行っていないか	MSy021 再帰パス数
	障害許容性	…	…	…	…
保守性	解析性	対象：ソースコードを測定し 論点：スタイル，構造，振舞い，保守する部分の特定がしやすいかを 視点：開発者の視点から 目的：評価する	Q3700 関数が複雑すぎないか	Q3701 関数の呼び出しネストが深くないか	MFn095 コールグラフの階層の深さ
				Q3702 処理は複雑すぎないか	MFn066 制御構造の最大ネスト数 MFn072 サイクロマティック数 MFn069 推定静的パス数
				…	…

よび共存性を除外した．

- 再利用性：全体もしくは部分の別環境における再利用のしやすさ，特に，部分をブラックボックス的に別環境で利用できる能力．ISO9126-1 では規定されておらず，保守性/移植性の評価結果と似た傾向を持つ可能性があるが，同種問題領域における開発効率を重視して再利用のしやすさ単独を詳細に取り上げることを可能とするため加えた．

続いて Goal-Question-Metric（GQM）法[19] を適用し，品質モデル上の各品質副特性から，利用可能な測定法へと対応付けを行った．GQM 法は，明確にしたい目標（Goal）を，目標達成のために評価すべき質問（Question）を経て，測定法（Metric）へと結びつけるゴール指向の対応付け手法であり，枠組みにおいて評価する品質特性と測定法の対応付けに適している．我々は GQM 法を用いて以下の 4 種を順に識別し，それぞれ階層的に対応付けた．

- 目標：各品質副特性を予測/評価したいという要求事項．
- 質問：目標を達成するにあたり，ソースコードについてプログラミング言語に独立なままで検証すべき事柄．
- 副質問：上位の質問が他の質問に比べて抽象度が高い場合の，上位の質問を分解/具体化して検証すべき事柄．もしくは上位の質問が，想定されるプログラミング言語依存の測定法との乖離が大きい場合の，ソースコードについてプログラミング言語に依存して検証すべき事柄．
- 測定法：上位の質問/副質問に答えるために必要な情報/データを識別したうえでの，同情報/データを測定可能な（プログラミング言語依存の）測定法．

このようにスイートを 4 階層より構成し，目標と質問は言語独立，副質問と測定法は基本的に言語依存とすることで，枠組みの可変部/不変部を明確にし再利用性を高めている．これらのすべての対応付けにあたり，その作業の属人性を極力排するため，実際のプログラム品質改善業務の従事経験を有する複数人が，レビュー/修正を約 1 年間繰り返してスイートを構築した．

構築したスイートの抜粋を**表 1** に示す☆．スイートは，47 の質問，101 の副質問，236 の測定法より構成される．測定法として，以下の 3 種を用いた．

- 各種 C 言語対応測定ツール（QAC や RSM）が扱う測定法：たとえば「再帰パス数」
- C 言語対応コーディング作法ガイド[18] における各ルールの適合度合いを判定するためのデータを提供する測定法：たとえば，ルール「const 型変数は宣言時に初期化する」の適合度合いを判定するためのデータを与える測定法「未初期化 const オブジェクトの数」
- 現段階で既存の測定ツールによって測定できないが，対象とする質問/副質問に回答するために必要と考えられるデータの測定方法：たとえば「マクロ記述内の条件分岐の数」や「アセンブラコードが書かれているモジュールの数」

現在利用可能な測定ツールによって測定できない測定法の割合は 19%（45 個）であり，また，質問のうちで副質問を経由して，あるいは直接に測定法を対応付けられていないものの割合は 34%である．今後，新たな測定ツールの開発により測定法/質問の非網羅率を 0%に近づける予定である．

☆ スイートの全体は文献 20) で公開した．

表 2 採用した測定法の例
Table 2 List of metrics used (excerpt).

ID	名称	水準	尺度	依存
MSy021	再帰パス数	最小	比	非
MMd027	直下要素数	閾値	比	非
MFl003	有効行数	閾値	比	非
MFn072	サイクロマティック数	最小	比	非

用いた測定法の例を**表 2** に示す．表形式で以下の詳細を網羅し，評価者による測定法の理解を支援する．

- 対象の種別：システム（ID: MSyXXX），ディレクトリ（MMdXXX），ファイル（MFlXXX），関数（MFnXXX）
- 評定水準の種別[21]：閾値（測定値が特定の値もしくは区間内が品質上最適と解釈），最小（小さいほど望ましい），最大（大きいほど望ましい）
- 尺度の種別[4]：名義，順序，間隔，比
- プログラミング言語への依存性の種別：非（言語に非依存），非 OO（非オブジェクト指向言語に依存），OO（オブジェクト指向プログラミング言語に依存），C（C 言語），C++（C++言語），C&C++（C または C++言語）

たとえば表 1 において，ソースコードの解析のしやすさを評価する目標に対して，いくつかの言語非依存な質問（たとえば Q3700）をあげている．さらに質問 Q3700 は，そのままでは抽象度が高く直接に測定法と結びつかないため，いくつかの副質問 Q3701，Q3702 などに分解されている．最後に，各副質問について回答するための材料となるデータを測定可能な測定法として，Q3701 については 1 つの測定法 MFn095 を，Q3702 については 3 つの測定法 MFn066，MFn072，MFn069 をそれぞれ対応付けている．これらの対応付けにより，測定値からソースコードの品質を品質副特性の単位で評価できる．たとえば，関数のサイクロマティック数[22),23)] の測定値を，ソースコードの解析性の評価に用いることができることが分かる．さらに表 2 より，サイクロマティック数の評定水準種別が「最小」であるため，その測定値が小さいほど解析性が高いと判定できることが分かる．

(b) 評定水準導出方法とツール

スイートの適用によって得られる測定値から特定の品質特性の許容度合いを評価するためには，個々の測定値について評定水準が必要である．枠組みでは，評価者にとってすべての品質特性が受け入れ可能な度合いに達していると明らかに判断可能なソースコード群を，評定水準の導出に用いる．しかしながら，任意のソースコードについて，すべての品質特性が評価者にとって受け入れ可能な度合いに達しているとは限らず，提案する枠組みとは別の何らかの方法（たとえば定性的レビュー）による品質評価を経て，受け入れ可能であると判断する必要がある．そのような別個の品質評価を経て受け入れ可能なソースコード群を用意することは，時間/労力のかかる作業であり容易ではない．

そこで枠組みでは，評価者にとって全品質特性が受け入れ可能な度合いに達していると明らかに判断可能なソースコード群が容易に得られない場合に，機能がほぼ保たれたままで何らかの品質上の観点から改訂されたソースコード群を，評定水準導出用の入力として代替的に用いる．そのような改訂後のソースコード群は，必ずしもすべての品質特性が改訂前と比較して向上しているとは限らないが，全品質特性について受け入れ可能な度合いに達していると近似的にとらえる．このようにして，任意のソースコード群の 1 つ 1 つについて品質特性達成の度合いを定性的に評価する代わりに，評定水準導出に必要なソースコード群を簡易かつ迅速に用意することができる．

枠組みは，上述の方針に従って準備された全品質特性について受け入れ可能なソースコード群（あるいは何らかの品質上の観点から改訂された後のソースコード群）における測定値の分布における統計上の第 1 四分位数および第 3 四分位数を用いて，対象測定法の評定水準の種別に応じて以下のように許容可能な評定水準を算出する．

- 最小：第 3 四分位数以下を許容可能な評定水準とする．同評定水準には，評定水準の算出に用いるソースコード群（算出用ソースコード群）の 75%が該当する．
- 最大：第 1 四分位数以上を許容可能な評定水準とし，同評定水準には算出用ソースコード群の 75%が該当する．
- 閾値：第 1 四分位数〜第 3 四分位数の間を許容可能な評定水準とし，同評定水準には算出用ソースコード群の 50%が該当する．

また，本導出法を，表計算ソフトウェア Excel のワークシート関数を用いて Excel 上で実装した．本導出法は，すべての品質特性が受け入れ可能な度合いに達していると判断可能なソースコード群が得られる場合，もしくは近似的に，何らかの品質上の観点から改訂されたソースコード群が得られる場合に適用可能である．

我々は本導出法を，機能をほぼ保ったままで品質上の観点から改訂された前後のプログラム対を入手可能な 3 件の組み込みソフトウェア S_1，S_2，S_3（プリンタ

表 3 用いたサンプル 3 件の規模
Table 3 Scale totals for samples used.

対象	ファイル数		関数の数		有効行数	
	前	後	前	後	前	後
S_1	444	125	688	300	55,535	5,162
S_2	8	55	249	386	7,530	6,707
S_3	151	662	2,332	4,187	111,585	104,146
合計	603	842	3,269	4,873	174,650	116,015

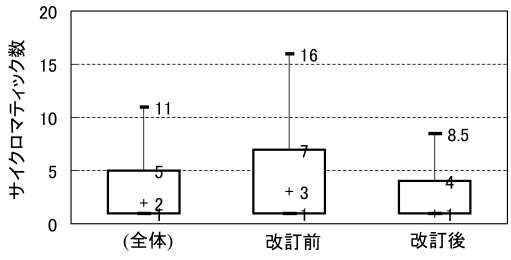

図 3 MFn072 サイクロマティック数の分布（箱ひげ図）
Fig. 3 MFn072-Distribution of cyclomatic numbers (box-plot diagram).

搭載ソフトウェア 1 件，車載ソフトウェア 2 件）に適用した．用いた計 6 つのプログラムの規模上の測定値を，改訂前後の群ごとに**表 3** に示す．表 3 より，改訂前と比較して改訂後では同等の機能がより多くの関数によって実現され，かつ，各関数のコード行数が短縮されていると推測できる．他の測定法の適用結果の例として，「MFn072 サイクロマティック数」の測定結果の分布を，改訂前後の群および全体について箱ひげ図として**図 3** に示す．箱ひげ図において，長方形の上側/下側の辺が第 3/1 四分位数を，辺の間の距離が四分位範囲を，長方形の上側/下側にある「ひげ」は上側/下側外れ値（第 3/1 四分位数 ± 四分位範囲 ×1.5）を，それぞれ表す．図 3 において，改訂後では改訂前と比較して小さな値をとる傾向にあることが分かる．この場合は，改訂後の群を評定水準導出用のソースコード群として用い，図 2 よりサイクロマティック数の評定水準の種別は「最小」であるため，改訂後の第 3 四分位数である 4 以下が許容可能な評定水準となる．

(c) 正規化/集計方法とツール

スイート中の異なる測定法の測定値を品質特性単位，および，モジュール単位で集計し総合的な品質評価を実現するため，各測定値を評定水準に従って 0〜100 の得点に正規化する手法を考案した．具体的には測定値が，上述の評定水準内に収まる場合の得点を 100，評定水準算出時の算出用の群における統計上の上側特異値（第 3 四分位数 + 四分位範囲 ×3）および下側特異値（第 1 四分位数 − 四分位範囲 ×3）をそれぞれとる場合の得点を 0 に設定し，その間を直線で結んだ得点グラフを測定法ごとに構築する．そして，構築した得点グラフ上で測定値に対応する得点を採用する．

例として，サイクロマティック数の測定値を正規化する様子を**図 4** に示す．図 3 において，導出用（改訂後）の群の第 3 四分位数が 4，四分位範囲が $4-1=3$ であるため，図 4 において得点グラフは測定値 4 から測定値 $4+3\times3=13$ を直線で結んだものとなる．ここでサイクロマティック数の測定値が 2 の場合，得点グラフ上で得点は 100 となる．

このように，測定値を線形かつ連続的な得点グラフ上で正規化された得点へと変換する手法により，測定値のわずかな違いを直感的に得点へと反映可能とし，さらに，異なる測定法の測定値を共通に比較可能とした．なお，測定値正規化時のグラフ形状として**図 5** に示すように，直線以外にも非線形曲線やステップ関数などの様々なものが考えられる[24]．しかし，測定値のわずかな変化が各品質特性にもたらす影響を精密に把握できない場合においては，考えられる種々のグラフ形状との違いが小さい直線を採用することが妥当と考えられる．その後，時間とコストをかけて，多数の測定実験および外部指標との照らし合わせにより，品質特性と測定値の関係が精密に明らかとなった場合は，最適な形状を品質特性ごとに採用することが望ましい．

続いて，正規化された個々の得点を，重み付けたうえで，品質特性/副特性単位およびモジュール単位で集計する．重みは，質問や測定法の単位でそれぞれの重要性を可変とする仕組みである．ただし，品質要求が与えられず，重視する品質特性や特徴が存在しない限りにおいて均等配分とする（つまり平均をとる）．得点の集計を実現するための基礎となる集計モデルを，UML クラス図として**図 6** に示す．また，図 6 の各クラスについて，得点に関する制約を OCL[25] によって以下に示す（簡略化のため副質問などを除外した）．

```
context 特性結果
 -- 得点は，全副特性の得点にそれぞれ重みをかけた
 -- 合計である．
 inv: 得点 = 副特性結果->iterate(c:副特性結果;result:
  Real=0 | result + c.得点 * c.品質副特性．重み)
 -- 副特性の重みの合計は，常に 1 である．
 inv: 副特性結果．品質副特性．重み -> sum() = 1

context 副特性結果
 inv: 得点 = 質問結果->iterate(q:質問結果;result:
  Real=0 | result + q.得点 * q.質問．重み)
 inv: 質問結果．質問．重み -> sum() = 1

context 質問結果
 inv: 得点 = 測定結果->iterate(m:測定結果;result:
```

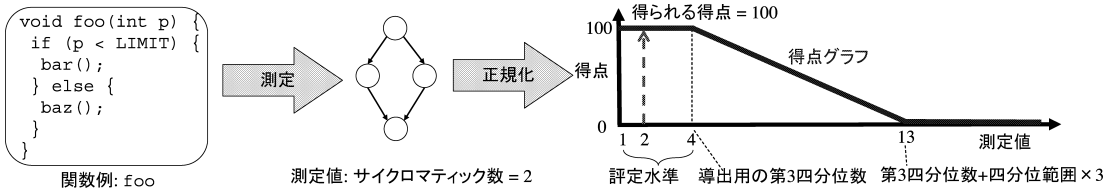

図 4 サイクロマティック数の得点化
Fig. 4 Calculating the score for cyclomatic number.

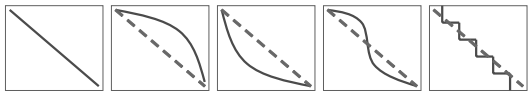

図 5 測定値の正規化における種々のグラフ形状の候補（点線は「直線」グラフを表す）
Fig. 5 Graph types for measurement normalization (each dotted line indicates the linear graph).

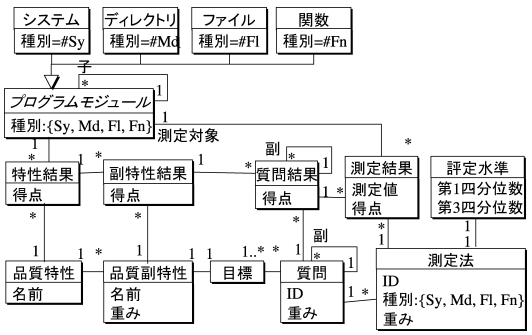

図 6 特性/モジュール単位についての得点集計モデル
Fig. 6 Characteristic/module unit score aggregation model.

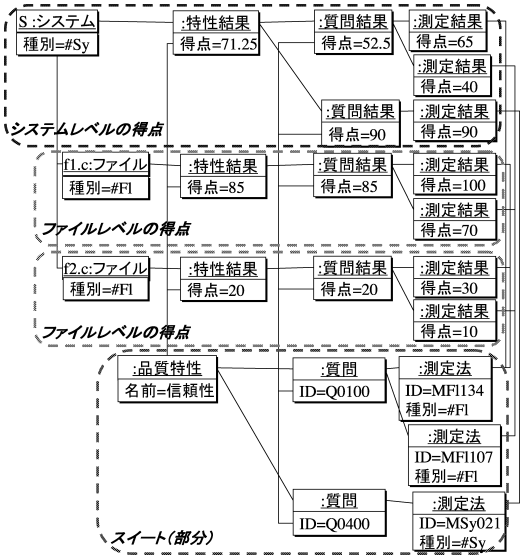

図 7 集計モデルを用いた得点集計例
Fig. 7 Example of calculating the score using the aggregation model.

```
  Real=0 | result + m. 得点 * m. 測定法. 重み)
 inv: 測定結果. 測定法. 重み -> sum() = 1

context 測定結果 inv:
 if 測定対象. 種別 <> 測定法. 種別
  -- 測定対象の全ての子の測定結果中で，測定法が自身の
  -- 測定法と同一の測定結果の得点の平均を，自身の得点
  -- とする．
   得点=測定対象. 子. 測定結果 -> select(測定法. ID=
      self. 測定法. ID). 得点 -> sum() /
      測定対象. 子 -> size()
 else
  -- 評定水準に照らして正規化した測定値を得点とする．
 endif
```

集計モデルに基づく得点の算出/集計例を，UML オブジェクト図として**図 7** に示す．簡略化のため，品質副特性や副質問，目標，評定水準を省略し，2 つのファイル `f1.c`，`f2.c` がディレクトリを経ずに直接に全体を構成するシステム S の信頼性のみの評価を表している．図 7 において，S の信頼性の得点（71.25）は，2 つの質問に対応した 3 つの測定法の得点より得られている．ただし，測定法のうちで MFl134，MFl107 は直接にはファイルを対象とする測定法であるため，それらの得点として `f1.c`，`f2.c` それぞれの得点の平均を採用している．さらに図 7 において，システムやファイルといった個々のモジュール単位それぞれに品質特性や質問単位で詳細な得点が得られることが分かる（たとえば `f1.c` の信頼性の得点は 85）．我々は上述の正規化/集計の方法を，Ruby によるスクリプトとして実装した．

(d) 可視化ツールと利用例

正規化と集計によって得られる得点を，個々のモジュール単位で品質特性ごとに表示し，かつ，モジュール単位の包含関係に従って詳細な単位で閲覧可能とする可視化ツールを実現した．可視化ツールは Ruby によって実装され，集計ツールから自動的に入力される集計済みの得点を使用し，相互にリンクされた HTML ページ集合を評価レポートとして生成する．可視化ツールは可視化の際，可変であり事前に設定する評価基準としての得点（初期値は 75 点）を下回る得点の

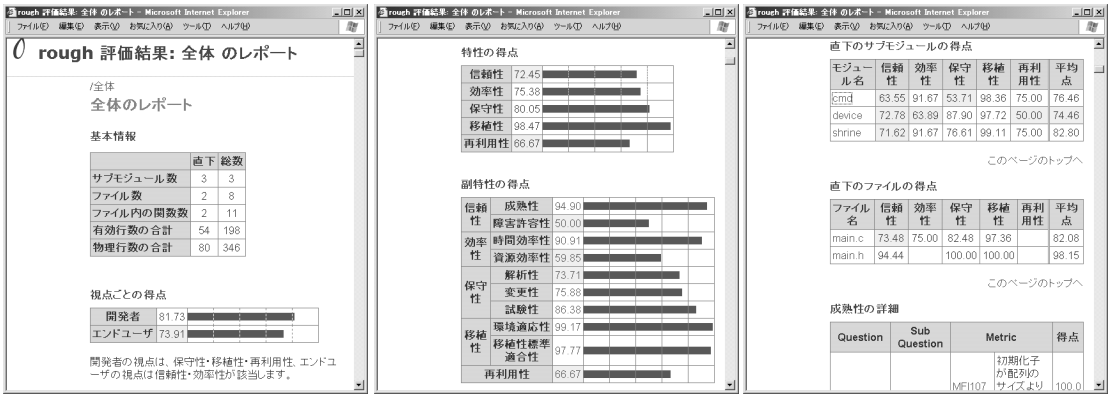

図 8 システム全体の評価レポート例（左：規模，中央：特性/副特性の得点，直下ディレクトリ/ファイルの得点）
Fig. 8 Example of system evaluation report (left: scales, center: characteristic/sub-characteristic scores, right: subordinate directory/file scores).

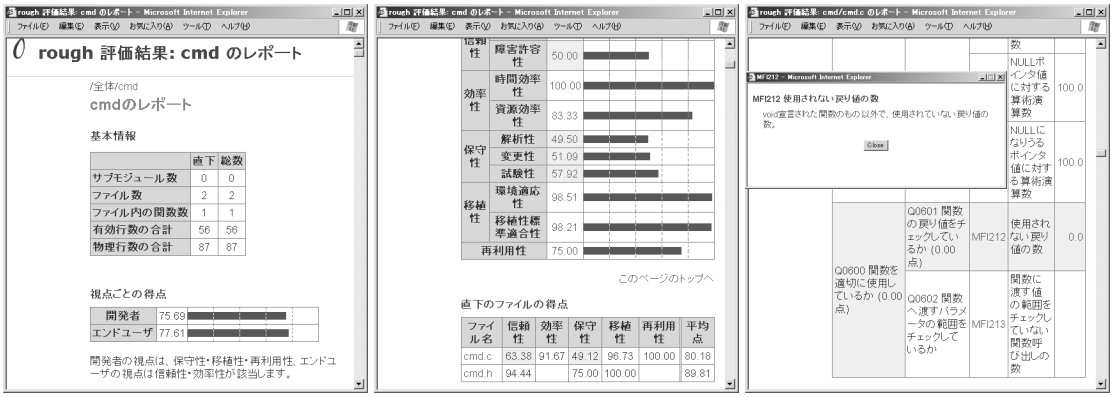

図 9 モジュール別の評価レポート例（左/中央：ディレクトリの規模/得点，右：ファイルの品質特性詳細）
Fig. 9 Example of module evaluation report (left: directory scales, center: directory scores, right: detailed characteristic of file).

箇所を強調する．

得られる評価レポートの例を**図 8** および**図 9** に示す．図 8 では，あるプロジェクト rough のシステム全体の規模，品質特性/副特性単位の得点，直下のディレクトリ/ファイル単位の得点の概要がそれぞれ示されており，評価者は全体的な品質の傾向を把握できる．この例では，特に信頼性が低いことが強調表示により分かるため，全体の信頼性を損なう原因追求のためディレクトリ一覧を閲覧すると，ディレクトリ（レポート中では “サブモジュール” と表記）cmd の信頼性が非常に低いことが分かる．そこで図 9 へのリンクをたどり cmd の得点傾向および直下のファイルの傾向を閲覧すると，cmd.c の信頼性が低いことが読み取れる．最終的に，同ファイルの品質副特性単位の詳細な得点状況を閲覧すると，障害許容性の質問「関数を適切に使用しているか」に対応付けられた 1 つの測定法「使用されない戻り値の数」の測定結果が 0 点となっており，信頼性を損なう要因の候補を特定できたことが分かる．さらにリンクをたどり，得点化前の測定値を閲覧することも可能である．なお，同測定法の下に位置する測定法「関数に渡す値の範囲をチェックしていない関数呼び出しの数」の得点は未設定であり，現状では測定できていない測定法の一種である．

このように，評価レポートが提供するページ間のリンクおよび強調表示の仕組みにより，評価者は特に問題のあるモジュールや品質特性を容易に識別し，より詳細を調査するために下位（もしくは上位）へとたどることを支援する．

3.3 枠組みの用途と適用範囲

枠組みは品質の全体から詳細まで網羅するため，管理者レベルの評価者から個々のモジュールを担当する開発者まで，幅広く品質評価に役立てることができる．

具体的には，得られる得点を用いて，重点的に改善すべき品質や問題のある箇所を特定できる．

また，組織/プロジェクトにおいて評価基準として許容可能な得点範囲を設定すれば，開発/調達時の非機能要件などに得点を活用できる．たとえば，枠組みでは評定水準を，すべての品質特性について受け入れ可能と明らか（もしくは近似的）に判断できるソースコード群の四分位範囲に基づいて設定する．さらに，スイートを構成する測定法の大多数（約 79%）について評定水準の種別は「最小」もしくは「最大」であり，それらの種別では算出に用いたソースコード群の75%が評定水準に該当することを意味する．そこで，枠組みにおいて品質上の受け入れ可能なソースコード群とはその 75%の得点が 100 点をとるものと近似することができ，残り 25%の得点が最低（0 点）であっても全体として 75 点以上は得られるため，評価基準の案として「75 点以上が合格」を考えることができる．

本枠組みは，以下にあげる様々な状況で利用できる．

- 組み込み領域で C 言語プログラムを実装/調達：枠組みのすべてを再利用できる．
- 非組み込み領域で C 言語プログラムを実装/調達：対象問題領域において品質上受け入れ可能と明らか（もしくは近似的）に判断可能なソースコード群が得られる場合は，本稿で導出済みの評定水準を除いて枠組みを再利用できる．そのようなソースコード群が得られない場合は，評定水準および評定水準導出法を除いた枠組みについて，他の評定水準導出法を組み込んで適用できる．
- 非 C 言語プログラムを実装/調達：枠組みのスイート中で言語独立な目標と質問を再利用できる．

4. 実験的評価

枠組みの妥当性および品質改善時の定量的な評価結果反映能力をそれぞれ，実際のプログラム集合を用いて実験し検証/評価した．その結果を以下に述べる．

4.1 枠組みによる定量的評価の妥当性

提案する枠組みによる品質の定量的評価の妥当性を，定性的品質評価の結果と照らし合わせることで検証した．

具体的には，品質副特性の単位で品質の高さを 4 段階評価（0 点，50 点，75 点，100 点）する質問票を作成し，評定水準の導出に用いた 3 つの組み込みプログラム（S_1，S_2，S_3）について，各プログラムの改訂前の開発（および改訂後の受け入れ）を担当した開発者が，質問票に回答することで改訂前後の品質を定性的に評価した．改訂前後における各プログラムの規模は表 3 に掲載済みである．得られた結果を品質特性単位に平均してまとめたものを**表 4** に示す☆．表 4 における「前」「後」は，それぞれ改訂前/改訂後の評価を表す．改訂前後の評価結果が得られる 2 つのプログラム S_1，S_2 について，プログラムの規模の違いにかかわらず，S_1 の信頼性を除くすべての品質特性が向上したと定性的に評価されていることが分かる．

表 4 質問票を用いた定性的品質評価の結果
Table 4 Qualitative quality evaluation results using the quality table.

対象	信頼性		効率性		保守性		移植性		再利用性	
	前	後	前	後	前	後	前	後	前	後
S_1	92	92	80	83	75	95	69	100	92	100
S_2	59	79	67	71	54	78	76	88	60	83
S_3	–	92	–	78	–	75	–	88	–	83

表 5 枠組みを用いた定量的品質評価の結果
Table 5 Quantitative quality evaluation results using the framework.

対象	信頼性		効率性		保守性		移植性		再利用性	
	前	後	前	後	前	後	前	後	前	後
S_1	79	83	96	92	80	88	87	86	80	92
S_2	88	99	99	96	74	89	94	96	0	95
S_3	85	90	96	86	67	75	77	82	0	0

続いて上述の定性的評価結果を，枠組みによる定量的評価結果と照らし合わせて，枠組みの妥当性を検証した．各プログラムの定量的評価結果を**表 5** に示す．枠組みの妥当性を，品質特性ごとに以下に考察する．

- 信頼性，保守性，再利用性：定性的評価結果における傾向と同様に，全プログラムの定量的評価結果に改訂後の向上がみられるため，枠組みはおおむね妥当と考えられる．ただし S_3 について，改訂後における再利用性の向上をとらえられていない．これは，スイート中の再利用性に関する測定法がすべてディレクトリに関するものであり，S_3 がディレクトリを持たないためである．そのようなプログラムの詳細な再利用性を評価するために，測定法の追加が必要と考えられる．
- 移植性：定性的評価結果における傾向と同様に，S_2，S_3 の定量的評価結果に改訂後の向上がみられ，枠組みを品質評価に有効活用できる可能性がある．ただし S_1 について，定性的な品質向上の傾向が定量的に得られていないため，測定法の修正/追加と対応付けが必要と考えられる．
- 効率性：全プログラムについて改訂前後で定性的

☆ 諸事情により S_3 の改訂前の定性的評価は得られなかった．

評価結果とは異なる定量的評価結果の傾向が得られたため，用いた測定法が妥当ではなかったと考えられる．原因として，ソースコードから最終システムの効率性を本質的に予測困難なこと[26]があげられ，測定法の修正が必要である．

以上より提案する枠組みを，プログラムソースコードの特に信頼性，保守性，再利用性および移植性の定量的評価に有効利用できることを確認した．ここで，実験に用いたソースコード群は現実のプリンタ/車に搭載された 7,530〜111,585[LOC] 規模のものであり，枠組みは上述の品質特性群の測定/評価について一定の実用性が認められたと考えられる．

4.2 定量的評価における品質改善の反映

上記のサンプル以外を対象とし，特定の品質特性の向上を目的とした改善時の例として，神棚を制御する組み込みプログラム[20]を取り上げる．最初の開発時には，グローバル変数の多用や巨大な main 関数などのため変更しにくいという保守上の問題を持っていた．そこで機能を保ったまま，(1) グローバル変数を多用しない，(2) 処理を適度な粒度の関数に分割，という改善を主に実施した．改善前後のプログラムとして主要構成ファイル shrine.c の抜粋を以下に示す．

```
/****************** 改善前 ******************/
...
extern int mic_threshold;
extern int show_mic_value;
extern int monitor_period;

void main(void) {
  MY_ADCSR.BYTE = 0x31;
  while(!MY_ADCSR.BIT.ADF);
  PADDR = 0x7F;
  PADR.BIT.B2 = 0;
  PADR.BIT.B3 = 0;
  PADDR = 0x0C | PADDR;
  mic_task();
  while (1) {
    char c;
    int  i, len = 0;
    char buffer[1024];
    char *cmd, *arg;
    for(i=0; i<64; i++) buffer[i] = '\0';
    syscall(serial_rea_dat(TASK_PORTID, &c, 1));
    while( c != '@' && len < 63) {
  ...
}

/****************** 改善後 ******************/
int main() {
  start_microphone();
  init_switch();
  init_motor();
  mic_task();
  while (1) {
    int finish;
    char buffer[COMMAND_BUFFER_SIZE];
    read_command(buffer, COMMAND_BUFFER_SIZE);
  ...
}
void read_command(char *buffer, int size)
{
  int i, len;
  ...
```

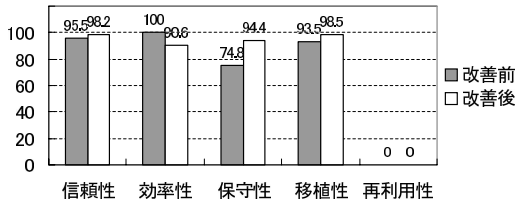

図 10 枠組みを用いた shrine.c の定量的品質評価の結果
Fig. 10 Quantitative quality evaluation results using the framework for shrine.c.

同ファイルの改善前後における評価結果を**図 10** に示す．図 10 より，改善による保守性の大幅な向上を枠組みがとらえていることが分かる．これは，関数の複雑さが低減されたことなどが，スイート中の種々の測定法の測定結果として得られたためである．なお，関数分割の副作用として，効率性がわずかに低下したと評価されている．

5. 関 連 研 究

本稿で提案する枠組みは，プログラムソースコードのみが得られる状況において適用可能であり，扱う品質特性について一定の網羅性を持ち，評定水準を容易に導出可能であり，かつ，モジュールからシステム全体までを統一的に扱う具体的な品質評価枠組みとしては初の成果である．しかしながら 2 章で述べたように，品質の測定と評価の重要さは広く認識され，これまでに数多くの測定法やいくつかのスイートが提案されている．以下に代表的なスイートを取り上げて，我々の枠組みと異なる側面および関係について言及する．

欧州の ESPRIT プロジェクトにおける REBOOT（Reusable Based Object Oriented Technology）プロジェクトでは，オブジェクト指向技術に基づいて再利用のための開発および再利用による開発を実現するための活動やツール，環境をまとめている[6),7)]．その中では，ソフトウェア部品の品質特性全般を測定するためのスイート，および，再利用性のみに特化して測定するための再利用性メトリクスがそれぞれ提案されている．しかし REBOOT のスイート/メトリクスは，部品化や再利用を必ずしも目的としないソフトウェア

製品全般について全面的には適合せず，扱う品質特性もたかだか 3 種と限定されている．

日科技連 SPC 研究会では，REBOOT を参考として，主に C 言語で記述されたソフトウェア部品の品質モデルおよびメトリクスからなるスイートを提案している[10]．しかし REBOOT と同様，部品に限らないソフトウェア製品全般についての適合性を欠く．また，スイートを構成する測定法の一部はソースコード以外の入力（たとえば仕様書や説明書）を必要とする．さらに SPC 研究会では部品向けのスイートに加えて，開発プロセスを構成する工程単位で，ソフトウェア製品全般を扱うメトリクスをそれぞれまとめている[10]．それらの一部はプログラムソースコードを測定対象とするが，品質特性と明確には対応付けられていない．

上述の SPC 研究会の成果に類似して，Lee らは，コンポーネントベース開発手法を採用する開発プロセス中の工程単位で，ソフトウェア製品全般を扱うスイートの構築手法を提案している[27]．Lee らの工程単位のスイートでは，ISO9126-1 品質モデルを網羅するように品質特性を取り上げ，ソフトウェアの内部品質と外部品質の両者の測定法を対応付けて扱い，両者の測定結果の集約および品質特性単位の集約に重み付けを導入している．しかし，用いる測定法の大多数は（実装工程における測定法についても）ソースコード以外の入力を必要とし，また，得られるスイートの具体例の全体は公開されていない．なお Lee らは重み付けにあたり，複数の熟練者/利害関係者より得られた重みの推薦値集合より，階層分析法（AHP）に基づいて適切な重みを決定する方法を提案している．この重み付けの決定手法は，我々の枠組みにおける重みの決定に適用できる可能性があり，今後その適用を検討する予定である．

Ortega らは，ソフトウェア開発における品質評価にあたり，プロダクトとプロセスの両方を統一的に扱う品質モデルを提案し，同モデルを構成する品質特性に対して，ISO/IEC TR 9126-2[28]，3[12] や ISO/IEC TR 15504-2[29] で提案される測定法を結びつけてスイートを構築している[9]．同スイートでは，我々の枠組みに類似して，異なる測定法測定値の品質特性単位の集計および比較を実現するために正規化（測定値の 5 段階評価へのマッピング）を行っている．ただし，技術報告書 ISO/IEC TR 9126-2/3 におけるスイートを構成するソフトウェア外部品質/内部品質測定法の大多数はソースコード以外の入力を必要とする．したがって，ソースコードのみが得られる場合に，Ortega らのスイートによりソースコードの品質を多面的かつ詳細に測定/評価することは困難である．

Bansiya らは，オブジェクト指向ソフトウェアの設計上の品質を，オブジェクト指向の観点から測定し評価するスイート QMOOD（Quality Model for Object-Oriented Design）を提案し，C++言語で記述された複数のプログラムに対して品質評価を実施した結果を報告している[8]．同スイートにおける品質モデルは，我々の枠組みと同様に ISO9126-1 の品質モデルを出発点とし，オブジェクト指向設計特有の事柄を考慮したうえで，品質特性を追加/削除し構築されている．また，同スイートは品質特性と測定法を複数の段階（品質特性 – 設計上の性質 – 設計測定法 – 設計要素）を経て結びつけており，我々の枠組みと同様に途中の段階において重みを変更可能としている．ただし QMOOD は，品質評価の対象として最終的にオブジェクト指向ソフトウェア（プログラム）全体のみを扱い，ソフトウェアを構成する個々のモジュールから全体までの任意の単位における体系的な品質測定/評価を実現しない．また，QMOOD はオブジェクト指向設計に特化したものであり，品質特性として（我々の枠組みでは扱っている）ISO9126-1 品質モデルにおける信頼性を除外している．

EASE プロジェクトでは，開発において日々得られるプログラム変更履歴や障害情報などに対する測定法の測定値を，GQM 法に基づいて品質上の評価目的へと結びつけて解釈/分析した事例が報告されている[11]．同報告において扱う測定法は，時系列に沿って変化したプログラムなどの履歴から得られる情報を加工するものが中心であり，単一時点におけるプログラム単体から得られる情報を中心とする我々の測定法とは情報源が異なる．

6. おわりに

本稿では，従来の品質測定/評価の取り組みがかかえる問題を解決するため，直接には C 言語プログラムソースコードの内部品質の評価を目的として，スイートや各種ツールより構成されるプログラムソースコード品質評価枠組みを提案した．我々は問題解決にあたり，ソフトウェア工学関連の基礎理論（および手法）として品質測定/評価の概念，品質モデル，プログラムソースコードの特徴と言語依存性，GQM 法および統計的アプローチを活用し，それらの組合せの上で具体的かつ実用的な枠組みを実現した．具体的には枠組みは，ISO9126-1 に基づく品質モデルの採用により，「網羅性の低さ」問題について，扱う品質特性の網羅性を改善した．さらに，測定値の正規化/集計によって

「分解性/総合評価能力の欠如」の問題を解決し，品質上受け入れ可能なソースコード群もしくは改訂によって受け入れ可能と近似的に判断されるソースコード群を用いる評定水準導出法により「評定水準導出の非簡便さ」の問題を解決した．枠組みは，品質全体から品質副特性の詳細まで，システム全体からモジュール単位の詳細までを網羅し，直接にはC言語ソースコードの内部品質評価に役立てられる．また，スイート中の副質問や測定法を変更することで，他言語ソースコードについても活用可能と考えられる．複数の組み込みプログラムへ適用した結果，枠組みを特に信頼性，保守性，再利用性（および移植性）の評価に有効利用できることを確認した．

今後は，特に効率性についてスイート中の測定法を見直し，品質評価の精度を向上させる予定である．その際，測定値の分布に基づいて測定法の冗長性や「優れた」プログラムの判別能力をあわせて検証する．また，多数のプログラムへの適用，および，実際のパフォーマンスや欠陥率などの枠組み外の定量的測定結果と照らし合わせることを通じて，評定水準の問題領域への依存性や枠組みの有効性を詳しく検証する．

さらに，枠組みにおける得点の集計方法について次の2点を検討する予定である．まず，モジュール間包含階層に関する得点集計の方法は，直下の複数の構成要素の得点を主に平均化するものであり，極端に品質の悪い要素の存在を上位レベルでは特定しにくくなる可能性がある．この問題について今後，標準偏差などにより要素集合の得点分布の広がりが大きい場合に（広がりが小さい場合と比較して）得点を小さく集計する仕組みや，各要素の規模（たとえば行数）に応じて要素単位で得点を重み付けする仕組みを検討する予定である．また，測定値が評定水準に該当する場合は，測定値の違いにかかわらず一律に100点へと得点化しており，品質の良さの微細な違いを厳密には得点へと反映していない．今後，評定水準の種別が「最大」もしくは「最小」である場合に，評定水準内の区間においても，測定値の違いが得点に反映されるようなグラフ形状を採用することを検討する予定である．

謝辞 論文の洗練にあたり有益なご指摘を多数くださった査読者の方々に御礼申し上げます．

参 考 文 献

1) ISO/IEC 9126-1: 2001, Quality Characteristics and Guidelines for their use.
2) 小笠原秀人ほか：ソフトウェアメトリクスの有効性評価，20SPC研究会分科会報告，日科技連 (2004). http://www.juse.or.jp/software/pdf/20_spc/1/1_a_report.pdf
3) 野中 誠：ソフトウェア品質に関する国際規格の紹介，Quality One 2006, 日科技連 (2006).
4) ISO/IEC 15939:2002, Software engineering—Software measurement process.
5) Chaudron, M.: Evaluating Software Architectures. http://www.win.tue.nl/~mchaudro/swads/
6) Sindre, G., et al.: The REBOOT Approach to Software Reuse, *Journal of Systems and Software*, Vol.30, No.3 (1995).
7) Mora, A. and Coscuella, A.: A Metrics Approach to the Software Reuse Problem, *Proc. 3rd European Conference on Software Quality* (1992).
8) Bansiya, J. and Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE Trans. Softw. Eng.*, Vol.28, No.1 (2002).
9) Ortega, M., Perez, M. and Rojas, T.: Construction of A Systematic Quality Model for Evaluating A Software Product, *Software Quality Journal*, Vol.11, No.3 (2003).
10) 菅野文友，吉澤 正（監修）：21世紀へのソフトウェア品質保証技術—日科技連ソフトウェア品質管理研究会10年の成果，日科技連出版社 (1994).
11) 門田暁人：EPMによるデータ収集・GQMによる分析の事例報告，第4回エンピリカルソフトウェア工学研究会 (2005). http://www.empirical.jp/research/publicdata/2005.4th_kenkyukai/publicdata_4.pdf
12) ISO/IEC TR 9126-3: 2003, Software engineering—Product quality—Part 3: Internal metrics.
13) ISO/IEC 14598-1: 1998, Information technology—Part 1: General overview.
14) Washizaki, H., Yamamoto, H. and Fukazawa, Y.: A Metrics Suite for Measuring Reusability of Software Components, *Proc. 9th IEEE International Software Metrics Symposium*, pp.211–223 (2003).
15) 平山雅之，佐藤 誠：ソフトウェアコンポーネントの利用性評価，情報処理学会論文誌，Vol.45, No.6 (2004).
16) Programming Research Ltd.: QAC. http://www.programmingresearch.com/
17) M Squared Technologies LLC.: Resource Standard Metrics. http://msquaredtechnologies.com/m2rsm/
18) 情報処理推進機構ソフトウェアエンジニアリングセンター（編）：組込みソフトウェア用C言語コーディング作法ガイド，翔泳社 (2006).

19) Basili, V.R. and Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data, *IEEE Trans. Softw. Eng.*, Vol.10, No.6 (1984).
20) http://www.ogis-ri.jp/solution/QAFramework.html
21) Emi, K. and Lewerentz, C.: Applying Design-Metrics to Object-Oriented Frameworks, *Proc. 3rd IEEE International Software Metrics Symposium* (1996).
22) McCabe, T.J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol.2, No.4 (1976).
23) McCabe, T.J. and Watson, A.H.: Software Complexity, Crosstalk, *Journal of Defense Software Engineering*, Vol.7, No.12 (1994).
24) Kazman, R., et al.: Making Architecture Design Decisions: An Economic Approach, CMU/SEI-2002-TR-035 (2002). http://www.sei.cmu.edu/publications/documents/02.reports/02tr035.html
25) OMG: UML 2.0 OCL Specification. http://www.omg.org/docs/ptc/05-06-06.pdf
26) Washizaki, H., Kobayashi, Y., Watanabe, H., Nakajima, E., Hagiwara, Y., Hiranabe, K. and Fukuda, K.: Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest, *Proc. 28th International Conference on Software Engineering* (*ICSE 2006*), pp.551–560 (2006).
27) Lee, K. and Lee, S.J.: A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development, *Proc. 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Paralle/Distributed Computing, and 1st ACIS International Workshop on Self-Assembling Wireless Networks* (2005).
28) ISO/IEC TR 9126-2: 2003, Software engineering—Product quality—Part 2: External metrics.
29) ISO/IEC TR 15504-2:1998, Process assessment—Part 2: Performing an assessment.

(平成 18 年 12 月 6 日受付)
(平成 19 年 5 月 9 日採録)

鷲崎 弘宜（正会員）

1976 年生．2003 年早稲田大学大学院博士課程修了，博士（情報科学）．2002 年同大学助手，2004 年国立情報学研究所助手．現在，同研究所助教．再利用と品質保証を中心としたソフトウェア工学の研究と教育に従事．他の活動に日科技連 SQiP 研究会運営小委員会副委員長，同研究会演習コース主査，情報処理学会ソフトウェア工学研究会幹事等．2004 年日本ソフトウェア科学会高橋奨励賞，2006 年情報処理学会 SES2006 優秀論文賞．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM，Hillside Group 各会員．

波木理恵子

1992 年株式会社オージス総研入社．現在，ソリューション開発本部組み込みソリューション部在籍．

福岡 呂之

株式会社オージス総研ソリューション開発本部組み込みソリューション部在籍．

原田 陽子

2006 年株式会社オージス総研入社．現在，ソリューション開発本部組み込みソリューション部在籍．

渡辺 博之

株式会社オージス総研ソリューション開発本部組み込みソリューション部在籍．