ソースコード静的メトリクス分析と単体テストアセスメント ——ソフトウェア品質向上への第三者評価による組織的取り組みの紹介

Static Code Metrics and Assessment for Unit Test Process —— Introduction of Organizational Efforts to Improve Software Quality

浦野隼人，沖汐大志

要　約　ソフトウェア品質管理の重要性に着目し，日本ユニシスでは，品質管理に関する社内標準を定め，各開発プロジェクトにてこれを実践することでソフトウェアの品質確保に努めてきた．これに加え近年，ソフトウェア品質の更なる向上へ向けて，ソースコードの定量的な測定・分析（ソースコード静的メトリクス分析）活動や，設計書の仕様に基づいた単体テストの妥当性に関するアセスメント（単体テストアセスメント）活動を，開発プロジェクト外の第三者的な組織が主体となって実施している．

これら二つの活動は，従来の品質管理手法を補完する取り組みとして有効なものであり，その直接的なアウトプットである分析・評価結果に基づき適切な是正策を講ずることが品質改善へと繋がる．また，これらの活動を通じた品質改善を効果的に実践する上では，現物確認の徹底・品質データの蓄積と再利用・品質を重視したプロジェクト運営・質の高い設計文書の整備といった基本動作が重要である．

Abstract　Nihon Unisys Limited focused on the importance of software quality control and has been improving the quality of software by developing the company standard for software quality control and practicing it in every development project. A third-party internal evaluation system, which provides the static code analysis and assessment for the unit-testing process, has been recently established and works company-wide.

Implementing appropriate measures based on outcomes of these two approaches improves the quality of software. Furthermore, to achieve the desired effect, it is important to perform the basic actions such as complete review of deliverables, accumulation and reuse of software quality analysis reports, the project management emphasizing on the quality of software, production of the high-quality software design documents.

1. はじめに

ICT（情報通信技術）が日常生活や経済活動の様々な場面で必要不可欠な存在として社会に広く普及している現在，情報処理システムには大規模で複雑性の高い仕組みの実現が要求されることから，システム開発の現場では，高いレベルの生産性と品質の確保が重要となる．

日本ユニシスでは，開発プロジェクトにおける開発規約・開発標準の一つとして品質管理標準を規定しており，設計からテスト工程における品質データを集計・分析・評価し，その評価

結果を是正策に繋げることで一定の品質を確保してきた．近年では，このような品質管理標準に準拠した活動に加えて，製造から単体テストの工程におけるソースコードの品質をより効果的に保証するための活動として「ソースコード静的メトリクス分析」と「単体テストアセスメント」を実施している．

ソースコード静的メトリクス分析は，ソースコードを定量的な指標で分析するツールを用いて評価する活動であり，単体テストアセスメントは，大量に作成されるソースコードの中から分析対象をサンプリングし，それに該当する設計書からテストケースを作成後，プロジェクト側が作成したテストケースと比較してアセスメントする活動である．

これらの活動の大きな特徴は，第三者が客観的にソースコードやテストケースを評価することにある．品質管理標準に準拠した品質管理は開発プロジェクトが主体となって実施するのに対し，ソースコード静的メトリクス分析と単体テストアセスメントの活動では，開発プロジェクトとは異なる組織が第三者的な立場から実施する．従来の品質管理標準による品質管理手法に加え，第三者組織がこれらの活動を通して，横断的に開発プロジェクトの品質管理に携わることで，ソフトウェア品質の底上げを図ることができると考える．

本稿では，ソースコード静的メトリクス分析と単体テストアセスメントの各々について，その活動内容を概説するとともに，活動によって得られる効果や有効な適用方法について，事例を交えて紹介する．

2. Java ソースコード静的メトリクス分析

ソースコード静的メトリクス分析とは，「ソースコードが構造化され，欠陥を含みにくいソフトウェアであること」を定量的に測る手段である．過去のシステム開発経験から，ソフトウェアに含まれる欠陥は，ソフトウェア全体に均一に含まれているのではなく，むしろ欠陥が多い「問題モジュール」と呼ぶべき一部に集中していることが知られている[1]．従って，ソフトウェアの品質を効率よく改善するためには，大量のソースコードの中から，品質上の問題があると推定される「問題モジュール」を短時間で抽出して重点的にレビューすることが効果的である．ソースコード静的メトリクス分析を適用することで，ツールによって取得した客観的な指標値を類似機能毎，あるいは開発チーム毎に相対比較することができ，大量のソースコードの中から特異な数値を示すソースコードを抽出できる．さらに，開発完了時に他プロジェクトとの比較を実施することで，保守や後続する 2 次，3 次開発に向けた基礎データとしても活用できる．

今日では，Java や C# など，さまざまなプログラミング言語でコーディングされたソースコードのメトリクスを測定するツールが有償，または無償で提供されている．日本ユニシスでは，これらのツールを評価検討し，個々の開発プロジェクトに対してソースコード静的メトリクス分析を実施するよう働きかけているが，本稿では，Java のソースコード静的メトリクス分析（以下，Java メトリクス分析）に焦点を絞って説明する．

2.1 活動紹介

2.1.1 ソフトウェア品質特性における Java メトリクス分析の位置付け

活動を紹介する前に，Java メトリクス分析とソフトウェア品質との関係について述べる．ISO/IEC9126 ではソフトウェアの品質属性を六つの特性に分類し，それぞれの特性について更に副特性に分割し定義している[2]．特性の一つである保守性の定義は次の通りである．

保守性：修正のしやすさに関する能力．修正は，是正もしくは向上，または環境の変化，要求仕様の変更及び機能仕様の変更にソフトウェアを適応させることを含めてもよい．

本稿で紹介するJavaメトリクス分析では，品質特性のうち，この保守性に関する評価・改善を通じたソフトウェア品質向上を目的としている．Javaメトリクス分析に使用している指標の中でも循環的複雑度（詳細は次項）は保守性に関する品質副特性のうち，解析性・変更性・試験性などに影響を与える．

2.1.2　Javaメトリクス分析に採用した指標

一般的なメトリクス分析では，ソースコードの構造や，ソースコード間の関連性などを表す測定指標が数十種類ある．Javaメトリクス分析では，品質の評価と改善に有効であり，かつ，一般的に用いられている指標を複数採用している．その中でもMcCabeの提唱する循環的複雑度[1][3][4]（以下，複雑度）は，Javaメトリクス分析の活動を通して最もソフトウェア品質の向上に効果的であったため，複雑度を例に挙げ説明する．

複雑度とは，分岐の数に基づいてソースコードの実行経路の複雑さを測る指標である．ソースコードの入口・出口・分岐をノード（N），ノード間をリンク（L）として捉え，以下の式で算出される．

複雑度（C）＝ L － N ＋ 2

複雑度の算出例として，ソースコード例とその有向グラフを図1に示す．図中の□がノード（N）を，○がリンク（L）を表す．図1では，ノード（N）＝6，リンク（L）＝7となり，複雑度（C）＝3となる．複雑度が高くなるにつれて，バグが発生しやすく，テスト項目数も増加し，ソースコードの保守性も低下すると言われている．

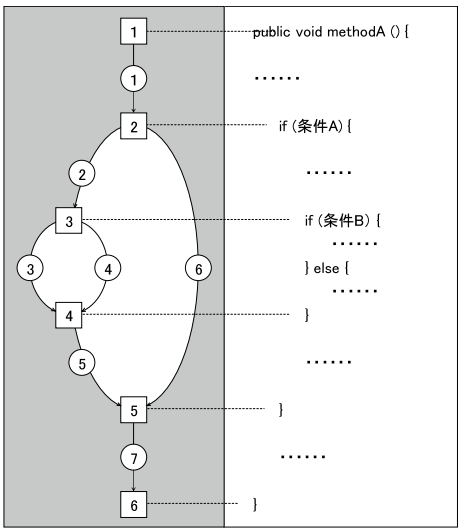

図1　複雑度

なお，McCabe の提唱している複雑度の一般的な目安は表 1 の通りであるが，複雑度を取り扱っている書籍によって若干の違いがある．

表 1　複雑度の目安

循環的複雑度(C)	リスク評価
C ≦ 10	単純な構造であり，リスクは高くない
10 ＜ C ≦ 20	やや複雑な構造であり，多少のリスクがある
20 ＜ C ≦ 50	複雑で，高いリスクがある
50 ＜ C	テストが不可能なほど複雑で，非常に高いリスクがある

2.1.3　Java メトリクス分析の実施内容

現在実践中の Java メトリクス分析活動では，最終的には測定・評価結果を報告書としてまとめ，プロジェクトに提出している．プロジェクトで報告書を活用する目的は測定を実施する工程により異なるが，これまでに実施した測定では，その活用目的は二つに大別できる．

目的の一つは，実装/単体テスト工程で，不具合のリスクが高く，テストが難しいといった品質上の問題を含むソースコードを抽出することによって，コードレビューを実施する対象を効率的に絞り込むことである．特に大規模システム開発の場合は，作成される膨大な数のソースコードのすべてに対して詳細にレビューするのは不可能であるが，単純にサンプリングしただけでは品質に問題のあるソースコードに当たる確率は低い．Java メトリクス分析を活用すれば，品質上問題がありそうな注意すべきコードを含むソースファイルを機械的に抽出することができる．測定は実装/単体テスト工程で一定期間を置きながら複数回実施し，プロジェクトでは，報告書で指摘された箇所を重点的にチェックし，必要があればソースコードを修正して品質を確保する．

もう一つの目的は，システム開発が終了したプロジェクトの総括をすることである．測定は開発終了時に実施し，報告書の内容を，後続する二次・三次開発や他の開発プロジェクトの品質管理に活用する．

2.2　事例紹介

Java メトリクス分析は 2007 年度に開始し，現在までに 5 件の大規模案件と，中規模・小規模案件を各 1 件の，合計 7 件のプロジェクトに対して適用してきた．以下ではこの中から，分析作業の実施タイミングや目的・回数の異なる二つの大規模プロジェクトの事例を挙げ，その概要を紹介する．一つは，製造/単体テスト工程にて定期的に測定した事例であり，もう一つはプロジェクト完了時に測定した事例である．

2.2.1　製造/単体テスト工程での定期的な測定

製造/単体テスト工程で定期的に測定したプロジェクトでは，5 ヶ月間に渡り月 2 回のペースで計 10 回のメトリクス分析を実施した．分析によって抽出された注意すべきソースコードを集中的にチェックし，修正の必要が認められた部分は次回の測定までに修正し，反映された結果の確認を繰り返した．

このプロジェクトでは，システムの規模が大きく，膨大な量のソースコードによって構成されるため，複雑度が 50 を越えるソースコードを重点的にコードレビューする対象として定め

た．分析結果に関する報告書では，該当するソースコードを複雑度の高い順に一覧表にまとめているため，プロジェクト側ではこの一覧表からレビュー対象とすべきソースコードを効率よく抽出することができる．

一覧表を基に重点的なコードレビューを実施した結果，即時に修正すべきバグが約3割のソースコードに含まれていた．この事実からも，複雑度はレビュー対象を絞り込むための指標として有効であることがわかる．

但し，複雑度の高低がソースコードの品質の良し悪しに必ずしも直結しないことに注意する必要がある．複雑度が高くなる原因が，「単に処理が構造化されていないため」なのか「業務ロジックそのものが複雑な処理であるため」なのかを見極めた上で品質を判断する必要がある．Java メトリクス分析は，品質に問題のありそうな対象を絞り込むための手段に過ぎないことを認識しなければならない．また，複数のシステム間でメトリクス値を比較する場合には，そのアーキテクチャやプログラミング規約に起因するプログラム構造の違いにも注意を払う必要がある．

ソースコードレビューの結果，単に処理が構造化されていないことが原因で複雑度が高くなっている場合は，リファクタリング[*2]により当該処理部分を構造化することで解決できる．他方，業務ロジックそのものが複雑な処理である場合には，開発プロジェクトが品質管理標準に従って収集しているレビュー実施時間や単体テスト件数などの品質データと Java メトリクス分析による評価データを組み合わせて検証し，是正の要否を判断する必要がある．

表2は，プロジェクトにて収集していた品質データと Java メトリクス分析による評価データの対比である．プロジェクトにて収集していた品質データの項目は，表2のプログラム仕様検証実績時間～単体テスト件数の項目である．複雑度がメソッド単位で測定される値であるのに対し，プロジェクトではコンポーネントというメソッドよりも粒度が大きい単位で品質データを管理していたため，「そのコンポーネントに含まれる最大の複雑度」や「複雑度が 20 を超えるメソッドが1コンポーネントあたりいくつ含まれているか」という集計処理をしている．表2は，ComponentD の複雑度が高いにもかかわらず，プログラム仕様検証実績時間が他のコンポーネントと同等であり，不足していると推定できる例である．

表2　他の品質データとの分析

コンポーネントID	Javaメトリクス分析による評価データ				プロジェクト側の品質データ				
	複雑度最大	複雑度20を超えるメソッドの数	メソッド数	ステップ数	プログラム仕様書検証実績時間	プログラム仕様書指摘件数	コード検証実績時間	コード不具合件数	単体テスト件数
ComponentA	12	0	5	112	7.2	4	3.7	0	14
ComponentB	17	0	5	291	8	12	7.1	2	28
ComponentC	17	0	3	110	3.6	7	3.1	1	19
ComponentD	72	1	2	202	8.6	5	11.1	5	77

2.2.2　プロジェクト完了時の測定

開発完了後に測定を実施したプロジェクトでは，Java メトリクス分析のデータをプロジェクト総括の参考データとして利用し，類似の後続プロジェクトでの品質データの目安としている．この事例では，複数プロジェクト毎の比較や，開発作業を発注した協力企業毎の比較を実施した．

1） プロジェクト毎の比較

図2は4件の大規模プロジェクトについて，複雑度の範囲を4段階で区分したときのシステム全体の複雑度の内訳を比較している．すべてのプロジェクトにおいて複雑度が1～10のメソッドが全体の85%以上を占め，また複雑度が50を超えるメソッドは，ほぼ1%以下に収まっていることがわかる．今後，Javaメトリクス分析を実施する上で，類似したプロジェクトと異なる傾向の結果が得られた場合には，プロジェクトに対して警告する必要がある．

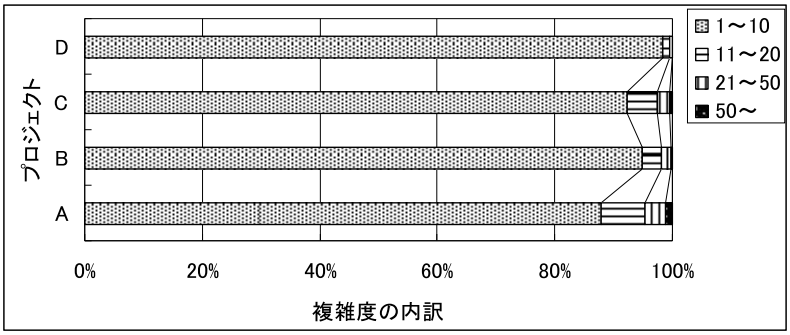

図2　各プロジェクトの複雑度内訳

2） チーム間の傾向比較

大規模な開発プロジェクトでは，システムをいくつかのサブシステムに分割し，サブシステムに対してチームを割り当てて開発を進める．図3は，各担当チームが開発したソースコードに対して1メソッドあたりの複雑度の平均値を算出したものである．この図では，担当チームBの平均複雑度の突出が目立つ．このデータだけでは“担当チームBの作成したソースコードは品質が悪い”と直接判断することはできないが，担当したサブシステムの複雑性や生産性，その他の品質データと組み合わせて判断することで，次回以降のチーム構成を考える上での材料として活用することができる．

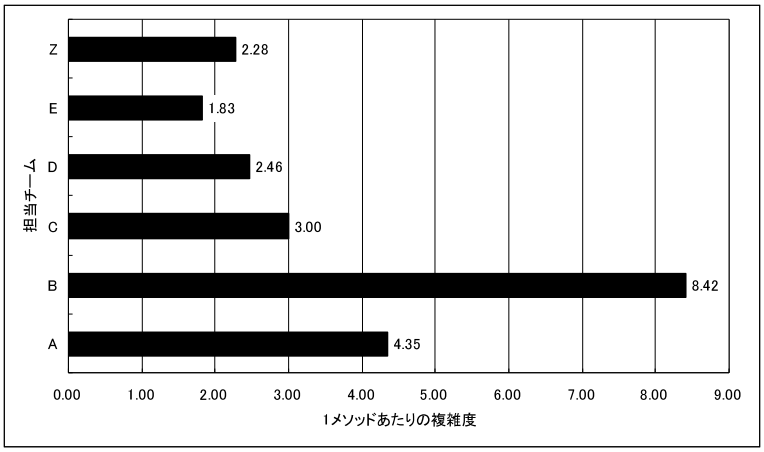

図3　1メソッドあたりの複雑度（担当チーム別）

2.3 Javaメトリクス分析の適用上のポイント

Javaメトリクス分析の結果だけでは品質の良し悪しを判断することはできないが，分析によって得られた結果を有効に活用し，「現物確認の徹底」と「品質データの蓄積と再利用」を遵守することによって，初めてシステム開発でのソフトウェアの品質向上という目的に対して成果を上げることが可能となる．Javaメトリクス分析を適用して品質を向上するために必要なこれらのポイントを次に示す．

1)　現物確認の徹底

品質管理において陥りやすいのは，分析結果や管理のデータばかりを見て成果物の品質状況を確認しなくなることである．重要なことは品質データの管理をした上で，現場で現物を見て，現実を見極めること（三現主義）を確実に実施することにある．これは品質管理の鉄則であり，Javaメトリクス分析を適用する際にも例外ではない．大規模プロジェクトの事例で紹介したように，Javaメトリクス分析のメリットは大量のソースコードから重点的にレビューする必要があるソースコードを効率よく抽出することである．しかし，抽出された部分の現物を確認し，必要な是正措置を講じなければ，品質の向上には結びつかない．

2)　品質データの蓄積と再利用

品質の向上を実現するには，測定したデータを蓄積し再利用することが重要であり，そのための仕組みを用意しなければならない．本活動によって測定した品質データはデータベースに格納することで，過去に測定した他のプロジェクトの品質データと比較して傾向を判断することができるようにしている．品質データを継続的に蓄積することで，現状よりも様々な角度からの分析も可能になり，比較の精度も向上する．

3. 単体テストアセスメント

日本ユニシスの品質保証部門では，品質保証活動の一環として，プロジェクトに対する第三者評価を実施しており，プロジェクトの進捗や標準開発プロセスの遵守，そしてレビュー密度やテスト密度のような品質管理のメトリクスを評価している．しかし，品質管理面の評価が悪くない場合でも結合テスト工程以降で欠陥が多数検出されるなど，最終的なテストの段階で問題が表面化することがあり，その原因は単体テストが不十分であることも少なくない．これは，テスト密度のような品質管理のメトリクスは統計的な指標であり，個々のモジュールや開発作業の品質そのものを表していないためである．指標が基準値を満たしていても，必要なテストケースを網羅しているとは限らない．

単体テストアセスメントでは，テスト密度のようなメトリクスだけでは把握しきれない，テストそのものの品質を評価している．本章では，単体テストアセスメントの活動内容と事例を紹介し，評価結果に基づいた改善の方法について提言する．

3.1 活動紹介

3.1.1 アセスメント対象の選択

単体テストアセスメントの対象となるプロジェクトは，大規模プロジェクト，及び社内の品質保証レビューにて特に品質に注意する必要があると判断されたプロジェクトである．

単体テストアセスメントでは，システム全体をすべて評価するのではなく，サンプリングして評価する．サンプリングは，システム全体の単体テスト品質を把握するため，主に網羅性と

リスク（使用頻度が高いなど）を考慮して実施している．図4にその概要を示す．

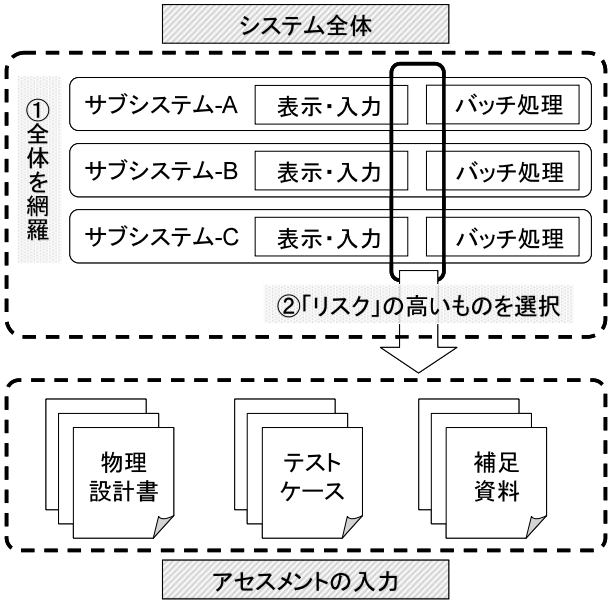

図4　アセスメント対象のサンプリング

まず，網羅性の観点では，サブシステムや処理形態（表示・入力処理とバッチ処理）を網羅するようにサンプリングする．大規模な開発プロジェクトではサブシステムを単位として協力企業に開発作業を発注することが多く，サブシステムとサブシステム内の処理形態を網羅すれば，協力企業も網羅することになる．

次に，リスクの観点では，プロジェクト側の判断により重要性の高いもの（使用頻度が高い，本番での障害発生時の影響が大きい，等）を優先して選択するケースや，ソースコードが存在する場合は，循環的複雑度（Javaメトリクス分析で用いた指標）の高いモジュールを選択するケースがある．

また，網羅性やリスク以外の観点として，開発スケジュールなども考慮している．プロジェクトが大規模になると作成する成果物（たとえば物理設計書）の数も多くなり，その完成時期にずれが生じる．なるべく早期にアセスメントを実施することができれば，並行する作業にもその結果を展開し，手戻りを極小化することができ，アセスメントによる品質改善の効果も大きくなる．こうした理由から，同等の重要度であれば，早い段階で成果物が揃うモジュールを選択することもある．

3.1.2　アセスメントの作業内容

単体テストアセスメントでは，ブラックボックス技法（デシジョンテーブル，同値分割，境界値分析，エラー推測など）[6][7][8]を用いた機能性に着目したアセスメントを実施している．

アセスメント作業の流れは，図5に示す通り，以下の手順で実施する．

①　入手した設計書を基にテスト設計を行なう．

②　テスト設計時には，設計書の不備（情報不足や曖昧さなど）も検証する．

③　作成したテストケースと，入手したテストケースを対比して，テストケースの過不足や不備（条件や期待値が曖昧など）を洗い出す．

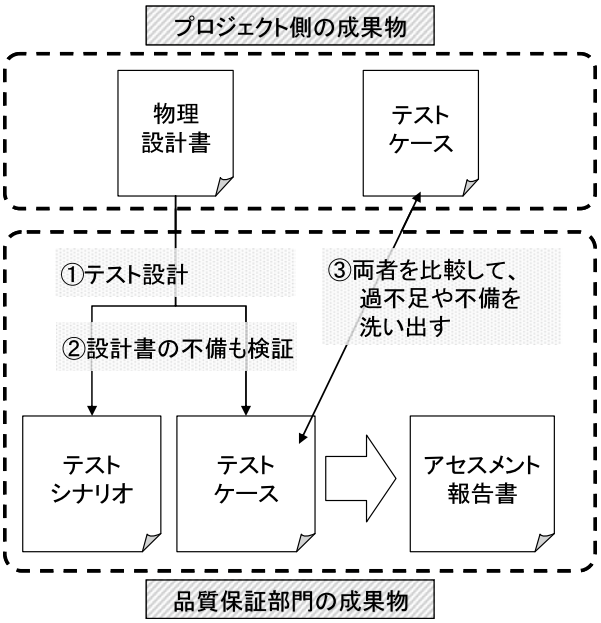

図5　アセスメントの作業内容

最初に，プロジェクトから入手した設計書を基にして，品質保証部門がテスト設計（テストケースの作成）を行なう（図5の①）．このとき，設計書の不備（曖昧，矛盾，考慮漏れ）についても評価している（図5の②）．テストケース不足が設計書の考慮不足に起因することも少なくないため，設計書の評価は，品質保証部門が実施する単体テストアセスメントのポイントの一つである．実際，プロジェクトの終盤やリリース後に検出される障害の中には「単体テスト工程で発見するべき」と判断される障害もあるが，テストだけの問題ではない場合もある．設計書の評価は，このようなことを未然に防止するためにも有効である．

次に，プロジェクトから入手したテストケースと，品質保証部門が作成したテストケースとを突き合わせ，その妥当性を評価する（図5の③）．具体的には，まず，入手したテストケースに過不足がないかを評価する．テストケースの過不足がない場合でも，テストの入力条件や期待結果の記述が曖昧であったり，説明が不足したりしていないかも評価する．このように，存在するテストケースの評価もポイントの一つである．たとえば，テストケースに定義された期待結果が「設計書の記述通りになること」のように曖昧な記述だと，テスト結果の確認に漏れや不足が生じ，単体テストで障害を検出できないことがある．期待結果には，確認する項目やその値や出力先などを，なるべく具体的に記述する必要がある．

このようにして評価した内容をアセスメント報告書としてまとめ，プロジェクト側にフィードバックしているが，報告書では，問題点を具体的に説明し，改善するための方法を示すようにしている．単体テストアセスメントでの評価内容は，図6にまとめる通り，プロジェクト側の成果物である設計書とテストケースについて，曖昧な点や考慮漏れを指摘して妥当性を評価し，さらに設計書とテストケースの間での一致性や網羅性も評価する．

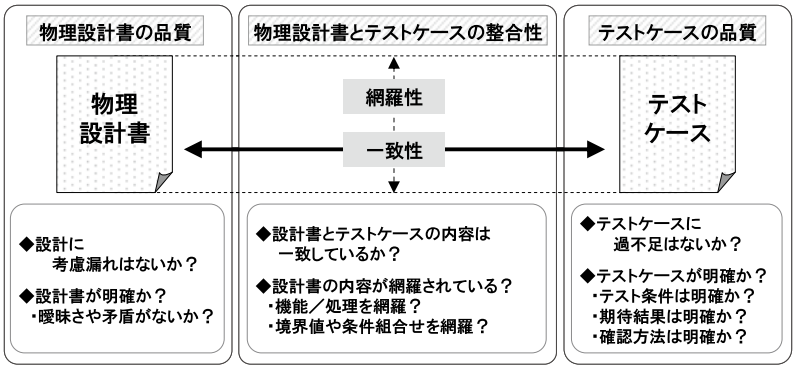

図6　アセスメントでの評価内容

3.2　事例紹介

本節では，これまでの単体テストアセスメントの事例を紹介する．

3.2.1　アセスメントの効果

あるプロジェクトでは，単体テストの品質改善を目的に，全協力企業を対象にして，最大4回のアセスメントを実施した．アセスメントで指摘した項目の中には，評価対象外のもの（別途テスト済みのものや，結合テスト以降でテスト予定のものなど）が含まれているため，プロジェクト側と対面にて対応の必要性を確認した上で，協力企業ごとに，どのような問題点があるのかを整理した．

同じ協力企業の中でも，経験年数やスキルが異なる複数の担当者がおり，問題点の量や傾向には多少の差が見られたが，全体的に見ると協力企業ごとに傾向や特徴が見られた．その特徴の構造を表3に示すような形式にまとめた．物理設計書の問題点は記述の曖昧さや矛盾などである．テスト設計の問題点は，テストケースの単純漏れや，テストデータの選択が不適切なことなどである．

表3　アセスメント結果（対応必要数集計）

	物理設計書			テスト設計		
協力企業／機能	問題1	問題2	問題3	問題1	問題2	問題3
A社／a機能	1件					
B社／b機能				2件	1件	
C社／c機能				1件		
D社／d機能		5件	4件		2件	5件
E社／e機能						
F社／f機能				1件		

これにより，

- どの協力企業，もしくは，機能の改善が必要であるか
- どのような指導（設計書のレビュー，テスト技法など）が有効であるか

を把握することができ，問題の多い協力企業（本事例ではD社）に対しては，指導と改善効

果確認のためのアセスメントを継続することは有効であった．

このプロジェクトでは，D 社に対して，4 回のアセスメントを実施することになった．アセスメントそのものだけでなく，複数の協力企業が横並びで評価されるという緊張感や，プロジェクト内での横展開や指導の効果もあり，対象企業のテスト技術や意識も次第に向上していった．そして，最終的には，品質保証部門が設計したテストケースと比較して，機能的なテストの漏れは見られなくなった．

3.3　単体テストアセスメントのポイント

前述した事例を通して，単体テストアセスメントによって品質向上の効果を得るプロジェクトには共通した特徴があることがわかった．それは，テストも含めたプロジェクト全体の方針や戦略が明確で，適時見直しが行なわれており，開発の標準プロセスを遵守するなど，基本を忠実に守っているということである．

以下に，単体テストアセスメントで効果を上げるために必要なポイントを示す．

1）　改善プロセスの運営

単体テストアセスメントの効果を確実にするには，利用するべきテスト技法や，作成するテストケースの粒度を記述したテスト規約を作成して，テスト設計を開始する前にプロジェクト全体に展開するとともに，次のような PDCA サイクルを徹底する必要がある．

- テスト規約による明確なゴール設定と周知徹底（Plan）
- テスト規約の遵守（Do）
- テスト規約を満足しているかの受入チェック（Check）
- アセスメントでの指摘事項への確実な対応と，プロジェクト内での横展開（Act）

品質保証部門がアセスメントを精緻に実施し，数多くの指摘項目を示したとしても，その対策が実行されなければ意味がない．実際のプロジェクトでは，プロジェクト開始時にテスト規約や受入基準を明確にしていないと，コストや期間の制約により，対策を実行できないことがある．また，アセスメントはサンプリングしたものが対象なので，その対策をプロジェクト全体に横展開しなければ，部分的な改善にとどまってしまう．

テスト設計のみをどれほど充実させても，それだけで製品の品質は確保できないことは明白であり，単体テストの品質と，プロジェクト全体の質は強く関係している．製造しているソフトウェアだけでなく，プロジェクトを確実に遂行するプロセスも重要である．図 7 に，プロジェクトの運営面の視点でのプロジェクトと協力企業，品質保証部門で取り組むべきポイントをまとめる．

図7　アセスメント効果を確実にするポイント（プロジェクト運営）

2)　質の高い設計書によるアセスメント実施

アセスメントの前提となるシステムの要件定義や設計書も重要なポイントの一つであり，以下に示す条件を確認しておく必要がある．

- 要件定義から物理設計まで一貫性があり，要件に漏れ・抜けがないこと．
  （上位の設計書との不一致がないこと）
- ブラックボックスのテスト設計は，「確定」した物理設計書を基にしていること．
  （機能の追加や修正が収束した状態の設計書でテスト設計すること）

完全ではなくとも，これらの条件を満たしていないと，アセスメントの意味が薄れてしまう．単体テストアセスメントで参照する設計書に機能や考慮の漏れがあると，テストケースの過不足や不備を検出できず，テストが確実に実施されたとしても，システムの要件を充足することができない．また，物理設計書が不確定のまま製造を進めている場合，プロジェクト側の設計書とテストケースとが一致せず，さらに，品質保証部門が作成したテストケースと対比することも難しくなってしまう．対比できない部分が多い場合は，アセスメント不能ということになる．

ソフトウェアの品質を維持・向上させるためには，これまで以上に「不備の少ない」設計書の記述や，「正しい」テスト設計に対する意識を高めて実践することが重要であり，こうした取り組みは，開発・テストでの生産性を向上させることにも期待できる．

4. お　わ　り　に

本稿では，第三者による客観的な視点での品質保証活動として，Java のソースコード静的メトリクス分析や，単体テストアセスメントによる品質の評価を紹介し，それを効果的に実施するためのポイントを示した．Java ソースコード静的メトリクス分析では，測定したデータに対して現物を確認して適切に対応すること，また，単体テストアセスメントでは，指摘事項の対策を確実に実行していくプロジェクトの運営面での改善プロセスや，要件定義から単体テストに至る各工程での設計情報の品質が，これらの品質保証活動を効果的にする上での重要なポイントである．

本稿で紹介した二つの活動は，それ自体がソフトウェアの品質を直接的に改善するものでは

なく，その目的や適用のポイントを正しく理解し，適切な是正措置や品質改善プロセス運営を実施することで初めて効果が得られるものである．これらの活動を，従来の品質管理手法を補完する実践的かつ有効な取り組みの一つと位置づけ，更なる強化・発展を図りたい．

＊1　ISO/IEC9126．国際標準となるソフトウェア品質モデル．1991年に策定．1994年に日本語に翻訳された「JIS X 0129」として発行された．
＊2　プログラムの振る舞いを変えることなくソースコードを変更すること．

参考文献
［1］ Caper Jones，鶴保　征城，富野　壽，ソフトウェア開発の定量化手法　第2版，構造計画研究所，共立出版，1998年12月，P230，354，388
［2］ 日本工業標準調査会，http://www.jisc.go.jp/
［3］ Caper Jones，富野　壽，ソフトウェア品質のガイドライン，構造計画研究所，共立出版，1999年4月，P99～100
［4］ 大塚俊章，荻野富二夫，ソフトウェアテスト技術，ユニシス技報，日本ユニシス，Vol.27 No.2 通巻93号，2007年8月
［5］ 町田欣史，ソースコード・メトリクスを用いた品質定量化モデルの提案，開発の現場，翔泳社，Vol.10，2007年10月，P186～192
［6］ G. J. Myers他，ソフトウェアテストの技法第2版，近代科学社，2006年8月
［7］ Lee Copeland，はじめて学ぶソフトウェアのテスト技法，日経BP社，2005年11月
［8］ Rex Black，ソフトウェアテスト実践ワークブック，日経BP社，2007年1月

執筆者紹介　浦　野　隼　人（Hayato Urano）
2005年日本ユニシス(株)入社．金融案件を中心に各業種のシステム開発プロジェクトに対して，提案支援・技術支援を手掛ける．現在，総合技術研究所OSSセンターに所属し，Javaソースコード静的メトリクス分析を担当．

沖　汐　大　志（Motoji Okishio）
1990年日本ユニシス(株)入社．CAD/CAMシステムの開発，適用支援に携わった後，2007年より品質保証に関する作業に従事し，現在は品質保証部品質技術室に所属．