定量的ソフトウェアテスト完了判断基準の一考察

堀 明広† 仲 孝浩‡ 向井 清‡ 金子 喬‡

†パナソニックモバイルコミュニケーションズ（株） 〒224-8539 横浜市都筑区佐江戸町 600 番地
‡シャープビジネスコンピュータソフトウェア（株）〒639-1186 奈良県大和郡山市美濃庄町 492 番地
‡住商情報システム株式会社 〒104-6241 中央区晴海 1 丁目 8 番 12 号
‡株式会社 日立システムアンドサービス 〒212-0058 神奈川県川崎市鹿島田 890

E-mail: †hori.akihiro@jp.panasonic.com, ‡naka@sbc.nara.sharp.co.jp, ‡mukai@jpta.scs.co.jp, ‡ta-kaneko@hitachi-system.co.jp

あらまし ソフトウェアのテストに明快な終わりはない。テストでそのプログラムにバグがあることは証明できるが，バグが存在しないことは証明できない。しかし，経済的な理由からテストを終わり無く継続することはできず，何かをきっかけに，テストを継続することをあきらめなければならない。テストの「完了」を適切に判断するにはどう考えるべきか，また，定量的な尺度を用いてテストの完了判断を行うには何が必要なのだろうか。本稿では，ソフトウェアテストの目的を４つ定義してそれをゴールに据え，メトリクスを設計・検討するためのＧＱＭパラダイムを利用して考察した結果，１つの前提条件と７つの基準を導出した。検討結果を一提案として発表する。

キーワード ソフトウェアテスト，メトリクス，完了判断基準

A consideration of quantitative standard for software testing termination judgment

Akihiro Hori† Takahiro Naka‡ Kiyoshi Mukai‡ Takashi Kaneko‡

†Panasonic Mobile Communications Co., Ltd. 600 Saedo-cho, Tsuzuki-ku Yokohama-city, Kanagawa 224-8539, Japan
‡Sharp Business Computer Software Co., Ltd. 492, Minoshou-cho, YamatoKooriyama-city, Nara, 639-1186, Japan
‡Sumisho Computer Systems Corporation 1-8-12, Harumi, Chuo-ku, Tokyo 104-6241, Japan
‡Hitachi Systems & Services,Ltd. 890, Kashimada, Saiwai-ku, Kawasaki-city, Kanagawa, 212-0058, Japan

E-mail: †hori.akihiro@jp.panasonic.com, ‡naka@sbc.nara.sharp.co.jp, ‡mukai@jpta.scs.co.jp, ‡ta-kaneko@hitachi-system.co.jp

Abstract There is no clear end in software testing. Testing can provide the proof of program has bugs inside but does not provide proof of that programs have no bugs at all. Testing will not keep continue forever beyond the economical reason and testing must be terminated by some triggered event. How do we make decision about software testing termination adequately? What kind of quantitative measurements are required for testing termination? In this paper, we define objectives of software testing in four ways. We set GOALs for them using GQM paradigm in order to design software metrics and we break down the Goals into details. As the result of using this approach, we have derived one prerequisite and seven standards for software testing termination decision. We would like to show our consideration and recommendation as one of new proposal about software termination decision.

Keyword Software test, Metrics, Standard of termination judgment

1. はじめに

1.1. テストの制約と完了判断

ソフトウェアのテストに明快な終わりはない。テストでその対象とするプログラムにバグがあることは証明できるが，バグが存在しないことは証明できないことは周知のとおりである。しかし，経済的な理由からテストを終わり無く継続することはできず，どこかで何かをきっかけに，テストを継続することをあきらめなければならない。上記を踏まえ，テストの「完了」を適切に判断するにはどう考えるべきか，また，定量的な尺度を用いてテストの完了判断を行うには何が必要か，これが本稿の主眼である。

1.2. 対象とするテストフェーズの明確化

本稿で考察の対象とするテストフェーズは，顧客にリリースする直前のものを対象とする。ここで言う顧客とは，エンタプライズ系ソフトウェアではエンドユーザーやシステム管理部門が該当する。組込み系やパッケージでは，ソフトウェアを製品に載せ込んで製造

する工場等が該当する。

開発プロセスのモデルは，ウォーターフォール型，プロトタイプ型，反復型などいくつか存在するが，何れの開発モデルも，開発フェーズは下記のように大別できる。

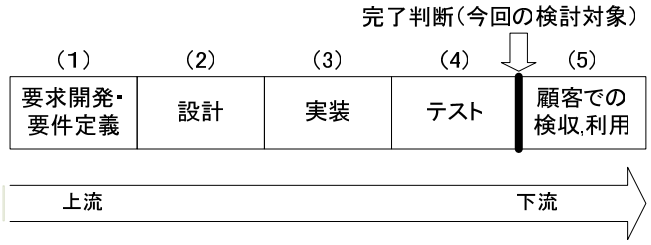

図 1 開発フェーズと検討対象

上記(1)～(5)は開発モデルによってオーバーラップしている箇所もあるが，テストがソフトウェア開発の最下流のフェーズに位置する点で共通している。

テストは「verification & validation」（検証と妥当性確認）の二面を持っているが，本稿では上記(4)に着目して"verification"を主な対象に据え，顧客にリリースする直前のテストフェーズの完了判断を定量的に行うための考え方と具体的な方策，前提条件について考察する。これらを一提案として発表する。

2. テストの目的と完了判断基準

「テストを完了した」とは，何を持って言うことができるだろうか。本稿では，「テストで目的としていたものを達成していると言える状態」と定義する。

上記 1.2.(4)のテストフェーズを対象とすると，テストの目的は下記の４つが挙げられる。

- (G1)開発中のソフトウェアに潜むバグを検出し，修正する。
- (G2)そのソフトウェアに求められる要件（機能，性能等）を満足していることを確認する。
- (G3)テストを実施した結果検出されるバグの発生状況から，未だソフトウェア製品に潜むバグの存在を推測し品質の成熟度を判断する。
- (G4)バグの作り込み要因と，より上流の検証フェーズでの流出要因を分析することにより，開発プロセス上に存在する問題を特定し対策する。

上記(G1)(G2)は，従来から言われてきているテストの大きな目的である。(G3)(G4)は，テストをその製品と開発プロセスの成熟度を判断するための情報を収集してアクションにつなげる活動として捉えているものである。

テスト完了判断基準は，そのテストの目的を達成しているか，判定する材料と言い表すことができる。

本稿では，上記のテストの目的をソフトウェアテストのゴールと捉え，メトリクスを設計・検討するためのＧＱＭパラダイムに当てはめ，定量的ソフトウェア完了判断基準を構築するアプローチをとった。

3. ＧＱＭパラダイムとは

ＧＱＭ(Goal/Question/Metric)パラダイムは，Basiliらによって提案されたソフトウェア測定および評価のための総合的な枠組みである。ＧＱＭパラダイムに基づいて測定を行う際には，まず，測定の目標(Goal)を明確にする。次に，その目標の評価方法または達成方法を，質問(Question) の形式で記述する。最後に，その質問に定量的に答えるための測定量(Metric)を明確化する。目標，質問および測定量の階層構造をＧＱＭモデルとして表すことにより，ソフトウェア測定および評価の目的を明確化することができる。ＧＱＭパラダイムの適用事例はこれまでに数多く報告されている。それらの多くは従来からプロセス改善に取り組んできた企業であるが，ＧＱＭパラダイムの導入により，ソフトウェア評価手法と測定量との関係がより明確になり，ソフトウェア測定活動が定着したとされている。

4. ソフトウェアテストの Goal と Question

ソフトウェアテストの目的を起点に，ＧＱＭパラダイムに則ってソフトウェアテストに求められるアクティビティを列挙して考察した結果，表 1が得られた。これら各々について考察し，Questionを定量的に把握するためのメトリクスを得ると同時に，得られたメトリクスを完了判断で用いる方法を検討した。次章以降にその検討内容と導出したソフトウェアテスト完了判断基準を記す。本稿の最後には，全体の本研究の総括と，今後の課題，展望を記す。

表 1 テスト完了判断 Goal-Question 対応表

G1:開発中のソフトウェアに潜むバグを検出･修正する
G1Q1:妥当な質と量のテスト項目を設定する
G1Q2:設定したテスト項目を実行する
G1Q3:検出したバグを修正する
G1Q4:検出したバグを正しく修正していることを確認する
G2:そのソフトウェアに求められる要件(機能,性能等)を満足していることを確認する
G2Q1:要件を網羅するテスト項目を設定する
G2Q2:プログラムの全機能を動作させるテスト項目を実行する
G3:テストを実施した結果検出されるバグの発生状況から，未だソフトウェア製品に潜むバグの存在を推測し品質の成熟度を判断する
G3Q1:予め計画したバグの数だけ，バグを検出できている
G3Q2:信頼度成長曲線が収束している
G4:バグの作り込み要因と,より上流の検証フェーズでの流出要因を分析することにより，開発プロセス上に存在する問題を特定し対策する
G4Q1:バグの発生状況を，開発プロセスにフィードバックしている

5. ソフトウェアに潜むバグを検出・修正

5.1. G1Q1:妥当な質と量のテスト項目を設定

5.1.1. 前提条件

テストを効果的・効率的に実施するには，質の良いテストを計画・設計することが必要になる。テストフェーズが完了した後であれば，テストフェーズ以降に発生した不具合の発生状況によってテストの質を検証することは可能であるが，テストを開始する前の時点でテストの“質”を判断することは難しい。

本稿では，網羅すべきテストのカテゴリを計画し，それに沿ってテスト設計が行えているかどうかを判定することで，テストの質を判定するという考えを取った。定量的尺度は，テスト項目の量で換算する。ここで言うテストのカテゴリとは，例えば，正常ケース・異常ケース・限界ケース，機能テスト・構成テスト・限界テスト等のことある。これらのカテゴリは，採用する観点で様々なものが考えられるため，どれか一つに特定する必要はない。ポイントは，どこに着目したテストを行うかを予め計画して，それに沿ってテスト設計を行うことと，その実施状況を表す尺度として，テスト項目数を採用していることである。

前述しているとおり，テストのリソースは有限であるため，意図したとおりにテストを設計し，実行することが困難な状況も考えられる。

このような場合にはテストを間引く必要性に迫られるが，何を間引くかは，その製品の特性，開発プロセスの実施状況等を鑑み，根拠に基づいて合理的に行うことが肝要である。

5.1.2. テスト項目の質と量

テスト項目の量は，対象とする製品の規模に見合ったものでなければならない。

また，単純に，テスト項目数だけに着目するのは危険である。例えば正常ケースだけをむやみにテスト項目を増やして量を稼ぐような状態を識別できないためである。

そのため，上記で示したようなテスト計画・設計の指針に則り，それぞれのカテゴリで何件のテスト項目を設定したか，明らかにする必要がある。

よって，テスト項目の量は，テストカテゴリごとに，ソフトウェア規模との対比で表すことが必要である。例えば，限界ケーステスト項目数／ソフトウェア規模（ESLOC，FP 等）などが考えられる。

テスト計画を策定してテスト設計を行う際は，そのソフトウェア製品の特性や開発プロセスの実施状況，テストしない領域に対するリスク，テストリソースのバランスを取り，実施するテストをデザインすることが必要である。

［本節で導かれたメトリクスと付随する情報］

- カテゴリ別テスト件数 及び プログラム規模との比率
- 実施しないテストカテゴリの明確化と，そのリスクの度合い・対応策

5.2. G1Q2:設定したテスト項目を実行

本節で着目するのは，実際に実施したテスト項目数である。

予め計画・設計したテスト項目が，テスト環境や時間等の制約で全て実施できないこともある。

これは実質的にテスト設計の段階で間引くことと同義であるため，テストを実施しなかったことに対するリスクを検討する必要がある。

［本節で導かれたメトリクスと付随する情報］

- 設定したテスト項目数と実際に実施したテスト項目数の比率（全体とカテゴリ別）
- 実施したテスト項目数と，プログラム規模との比率
- 実施しなかったテストの明確化と，そのリスクの度合い・対応策

5.3. G1Q3:検出したバグを修正

本節で着目しているのは，テストの終盤で発生したバグの重要度とバグ修正のトレードオフである。

言うまでもなく，テストで発見したバグは全て修正することが原則であるが，テストの終盤になって検出されたバグの対処方法には格段の注意を要する。バグの修正にはデグレードの危険が常につきまとう。よって，何時でもバグの修正は細心の注意を払って行う必要があるが，テスト終盤では納期のプレッシャーからの焦りで,不適切に修正してしまう危険がより高まる。更には，バグ修正後の再テストやリグレッションテストも十分に行えない状況が生じることもある。

ここで，バグの重要度を加味して考察する。テスト終盤に発見したバグが重大なものであれば，何が何でも修正しなければならない。しかし，重要度が軽微なものであれば，それを修正するメリットと，修正のコスト・リスクのトレードオフを検討する必要が生じることもある。それがユーザーにとって影響が小さい部類のものであれば,デグレードの危険を冒して修正し，その検証で納期が延びるよりも，関係者の合意を得た上で利用のフェーズに移行する方が目的に沿う場合がある。よって，検出したバグは，無条件に全て修正しなければならないというのは，あまり現実的でない。

しかし，テスト終盤で発生したバグの現象が軽微であったとしても，その原因を特定していることは必須である。現象的には軽微なものであっても，その原因はクリティカルなところにあり，同一のバグが別の条件で極めて重大な現象を引き起こす可能性もあるからである。バグの原因を特定せず，現象面だけで判断す

ることは極めて危険である。

更に，重要度の定義についても，予め定義し，プロジェクト計画を立案する段階で関係者と合意する必要もある。重要度の定義は,一種の価値観の表れであり，それが発露した際のユーザー・環境に及ぼす影響度，発生頻度（確率）を勘案して設定する必要がある。何が重要かそうでないか，それを判定する基準は，そのソフトウェアの目的・特性・環境・人によって異なるものである。ある一つのバグの現象をとってみても，ある製品では"軽微"と判断できるものが，別の製品ではその目的と照らし合わせて"重大"と判定すべきものに変わる可能性もある。

バグの重要度を割り振る基準を策定したとしても，誰がそれを判定するかという問題が残る。こと細かく基準を策定することはできたとしても，微妙な判断が必要な場合もあり得る。よって，例えば SCCB のような，バグの影響度と発生頻度を踏まえて判断する仕組みが必要となってくる。

以上の考え方は，検出したバグを全て修正するという，いわば 100 点満点を基準にするのではなく，"合格点"を合理的に設定する，ということが根底にある。もちろん，未修正のバグが存在することを，特別採用として顧客を含む関係者と予め合意するというように，一定のプロセスを踏むことは不可欠である。

本節では，テスト終盤で検出したバグについて重要度を軸に考察したが，ソフトウェアの品質の成熟度という面では，また別の考察が必要である。観念的に言い表すと，90 点でも合格とする基準を作ったとしても，差し引いた残りの 10 点が本当に 10 点で済んでいるかどうかは，別の観点で検証する必要があるということである。これについては G3Q2 にて考察する。

［本節で導かれたメトリクスと付随する情報］

- バグ発生件数（修正済みの件数と未修整の件数）
- 未修整バグの各々の重要度と，使用者や他システム等への影響
- 重要度を判定する基準と，その基準に則って判定する仕組み

5.4. G1Q4:検出したバグを正しく修正しているか

本節で着目しているのはデグレードである。デグレードはあってはならないが，現実には起こりうる。発生させてしまったデグレードの件数を計測することで，バグ修正プロセスの品質と，プログラムの変更容易性を伺い知ることができる。

デグレードの発生頻度や部位等の傾向を分析することで，プロジェクトに内在する問題を明らかにし，対処することも可能になる。また，これらの分析結果は，回帰テストの範囲を判断する材料に用いられることも期待できる。

よって，バグを検出してその原因を解析する際には，それが新規に検出したバグなのか，別のバグ修正で悪影響が及んだことで発生したものなのかを切り分けし，それを記録する取り組みが必要となる。

［本節で導かれたメトリクスと付随する情報］

- デグレード発生件数
- デグレード発生傾向の分析結果

6. G2:要件を満足していることを確認

6.1. G2Q1:要件を網羅するテスト項目を設定

本節では，ソフトウェアの再利用に着目している。

ソフトウェアの再利用は，品質・生産性において重要な技術であることは言うまでもない。再利用したコンポーネントを，何も手を加えていないからといってテストを省略することは危険である。

- 再利用コンポーネントを，システムに的確に組み込めているかどうかは，そのコンポーネント単体の品質とは別の問題
- 新規開発部分・改造部分に誤りがあり，それが再利用するソフトウェアに悪影響を及ぼす可能性
- 再利用するコンポーネントに論理的な誤りがあり，改造したことでバグが顕在化する可能性
- 再利用するコンポーネントにバグが存在していない保証は，どこにも無い
- 機能の実装漏れを検出する最後の砦はテストである

上記のように，たとえ信頼性が高いと思われるコンポーネントを再利用していたとしても，修正がごく僅かで容易に見えたとしても，テストリソースが限られていたとしても，全ての要件は少なくとも一回はテストする必要性はあるのである。

［本節で導かれたメトリクスと付随する情報］

- 要件の数，粒度
- テスト項目数

6.2. G2Q2:プログラムの全機能を動作させるテスト項目を実行

本節で着目しているのはカバレッジである。

カバレッジの利点としては，下記が考えられる。

- カバレッジの指数をより高くするようにテスト設計することは，網羅度を高める指針になるため，テストの抜けを抑制する，又はテストの質を測る面で有効
- 論理的に到達できないパスの存在を明らかにすることも期待できるため，カバレッジの結果を分析することで，バグを摘出できる可能性もある

しかし，論理の抜け（パスの抜け）はカバレッジでは検出できない。

上記を踏まえて，カバレッジを計測してデータを積み重ね，テスト項目の評価ができる環境を作り上げることが必要と考える。

［本節で導かれたメトリクスと付随する情報］

- カバレッジ
- カバレッジとテスト項目数等との対比，データの積み重ねによるテストの質と量の検証

7. G3:バグの発生状況から残存バグの存在を推測して品質の成熟度を判断

7.1. G3Q1:予め計画した数のバグを検出

本節の主眼は，そのソフトウェアに仕込まれたバグの総量を見積もり，レビューを含めて各フェーズで除去する数を想定し，摘出バグ数を目標管理することである。

ここでポイントとなるのは，如何に合理的にバグの総量を見積もるか，という点にある。その一方法として，過去のプロジェクトデータからパターンを見出して予測する方法がある。

バグの数が予測値より少なければ，テストの内容を見直す。予想した値にまで達したら，テストは十分行ったと判断するものであるが，これには以下の問題も含んでいる。

- バグの数が目標にまで達したらそこで安心感が芽生え，それ以上のバグを検出しようとする意欲が薄らぐ（過去の実績を盲信する危険）
- 想定以上の数のバグが検出された場合には，この基準は余り意味を持たなくなる

前者の問題は，この基準単独では解決できないため，他の基準と複合して勘案する必要があると考える。あまりにバグが多く検出された際には，一度テストをストップし，設計工程に立ち返ることが必要と考える。

［導かれたメトリクスと付随する情報］

- バグの数の見積り値と実績値の差異

7.2. G3Q2:信頼度成長曲線が収束している

ゴエル・オクモトモデルに代表される信頼度成長曲線は，確率過程をベースにしたモデルである。このモデルを適用するためにはいくつかの前提条件を満足する必要があるが，テストフェーズのある時点でバグを検出するペースはピークに達し，その後なだらかになることは，直感的に理解できる。

この収束判断を定量的に判断する方法は，信頼度成長曲線の全体の傾きと終盤の傾きの比や，MTBF（平均故障間隔）等，既に別の研究で考察されている。

しかし，実際にはテストの終盤になってもバグが検出され，収束判断をするに至らないケースもある。

そもそも，なぜテストフェーズ終盤になってもバグは検出されるのか。

- 単純に，バグを検出するテストが完了してなく，信頼度成長曲線が未だ成長過程にいる
- デグレードにより新たにバグが作りこまれ，それが終盤になって検出される
- テストを進めるにつれて，そのプログラムの理解がより深まり，テストを追加することによって，新たにバグを検出する

G1Q3 では，バグの重要度によっては，それを修正することには検討の余地があることを示したが，本節では信頼性の面で考察する。

まず，バグをテスト終盤になって発生させないようにするには，バグをより早期に検出するように努めることが必要になってくる。それには，バグを検出したら他にも同種のバグが残存することを疑い，芋づる的にバグを検出することが必要になる。その一方法として，バグの発生状況を分析することが有効と考える。これを本稿では欠陥分析アクティビティと呼称する。

バグには一定のパターンがあることが，経験的に分かっている。例を下記に記す。

- バグは，ある一点に集中して存在する
- 同種のミスによるバグは，他にもある

バグの発生状況を上記の観点を含んで分析することにより，他に潜在するバグ（類似バグ）を予測することができる。類似バグを摘出するようなテストを設計し，追加する。このような欠陥分析のアクティビティを，テストプロセスで常時機能させることで，潜在するバグを早期に検出することが期待できる。

同様に，テストフェーズ終盤になって検出したバグについても同様に分析し，未だ残存するバグの可能性をリスクとして捉え，テスト完了判断の材料にする。テスト終盤になって検出したバグを，未検出のバグのシグナルとして捉える考え方である。この場合，分析の元となるバグの重要度とは分けて考慮する必要がある。観念的に言えば，軽微なバグが数件未修整で，そのプログラムが 90 点の出来と捉えることができたとしても，未だ検出されていないバグのリスクを加味した時，引き算した 10 点が本当に 10 点ではないのかもしれないのである。

［導かれたメトリクスと付随する情報］

- 信頼度成長曲線の収束判断（曲線収束度の評価方法，MTBF）
- 欠陥分析アクティビティの分析結果と，リスクの評価

8. G4:開発プロセス上に存在する問題を特定・対策する

8.1. G4Q1:バグ発生状況を開発プロセスにフィードバック

前述している欠陥分析アクティビティは，分析結果を主にテストプロセスにフィードバックすることを主眼に置いているが，これを開発プロセスにフィードバックするように拡張する。

欠陥分析アクティビティを掘り下げると，そのバグを作り込んだプロセスと，上位の検証プロセスに存在する問題点が分かってくる。その作り込み要因と流出要因を特定できれば，プロセスに潜む欠陥から生成された同種のバグを机上で検出することができるようになる。

開発とテストを別の組織で行う場合，テストチームが開発チームにプログラムを差し戻すことがあるが，この活動が具体例としてあげられる。この取り組みを行うには，開発チームとテストチームに二人三脚の協調関係がなければ実行できない。この関係を築き上げるには，テストチームには献身の心構えが，開発チームには最終的に品質を作り込むのは自身にあるとの自覚が必要であると，筆者らは考える。こう言う意味では，テストフェーズは単に検証工程でなく，生産工程なのである。

［導かれたメトリクスと付随する情報］

- 差し戻し回数
- 工程別アクション実施件数
- 指摘項目改善率

9. 抽出したメトリクスのまとめ

ソフトウェア完了判断をＧＱＭパラダイムに則って考察した結果を，表 2にまとめる。

表 2 テスト完了判断 Goal-Question-Metrics

G1:開発中のソフトウェアに潜むバグを検出・修正	
G1Q1:妥当な質と量のテスト項目を設定する	
(1)	カテゴリ別テスト件数，プログラム規模との比
(2)	実施しないテストカテゴリの明確化，リスクの度合い・対応策
G1Q2:設定したテスト項目を実行する	
(3)	設定テスト項目数と実施テスト項目数
(4)	実施しなかったテストカテゴリの明確化，リスクの度合い・対応策
G1Q3:検出したバグを修正する	
(5)	バグ発生件数（修正済みの件数と未修整の件数）
(6)	未修正バグの重要度，他への影響
(7)	重要度を判定する基準と，判定する仕組み
G1Q4:検出バグを正しく修正していることを確認	
(8)	デグレード発生件数
(9)	デグレード発生傾向の分析結果
G2:求められる要件を満足していることを確認	
G2Q1:要件を網羅するテスト項目を設定	
(10)	要件の数とテスト項目数
G2Q2:全機能を動作させるテスト項目を実行	
(11)	カバレッジ
(12)	カバレッジとテスト項目数等との対比，データの積み重ねによるテストの質と量の検証
G3:バグの発生状況から品質の成熟度を判断	
G3Q1:予め計画したバグの数だけ，バグを検出	
(13)	バグの数の見積り値と実績値の差異
G3Q2:信頼度成長曲線が収束	
(14)	信頼度成長曲線の収束判断
(15)	欠陥分析アクティビティの分析結果とリスク
G4:開発プロセス上に存在する問題を特定・対策	
G4Q1:バグ状況を開発プロセスにフィードバック	
(16)	差し戻し回数，工程別アクション実施件数，指摘項目改善率等

10. テスト完了判断基準の考察

ＧＱＭパラダイムによって考察した結果，下記のようにソフトウェアテスト完了判断をまとめた。カッコ書きの番号は，表 2のメトリクスに対応している。

＜前提条件＞

テスト項目は，的確・効率よくバグ検出できるように，論理立てて設計していなければならない。これを測るため，着目したカテゴリ別に件数が分かるようにすることが必要となる。(1)

同時に，要件を満たしていることをくまなく網羅していることを，要件とテスト項目数を対比できるようにしなければならない。(10)

これらはテスト対象の規模に見合った量でなければならない。上記の条件を満たしつつ，有限なリソース内で収まるように，計画する必要がある。(1)

バグを終盤で発生させると，その対処にはコストと危険が伴うため，バグを可能な限り早期に検出する必要がある。そのためムダにテストを行うと時間のロスにつながるため，テスト項目数はむやみに増やすことはできない。リソースと効率的なテストとのトレードオフで，テストを実施しないと意志決定した領域がもしあれば，それはリスクとして管理下に置き，モニターする必要がある。(2)

＜基準１＞テストの実施状況

設定したテストは，全て実施済みでいること。(3)

必要なテストが環境面や他の事情で実施できない項目があれば，その理由とリスクの度合い・対応策を明確にする。(4)

カバレッジとテスト項目数，過去の実績と照らし合わせ，一定の基準を満たしている。(11)(12)

＜基準２＞バグ発生状況

過去の実績と照らし合わせて目標に定めた件数に見合った数のバグを検出できていること。バグとテスト項目数の比を検討し，妥当な基準にあること。少数のテストであまりに多くのバグが検出されていたり，バグの検出数が目立って少ないなどが見受けられた場合には，その理由を分析し，対策する。(13)

＜基準３＞バグの修正状況

検出したバグは原則として全て修正していなければならない。ただし，重要度と他への影響に応じ，残存するバグ件数を許容する基準を設けても良い。(5)(6)

その場合には予め下記を実施していること。(7)

- 重要度の定義，個別のバグのランクを意志決定する仕組みを予め計画に盛り込み，関係者間で合意している
- 顧客に残存するバグの存在を通知し，了承されている

＜基準４＞バグの修正品質

デグレードが発生した件数と，発生状況の分析結果により，未だ存在するデグレードバグを類推し，リスクが許容できる範囲であること。(8)(9)

＜基準５＞信頼度成長曲線の収束度合い

信頼度成長曲線が収束傾向にあること。(14)

＜基準６＞検出バグの発生傾向分析

欠陥分析アクティビティの分析結果によりリスクを評価し，リスクが許容範囲であること。(15)

＜基準７＞開発プロセスへのフィードバック状況

欠陥分析アクティビティの分析結果を開発プロセスにフィードバックしたものに,積み残しがないこと。(16)

11. テスト完了の意志決定

以上の結果，様々な観点でテストプロセスを掘り下げた結果，１つの前提条件と７つの基準を導出した。これらを全て満足するものが，完了判断基準であるとは，筆者らは考えていない。

これら導出したメトリクスは，ある観点での側面を切り出して表したものである。それぞれのメトリクス単独では全体像を正しく見渡せるものではないため，様々な角度からの情報を集約し，総合して全体像を捉えることが，合理的なテスト完了判断につながる。

そのため，単に数値と基準の差異を見るのではなく，その意味するところを，開発チーム，テストチーム，品質保証チーム等の関係者で議論を尽くすことが必要である。その中で，出来ていることと出来ていないことを明確化し，テストを継続しないことに対するリスクを評価する。最終的にはそのソフトウェアの出荷責任者が決裁することが妥当であろう。

12. メトリクスとマネージメント

導出した７つの基準は，テストフェーズのタイムリミット間近になってからデータをかき集め，後追いで基準に照らし合わせて判断に用いるものではない。

テスト完了判断基準は，それ単独で機能するものではない。ソフトウェア開発のライフサイクル全体を通して品質保証計画を立て，その中の一要素として基準を設計することが必要である。

また,「テストには終わりが無いこと」と「テストはいつか終わらせなければならない」というパラドックスがあるため,テスト完了判断基準を設計する際は,その基準を満たしていてもなお未知のバグが存在する可能性を考え合わせることが必要になる。よって，万能な完了判断基準は存在しないと考えるのが妥当である。これを踏まえた上で,自らが置かれた状況で,日々きめ細かい軌道修正をし，マネージメントすることこそが，何よりも重要なのである。

13. 今後の展開・課題

今回,「テストの目的とは何か」を出発点として,「テストの制約条件」を加味しながらＧＱＭパラダイムを利用して定量的テスト完了判断基準の導出を試みた。

本稿ではテストの目的を４つ挙げたが，更に別の観点があるかもしれない。また，４つの目的から更にいくつかのQを検討したが，他の着眼点もあるはずである。したがって,この完了判断基準は一例に過ぎない。

今後も本フレームの検討を進めていくが，現時点では下記を課題として認識している。

- 「テストの質の評価」と「プログラムの規模の表し方」に，まだ曖昧な点が多い
- 今回,欠陥分析アクティビティでバグを分析し，そのリスクの評価を完了判断に加えることを提案したが，それを定量的に表す方法が確立できていない

これらを考慮しつつ，本検討結果を実際のプロジェクトに当てはめ，有効に機能するか否かを検証し，別の機会で発表したい。

14. 謝辞

本検討は，ソフトウェア技術者ネットワーク（S-open）のソフトウェアメトリクス SIG によるものです。メンバー皆様方に感謝いたします。

文　献

[1] G.J.Myers，ソフトウェア・テストの技法，近代科学社，東京，1980

[2] 玉井哲雄，三嶋良武，松田茂広，ソフトウェアのテスト技法，共立出版，東京，1988

[3] 真野俊樹，誉田直美，見積りの方法，日科技連出版社，東京，1993

[4] B.Beizer，ソフトウェアテスト技法，日経 BP，東京，1994

[5] 日科技連ソフトウェア品質管理研究会，２１世紀へのソフトウェア品質保証技術，日科技連出版社，日科技連出版社，東京，1994

[6] C.Kaner，H.Q.Nguyen，J.Falk，基本から学ぶソフトウェアテスト，日経 BP，東京，2001

[7] C.Kaner，J.Bach，B.Pettichord，ソフトウェアテスト 293 の鉄則，日経 BP，東京，2003

[8] 大西建児，ステップアップのためのソフトウエアテスト実践ガイド，日経 BP，東京，2004

[9] R.Black，基本から学ぶテストプロセス管理，日経 BP，東京，2004

[10] K.E.Wiegers，ピアレビュー，日経 BP，東京，2004

[11] R.Pressman，実践ソフトウェアエンジニアリング，日科技連出版社，東京，2005

[12] 高橋寿一，知識ゼロから学ぶソフトウェアテスト，翔泳社，東京，2005

[13] R.Black，ソフトウェアテスト 12 の必勝プロセス，日経 BP，東京，2005

[14] R.Craig，S.P.Jaskiel，体系的ソフトウェアテスト入門，日経 BP，東京，2005

[15] 伊藤拓也，“どこでテストをやめるか？,”，ソフトウェア・テスト PRESS，Vol.1，pp.58-64，Jun2005

[16] 西康晴，“テストの改善・テストによる改善,”，第 21 回 SEA 関西プロセス分科会講演，Sep2005