ソフトウェアメトリクスを用いた単体テストの品質リスク評価

Software Quality Risk Evaluation on Unit Testing using Product Metrics

日本電気株式会社　ソフトウェア技術統括本部
NEC Corporation Software Engineering Management Division
○下村 哲司　森 岳志[1)]　野中 誠[2)]　佐藤 孝司[3)]
○Tetsuji Shimomura　Takeshi Mori[1)]　Makoto Nonaka[2)]　Takashi Sato[3)]

Abstract Software defects injected in detail design or coding phases are often overlooked in unit testing. Rework to correct these defects is occurred in the latter stage of a testing phase and requires substantial efforts. Quality risk evaluation on unit testing allows developers to identify risky software modules and reduces such rework. Software metrics such as product metrics and testing metrics are promising techniques to evaluate the risk. In this paper, the authors describe the experiences on analyzing the relationship between product metrics and overlooked defects in unit testing. The result showed that the lines of code was the most useful metric to explain the trend of overlooked defects. The unit testing density and the unit testing coverage were also demonstrated as useful metrics. Action plans to be taken in unit testing for each segment divided by using these metrics are derived based on both the result and the authors' experiences.

1. はじめに

実際のソフトウェア開発現場では、ソフトウェア単体テスト後に詳細設計とコーディングで作りこんだ欠陥が流出し、テストの中盤から終盤にかけて、これら見逃し欠陥の修正による手戻りが発生することがしばしばある。このような手戻りを削減するには、そもそも、詳細設計書およびソースコードの内部品質と単体テストの質を高め、欠陥見逃しのリスク要因をできるだけ排除することが正しい方法だが、現実には難しい場合もある。そのため、プロダクトメトリクスを用いるなどして、単体テストでの欠陥見逃しリスクを評価し、リスクの度合いに応じた対策を早期に施すことも必要である。

過去に、プロダクトメトリクスに基準値を設けてソフトウェア品質の判断する方法論[1][2][3]が報告されているが、実際の製品データに適用して品質との関連を実証した事例は多くはない。実製品データを用いてメトリクスによる品質評価が行われた事例として、ソフトウェア設計品質の評価[4]や、Fault-Prone 予測[5]などが行われている。しかし、単体テストのメトリクスとソースコードのメトリクスを組み合わせて評価を行った事例は多くはない。

筆者らの組織では、近年、クラウド型ソフトウェア開発環境（以下、ソフトウェアファクトリと称す）を利用することで、ソースコードのメトリクスを自動計測する仕組みと、単体テストの自動化による単体テストのメトリクス（単体テスト密度、単体テストカバレッジ）を自動計測する仕組みを実現している。そこで、本稿では、単体テスト後のリスク評価に役立つメトリクスについて、弊社製品の分析によって得られた結果を報告する。

まず、単体テストで見逃した欠陥数とソースコードメトリクスの関係を分析し、単純ではあるが規模が有用なメトリクスであることを確認した。そして、規模に加えて、単体テスト密度や単

日本電気株式会社 ソフトウェア技術統括本部
NEC Corporation, Software Engineering Management Division
神奈川県川崎市中原区下沼部 1753　Tel: 044-431-7692
1753,Shimonumabe, Nakahara, Kawasaki, Kanagawa Japan
1) 日本電気株式会社 品質推進本部
NEC Corporation, Total Quality Management Division
2) 東洋大学 経営学部
Toyo University, Faculty of Business Administration
3) 日本電気株式会社 ソフトウェア技術統括本部
NEC Corporation, Software Engineering Management Division

体テストカバレッジといったメトリクスを用いることで、単体テスト見逃し欠陥の数と修正内容をある程度説明できることを示した。この分析結果をもとに欠陥見逃しリスクに応じた必要な対策を導出した。今後は、ソフトウェアファクトリの適用製品拡大に伴い、本稿の結果を適用できる範囲に展開していきたいと考えている。

2. ソフトウェア開発プロセス

筆者らの組織におけるソフトウェア開発プロセスは、図 1 の V 字モデルを採用している。基本設計工程からコーディング工程までの上工程、単体テスト工程からシステムテスト工程までのテスト工程から構成される。この V 字モデルに、品質会計制度[6]を適用することで、ソフトウェア製品の品質保証活動を行っている。

本稿では、テスト工程の最初の工程である単体テスト工程（以下 UT と称す）に着目し、V 字底（詳細設計工程～コーディング工程～単体テスト工程）の品質について、プロダクトメトリクスを用いて分析する。

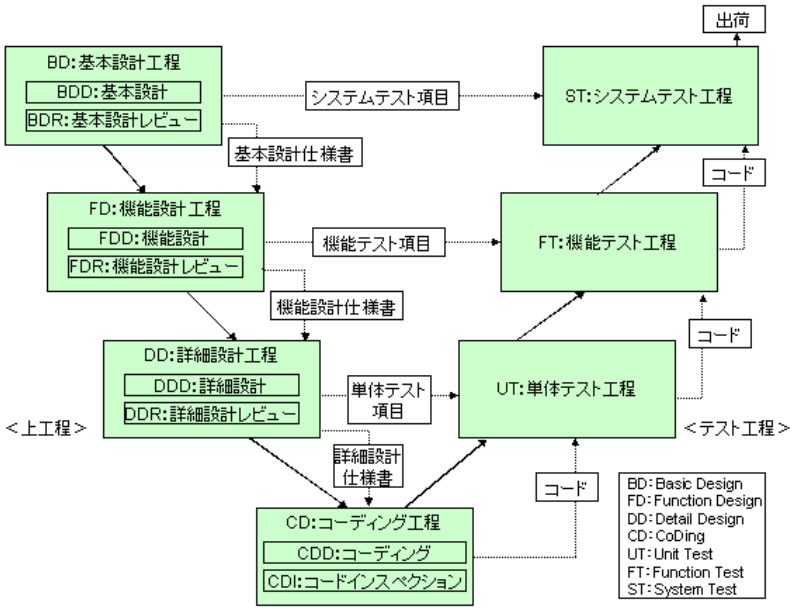


図 1 ソフトウェア開発プロセス

3. 分析対象データおよびメトリクス

3.1 分析対象データ

本稿では、ソフトウェアファクトリを利用している汎用的なシステムソフトウェア製品のうち、以下の条件に合致する製品を分析対象データとして選定した。

(1) UT がテストコードを用いて自動化されている
(2) カバレッジを自動計測している
(3) ソースコードが構成管理サーバに格納されている
(4) UT 完了後から開発完了時までに摘出した欠陥がバグトラッキングシステム(以下 BTS と称す)に登録されている
(5) BTS から当該欠陥に対する修正ファイル、修正内容、および修正差分をトレースできる

(1)のテストコードに記載されたテストメソッド数をテスト項目数としてカウントした。テストメソッド数をカウントすることで、テスト項目数のカウント方法の曖昧さを排除した。(2)ではステートメントカバレッジとデシジョンカバレッジを採用した。(3)の構成管理サーバに格納されたソースコードから各種プロダクトメトリクスを測定した。(4)の BTS に登録されている欠陥から、UT で摘出すべき欠陥の数をファイル単位に集計した。ファイル単位の欠陥数は(5)の修正ファイ

ル数を集計した。なお、UT で摘出すべき欠陥とは、詳細設計およびコーディング工程で作りこまれた欠陥のことである。

なお、今回の分析対象規模は 24KLOC、開発言語は Java である。

3.2 分析対象メトリクス

本稿で分析対象としたメトリクスを表 1 に示す。まず、UT の品質判断尺度として、UT 見逃し欠陥数を採用した。ソースコードメトリクスおよび UT メトリクスはソフトウェアファクトリを用いて組織的に自動収集可能なメトリクスを採用した。全てのメトリクスはファイル単位に集計を行った。本稿では、UT 見逃し欠陥に影響を与えるメトリクスおよびそれらの組み合わせの導出を試みる。記号は、次章以降の図表にて使用する。なお以下では、C0、C1 をまとめて表現する場合は UT カバレッジと称す。

表 1 分析対象メトリクス

種別	メトリクス	記号	説明
欠陥に関するメトリクス	UT 見逃し欠陥数	DEF	UT で見逃され、UT 完了後から開発完了時までに摘出された欠陥の数
ソースコードメトリクス	規模	ELOC	コメントを除く有効行数
	サイクロマチック数	CYC	McCabe のサイクロマチック複雑度[7]
	分岐条件数	BRA	複合条件を考慮した分岐条件の数
	最大ネスティング数	NEST	ファイル内の最大ネスティング数
UT メトリクス	UT 項目数	UTI	UT 用のテストコードに記述されたテストメソッド数
	UT 密度	UTD	UT 項目数/KLOC
	C0	C0	UT のステートメントカバレッジ
	C1	C1	UT のデシジョンカバレッジ

4. UT 見逃し欠陥のリスク評価

4.1 メトリクス間の相関分析

まず、収集したメトリクスを俯瞰するため、各メトリクス間の関係を調べた。図 2 にメトリクス全体の散布図行列を示す。散布図行列とは 3 つ以上のメトリクスを行列形式に配置することでメトリクス間の相関関係を俯瞰する可視化手法である。対象としたメトリクスは、規模、サイクロマチック数、分岐条件数、最大ネスティング数、UT 項目数、C0、C1、および UT 見逃し欠陥数である。図 2 上の各メトリクスは、表 1 の記号を用いて表現している。図 2 の右上は相関係数の組み合わせ、対角部分はヒストグラム、左下は散布図の組み合わせである。目盛は平均を 100 とした場合の相対値である（ただし、UT 見逃し欠陥数のみ実測値）。

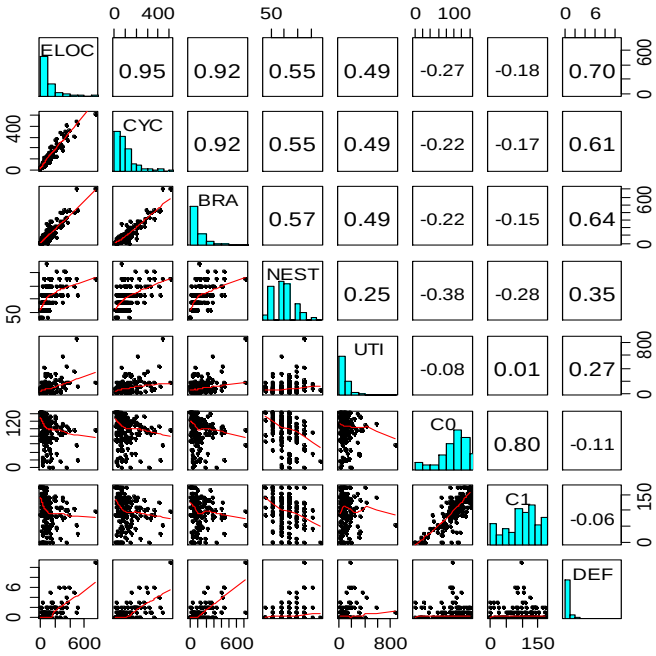


図 2 全メトリクスの散布図行列

UT 見逃し欠陥数と各メトリクスの相関を見ると、規模、サイクロマチック数、および分岐条件数との間に正の相関がある。しかし、これら 3 つのメトリクスは互いに強い正の相関がある。そのため、UT 見逃し欠陥数を説明するときには、これら 3 つのメトリクスのすべてを利用する必要はない。ここでは、UT 見逃し欠陥数を説明するときには、最も相関係数が大きい規模を用いる。

また、最大ネスティング数は UT 見逃し欠陥数と弱い正の相関があることが分かった。一方、UT 項目数、C0、および C1 などの UT メトリクスに関して、UT 見逃し欠陥との相関は見られなかった。

4.2 UT 見逃し欠陥の層別分析

図 2 の UT 見逃し欠陥のヒストグラムに着目すると、見逃し欠陥数が 0 件の場合に突出していることが分かる。そこで、UT 見逃し欠陥数に関して、1 件 1 件ではなくデータ群として層別化を行った。具体的には、UT 見逃し欠陥数が 0 件、1～2 件、3 件以上の 3 つに層別し、規模、最大ネスティング数、UT 密度、C0、および C1 に対して分析を行った。なお、UT 項目数ではなく、UT 密度を採用した理由は、規模による影響を排除するためである。各メトリクスの中で最も特徴がみられた規模について、UT 見逃し欠陥数の層別による分布（箱ひげ図）を図 3 に示す。横軸が 0 の場合は、UT 見逃し欠陥数が 0 件であったファイルの分布、1 の場合は UT 見逃し欠陥数が 1～2 件であったファイルの分布、2 の場合は UT 見逃し欠陥数が 3 件以上であったファイルの分布を示す。縦軸は、ファイル規模の平均値を 100 とした相対値である。

図 3 を見ると、UT 見逃し欠陥数が 0 件のファイルは、そのほとんどが規模の相対値=160（破線）以下に、約 75%が規模の相対値=100（一点鎖線）以下に分布していることが分かる。また、UT 見逃し欠陥数が 1～2 件のファイルは、約 50%が規模の相対値=100 以下に分布しており、UT 見逃し欠陥数が 3 件以上のファイルは、その約 75%が規模の相対値 160 以上に分布していることが分かる。

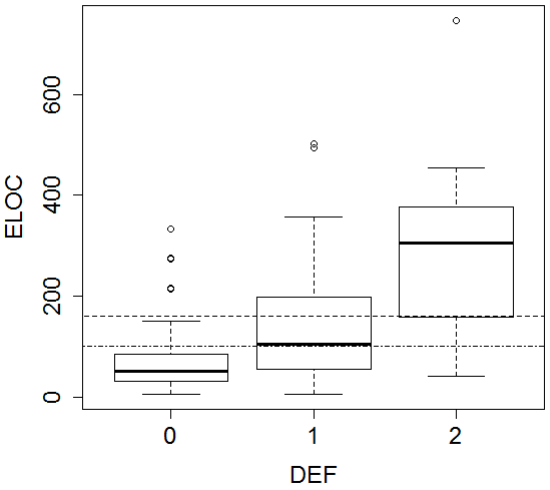


図 3 規模と UT 見逃し欠陥数の分布

4.3 分析結果の評価

4.2 節で分析した規模に関して基準値を設けた場合の評価を行う。評価指標は、表 2 の適合率、再現率および、F 値を用いる。

表 2 規模の評価指標

評価指標	説明
ファイルの適合率	基準値外ファイルのうち、UT 見逃し欠陥を（1 件以上または 3 件以上）含むファイルの割合
ファイルの再現率	UT 見逃し欠陥を（1 件以上または 3 件以上）含む全ファイルのうち、基準値外ファイルの割合
ファイルの F 値	ファイルの適合率と再現率の調和平均
欠陥の再現率	全 UT 見逃し欠陥数のうち基準値外ファイルに含まれる UT 見逃し欠陥数の割合

表 3 に 4.2 節で分析した規模の相対値（100～160）近傍で最も F 値が高かった値を基準値とした場合の評価結果を示す。本基準値を用いて基準値外のファイルを抽出した場合、以下のことが分かる。

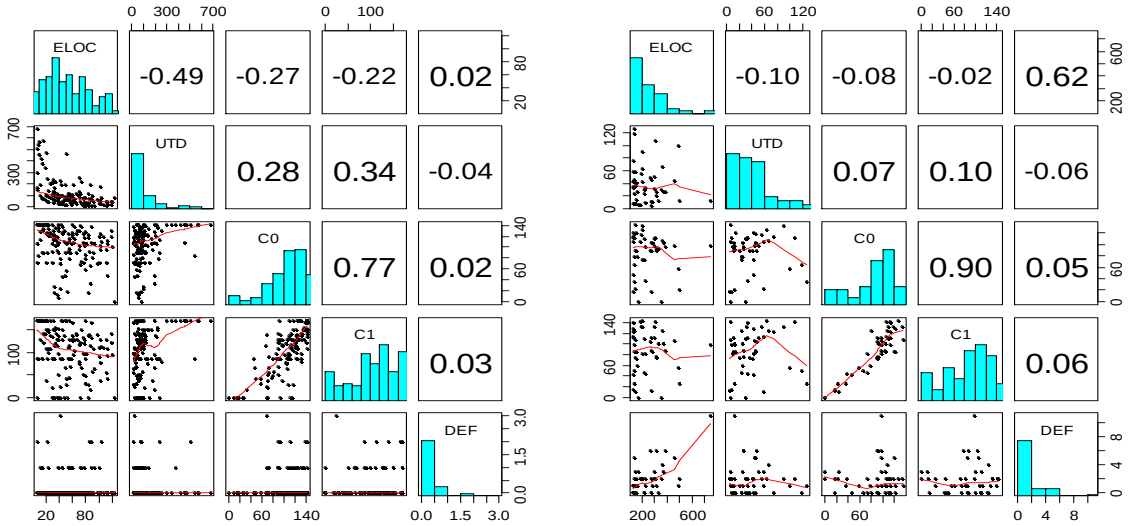
(1) ファイルの適合率より
　基準値外ファイルの 7 割強は、UT 見逃し欠陥を含んでいる。
(2) ファイルの再現率より
　UT 見逃し欠陥を 3 件以上含むファイルの 9 割強が基準値外のファイルである。
(3) 欠陥の再現率より
　基準値外ファイルには、全 UT 見逃し欠陥の 7 割強が含まれている。
(1)～(3)より、本基準値を用いれば、高い確率で UT 見逃し欠陥のあるファイルを検出可能であることが分かった。
　なお、他のメトリクス（最大ネスティング数、UT 密度、C0、C1）を用いた場合、規模以上の適合率と F 値を示すメトリクスは存在しなかった。以上より、UT 見逃し欠陥との関係において、規模が最も有用なメトリクスの一つであることが分かった。

表 3 規模の評価結果

基準値	UT 見逃し欠陥数	ファイル			欠陥
		適合率	再現率	F 値	再現率
125	>0	72%	58%	64%	74%
	≥3	23%	92%	37%	N/A

5. 欠陥見逃しリスク対策
　4 章で、UT における欠陥見逃しリスク要因として、単純ではあるが規模が最も有用なメトリクスであることを確認した。本章では、さらにリスク対象ファイルを抽出後の対策について議論する。

5.1 規模の大小で分けた場合の分析
　表 3 より、規模が大きいファイルに含まれる UT 見逃し欠陥は全体の 7 割であるが、規模が小さいファイルにも残りの 3 割の UT 見逃し欠陥が含まれることが分かる。場合によっては規模が小さいファイルに対しても対策が必要な可能性もあるため、本節では、規模の大小で分けた場合のそれぞれの対策について分析を行う。
(1) メトリクス全体の分析
　図 4 に規模の大小で分けた場合の散布図行列を示す。メトリクスは規模、UT 密度、C0、C1、および UT 見逃し欠陥数である。目盛は図 2 と同様に各メトリクスの平均を 100 とした場合の相対値である。ただし、UT 見逃し欠陥数のみ実測値である。
　まず、規模が小さい場合を分析する。図 4(a)を見ると、以下が読み取れる。
A) UT カバレッジが高く UT 密度が高い場合、UT 見逃し欠陥数が少ない。
B) UT 密度が低い場合、UT カバレッジが高くても UT 見逃し欠陥は存在する。
規模が小さい場合は、A)より、UT カバレッジの向上が UT 見逃し欠陥を削減する一つの対策と考えることができる。また、B)より、UT カバレッジが高くても UT 密度が低いファイルは UT 密度を向上させることがもう一つの対策と考えられる。
　次に、規模が大きい場合を分析する。図 4(a)と(b)を比較すると、以下が読み取れる。
C) 規模が小さい場合と比較して、全体的に UT 密度が低い。
　また、図 4(b)を見ると、以下が読み取れる。
D) C0、C1 共に、相対値 100 近辺で UT 見逃し欠陥が多い。
規模が大きい場合は、C)より、全体的に UT 密度の底上げが必要であることと、D)より UT カバレッジだけでは判断できないことが分かる。

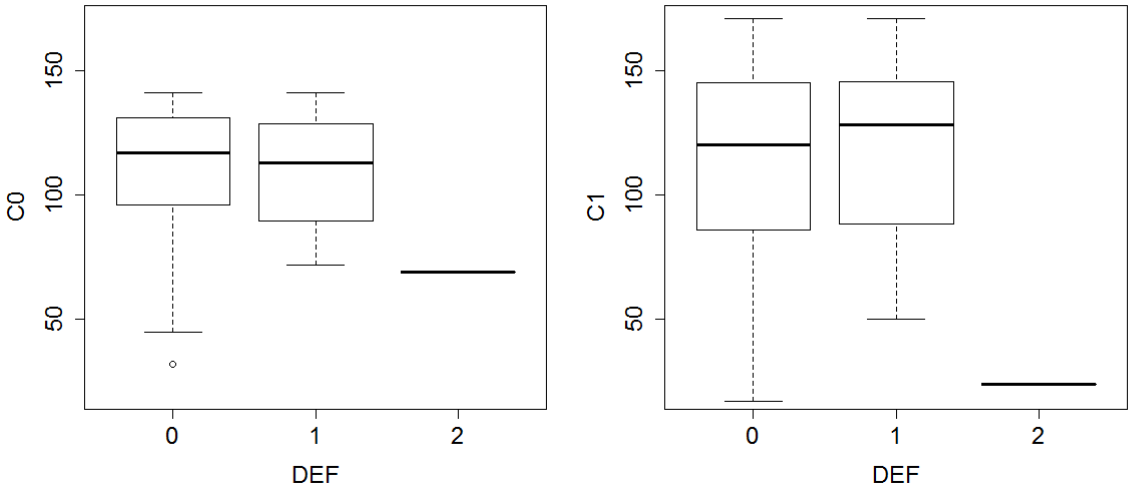


(a) 規模小　　(b) 規模大

図 4 規模の大小で分けた場合の散布図行列

(2) UT カバレッジの分析

次に、規模が小さい場合の UT カバレッジについて UT 見逃し欠陥数の分布を分析する（図 5)。なお､UT カバレッジが極端に低いファイルは､別の要因が考えられるため外れ値として除外した。横軸は図 3 と同じであり、縦軸は、(a)が C0、(b)が C1 である。なおそれぞれ平均値を 100 とした相対値である。UT 見逃し欠陥数が 3 件以上の場合は、C0、C1 ともに低い値(平均以下)を示している。したがって､規模が小さい場合､UT カバレッジを向上させる施策の必要性が裏付けられた。



(a) C0　　(b) C1

図 5 UT カバレッジと UT 見逃し欠陥数の分布（規模小の場合）

(3) UT 見逃し欠陥の修正内容分析

次に､UT 見逃し欠陥のテスト漏れ観点を分析するため､UT 見逃し欠陥の内容を BTS より抽出し、UT にて摘出すべきテスト区分を調査した。テスト区分は、構造ベースドテスト[8]と仕様ベースドテスト[8]のどちらか一方に分類した。その結果を表 4 に示す。また、それぞれのテスト区分に分類した事例を表 5 に示す。具体的には、BTS から UT 見逃し欠陥の原因を分析し、その原因を除去する対策（テスト技法）を導出した。そして、そのテスト技法をテスト区分に分類した。

表 4 見逃し欠陥を摘出すべきテスト区分

	構造ベースドテストの割合	仕様ベースドドテストの割合
修正ファイルが全て規模小の欠陥	100%	0%
修正ファイルが全て規模大の欠陥	22%	78%

表 5 テスト区分分類事例

テスト区分	欠陥の原因	テスト技法
構造ベースドテスト	クラス名称誤り、設定値誤り 判定ロジック評価漏れ	ステートメントカバレッジ デシジョンカバレッジ
仕様ベースドテスト	状態遷移評価漏れ 例外処理評価漏れ コマンドオプション評価漏れ 入力バリエーション評価漏れ	状態遷移テスト ユースケーステスト デシジョンテーブル 同値分割/境界値テスト

表 4 を見ると、規模小の場合は、構造ベースドテストが不足していることが分かる。一方、規模大の場合は、仕様ベースドテストが不足している割合が高いことが分かる。

次に、各欠陥の修正内容を確認した。規模小のファイルの修正内容は、条件文を含む修正はなく、修正行数が実質 1～2 行であったのに対し、規模大のファイルの修正は、条件文を含む修正、すなわちロジックに影響する修正がほとんどであった。

以上の確認結果は、規模小のファイルに対しては構造ベースドテスト、規模大のファイルに対しては仕様ベースドテストの必要性を示唆している。

5.2 定性的リスク分析と対策

5.1 節で、ファイル規模の大小による、リスクと対策について議論した。この結果を定性的リスク分析として発生確率-影響度マトリクスにマッピングすると表 6 のようになる。ここで、発生確率とは、見逃し欠陥が UT 後に発生する確率であり、影響度とは、対策を実施する場合の手戻り工数を意味している。

表 6 発生確率・影響度マトリクス

発生確率＼影響度	大	小
高	規模大、UT 密度小	規模大、UT 密度大
低	規模小、UT カバレッジ小	規模小、UT カバレッジ大

本稿では、表 6 のマトリクスに組織別・製品別の優先度をあてはめて、リスク管理を行うことを提案する。そして、それぞれのリスクに対する対策は以下の通りである。

(1) 規模が小さい場合、
UT カバレッジ（C0、C1）を向上させることを施策とする。

(2) 規模が大きい場合
UT 密度の底上げが必要。ただし、闇雲に UT 項目数を追加して UT 密度を向上させても効果が薄いため、仕様ベースドテスト観点の強化を行う。

6. 考察

本稿で得られた結果を以下に考察する。

(1) UT 品質に影響を与えるメトリクスについて

・ UT 完了時点のプロダクト品質に影響を与えるメトリクスとして規模が最も有用である。
・ 規模が一定未満の場合は、UT の品質を確保できる可能性が高い。
・ 規模が一定以上の場合は、UT の品質確保が困難になる可能性が高い。

(2) UT の十分性について

・ 単純にカバレッジだけでは UT の十分性を判断できない。ただし規模が一定未満の場合は、カバレッジを用いて十分性を判断できる可能性がある。
・ 規模が一定以上の場合は、仕様ベースドテストが UT の十分性に影響を与える可能性が高い。

(3) UT の十分性を確保する対策について

・ 単体テスト対象の規模（本稿ではファイルの規模）を可能な限り一定以内に収める。
・ 一定規模未満の場合は、カバレッジ強化を重点的に実施する。
・ 一定規模以上の場合は、カバレッジだけでは不十分であり、仕様ベースドテストをより強化する必要がある。

7. おわりに

本稿では、UT 完了時点の品質リスクについて、ソフトウェアプロダクトメトリクスを用いて評価を行った。様々なメトリクスを用いて評価した結果、単純ではあるが規模が最も有用なメトリクスであることを示した。そして、規模の大小に応じたリスク評価を行ったあと、それぞれに必要な対策を導出した。この規模の基準値を活用すれば、UT 以降にリスクを含んだモジュールやファイルをある程度特定でき、それらに必要な対策を実施することが可能となる。

今後、ソフトウェアファクトリの利用拡大に伴い、UT が自動化された製品数の増加が見込まれる。その結果、これらの製品に対しても今回対象としたメトリクスを今回と同じ粒度で測定可能となる。その際には、今回対象としたメトリクスに関する組織全体としての基準値および対策方法を立案したいと考えている。ただし、画一的な基準値と対策だけではなく、製品特性（事業領域、開発言語、使用するフレームワークなど）も踏まえた議論も合わせて必要と考えている。

参考文献

[1] Watson, A.H. and McCabe, T.J., Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication, Vol.500-235, 1996.

[2] Rosenberg, L., Stapko, R., and Gallo, A., Applying Object-Oriented metrics, 6th Int'l Symposium on Software Metrics, Nov. 1999.

[3] Lorenz, M., Object-Oriented Software Development: A Practical Guide, Englewood Cliffs, N.J.; PTR Prentice Hall, 1993.

[4] 倉下亮, 吉村博昭, 野中誠, 誉田直美, CK メトリクスの分布に基づくソフトウェア設計の質の定量的評価, ソフトウェア品質シンポジウム 2011 報文集, 2011.

[5] 纐纈信子, 川村真弥, 野村准一, 野中誠, プロセスメトリクス利用による Fault-Prone クラス予測精度の向上, ソフトウェア品質シンポジウム 2010 報文集, 2010.

[6] 誉田直美, ソフトウェア品質会計, 日科技連出版社, 2010.

[7] McCabe, T.J., A Complexity Measure, IEEE Transactions on Software Engineering, Vol. SE-2, No.4, 1976.

[8] Foundation Level Syllabus, International Software Testing Qualifications Board, 2011.