Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

テスト工数配分方法の評価手法と組込みソフトの派生開発プロジェクトへの適用

八木将計† 小川秀人†

近年のソフトウェア開発では，開発規模が増加する一方，開発工数が限られるため，限られたテスト工数での品質確保が課題となる．テストにおける品質確保の一側面は不具合を多く発見することであり，不具合の多い傾向のある部分にテスト工数を多く配分することが，限られたテスト工数での品質確保につながると考えられる．より多く不具合を摘出できるテスト工数配分方法を選択するには，様々な配分方法の評価が必要となる．よって，本論文では，実際のプロジェクトデータから評価モデルを作成することで，様々なテスト工数の配分方法を評価可能な手法を提案する．既存機種のコードを改変する派生開発を行っている実際の組込みソフト開発プロジェクトに提案手法を適用し，改変母体の過去の不具合修正回数に基づいてテスト工数配分を行うと摘出不具合数を増加できることを示した．

Method for Estimating Test Effort Allocations and its Application to Derivative Development Project of Embedded Software

Masakazu Yagi† and Hideto Ogawa†

This paper proposes a method for estimating test effort allocations to enhance the reliability of software systems. Our method calculates the defect removal efficiency of different test allocations by using the estimation model based on real project data. Thus, we can compare the traditional test allocation with other ones. We applied our method to the derivative development project of embedded software. The result shows that the test allocation based on the number of past defects in the base code enables us to find defects most effectively.

1. はじめに

年々，ソフトウェアの開発規模の増大に伴い，ソフトウェアテスト工数も増加傾向にある．一方で，開発工数は限られており，テスト工数を十分に確保することが困難になりつつある．よって，限られたテスト工数で品質を確保することがソフト開発における課題の一つとなっている．

テストにおける品質確保の一側面は，より多くの不具合を発見することといえる[1]．そのため，限られたテスト工数における品質確保には，テストケース一件当たりの摘出不具合数（不具合摘出率）を向上することが一つの対策である．不具合摘出率向上のため，境界値分析，HAYST 法など，不具合摘出率向上をめざしたテスト設計技法が提案されている[2][3][4]．

一方，不具合は一部のソフトウェアモジュールに偏在することが知られている[5]．そのため，不具合の多い傾向にある部分にテストを多く割り当てることができれば，上記テスト設計技法がより効果を発揮し，多くの不具合を摘出できると考えられる[6]．通常，テスト工数の割り当ては，開発規模，リスク，重要度など，その組織で定めた基準で決定している．しかし，それらは経験に基づいている基準であるため，より効果的に不具合を摘出できるテスト工数の配分方法が存在する可能性がある．より良いテスト工数配分方法を見付けるためには，様々なテスト工数配分による不具合摘出の効果を評価することが必要となる．

テスト工数と摘出不具合の特性を表現したものとして，ソフトウェア信頼度成長曲線(以下，SRGC: Software Reliability Growth Curve)が知られている[7][8]．この SRGC は，曲線の収束に基づく総不具合数の推定やテストの進捗に対する品質評価などに応用されており，テスト工数に対する不具合数の評価を可能にする．しかし，この SRGC は，ソフト開発プロジェクト全体のテスト工数と不具合の関係を表現したものであり，テスト工数配分方法については考慮されていない．

本論文では，SRGC を個々のソフトウェアモジュールに応用し，テスト工数配分方法を評価する手法を提案する．提案手法では，実際のプロジェクトデータに基づき，様々なテスト工数配分方法を適用した場合の摘出不具合数を算出する．

本手法を，既存機種のコードを一部改変する派生開発を行なっているプロジェクトに適用した．結果，一般に不具合と相関すると考えられているソースコードの複雑さ[9][10]よりも，改変母体の過去の不具合修正回数に応じてテスト工数を配分することで摘出不具合数を増加できることがわかった．

以降，第 2 章では，ソフトウェア信頼度成長曲線を説明する．第 3 章では，テスト

† 株式会社日立製作所 中央研究所
Hitachi, Ltd., Central Research Laboratory

Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

工数配分方法評価モデルとそれを用いたシミュレーション方法を説明する．第 4 章では，実際のプロジェクトへの適用事例を述べ，第 5 章で，適用結果について考察する．最後に第 6 章で本論文をまとめる．

2. ソフトウェア信頼度成長曲線

ソフトウェア開発プロジェクトでのテスト工数と摘出不具合数は，ソフトウェア信頼度成長曲線（SRGC）で表現される（図 1）．図に示すとおり，一般に SRGC は飽和曲線となる[7][8]．出荷時までにテストにより摘出しきれなかった不具合が，出荷後不具合になり得る．よって，不具合摘出に効率的なテスト工数配分を行うことで，不具合摘出率が向上すると，出荷後不具合を削減できる（図 2）．

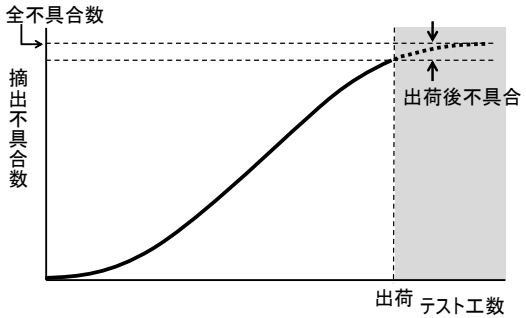

図 1　ソフトウェア信頼度成長曲線と出荷後不具合

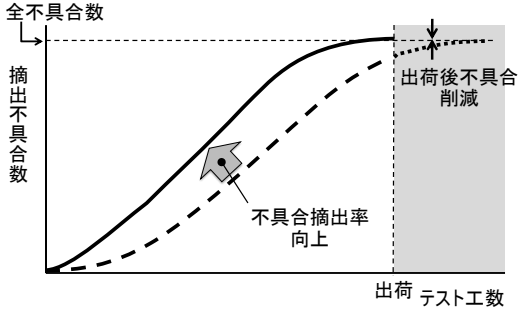

図 2 不具合摘出率向上と出荷後不具合の削減

テスト工数に対する不具合の収束状況の評価などを目的として，SRGC の決定論的モデルにロジスティック曲線とゴンペルツ曲線が用いられる[8].

SRGC のロジスティック曲線モデルは以下となる．

$$D(Z)=\frac{D_{\mathrm{all}}}{1+\alpha\exp(-\beta Z)}, \tag{1}$$

また，ゴンペルツ曲線モデルは以下となる．

$$D(Z)=D_{\mathrm{all}}\gamma^{\exp(-\delta Z)}, \tag{2}$$

ただし，Z がテスト工数，D がテスト工数 Z における摘出不具合数，D_{all} が全不具合数，α と β がロジスティック曲線モデルの特性を表わす係数，γ と δ がゴンペルツ曲線モデルの特性を表わす係数である．不具合の収束状況の評価ため，SRGC の実績曲線を各曲線モデルに近似する場合は最小二乗法などで $\alpha,\beta,\gamma,\delta$ を調整する（飽和していない場合は D_{all} も調整する）．

3. テスト工数配分方法の評価手法

3.1 テスト工数配分方法評価モデル

本論文では，テスト工数と摘出不具合の関係を簡潔に表現している SRGC を，プロジェクト全体の特性としてではなく，ソフトウェアモジュール毎でも同様の特性を持つものとして，テスト工数配分方法の評価モデルを作成する．つまり，一つのモジュールにおけるテスト工数と摘出不具合の関係も SRGC と同様に上記ロジスティック曲線モデル(1)やゴンペルツ曲線モデル(2)で近似できるものとする．ここで，評価モデルは，ロジスティック曲線モデルベースとゴンペルツ曲線モデルベース，二つのモデルを対象プロジェクトに適したものを使い分けるものとする．

式(1)より，一つのモジュールにおける不具合摘出のロジスティック曲線モデルを

$$d(z)=\frac{d_{\mathrm{all}}}{1+\alpha\exp(-\beta z)}, \tag{3}$$

とし，式(2)より，ゴンペルツ曲線モデルを

$$d(z)=d_{\mathrm{all}}\gamma^{\exp(-\delta z)}, \tag{4}$$

とする．ただし，z がモジュールに対するテスト工数，d がテスト工数 z における

Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

そのモジュールの摘出不具合数，d_{all} がそのモジュールの全不具合数である．

ここで，全てのモジュールの摘出不具合数の総和がプロジェクト全体の摘出不具合数に一致するものとする[a]．また，対象とするプロジェクトで開発した全てのモジュールが同じ特性係数（α,β （or γ,δ））をもつと仮定する．

さらに，テスト工数の配分を考慮するため，プロジェクト全体のテスト工数 Z を配分してモジュールのテスト工数 z を決定することとする[b]．

よって，プロジェクトで開発したモジュールが n 個の場合，プロジェクト全体の摘出不具合数のロジスティック曲線モデルは，

$$D(Z)=\sum_{i=1}^{n}\frac{d_{\mathrm{all},i}}{1+\alpha\exp(-\beta' r_i Z)}, \tag{5}$$

となり，ゴンペルツ曲線モデルは，

$$D(Z)=\sum_{i=1}^{n} d_{\mathrm{all},i}\,\gamma^{\exp(-\delta' r_i Z)}, \tag{6}$$

となる．本論文では，これら二つのモデルを『テスト工数配分方法評価モデル』と呼ぶ．ただし，モデルをプロジェクト全体のテスト工数 Z の式とするため，式(3)の β を $\beta'(=n\beta)$，式(4)の δ を $\delta'(=n\delta)$ としている．また，r_i がモジュール i のテスト工数配分係数，$d_{\mathrm{all},i}$ がモジュール i の全不具合数，式(1)，(2)の D_{all} と式(5)，(6)の $d_{\mathrm{all},i}$ 関係は，以下の通りである．

$$D_{\mathrm{all}}=\sum_{i=1}^{n} d_{\mathrm{all},i}\,. \tag{7}$$

なお，テスト工数配分係数 r_i は以下の条件を満さなければならない．

$$\sum_{i=1}^{n} r_i=1 \ \text{かつ}\ r_i\geq 0\,. \tag{8}$$

3.2 テスト工数配分方法評価モデルを用いた評価手法

本論文では，前節で示したテスト工数配分方法評価モデルを用いて，テスト工数配分方法を様々に変更した場合の摘出不具合数の変化をシミュレーションする手法を提案する．本提案手法では，実際のプロジェクトデータを元に評価モデル(5)(or (6))のテスト工数配分係数 r_i を変更した場合の摘出不具合数を算出し，テスト工数配分方法を評価する．

具体的なアルゴリズムを以下に示す(図 3 参照)．

STEP1: 実際のプロジェクトのデータから，SRGC の実績値をプロットする．

STEP2: STEP1 で作成した SRGC について，テスト工数配分方法評価モデルを最小二乗法で近似する（評価モデルの特性係数 α,β'(or γ,δ')を決定）．

STEP3: STEP2 で決定した特性係数 α,β'(or γ,δ')を固定し，テスト工数配分係数 r_i を変更して，摘出不具合数をシミュレーションする．

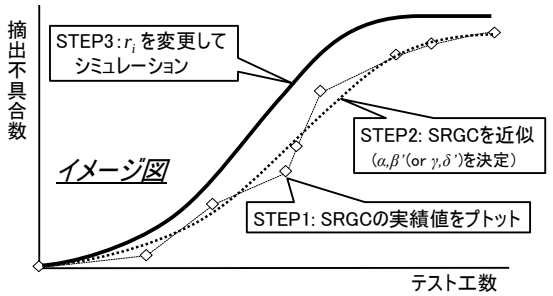

図 3 テスト工数配分方法評価シミュレーション手順のイメージ図

a) 通常，プロジェクトの不具合は，複数のモジュールに関係するため，[全モジュールの不具合数総和]>[プロジェクトの不具合数]となるが，本論文では簡単化のため，[全モジュールの不具合数総和]=[プロジェクトの不具合数]とする．第 4 章のシミュレーションでは，全モジュールの不具合数の総和がプロジェクトの不具合数に一致するよう，正規化を行う．例えば，N 個のモジュールに関わる不具合は，1 つのモジュールでは 1/N 個としてカウントする．

b) 実際のテスト工数配分方法としては，テストケース数，テストの範囲，テストレビュー回数・時間など，が考えられる．

Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

4. テスト工数配分方法シミュレーション事例

4.1 シミュレーション対象

本論文では，ある組込み製品におけるソフトウェア開発プロジェクト二つをシミュレーション対象とする．

対象プロジェクトの開発上の特徴は以下の通りである．

- 既に出荷した機種のコードを改変母体とする世代型派生開発を行なっている．
- 品質保証部を設けており，設計部でのテストの後，品質保証部による妥当性確認を行なっている．
- 設計部で摘出しきれなかった不具合が品質保証部の妥当性確認で摘出される．
- 設計部のテストは“開発規模[c]”を基に工数配分している．つまり，開発規模の大きなモジュールに多くのテスト工数をかけている．
- 結合テスト以降の情報を収集している．つまり，ユニットテスト以前の情報は収集対象外としている．
- 設計部では，SRGC をプロジェクトの進捗確認のために使用している．
- テスト終了条件を“計画したテストケースを消化したとき”としており，上記 SRGC をテスト終了条件としては使用していない．
- 上記 SRGC には，品質保証部の妥当性確認は含まれない．

対象プロジェクト二つの詳細データを表 1 に示す．表中の全不具合数は，結合テスト以降で発見した不具合，つまり，結合テスト以降および品質保証部の妥当性確認での摘出不具合と出荷後不具合の総和である．

表 1　シミュレーション対象プロジェクトの詳細

	プロジェクト A	プロジェクト B
開発規模	十数 KStep	百数十 KStep
開発で変更のあったファイル数（モジュール数）	約 350 ファイル	約 1700 ファイル
開発言語	C++, ほか	C++, ほか
全不具合数	数十件	数百件
不具合の偏在性	約 50%の開発モジュールに全不具合が含まれる	約 40%の開発モジュールに全不具合が含まれる

c) 開発時に実際に実装したコード量．改変母体の改造コード量(ステップ数)と新規追加コード量(ステップ数)の和．

4.2 シミュレーションの前提条件

本事例では，“ファイル”をソフトウェアモジュールとする．よって，モジュール数 n は，表 1 の“開発時変更のあったファイル数（モジュール数）”を用いる．

また，本事例では，対象プロジェクトにおける実際の SRGC をより良く近似できるロジスティック曲線ベースの評価モデル(5)を用いる．ただし，SRGC が設計内テストのデータのみであるため，SRGC の実績曲線に品質保証部の担当部分は含めない．

対象プロジェクトでは，プロジェクトレベルの不具合が複数モジュールに関わっているため，全モジュールの不具合数総和がプロジェクト全体の不具合数よりも多い．よって，評価モデル(5)への対応のため，全モジュールの不具合数の総和がプロジェクトの不具合数に一致するよう，各モジュールの全不具合数を正規化する．具体的には，N 個のモジュールに関わる不具合を 1 つのモジュールでは 1/N 個とカウントする．

4.3 メトリクスに基づくテスト工数配分方法の評価

本事例では，テスト工数配分方法の基準として用いるメトリクスを変更した場合の摘出不具合数を評価する．つまり，実際のテスト工数配分は“開発規模”を基準としているものとし，様々なメトリクスを基準にテスト工数を配分した場合の摘出不具合数(設計のテスト終了時点)をシミュレーション・比較する．

ここで本事例では，モジュールをメトリクスの値に応じて 10 のグループに分類し，グループ毎に 10 段階でテスト工数配分係数を設ける．開発規模の大きい順でモジュールをソートしたところ，ほぼ反比例特性を有していたため，グループ 1 を基準に，グループ 2 は 1/2 のテスト工数，グループ 3 は 1/3…という比率でテスト工数 Z を配分するものとした(図 4)。このグループ毎のテスト工数配分係数を変更せずに，この分類のためのメトリクスを変えることで不具合摘出率を評価する．

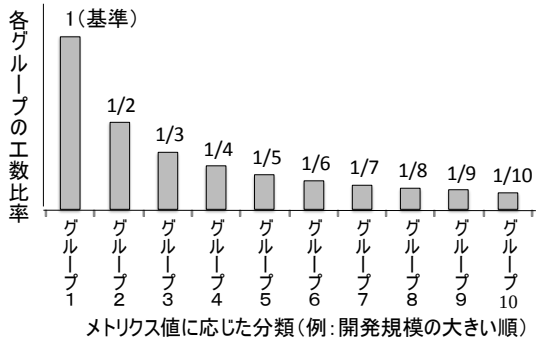

図 4　メトリクス値に応じた分類とテスト工数配分係数の比率

Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

評価したメトリクスを表 2 に示す.

メトリクス#2～#11 は，ソースコードメトリクスを計測するフリーソフト SourceMonitor[11]で計測したメトリクスであり，メトリクス名もツールで使用されているものを用いている．特に#8～#11 は，ソースコードの複雑さを表わすメトリクスであり，一般に不具合発生数はソースコードの複雑さと相関があると考えられている[9][10]ため，評価対象とする．

また，メトリクス#12,#13 は，派生開発特有のメトリクスである．#12 の“改変母体の規模”は，派生開発における改変母体としたモジュールの全コード量であり，#13 の“改変母体の過去の不具合修正回数”は，派生開発における改変母体とした機種で過去開発時に発生した不具合の修正回数である.「改変母体の特徴を派生開発時にも引き継いでいる」という仮説の下，この二つのメトリクスを評価する．

表 2 テスト工数配分方法評価シミュレーションで評価したメトリクス

#	メトリクス名	説明	備考
#1	開発規模	母体の改変コード量+新規追加コード量	既存のテスト工数配分方法
#2	Statements	命令数	SourceMonitor で計測.
#3	Percent Branch Statements	命令数における分岐の割合	SourceMonitor で計測.
#4	Percent Lines with Comments	コメント行の割合	SourceMonitor で計測.
#5	Classes Defined	宣言されているクラスの数	SourceMonitor で計測.
#6	Methods Implemented per Class	1 クラスあたりのメソッド数	SourceMonitor で計測.
#7	Average Statements per Method	1 メソッドあたりの命令数	SourceMonitor で計測.
#8	Maximum Complexity	循環的複雑度の最大値	SourceMonitor で計測. 循環的複雑度は，McCabe のサイクロマチック数[12][13].
#9	Maximum Block Depth	分岐階層の深さの最大値	SourceMonitor で計測.
#10	Average Block Depth	分岐階層の深さの平均値	SourceMonitor で計測.
#11	Average Complexity	循環的複雑度の平均値	SourceMonitor で計測. 循環的複雑度は，McCabe のサイクロマチック数[12][13].
#12	改変母体の規模	改造母体モジュールのコード規模	
#13	改変母体の過去の不具合修正回数	改造母体モジュールに過去に不具合修正した回数	

4.4 シミュレーション結果

評価対象のメトリクスの内,代表例として,プロジェクト A における#8 の“Maximum Complexity”と#13 の“改変母体の過去の不具合修正回数”のシミュレーション結果を図 5 に示す．設計内でのテスト終了時，“Maximum Complexity”では摘出不具合数が“開発規模”に比べて 13%向上し，“改変母体の過去の不具合修正回数”では，摘出不具合数が“開発規模”に比べて 17%向上するシミュレーション結果となった．

また，開発規模に基づくテスト工数配分におけるテスト終了時の摘出不具合数を 100%とした場合の各メトリクスによるテスト工数配分での摘出不具合数の割合を表 3 に示す.

表より，プロジェクト A では，ソースコードの複雑さを示すメトリクス(#8～#11)でテスト工数配分を行うと,開発規模よりも不具合摘出率を向上できることがわかった．しかし，プロジェクト B においては，開発規模とあまり差がなかった．

ソースコードの複雑さメトリクスよりも不具合摘出率を向上できるという結果を示したのが,“改変母体の規模”と“改変母体の過去の不具合修正回数”であった．特に“改変母体の過去の不具合修正回数”は，プロジェクト A，B ともに最も摘出不具合数が多くなるという結果を得た．このことについて次章で考察する．

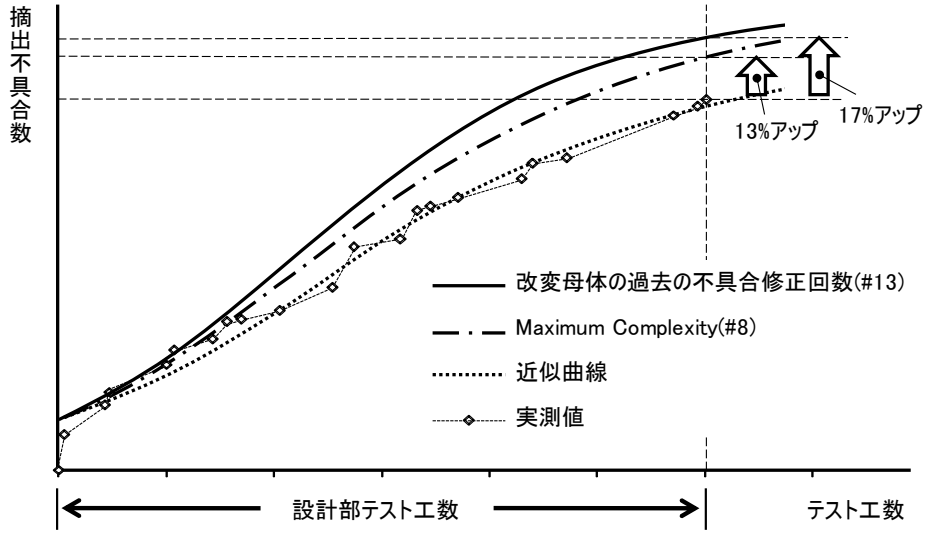

図 5 テスト工数配分シミュレーション結果代表例（プロジェクト A の#8“Maximum Complexity”，#13“改変母体の過去の不具合修正回数”）

Vol.2011-SLDM-149 No.39
Vol.2011-EMB-20 No.39
2011/3/19

表 3 テスト工数配分シミュレーション結果（各メトリクスの摘出不具合数の割合）

#	メトリクス名	摘出不具合数の割合	
		プロジェクト A	プロジェクト B
#1	開発規模	100.0%(基準)	100.0%(基準)
#2	Statements	113.8%	109.7%
#3	Percent Branch Statements	109.2%	85.1%
#4	Percent Lines with Comments	97.2%	50.8%
#5	Classes Defined	60.1%	45.2%
#6	Methods Implemented per Class	112.3%	105.5%
#7	Average Statements per Method	110.5%	91.3%
#8	Maximum Complexity	112.8%	104.8%
#9	Maximum Block Depth	116.6%	105.2%
#10	Average Block Depth	114.1%	96.3%
#11	Average Complexity	113.2%	93.5%
#12	改変母体の規模	115.7%	111.0%
#13	改変母体の過去の不具合修正回数	**117.1%**	**115.5%**

5. 派生開発における改変母体に関するメトリクスについての考察

前章のシミュレーション結果より，改変母体の過去の不具合修正回数に基づいてテスト工数配分すると最も摘出不具合数が多くなることがわかった．つまり，改変母体としたモジュールで過去に不具合があった場合，新規にも不具合を作り込みやすいといえる．

このことについて，シミュレーション対象とした開発プロジェクトの開発者にヒアリングし，開発現場での感覚と一致しているということを確認している．

過去に不具合のあったモジュールに新規にも不具合を作りこみやすいことについては，以下のような要因が考えられる．

- 改変母体のソースコードが複雑になっている
- 改変母体の仕様を記録したドキュメントが残ってない
- 人的な要因によるもの

前章のシミュレーション結果より，“改変母体の過去の不具合修正回数”というメトリクスが，上記の要因を端的に表現するものになっていると考える．ただし，この考察に対する裏づけは，今後の課題である．

また，本考察は，一つの製品における事例にすぎない．よって，様々な製品に対し，同様に提案手法を適用することも今後の課題となる．

6. まとめ

本論文では，様々なテスト工数配分方法を評価する手法を提案した．本提案手法では，ソフトウェア信頼性成長曲線をベースにテスト工数配分方法評価モデルを作成し，実際のプロジェクトデータを元に様々なテスト工数配分方法をシミュレーションして，摘出不具合数を算出する．これにより，実際のプロジェクトのテスト工数配分との差を評価することができる．

本論文では，派生開発を行っている，ある組込み製品のソフト開発プロジェクトに提案手法を適用し，改変母体の過去の不具合修正回数に基づいてテスト工数配分を行うと最も摘出不具合数を増加できることを示した．

今後は，実際のプロジェクトにおいて，改変母体の過去の不具合に基づいてテスト工数配分し，効果を検証する予定である．

参考文献

1) Myers, G. J.: The Art of Software Testing, Wiley, New York (1980).
2) Kaner, C., Falk, J. and Nguyen, H. Q.: 基本から学ぶソフトウェアテスト―テストの「プロ」を目指す人のために，日経 BP 社 (2001).
3) 山本訓稔，秋山浩一: 直交表を活用したソフトウェアテストの効率化―HAYST 法の活用―，JaSST'04，東京 (2004).
4) 川上真澄，小川秀人，加藤正恭: デジタル家電に対する All-Pair テストの改善と適用，JaSST’08，東京 (2008).
5) Endres, A. and Rombach, D.: A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories, Addison Wesley (2003).
6) 山本徹也，岡安二郎，新井本武士，平山雅之: 機能モジュールの優先度評価に基づくテスト作業の効率化手法，IPSJ SIG Notes, Vol.2002, No.23, pp.179-186 (2002).
7) Goel, L. A. and Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measure, IEEE Trans. Reliab., Vol.28, No.3, pp.206-211 (1979).
8) 山田茂: ソフトウェア信頼性モデル，日科技連 (1994).
9) Humphrey, W. S.: Managing the Software Process, Addison Wesley (1989).
10) Jones, C.: Applied Software Measurement Second Edition, McGraw Hill (1991).
11) SourceMonitor, http://www.campwoodsw.com/sourcemonitor.html
12) McCabe, T. J.: A Complexity Mesure, IEEE Trans. softw. Eng., Vol.SE-2, No.4 (1976).
13) McConnell, S.: Code Complete, p.395, Microsoft Press (1993).