RB-001

テスト設計手法 PROST!

PROST! - a Test Design Method

鷲見　毅†，加瀬　直樹†，市田　憲明†，小笠原　秀人†
Takeshi Sumi,　Naoki Kase,　Noriaki Ichida,　Hideto Ogasawara

1. まえがき

近年のソフトウェアの大規模化，複雑化に伴い，品質確保の最後の砦としてのテストの重要性は更に増している．ソフトウェア開発の現場では，限られた開発期間の中でできる限り大量のテストを行う事で，品質確保を行っている．しかし，開発の短納期化により開発期間内に行えるテストの量が限られる為，大量のテストをただ行うだけでは，品質確保が難しくなってきている．

テストには，トラブルの原因となるバグを発見できるテストと，そうでない無駄なテストがある．限られたリソースで妥当な品質を確保する為には，テストを選別する事で無駄なテストを削減し，効果的にバグを発見する必要がある．しかし，テストを選別する為の具体的な手法は提案されておらず，効果的なテストであるかどうかは，テスト担当者の経験やノウハウに依存せざるを得ない．テストの選別を客観的に，即ち誰もが同じ様に行う為には，個々のテストがどんなバグの発見に効果を発揮するかを明確にし，必要なテストと削減するテストを合理的に決定する手法が必要になる．本稿では，テストの選別に必要な情報を可視化し，合理的にテストを選別する手法として，テスト設計手法 PROST!を提案する．

本稿は，2 章で背景と問題点について述べ．3 章でテスト設計手法 PROST!とその効果について説明し，4 章で PROST!の適用例を示す．最後に 5 章でまとめと今後の課題について述べる．

2. 背景と問題点

ソフトウェアの大規模化，複雑化と開発の短納期化により，全てのテストを現実的な期間で実施する事は，極めて困難になっている．その為，テストを削減する事が求められる．しかし，闇雲にテストの削減を行ってはバグを見逃す可性が増大してしまう為，無駄なテストを選別する必要がある．無駄なテストとは，新しいバグを発見できないテストと言う事ができる．しかし，ソフトウェア開発の現場からは，「良く似たテストを幾つも行っているが，少しずつ違いがある為，同じテストとは言い切れない」との悩みが聞かれる．これは，個々のテストが何を狙っているのか，即ち，テストの目的が明確になっておらず，何の為のテストなのかが曖昧になっている事が原因と考えられる．従って，テストを選別するには，個々のテストの目的を明確にする為の可視化を行い，客観的に違いを比較できる手段が必要と言える

しかし，テストの目的を可視化し，比較できる手法やツールは提案されていない．従来から提案されているテスト設計手法としては，CFD 法[2], HAYST 法[3]が挙げられる．これらの手法は，デシジョンテーブル，直交表といった特定のテスト技法を用いてテストケースを作成する手法として提案されており，テストの目的を可視化し，違いを比較する事には言及されていない．その為，テストの選別を行うという目的に応用する事は難しい．

こうした問題に対して PROST!では，テストで確認する事が何か，テストの対象が何かの 2 点を明確する事でテストの目的を可視化する手段と，可視化された情報を用いてテストを選別する手順を提案する．図 1 に示す様に，テストの目的を可視化し，選別する作業をテスト実行前に行う事で，効果的なテストを実現する事ができる．

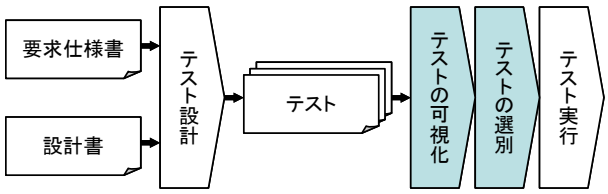

図 1：PROST!のテスト設計

3. テスト設計手法 PROST!

3.1 テストの可視化

3.1.1 LTMap（Layer Tier Map）

PROST!では，テストの目的を可視化する為の手段として LTMap（Layer Tier Map）を提案する．LTMap は，縦軸（Layer 軸）にテストの対象，横軸（Tier 軸）にテストで確認する事を取り，この 2 軸でテストの目的を表現する．即ち，何を対象として，何を確認するテストであるかを，LTMap 上の座標として可視化する（図 2）．LTMap において座標が同じテストは，その目的も同じと判断する．

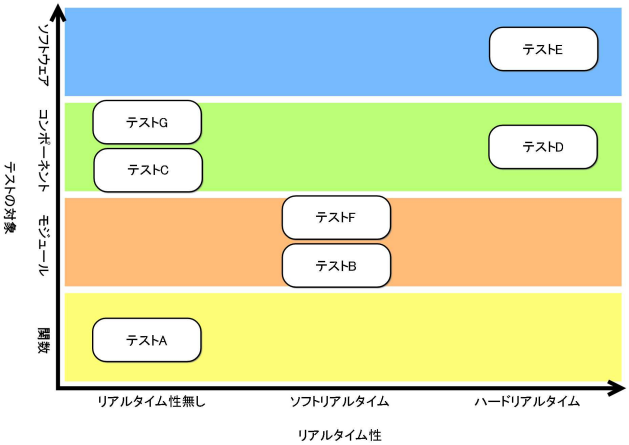

図 2：LTMap のイメージ

†（株）東芝 ソフトウェア技術センター

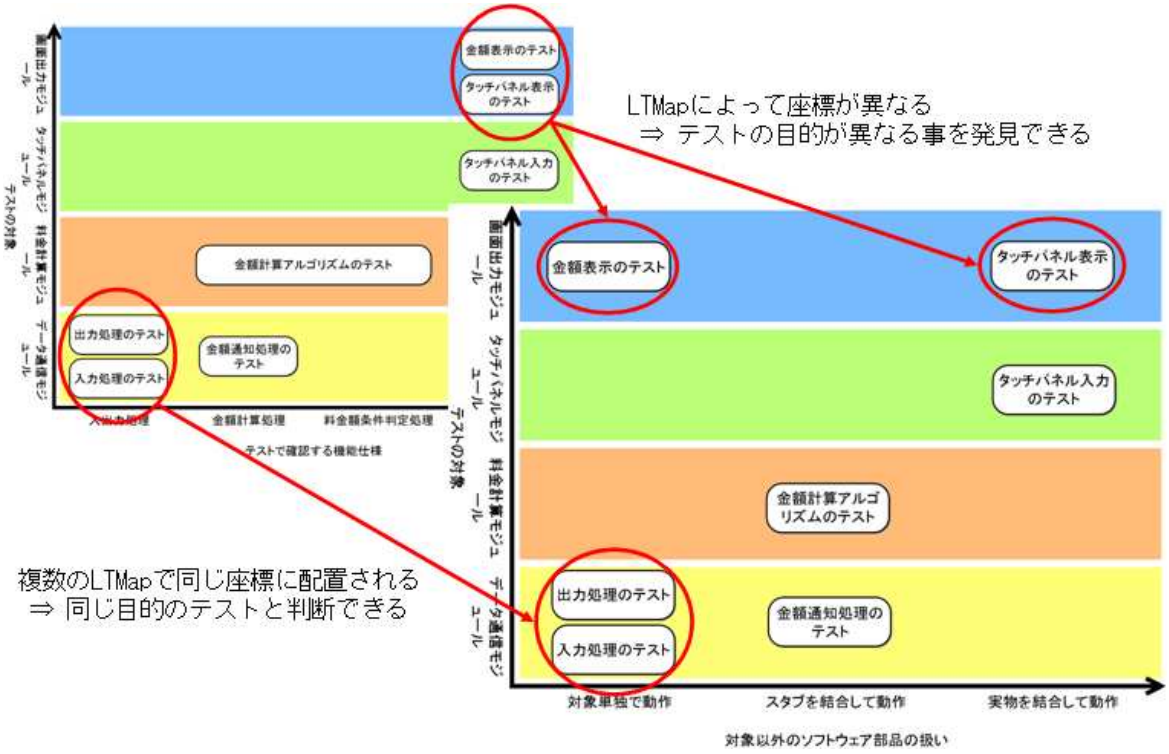

図 3：Tier 軸が異なる LTMap に配置したテスト

この時，テストで確認する事は複数存在する．例えば，機能性や信頼性，効率性等の品質特性は，それぞれの品質特性によって確認すべき事は異なると考えられる．その他にテストを実行する環境によっても，シミュレータ上での動作を確認する場合，実機上での動作を確認する場合では確認すべき事は異なると考えられる．従って，テストで確認する事は単独の観点だけで可視化できるものではなく，複数の観点から多角的に可視化する必要がある．そこで，Tier 軸が異なる LTMap を複数作成する．複数の LTMap を作成する事で，ある LTMap では同じ座標に配置されていても，別の LTMap では異なる座標に配置されているという様に，テストの目的をより詳細に把握する事ができる（図 3）．

複数の LTMap を作成する事で，良く似たテストについても，どの点で異なり，どの点で同じなのかを詳細に把握する事ができる．その為，「良く似たテストを幾つも行っているが，少しずつ違いがある為，同じテストとは言い切れない」という点に対する解決策とできる．

3.1.2 TRMap（Test Relation Map）

個々のテストは，完全に独立して存在する事は稀で，一方は機能性を確認し，他方は信頼性を確認するという様に，何らかの関係を持つ事が多い．こうした関係は，テストの実行順序等に影響する為，テスト設計でテスト間の関係を整理しておく必要がある．こうした関係を整理する例としては，大分類，中分類等にグルーピングして整理されたり，マインドマップを用いてツリー構造として整理されたりする[4].

PROST!では，テスト間の関係を整理する為の手段として TRMap（Test Relation Map）を提案する．TRMap で表現するテスト間の関係は，以下の様に定義した包含関係である．

『子テストは，親テストの一部』

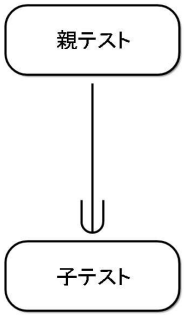

図 4：包含関係

この包含関係を，TRMap では図 4 の様に表現する．ここで，全ての子テストの集合は，親テストと等価であり，親テストは，子テストに分割できる事を意味する．また，親テストと包含関係を持つ子テストが 1 つの場合は，親テストと子テストは等価と言える．テスト間に包含関係を定義できる場合，親テストを実行する事で発見できるバグと，全ての子テストを実行する事で発見できるバグは同じになるはずである．従って，親テストと子テストのどちらを実行するかを選択する事ができる．

テスト間の包含関係を定義して行く事で，大きなテストを幾つかのテストに分割したり，幾つかのテストを統合してまとめあげたりして，テストの粒度を調整して行く事ができる．例えば，より大きな目的を持つ親テストとしてまとめて実行する場合は，テストケースの数は増大するが，ドライバやスタブ等は共通のものを使える可能性が高く，テスト準備が容易になると考えられる．一方で，目的が細分化された子テストを実行する場合はこの逆になると考え

られる．こうした点を考慮し，効率的にテストを実行する為に，テストの粒度を調整する．

しかし，こうした包含関係を最初から定義していく事が難しい場合も考えられる．その様な場合の為に，TRMap では，依存関係という関係を表現できる様にしている．依存関係は,『テスト間に包含関係を定義できる可能性があるものの，より詳細に分析しなければどの様に包含関係を定義すれば良いか分からない』という事を表現する．これは，包含関係を定義する途中過程にある事を記録する為の関係で，以下の様に表現する．

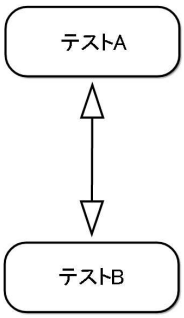

図 5：依存関係

TRMap は包含関係と依存関係を用いて，テスト間の関係を階層構造として整理し，テストの粒度を表現する．

3.2 PROST! によるテスト設計手順

PROST!では，LTMap と TRMap を用いて，テスト設計者がテストの目的，テスト間の関係をどの様に考えたかを可視化する．可視化した結果に対して，目的が重複するテストを選別し，より効果的なテストを設計していく．

LTMap と TRMap を用いて可視化と選別を行ない，効果的なテストを設計する手順の例を以下に示す．

0. テストを切り出す
1. 切り出したテストを TRMap で整理する
2. LTMap の軸を定義し，切り出したテストを LTMap に配置する
3. LTMap 上でテストが配置されていない座標に対してテストを補完する
4. 補完したテストの内容から LTMap に再配置する
5. LTMap の配置から，テストの選別を行う
6. テスト間の包含関係を再定義し，TRMap を修正する

PROST!では，この様にテストの可視化と選別を繰り返す事で，より効果的なテストの設計を行う．この手順を用いてテスト設計を行う詳細について，次章で適用例を用いて説明する．

4. PROST! の適用例

4.1 適用例の概要

PROST!の適用例として，エレベータ制御ソフトウェアの開発を用いた．これは，PC 上で動作するエレベータのシミュレータを制御するソフトウェアで，制御対象のエレベータには，乗客が押下したボタン等に応じてエレベータが移動する応答制御の他に，緊急時にエレベータを停止させる非常停止と，エレベータの状態を初期状態に戻す初期化という 2 つの制御がある．

開発するソフトの規模 3～5KLOC 程度で，開発の規模としては 3～4 人月程度になる．

比較の為に，PROST!を用いずにテスト設計を行った場合と，PROST!を用いた場合の 2 パターンでテストを実施し，テストケース数，テスト不合格数，バグ発見数を比較し，PROST!の効果を確認した．

4.2 適用の詳細

適用例の詳細の説明には，乗客にエレベータの到着を知らせる為に，エレベータの動作に応じてランプを点灯，消灯させる制御のテストを用いる．先に示したテスト設計手順に従って説明する．

0. テストを切り出す

テストを設計する最初の手順として，テストを切り出す．この段階では，要求仕様書，設計書等の開発ドキュメントから必要と考えられるテストをテストとして切り出す．この作業自体は，従来通りにテストを設計する場合にも行われている作業であり，PROST!に特有という訳ではない．

適用例では，以下の 4 つのテストを切り出した．

- ランプ制御のテスト
- 応答制御のテスト
- 非常停止のテスト
- 初期化のテスト

これらのテストは，この段階では，まだ十分に詳細なテスト，即ち目的が明確なテストとはなっていない．以降の手順で，LTMap，TRMap を用いてテスト設計者の意図を可視化し，これらのテストの目的を明確にしていく．

1. 切り出したテストを TRMap で整理する．

切り出したテストについて，TRMap を作成する．この時点では，切り出した際の意図に従って包含関係，依存関係を定義する．

適用例の場合，ランプの点灯と消灯は応答制御や非常停止，初期化からの制御要求によって制御されている．その為，「応答制御のテスト」を行うと，その中でランプの点灯と消灯も行われる．しかし，「ランプ制御のテスト」が完全に「応答制御のテスト」や「非常停止のテスト」，「初期化のテスト」の一部であるとは言い切れない為，図 6 の様に依存関係を定義した．

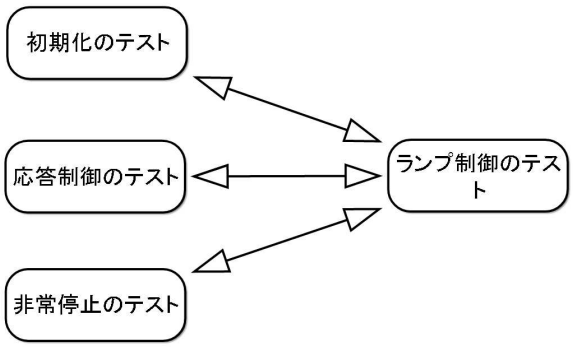

図 6：初期段階の TRMap

2. LTMap の軸を定義し，切り出したテストを LTMap に配置する

切り出したテストで確認する事と，テストの対象を LTMap の軸とし，軸の座標を定義する．この段階では，切り出したテストの特徴をそのまま LTMap の軸とその座標に用いれば良い．

適用例では，テストで確認する事として，“テストで確認する機能仕様”を考慮している．その為，LTMap の軸として“テストで確認する機能仕様”と“テストの対象”の 2 つを LTMap の軸として定義した．軸の座標は，「ランプ制御のテスト」がランプの点灯/消灯を制御する動作仕様を確認するテストであり，その為にランプ制御モジュールを動作させて行うテストとして切り出している事から，“テストで確認する機能仕様”の軸には“ランプの動作仕様”が，“テストの対象”の軸には“ランプ制御モジュール”が必要になる．他のテストについても同様に考え，LTMap の軸とその座標を定義した．

LTMap の軸を定義した後，切り出したテストを LTMap 上に配置する．この段階では，テストの特徴と LTMap の座標が対応している為，機械的に配置する事ができる．

適用例の場合は，図 7 の様になる．

3. LTMap 上でテストが配置されていない座標に対してテストを補完する

作成した LTMap でテストが配置されていない座標は，テストが抜けている可能性がある．その為，その座標が表す目的を持ったテストを補完する必要性を検討する．図 7 の場合，例えば，“応答制御モジュール”や“非常停止モジュール”，“初期化モジュール”を対象として“ランプの動作仕様”を確認するテストは LTMap には配置されていない．こうした座標に配置されるテストの有無を検討する事でテストを補完する．

適用例では，ランプの点灯と消灯が応答制御や非常停止，初期化からの制御要求によって制御されている事から，この一連の動作を確認するテストとして，「ランプの制御要求への応答のテスト」を補完した．また，応答制御の動作を確認する為には，応答制御モジュールだけでは不十分で，ランプ制御モジュールも一緒に動作させる必要がある事が分かった為，「応答制御のテスト」の対象を“応答制御モジュール”と“ランプ制御モジュール”の両方にまたがる様に補完した．「非常停止のテスト」，「初期化のテスト」についても同様である．その結果，LTMap は図 8 の様になる．

4. 補完したテストの内容を LTMap に再配置する

補完したテストを含めて包含関係を検討し，テストの粒度を調整して LTMap 上にテストを再配置する．その為に，要求仕様書や設計書に記載されている情報を用いる．

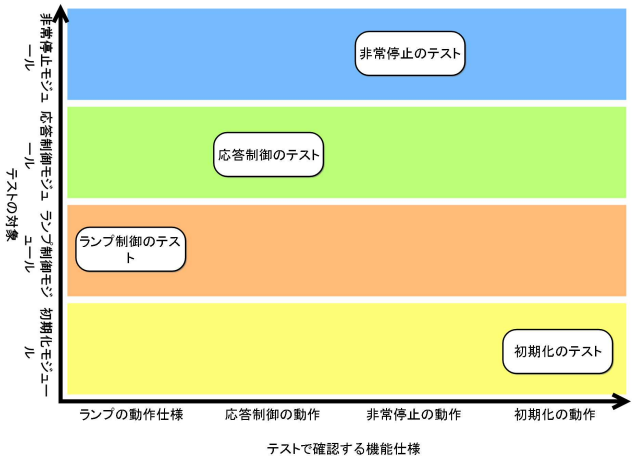

図 7：初期段階の LTMap

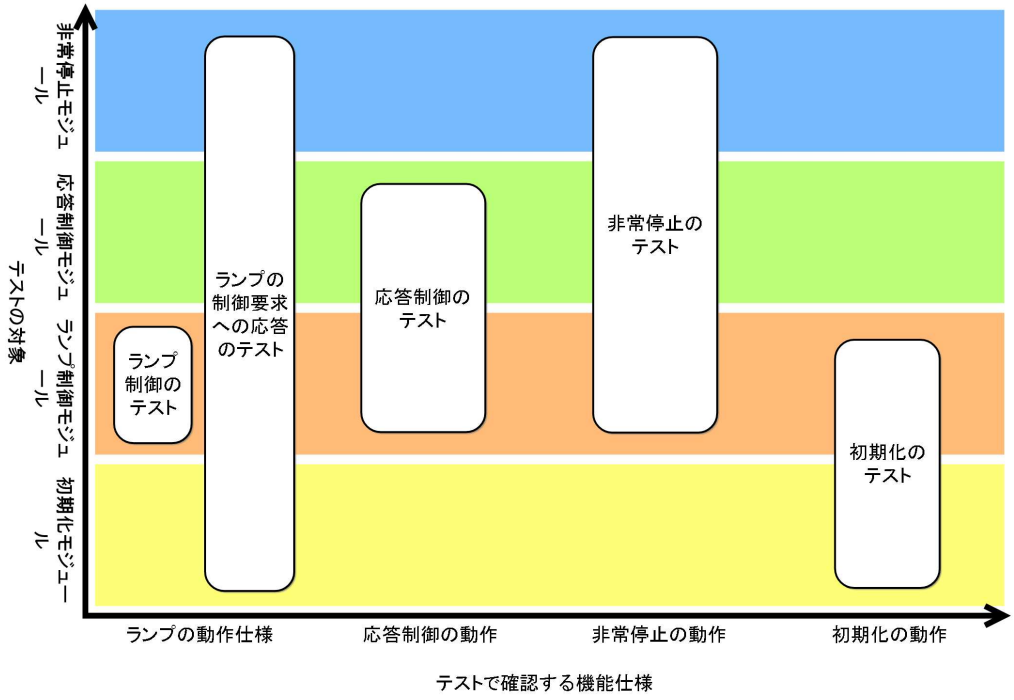

図 8：テストを補完した LTMap

この適用例では，「ランプ制御のテスト」と「ランプ制御要求への応答のテスト」の 2 つは，どちらも“ランプ制御モジュール”を対象として“ランプの動作仕様”を確認する事がテストの目的に含まれている．この部分について，テストの目的が重複しない様にテストの粒度を調整できないか検討した．その為に，ランプの制御要求が出されてからランプが点灯，消灯するまでを確認するテストの流れについて，設計書等を用いて確認した．その結果，「ランプ制御要求への応答のテスト」は，応答制御や非常停止からランプの制御要求が出されるまでを確認する「ランプ制御要求出力のテスト」と，出力された制御要求を受け取って適切なランプに振り分ける処理を確認する「ランプ制御要求の振り分けのテスト」に分割できると判断できた．また，「ランプ制御のテスト」についても，ランプの制御要求を I/F を通して正しく受け取れる事を確認する「モジュール間 I/F のテスト」と，受け取った制御要求に応じてランプを点灯，消灯させられる事を確認する「ランプの点灯/消灯のテスト」に分割できると判断した．

分割したテストは，LTMap 上に適切に配置できる座標が無い可能性がある．その為，LTMap の軸についても修正を検討する．適用例の場合も，「ランプ制御要求出力のテスト」，「ランプ制御要求の振り分けのテスト」，「モジュール間 I/F のテスト」，「ランプの点灯/消灯のテスト」を配置する適切な座標が無いと判断し，軸を修正した．“テストで確認する機能仕様”の軸の“ランプの動作仕様”を“ランプの点灯/消灯処理”と“I/F の動作”の 2 つに分割し，“テストの対象”に“モジュール間 I/F”の座標を追加した．

「応答制御のテスト」，「非常停止のテスト」，「初期化のテスト」についても同様の検討を行い，ランプの制御要求の出力を確認するテストと，制御自体の動作を確認するテストに分割し，最終的に図 9 の様に LTMap を修正した．

5. LTMap の配置から，テストの選別を行う

再配置した LTMap 上で座標が同じになるテストは目的が同じ為，これを選別する．

適用例では，「ランプ制御要求の振り分けのテスト」と「モジュール間 I/F のテスト」が同じ座標に配置されている．そこで，「ランプ制御要求の振り分けのテスト」削除して「モジュール間 I/F のテスト」のみを残した．

6. テスト間の包含関係を再定義し，TRMap を修正する

テストの選別を行った後，LTMap の配置されているテストについて包含関係を検討し，TRMap で整理する．

適用例の包含関係は，図 10 の様になる．「ランプ制御のテスト」は，「ランプの点灯/消灯のテスト」と「モジュール間 I/F のテスト」に分割した為，これらのテスト間には包含関係を定義できると判断した．また，「ランプの制御要求への応答のテスト」を分割した「ランプ制御要求出力のテスト」と「モジュール間 I/F のテスト」の間は包含関係までは定義できないと判断し，依存関係とした．

こうして定義した包含関係についてテスト設計者が妥当と判断できれば，包含関係を定義した範囲に対するテスト設計が完了する．適用例の TRMap では，「ランプ制御のテスト」に関する包含関係は妥当と判断した．

TRMap に残る依存関係についても，同様の手順でテストの補完，分割や統合，選別を繰り返してテスト全体の設計を行う．

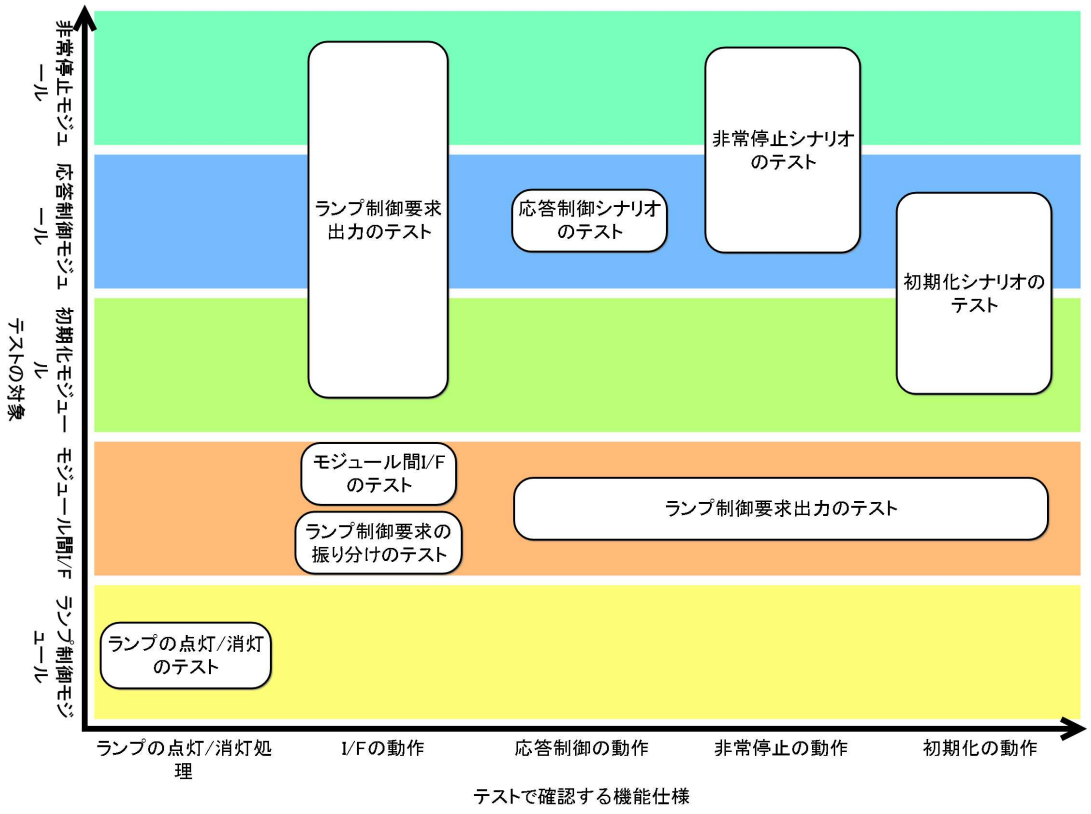

図 9：再配置後の LTMap

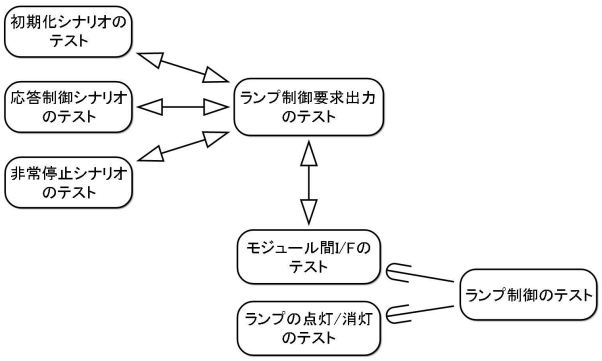

図 10：修正後の TRMap

4.3 適用結果

適用例では，エレベータ制御ソフトウェアの全体について PROST!を用いてテストを設計した．また，設計したテストに対して，具体的なテストケース（入力値と期待値のペア）を作成して実行まで行なった．適用例の結果をテストケース数，テスト不合格数，バグ発見数について従来手法と比較した結果をと表 1に図 11まとめた．この結果から，PROST!を適用する事で，従来手法より少ないテストケース数で，より多くのバグを発見できる事が分かった．

まず，バグ発見率の比較では，従来手法が約 3.2%（279 件のテストケースに対して 9 件のバグ発見数）に対し，PROST!を適用した場合は，約 14.6%（260 件のテストケースに対して 38 件のバグ発見数）と高い．また，「リフト制御」，「非常停止処理」のテストの様に，従来手法に比べて PROST!を適用した場合のテストケース数に非常に少ない場合でも，バグ発見数が大きく減る事は無い．こうした事から，PROST!を適用する事で無駄なテストを削減できたと考えられる．

テスト不合格数に対するバグ発見数の割合を比較すると，従来手法が 45.0%（テスト不合格数 20 件に対してバグ発見数 9 件），PROST!を適用した場合は約 66.7%（テスト不合格数 57 件に対してバグ発見数 38 件）と，従来手法と比較して十分に高いとは言えない．この点について分析すると，総合テストでは PROST!を適用した場合のテスト不合格数に対するバグ発見数が約 81.5%（テスト不合格数 27 件に対してバグ発見数 22 件）と高いのに対し，単体テストでは約 39.1%（テスト不合格数 23 件に対してバグ発見数 9 件）と非常に低い．

表 1：PROST!と従来手法のテスト結果の比較

		テストケース数		テスト不合格数		バグ発見数	
	テスト名	従来法	PROST!	従来法	PROST!	従来法	PROST!
単体テスト	上下ボタン点灯・消灯	2	4	0	0	0	0
	上下ランプ点灯・消灯	2	6	0	0	0	0
	フロアボタン点灯・消灯	2	2	0	0	0	0
	出力要求キュー	1	4	0	0	0	0
	上下ボタン要求発生判断	7	6	0	0	0	0
	フロアボタン要求発生判断	8	4	0	0	0	0
	非常停止ボタン要求発生判断	4	4	0	0	0	0
	初期化ボタン要求発生判断	4	2	0	0	0	0
	上下ボタン要求作成	7	18	0	10	0	3
	フロアボタン要求作成	6	23	0	2	0	2
	初期化ボタン要求作成	4	9	0	1	0	1
	非常停止ボタンが押下された時の処理	3	8	0	0	0	0
	移動要求キュー	3	3	0	1	0	1
	移動順序計算	19	29	0	9	0	2
	移動要求キューを介した移動順序計算	0	3	0	0	0	0
	ドア開閉	2	4	0	0	0	0
	リフト制御	38	6	0	0	0	0
結合テスト	初期化処理終了ロジック	5	13	0	0	0	0
	非常停止時の要求再割り当て	10	9	0	5	0	5
	上下ランプ点滅開始・終了	5	2	0	1	0	1
	ランプ・ボタン制御	6	9	0	0	0	0
	追加された移動要求をランプ・ボタンに反映	2	2	0	0	0	0
	削除された移動要求をランプ・ボタンに反映	5	4	0	0	0	0
	リフトが到着した時の処理	6	14	0	1	0	1
総合テスト	リフトの移動（単一要求）	18	4	0	0	0	0
	リフトの移動（複数要求）	20	25	0	9	0	7
	リフト進行方向15秒間保持	5	14	1	4	1	4
	初期化処理	13	9	11	6	4	3
	非常停止処理	35	6	1	1	1	1
	ログ出力	11	4	1	2	1	2
	シナリオテスト	0	7	0	3	0	3
	負荷テスト	26	3	6	2	2	2
	単体テスト合計	112	135	0	23	0	9
	結合テスト合計	39	53	0	7	0	7
	総合テスト合計	128	72	20	27	9	22
	合計	279	260	20	57	9	38

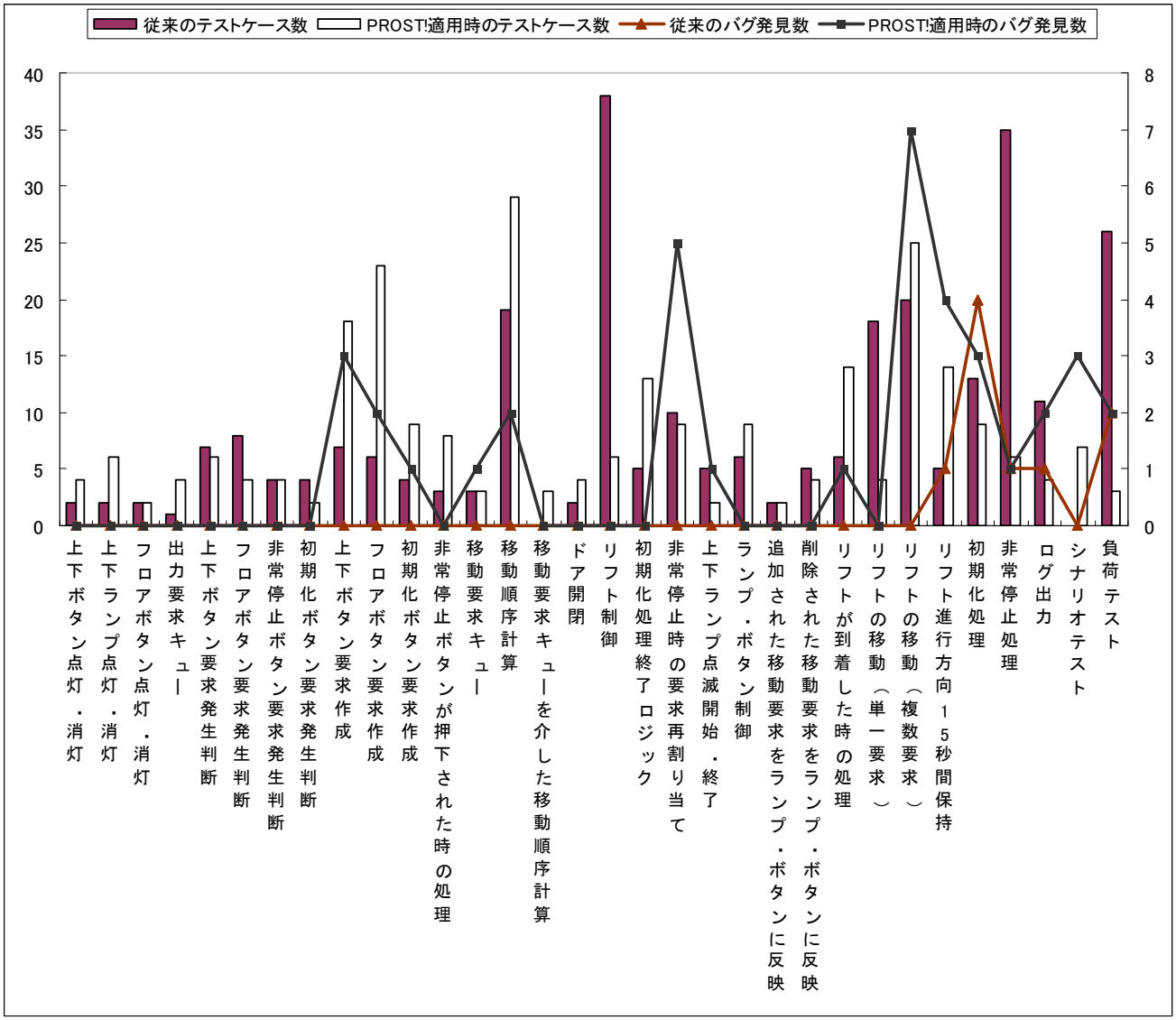

図 11：PROST!と従来手法のテストケース数とバグ発見数の分布

これは，モジュール間 I/F の様なモジュールの結合部分に原因があるバグを，単体テストで実行されるモジュール毎のテストでダブルカウントしている可能性が考えられる．この点を解決する為には，「ランプ制御のテスト」を「ランプの点灯/消灯のテスト」と「モジュール間 I/F のテスト」に分割した様に，バグの原因となる部分に対するテストを上手く切り出して定義する必要がある．しかし，単体テストの幾つかのテストでは，これが上手く行えなかったと考えられる．この点が解決されれば，単体テストでも総合テスト同様に，同じバグを複数回発見する事が少なくなると考えられる．

最後に，従来手法と PROST!を適用した場合で，テスト設計にかかった工数を比較すると，従来手法が 97 人時に対し，PROST!を適用した場合は 70 人時だった．PROST!の適用は従来手法でのテスト設計よりも後に行った為，学習効果で工数が少なくなったとも考えられる．しかし，学習効果のみで劇的に工数が削減されるとは考えにくい為，PROST!は従来手法とほぼ同程度の工数でテスト設計が行えると考えられる．

以上の結果から，PROST!の適用については，テストをどう切り出し定義するかについて，今後，手法として改善して行く必要がある事が分かった．一方，PROST!を適用する効果として，テストの目的を可視化，選別する事で，バグの発見により効果的なテストを設計できる事が判った．即ち，テストの目的を明確に定義する事で，確認すべき事を適切な対象に対して，重複を避けてテストできる様になる事が期待できる．

5. まとめと今後の課題

本稿では，効果的なテストを行う為に，無駄なテストの選別を含めたテスト設計手法として PROST!を提案した．また，エレベータ制御ソフトウェアのテストに PROST!を適用し，その効果について示した．適用例の結果からは，PROST!を用いてテストの目的を明確にする事で，バグ発見に効果的なテストを設計できる事が分かった．

本稿で提案した PROST!を用いてテストの可視化，選別を行う事は，W モデル[1]で主張されているテストの最上流に

相当する．これは，テストアーキテクチャを設計している事に他ならない．PROST!を用いる事で，W モデルに基づいたソフトウェア開発プロセスの実現が期待できる．

今後の課題としては，適用例の特に単体テストで同じバグを複数回発見してしまった点に対して，手法を洗練して行く必要がある．これは，LTMap にテストの目的を可視化する上で有効な軸と座標を用意する事で解決できると考えている．その為に，実際の開発への PROST!の適用を通して知見を集め，LTMap の軸を洗練する必要がある．

また，PROST!を用いてテスト設計を行う為には，LTMap と TRMap を作成する必要がある．この LTMap と TRMap の作成をサポートするエディタツールが必要と考えている．その為，エディタツールの開発を急ぐ．

参考文献

[1] The W Model Strengthening the Bond Between Development and Test，Andreas Spillner，STAReast2002
[2] 難しいテスト簡単にCFD法の極意【前編】, 松尾谷徹, ソフトウェア・テスト PRESS Vol.8, 技術評論社（2009）
[3] ソフトウェアテスト HAYST 法入門 品質と生産性がアップする直交表の使い方，吉澤正孝，秋山浩一，仙石太郎（著），日科技連出版社（2007）
[4] マインドマップから始めるソフトウェアテスト，池田暁，鈴木三紀夫（著），技術評論社（2007）