大規模テストへの PROST! の適用
ーテストアーキテクチャ設計工数をいかに削減するかー

市田 憲明† 鷲見 毅† 加瀬 直樹† 小笠原 秀人†

†株式会社東芝 ソフトウェア技術センター 〒212-8582 神奈川県川崎市幸区小向東芝町 1 番地
E-mail: †{noriaki.ichida, takeshi.sumi, naoki.kase, hideto.ogasawara}@toshiba.co.jp

あらまし ソフトウェアテストが大規模化する中、抜け漏れなく不具合を発見する効果的なテストを設計する手法が幾つか提案されている。そのひとつである PROST!は比較的小規模なテストにおいて、従来と同等の工数で効果的なテストを設計できることが示されている。しかし、大規模なテストに PROST!を適用する場合、PROST!が提唱するテストアーキテクチャの作成、更新に必要な工数が爆発的に膨れ上がり、人手では現実的な時間内に実施することが出来ないと考えられる。我々はテストアーキテクチャの作成、更新作業のうち、特に多くの工数が必要になる部分をアシストするエディタを開発し、大規模テストへ PROST!を適用した。

キーワード Windows，Word，ソフトウェアテストシンポジウム，テンプレート

Applying PROST! in a massive software testing
－How to reduce cost of test architecture design－

Noriaki ICHIDA† Takeshi SUMI† Naoki KASE† and Hideto OGASAWARA†

† Software Engineering Center, TOSHIBA CORPORATION 1, Komukai Toshiba-Cho, Saiwai-ku, Kawasaki, 212-8582, Japan
E-mail: †{noriaki.ichida, takeshi.sumi, naoki.kase, hideto.ogasawara}@toshiba.co.jp

Abstract With increasing size of software testing, some highly efficient methods of test architecture design are devised. Where, "highly efficient" means the method can find out many bugs without missing. One of these methods, PROST! can design a highly efficient test set at the same cost as conventional method. However, in the case that the size of a test is massive, the cost become too much to design a test by "PROST!". We develop an editor for test architecture design by "PROST!". The editor has some assist facility to reducing the work cost. We applied PROST! for massive test with This editor, and evaluated the effect.

Keyword test design，test architecture，PROST!，massive software testing

1. はじめに

近年、ソフトウェアの大規模化に伴い、そのテスト項目数も著しく増加している。テスト項目数が数千件またはそれ以上になると、テスト項目全体を把握することが難しくなる。テスト項目全体を把握できなければ、テスト項目の抜け漏れに気付くことが出来ず、不具合を見逃す恐れがある。

そのため近年、テスト項目全体の構造を把握しながらテスト設計を行う方法が提案されている[1][5][6]。これらの手法はテスト項目全体の構造を可視化するものである。テスト項目全体を可視化しながら洗練していくことで、抜け漏れのない効果的なテスト項目を設計するという考え方である。これは従来、テスト設計者が頭の中で行なっていたことを有形化する手段と捉えられる。ちょうど、UML を用いてソフトウェアアーキテクチャを設計することに似ている。すなわち、"テストアーキテクチャ"を設計していると捉えることができる。テストアーキテクチャを可視化する手段としては、既存の汎用的な記法であるマインドマップを用いた方法[5]や、NGT（Notation for Generic Testing）[6]という新しい記法を用いた方法が示されている。

この中で PROST![1]はテストアーキテクチャを表す手段として、LTMap（Layer Tier Map）、TRMap（Test Relation Map）という 2 つの新しい記法を提案している。これらの記法を用いることで、テストアーキテクチャ全体を俯瞰でき、抜け漏れのない効果的なテストアーキテクチャを設計することができる。ただし、テストアーキテクチャの可視化や変更のために追加の工数が必要となる。テスト項目数が少ない場合、この工数は従来テスト設計者が頭の中で行なっていた思考の工数と同程度になり、テスト設計全体の工数は増加しない。しかしテスト項目数が多くなるにつれて記述量

と複雑さが増加するため、工数が急激に増える。

したがって、大規模なテストに PROST!を適用するためには、この工数を削減することが求められる。これはテスト設計者が考えたテストアーキテクチャの可視化に付随する工数であるため、本来テスト設計で行う必要のない作業であり、ツールによるアシストが可能である。我々はこの作業をアシストするエディタを開発し、PROST!を大規模テストの設計に用いることを試みた。

本稿では、PROST!を大規模テストに適用するときに工数が大きくなる原因と、その対策としてのエディタについて述べる。また、エディタを用いて大規模なテスト設計に PROST!を適用した結果と考察について述べる。

2. PROST!とその課題

本章では、PROST!の概要を説明するとともに、PROST!を大規模テストに適用する場合に問題となる部分について説明する。

PROST!はテスト項目よりも抽象度が高いテスト仕様を設計する手法である。テスト仕様を扱うことで、テスト全体を把握しながらテストアーキテクチャを設計できる。テストアーキテクチャの設計で生成されたテスト仕様に対して既存のテスト技法である CFD 法[2][3]や HAYST 法[4]を適用することで、テスト項目を作成する。テスト設計プロセスにおける PROST!の位置づけを図 1に示した。

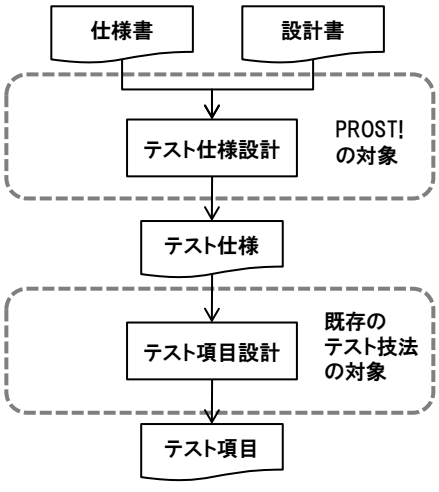

図 1 テスト設計における PROST!の対象範囲

テスト仕様を設計するために PROST!は LTMap および TRMap という 2 種類の図を用いる。これらの図は、テスト設計者がテストの抜け漏れや偏りを発見しやすいように全体の構造を示すものである。これらの図は PROST!独自のものであるため、効率よく可視化するエディタは存在しなかった。またこれらの図はテストアーキテクチャが変更されると、同時に関連する図を変更する必要があるため、変更にかかる工数が大きい。以下でテスト仕様、LTMap、TRMap についての詳細とその課題を述べる。

2.1. テスト仕様

テスト仕様はテスト項目よりも抽象度を高くした概念である。1 つのロジックや機能などをテストする場合でも、テスト項目は入力値や出力値のパターンの数だけ作成される。テスト仕様は、入力値や出力値だけが異なるテスト項目を 1 つのグループとして扱うものである。このようにテスト項目よりも抽象度を高くすることで扱う個数が減るため、テスト全体を把握しやすくなる効果がある。1 つのテスト仕様からは 5 件から 50 件程度のテスト項目が作成される。テスト仕様の数が 100 程度の場合、テスト項目数は数千件になる。テスト仕様の例を図 2 に示す。

名前	ヒーター状態決定
目的	ヒーター状態決定のロジックが以下の正常動作の仕様を満たしていることを確認する 1. センサー値が設定温度より 5℃以上低ければ、ヒーター状態を High にする 2. センサー値が設定温度以上ならヒーター状態を Off にする 3. 上記以外の場合はヒーター状態を Low にする
対象	ヒーター状態決定モジュール
手順	1. 以下をあらわす変数に入力値を設定する ・設定温度 ・センサー値 ・ヒーター状態 2. ヒーター状態決定関数を実行する 3. ヒーター状態が High、Low、Off のどの状態になったかを確認する。

図 2 テスト仕様の例

以下、テスト仕様に記述する内容について順番に説明していく。

テスト仕様の名前は、そのテスト仕様の内容を端的に表す文字列で、LTMap や TRMap 上にテスト仕様を表すときのシンボルとなる。

テストの目的は、そのテスト仕様では何を確認しているのか（どの仕様を確認するのか、など）を記述する。テストの目的は第三者がテスト仕様についての情報を理解するのに役立つ情報となり、レビューを正確に行えるようになる効果が期待できる。

テストの対象は、どの機能を確認するのか、どの関

数の動作を確認するなのか、などを記述する。テストの対象が明確に記述されることで､それらを列挙して、抜け漏れや偏りがないかを判断することができる。

テストの手順は、テストを実施するための環境、どの条件（入力や内部状態など）を変化させるのか、テストの合格条件として何（出力や内部状態、副作用など）を確認するのかを記述する。

これらの内容は、テスト設計を行う過程で徐々に作成・変更されるが、同一のテスト仕様は LTMap や TRMap 上で全て同一の名前に保たれる必要がある。この作業は単純ではあるが、テスト設計の本質的な作業ではない。また、テスト仕様の数が多くなると更新忘れなどのミスが発生する可能性がある。そのため大規模テストではこの点に対策が必要となる。

2.2. LTMap

次に LTMap について述べる。LTMap は様々な観点からテストの位置づけを表すための図である。ここで、テストの観点とは、テストの対象が利用するリソースやテストを実施する環境、関連するモジュールなど、テストに関する情報を表す様々なものが考えられる。どのような観点を用いるかについては今後の検討課題として挙げられており[1]、現状ではテスト設計者が状況に応じて作成する。それぞれの観点を軸として作成し､その中から 2 軸ずつを組み合わせて LTMap が作成される。そのため LTMap は複数存在する事になる。LTMap の例を図 3 に示した。

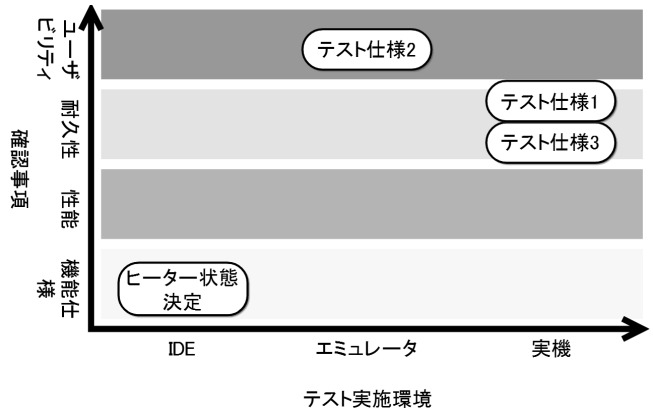

図 3 LTMap の例

LTMap 上のテスト仕様の位置は、テスト仕様に記述されている内容から判断する。テスト仕様の内容から位置を判断できない場合は、テスト仕様の内容が曖昧である可能性がある。テスト仕様の LTMap 上での位置が明確になるようにテスト仕様の記述を変更することでテスト仕様の曖昧さを取り除くことができる。

LTMap の軸は離散的なものを用いることを想定している。離散的な軸にすることで、LTMap 上の領域を有限個の領域に分割できる。有限個の領域とすることで､テスト仕様が配置されていない領域を発見できる。テスト仕様が配置されていない領域はテスト仕様が抜けている箇所である可能性があるため、その領域のテスト仕様が必要でないかを検討する。例えば、図 3には性能に関するテスト仕様が全く配置されていないので、新しくテスト仕様を追加し抜け漏れを防ぐ（変更後の図は省略)。また、1 つの領域に複数のテスト仕様が配置されている場合は、それらのテスト仕様は目的が重複している可能性がある。この場合は、テスト仕様の記述を確認し、統廃合が可能かを判断する。この過程によってテスト仕様の偏りや重複が防がれ、効果的なテストアーキテクチャを構成できる。

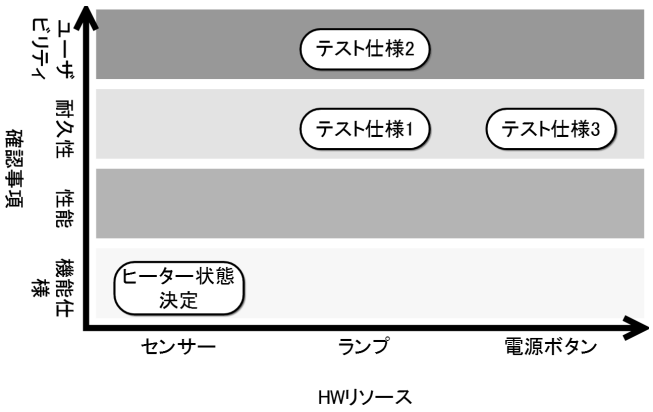

図 4 異なる軸で作成した LTMap の例

LTMap は異なる軸を使用して複数作成される。例えば、図 4 は図 3 の横軸を変えてテスト仕様を配置した LTMap である。このように LTMap を複数作成することで、テスト仕様を様々な軸で定義付ける。テスト仕様 1 とテスト仕様 3 は図 3では同じ領域に配置されているが、図 4では異なる領域に配置されている。この事によって、テスト仕様 1 及びテスト仕様 3 は、実機で行う耐久テストという点では同じだが、使用する HW リソースがランプと電源ボタンというように異なっている事が分かる。

LTMap を複数使用する場合、それらの間で整合性を保つ必要がある。例えば、図 3と図 4は縦軸が共通である。この場合、両方の LTMap に存在するテスト仕様の縦軸の座標について整合性を保つ必要がある。例えば図 3を作成した後に図 4を作成する場合、縦方向の座標は共通であるため、図 3のテスト仕様と縦方向の座標が同じになるように図 4の LTMap にテスト仕様を配置しなければならない。また、テスト仕様の座標がテスト設計の過程で変更になった場合、変更になった軸を使用する他の LTMap について､同じように位置を変更する必要がある。複数の LTMap 間の整合性を保

つための工数は、テスト仕様数や LTMap の数が増えるに従って増加する。テスト仕様の数が 100 を超えるような場合、整合性を保つために他の LTMap から目的のテスト仕様を探すだけでも、テスト設計者に大きな負担となる。また、人手で行う以上、人的ミスを完全に防ぐ事ができない。

また、LTMap はテストの構造を把握しやすくするための図であることから、その見やすさも重要である。テスト仕様数が増えると、重なりを少なくするなど視認性よくテスト仕様を配置しなければ、テスト設計者はテストの構造を把握することができなくなる。そのため、大規模なテストの設計では、テスト仕様を視認性良く配置する必要がある。

このように、大規模テストに LTMap を用いる場合、複数の LTMap 間の整合性を保つ工数、および LTMap 上のテスト仕様を視認性良く配置する工数が増加するという 2 つの課題がある。これらの作業はどちらもテスト設計者の負担となり、効率的なテストの設計作業を妨げる要因となる。そのためこれらの作業工数を削減するための対策が望まれる。

2.3. TRMap

LTMap が個々のテスト仕様の位置づけを座標として表現しているのに対して、TRMap はテスト仕様を、関連する他のテスト仕様との関係によって表現する。具体的には、1)包含関係、2)依存関係、3)類似関係の 3 種類の関係を用いる。これら 3 つの関係について、以下で順に説明する。

包含関係は TRMap において最も重要な関係である。包含関係はコ型の矢印を用いて表す（図 5)。図 5 は、テスト仕様 0 がテスト仕様 1～4 を包含している事を表している。これはテスト仕様 1～4 で確認する内容の集合（理想的には発見できるバグの集合）がテスト仕様 0 で確認する内容の集合に包含されているということである。これは、テスト仕様 0 で確認することを明確にし、より細かいテスト仕様に分割した状態であるとも言える。確認しなければならないことが多いテスト仕様は、分割することで単純化される。ただし分割すると、テスト項目数が増えてテスト実施工数が増える事も考えられる。テスト仕様を分割するかどうかは、テスト実施の工数を考慮して決定する。TRMap 全体で見た時、すべてのテスト仕様は基本的にこの包含関係によってツリー構造になる（一部は完全なツリー構造にならないため、正確にはグラフ構造である)。

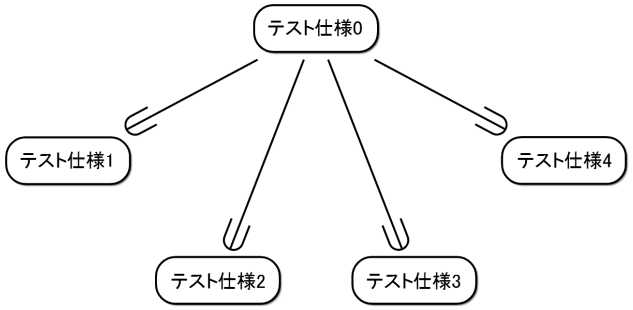

図 5 TRMap における包含関係の記法

次に、依存関係について述べる。依存関係は図 6 のように白抜三角矢印を用いて表す。図 6 はテスト仕様 5 とテスト仕様 6 が互いに依存していることを表している。これは、テスト仕様 5 で確かめる内容と、テスト仕様 6 で確かめる内容に一部重複があることを表している。依存関係がある場合、それらのテスト仕様を統合、または分割の仕方を変更することを検討する。統合する場合、2 つのテスト仕様で確認するべきことや、テストの手順をすべて含んだ 1 つのテスト仕様を作成する。この場合テスト仕様数が減る。一方、テストの手順が増え、テスト項目設計の難しさは増す。分割の仕方を変更する場合、重複している部分をどちらか一方のテスト仕様のみに含め、もう一方からは排除する。統合、分割方法の変更のどちらを選択するかは、テスト項目設計の工数、テスト実施の工数を勘案して決定する。

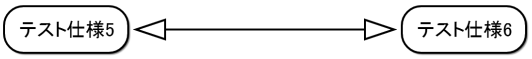

図 6 TRMap における依存関係の記法

次に、類似関係について説明する。類似関係は図 7 のように波線で記述する。類似関係はテスト仕様で確認すべき内容は異なるが、その実施手順が似ているテスト仕様が存在する場合に用いる。この関係を表現することで、テスト仕様を作成するときや、テスト項目を作成するときに、参考になるテスト仕様を参照できるようになる。類似するテスト仕様を参照することで、テスト仕様やテスト項目の内容の記述にかかる工数を削減できる。また、記述する内容の統一性を増すことができる。

図 5 TRMap における類似関係の記法

TRMap は包含関係、依存関係、類似関係を用いてテスト仕様間の関係を確認、変更しながらテスト仕様の内容を洗練していく。また、その過程で TRMap の構

造も変更される。TRMap は 1 つだけ作成することを想定しており、すべてのテスト仕様が 1 つの TRMap 上に配置される。そのため TRMap は LTMap のように他の図との配置の整合性を考慮する必要はない。しかし、テスト仕様同士が関連線で結ばれているため、その複雑さは LTMap よりも大きくなる。また、あるテスト仕様の位置を変更した場合は、視認性を良くするためにそのテスト仕様と関連している他のテスト仕様も併せて近くに移動させる必要がある。関連するテスト仕様の数は、テスト仕様の組み合わせに比例することが考えられる。そのため大規模なテストではこの工数が急激に大きくなる可能性がある。実際に大規模なテストに PROST!を適用するには、この工数を削減するための対策が必要と考えられる。

3. 大規模テストに PROST!を用いるための対策

3.1. 課題の分析

前章で述べたように PROST!は LTMap と TRMap を作成、編集しながらテスト仕様を設計していく。そして、これらの図は一度作成したら終わりではなく、図示と変更を繰り返しながら、何度もブラッシュアップして更新していくものである。その過程で、各図の編集や整合性確保のための工数が必要になる。対象のソフトウェアが比較的小さい場合、これらの工数はテスト設計全体に対して小さいためあまり問題にならない。これは図の編集にかかる工数増加と、テスト設計者が図を用いて考えを整理しやすくなる事による工数削減が同程度になるためである。

しかし、対象のソフトウェアが大規模になると急激に大きくなる工数や、対象のソフトウェアが小規模な場合には必要なかった新たな作業の工数が現れる。具体的には、前章で以下の工数が存在することを述べた。

1. テスト仕様名の変更を各図に反映する工数
2. LTMap 上のテスト仕様の座標の整合性を保つ工数
3. LTMap 上のテスト仕様を視認性良く並べる工数
4. TRMap 上のテスト仕様を視認性良く並べる工数

これらはいずれもテストアーキテクチャを記述することによって生じる工数であり、テスト設計の本質的な部分ではない。そのため本来テスト設計者が行う必要がない作業であると言える。また、その内容は機械的な作業であるため、これらの作業はツールでアシストすることが可能であると考えられる。

我々は PROST!のテストアーキテクチャを作成・編集するためのエディタを開発し、そのエディタにこれらの作業をアシストする機能を実装することで、PROST!を大規模テストに適用可能に出来ると考えた。

3.2. テストアーキテクチャ・エディタ開発

ここでは、我々がテスト設計工数を削減するために開発したエディタについて述べる。エディタの外観を図 8に示す。画面左には作成した TRMap、LTMap、テスト仕様がすべて列挙されている。右側では TRMap や LTMap を複数表示しながら、GUI でそれらを編集できるようになっている。

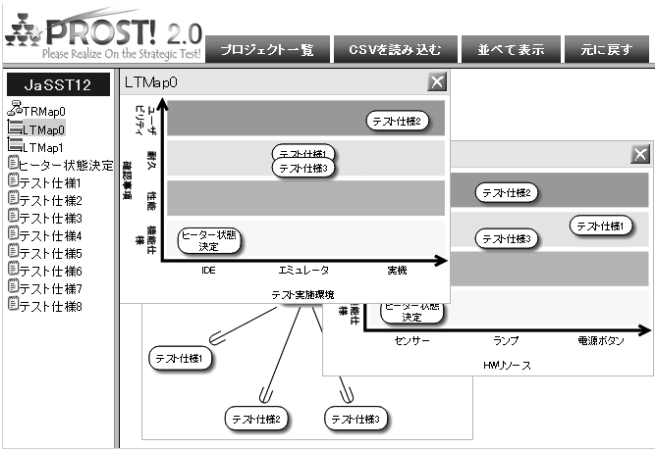

図 6 開発したエディタの画面

本エディタは PROST!のテスト仕様、LTMap、TRMap の作成・編集・保存を行うための基本的な機能を備えている。それに加えて、テスト設計工数削減のため以下の機能がある。

1. テスト仕様名変更時の一貫性維持機能：
   テスト設計の過程では、テスト仕様名が変更されることがある。テスト仕様の名前が変更されると、その変更を TRMap や LTMap に反映する必要がある。本機能はテスト仕様の名前が変更されたときに TRMap、LTMap 上のテスト仕様の名前を自動的に更新する機能である。反映していなければ、各マップ上のテスト仕様が、実際にはどのテスト仕様を指しているのかが不明になる。しかし、テスト仕様数が増えると、名前が変更されるたびに、各マップ上でそのテスト仕様を発見して名前を書き換えるのに非常に工数がかかる。本エディタはテスト仕様名が変更されると各マップも自動で更新されるため、必ず一貫性が保たれる。また一貫性を保つための作業も必要ない。また、テスト仕様名の変更は、左側の一覧、LTMap、TRMap のどの場所からでも書き換えられる。
2. LTMap 座標同期機能：
   本機能は同じ軸を共有する複数の LTMap においてテスト仕様の座標の整合性を保つ。2.2節で

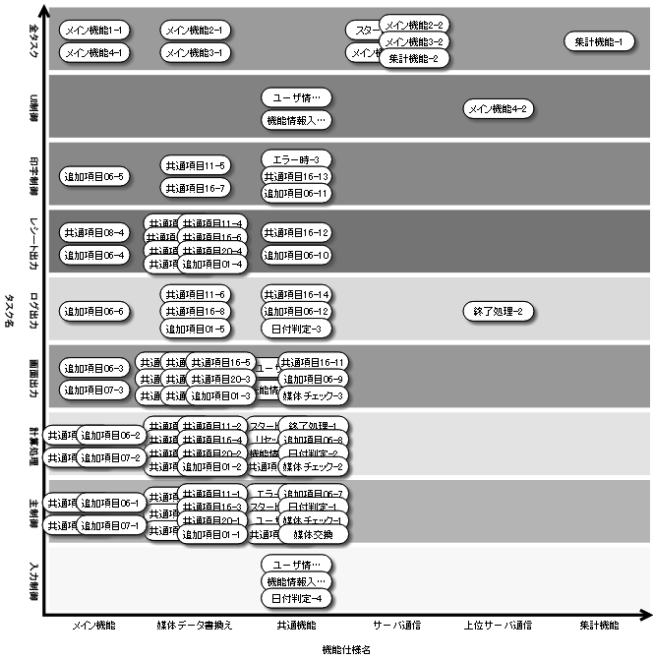

図 9　適用事例で作成した LTMap1

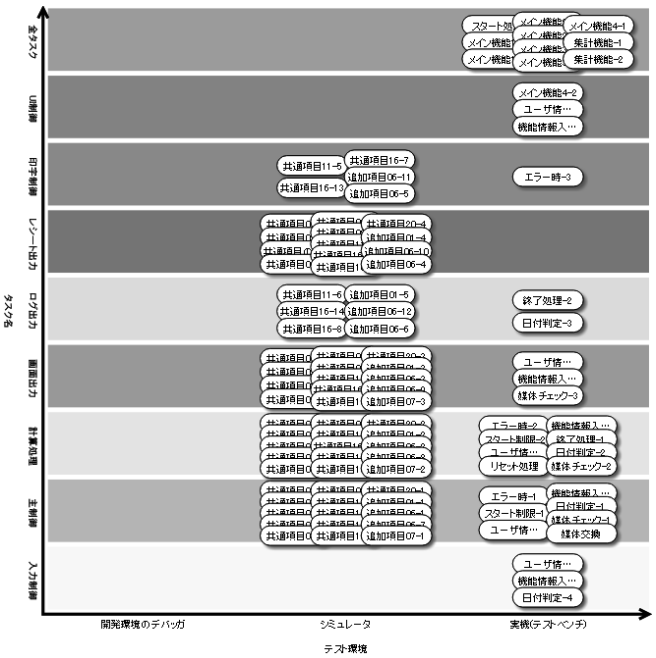

図 10　適用事例で作成した LTMap2

説明したように、共通する軸を使用する LTMap が複数ある場合は、そこに配置されているテスト仕様の座標の整合性を保つ必要がある。これは、初期配置だけでなく、テスト設計作業中にテスト仕様の座標が変更されたときにも必要になる。本エディタは共通の軸を持つ LTMap を作成するとき、初期配置は自動的に既に存在する LTMap に合わせて設定される。また、テスト仕様の位置を変更した場合は、軸を共有するすべての LTMap において、そのテスト仕様の位置が同時に移動する。テスト設計者は他の LTMap のことを意識すること無く、テスト設計に集中できる。

3. LTMap 上のテスト仕様の自動整列機能：
   LTMap は離散的な軸によって、幾つかの領域に分割されている。そして、それぞれの領域内には複数のテスト仕様が配置されることがある。テスト仕様が重なっていると、どの区画にどのテスト仕様が配置されているのか把握できなくなる。本機能は、それらが重ならないように整列する。テスト設計者はどのテスト仕様がどの領域に入るかのみを操作するだけでよく、領域内での細かな位置調節を行う必要がない。
4. TRMap 上のテスト仕様のツリー整列機能：
   本機能は TRMap 上の包含関係がツリー構造になっていると仮定して、選択したテスト仕様を整列する。ツリー構造になっていない部分については、無限ループに陥らない様に処理される。本機能を使用することで、テスト仕様を分割したときなどの再配置にかかる時間を短縮できる。本機能はテスト仕様の整列の大部分を行うが、全体を一度に自動的に整列しない。テスト設計者が選択した部分を、テスト設計者の操作に基づいて整列する。これは TRMap 上のテスト仕様の配置を、ある程度テスト設計者の意図通りにすることが、テスト設計者の思考を支援すると考えたからである。

我々は、これらの機能を持ったエディタを用いて、テスト項目が数千件になるテスト設計に PROST!を適用した。次章ではその結果について述べる。

4. 実適用結果

4.1. 対象ソフトウェア

我々が PROST!を適用したソフトウェアは社会インフラ系システムのソフトウェアである。このシステムの開発は、既存のシステムを変更する派生開発である。そのため、変更によって追加されたテストと回帰テストが混在している。今回は、派生部分と派生によって影響を受ける部分のテスト設計に PROST!を適用した。適用対象部分の規模は、総合テスト段階のテスト項目数で 2 千件程度となる。また、テスト仕様設計工数は 5 人日程度までであることが求められる。

4.2. 適用結果

LTMap の軸は、“タスク名”、“機能仕様名”、“テスト環境”の 3 つを用いた。これらから、“タスク名－機能使用名”および“タスク名－テスト環境”の 2 つの

組み合わせで LTMap を作成した（図 9、図 10)。

これら 2 つの LTMap は縦軸が共通である。また、これら 2 つの LTMap 上に配置さているテスト仕様の 9 割以上は 2 つの LTMap の両方に出現している。したがって、テスト仕様の座標の整合性を確保することが求められる状態である。図 9、図 10 のテスト仕様の座標は LTMap 座標同期機能により、整合性は完全に確保されている。

また、図 9 および図 10 は LTMap 上のテスト仕様の自動整列機能によって整列が行われた状態である。図 11 は図 9 に整列機能を適用する前の状態である。

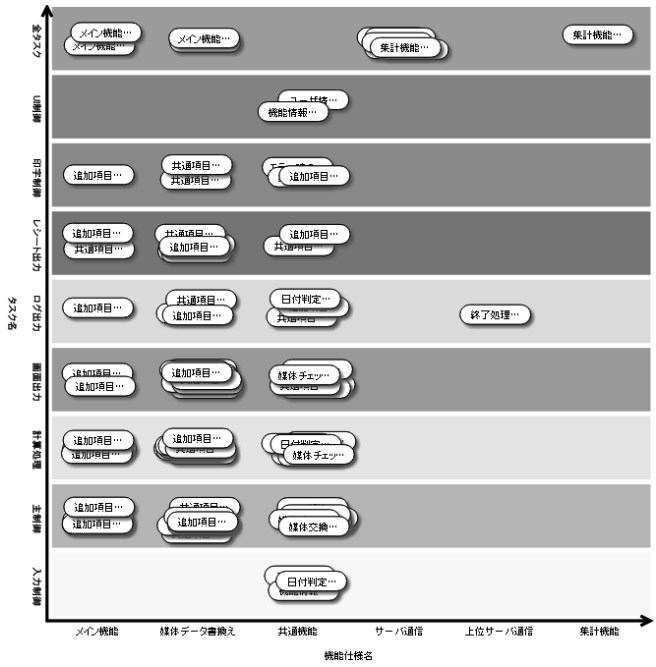

図 7　整列前の LTMap1

図 11 はテスト仕様が重なっている。そのため、各座標にいくつのテスト仕様が配置されているかが分かりにくい。また、各座標にどんなテスト仕様が配置されているか把握しにくい。両者を比較すると、図 9 の視認性が高いことが分かる。

また、作成した TRMap は図 12 のようになった。最終的な TRMap には依存関係および類似関係は現れなかった。テスト仕様の配置は包含関係をツリーとみなしたときに、親ノード側になるテスト仕様が中央付近に配置され、そこからテスト仕様が分割される様に配置されている。これはテスト設計者が TRMap 上のテスト仕様のツリー整列機能を利用しながら配置した結果である。アシスト機能を利用することで、容易にテスト設計者の意図を反映した配置にすることができた。

今回の適用では、LTMap と TRMap を 1 回作成するだけでなく、それらを用いてテストアーキテクチャのブラッシュアップを行なった。その過程でテスト仕様の追加・変更・削除が行われたが、最終的に作成された 105 個のテスト仕様すべてについて、LTMap と TRMap 上のテスト仕様名は一貫している。これはエディタが行なっているものであり、テスト設計者はこの作業を意識せずにテスト設計を行えた。

エディタを用いて PROST!を実適用した場合にかかった工数は約 3.5 人日である。またテスト設計にかかった工数のうち、今回課題にあげた作業にかかった工数は約 0.4 人日である。この 0.4 人日はすべて TRMap 上のテスト仕様の整列の配置調整によるものである。他の 3 つの作業はすべてエディタによって瞬間的に終わるため、工数は 0 となった。残りの 3.1 人日はテスト設計の工数である。

4.3. 考察

エディタのアシストによって、LTMap と TRMap を作成し、それらをブラッシュアップしながらテスト仕様を設計するというテスト設計の作業を現実的な工数で行える事がわかった。

今回は実適用であるため、エディタを用いない場合のテスト設計は行なっていない。しかし、エディタを用いない場合の工数を見積もるために幾つかの実験を行った。

まず、LTMap 間のテスト仕様位置の整合性を保つ作業を見積もるために、図 9 のテスト仕様をランダムに 10 個移動させ、その移動を図 10 の LTMap に人手で反映した。このとき、エディタが存在しない状態を想定し、LTMap 上のテスト仕様は整列していない状態で行った。この作業にかかった時間を測定したところ、約 7 分であった。今回のテスト仕様は全部で 105 個であるため、全体では約 75 分程度必要になると考えられる。初期配置も含めて、テスト設計作業全体で 5 回の整合性確保作業を行うとすると、約 0.78 人日必要になる。

次に LTMap 上のテスト仕様の整列工数を見積もるために、図 9 の LTMap に対して、各領域内のテスト仕様をランダムに配置しなおし、その状態から、人手で各テスト仕様が重ならないように整列しなおした。その結果、整列しなおすのに約 20 分必要であった。今回は LTMap を 2 つ作成しているため、両方の LTMap を整列するには約 40 分必要になると考えられる。今回は LTMap 上のテスト仕様の自動整列機能によってこの作業を工数 0 で行えるため、百回以上の再整列を行なっている。手作業の場合はこれほどの回数の再整列を行わないと考えられるので、全体で 10 回程度と見積もると、その工数は約 0.83 人日となる。

テスト仕様名変更、TRMap 上のツリー整列についても、それぞれ LTMap の整列作業と同程度の作業と仮定して 0.8 人日程度の工数が必要と見積もると、合計で

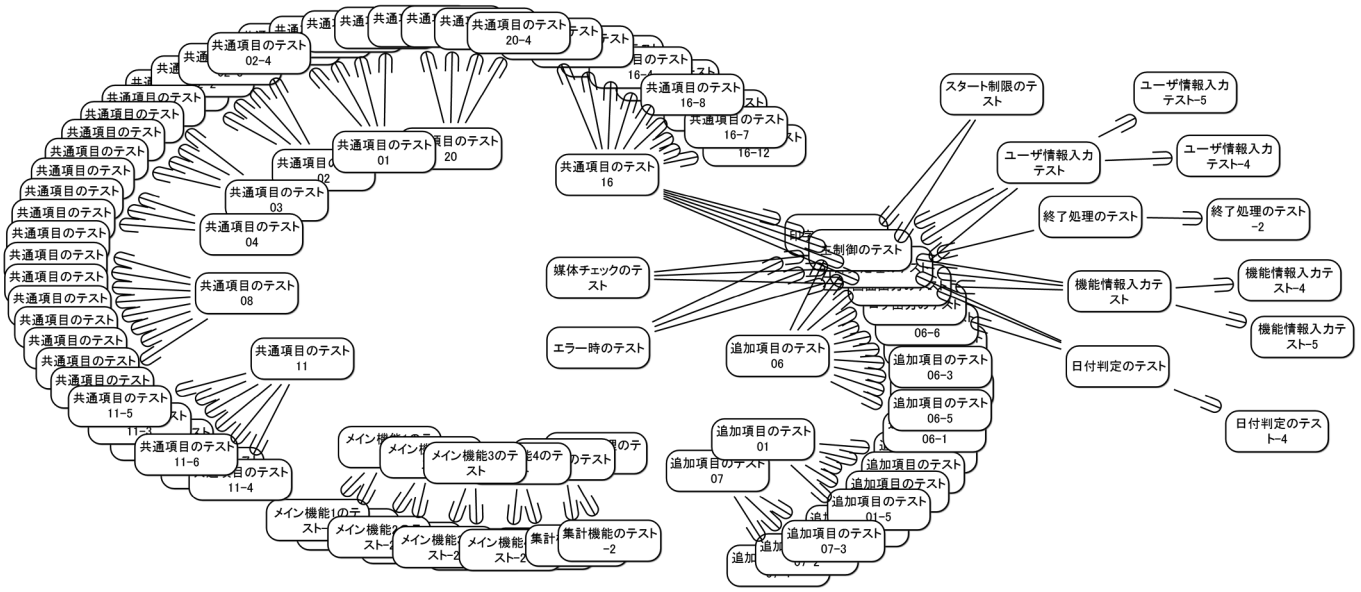

図 12　TRMap 上のテスト仕様のツリー整列機能を用いて整列した TRMap

3.21 人日程度の工数が追加で必要となる。この工数をテスト設計の工数 3.1 人日と合わせるとテスト仕様設計にかかる工数は 6.31 人日程度だったことになる。図 13 に工数を比較したグラフを示した。現在の短納期化が求められる開発現場において、5 人日程度の工数が期待される工程に対して 6.31 人日の工数が必要になる手法は受け入れがたいと思われる。これに対して、エディタ使用によって約 3.5 人日で PROST!を適用できた今回の成果は一定の評価ができるものと考えられる。

また図 13 では本稿で課題として挙げた作業以外の、テスト設計の本質的な工数については、エディタのアシストの有無にかかわらず同じ工数になると仮定している。しかし、エディタのアシストによってテスト設計者が作業に集中できるようになると考えられる。そのため定性的には、この部分の工数もツールアシストによってある程度縮小されていると考えられる。

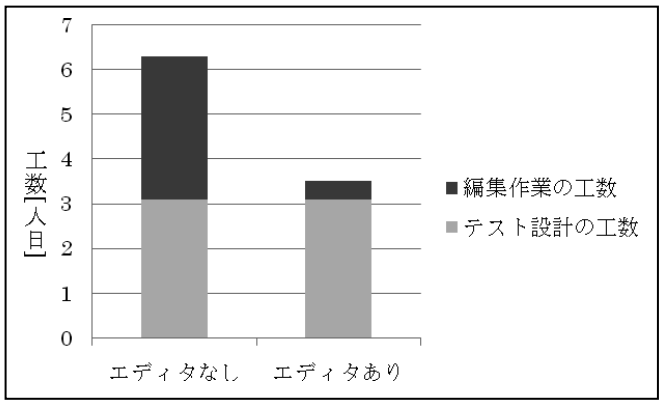

図 13　工数の比較

5. おわりに

本稿では大規模テストに PROST!を適用する場合に工数が大きくなる作業が存在することを述べ、それらの作業は機械的に行うことができることを説明した。また、それらの作業をアシストするエディタを使用することで、工数が削減され、PROST!を実製品のテストに適用可能になることを示した。

今回の事例では、テスト設計にかかる工数が現実的な範囲に収まるかどうかを評価した。作成したテストによって発見できた不具合は妥当と考えられるが、効果的であったかどうかをバグの発見率を用いて定量的に評価することも今後必要である。

文　献

[1] 鷲見毅，加瀬直樹，市田憲明，小笠原秀人，“テスト設計手法 PROST!,”FIT2011 第 10 回情報科学技術フォーラム講演論文集，第 1 分冊, pp.43-50, Sep.2011.

[2] 松尾谷徹，難しいテストを簡単に CFD 法の極意【前編】 ソフトウェア・テスト PRESS，vol.8, 2009.

[3] 秋山浩一，ソフトウェアテスト技法ドリル テスト設計の考え方と実際，日科技連出版社，2010.

[4] 吉澤正孝，秋山浩一，仙石太郎，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，日科技連出版社，2007.

[5] 池田暁，鈴木三紀夫，マインドマップから始めるソフトウェアテスト，技術評論社，2007.

[6] 西康晴，テスト設計におけるモデリングのための記法の提案，JaSST2006 Tokyo.