テスト詳細設計プロセスの手順定義によるテストケース抜けの防止

Definition of procedure for test detail design process to prevent test case omission

Test Engineer's Forum 北海道（TEF 道）　コミュニティ
Test Engineer's Forum in Hokkaido　Community
○水野 昇幸　中岫 信[1)]　情野 吉紀[2)]
○Noriyuki Mizuno　Makoto Nakakuki[1)]　Yoshinori Seino[2)]

Abstract Recent years, scale of software development has become larger and development period has become shorter. In this situation, design docs that used for test base often have fluctuations, scattered information and no required information to design test cases. These problems of test base cause test case omission.

This paper proposes the procedure for test detail design process to prevent test case omission. Each procedure is corresponding to the problem of test base. Templates of the procedure support to design test cases.

The result that we compare test cases designed by this proposed procedure and conventional test cases shows effectiveness of prevention of test case omission.

1. 概要

近年、ソフトウェア開発は規模が大きくなり、1 つのプロダクト構築に向けて複数名が作業を行うことは当たり前になっている。複数名で同一の設計文書をつくる場合、テストベースにもなる設計文書にばらつき（記述項目の粒度や表現の曖昧さの度合いの違い）が発生してしまうことが多い。また、開発期間の短さ、情報を提供する締め切りの厳格さによって、テストベースにおいて情報の散逸や、必要な情報が存在しない、という状況が発生する。

これらのテストベースにおける問題がテスト設計を困難にする。テスト設計の困難さがテストケース抜けを発生させ、最終的に不具合へつながってしまう。市場での不具合は大きな損害をもたらすリスクとなる。

本論文では、テストベースとなる設計文書にばらつき、情報の散逸、必要な情報が存在しない状況を想定し、テストケース抜けを防止する「テスト詳細設計プロセスの手順」を提案する。テストベースの各問題に対応したテストケース設計手順を定義し、各手順に適した設計用テンプレートを用意した。これらを活用して設計を行うことでテストケースの抜けを防止する。

また、提案する方法を用いてテストケースを実際に作成し、従来的なテスト設計方法と比較を行った。結果として得られたテストケース抜け防止の効果についても報告する。

なお、記載した問題は設計側での解決が理想的だが、本論文ではまずテスト側で対処を行った上で、検出した問題発生状況を活用して設計側の改善を行うという方針で提案を行う。

2. 背景

ソフトウェア開発は規模が大きくなりつつあり、1 つのプロダクト構築に対して複数の担当者にて要件定義や設計、開発作業を行うことが当たり前となっている。作業を行う担当者はドメイン知識や開発経験に差があり、設計文書を含めた開発成果物に対して記述項目の粒度や表現の曖昧さの度合いにばらつきを発生させてしまうことがある。大規模の開発では要件定義や設計に必要な情報をもつステークホルダも増えるため、情報獲得の活動も困難となる。

Test Engineer's Forum 北海道（TEF 道）　コミュニティ
Test Engineer's Forum in Hokkaido　Community

北海道苫小牧市新富町 1-7-19　Tel: 090-2690-9944　e-mail:mizu-nori@e-mail.jp
1-7-19, Shintomi-Cho, Tomakomai-City, Hokkaido, Japan
1) Test Engineer's Forum 北海道（TEF 道）　コミュニティ 工作員
Agent, Test Engineer's Forum in Hokkaido
2) Test Engineer's Forum 北海道（TEF 道）　コミュニティ 戦闘員
Fighter, Test Engineer's Forum in Hokkaido

【キーワード：】テストプロセス、テスト詳細設計、テスト設計、テストベース、テストケース

また、近年の競争激化によりプロダクトのリリース間隔が短縮されていることから、開発期間が短くなっている。この開発期間の短さによる「間に合わないモンスター」[1]が、担当者に対して強迫観念を発生させてしまい、必要な作業を省いてしまう場合もある。

加えて、特に日本では要件定義や設計、開発工程を分割して別組織（別企業）へ発注して行うことが多い。この工程を分割した仕組みでは、後工程のプロセス（図 2-1 では設計工程）を開始するために一括した情報（図 2-1 では要件定義書記載の機能一覧）を発注元から発注先へと渡す必要がある。この契約上の制約と開発期間の短さから、担当者は厳しい締め切り期間で一式の揃った情報を用意することが求められる。この条件下では、必要な情報すべてを担当者が十分に記載できない場合も存在する（図 2-1 参照）。

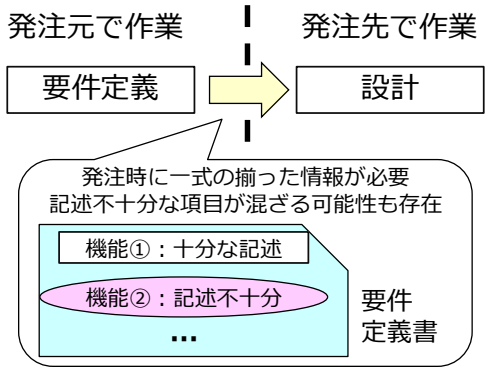

図 2-1. 工程の分割と契約上の制約（例）

3. 問題点

3.1 節および 3.2 節にて問題を整理する。その結果として 3.3 節に記載するテストケース抜けへとつながってしまう。本論文で解決するターゲットに設定する対象を 3.4 節に記載する。

3.1 テストベースに発生する問題点

「2. 背景」に記載した状況により、テストベースに対して次のような問題が発生しやすい。ここでの問題点は、主に要件定義や上流設計において発生しやすい項目を対象とする。

(1) テストベースとなる要件定義書や設計文書記述のばらつき

ドメインの知識や開発経験の異なる複数名の担当者で要件定義書や設計文書を作成する場合には、記述項目の粒度や表現の曖昧さの度合いにばらつきが発生しやすい。たとえば、図 3-1 で機能要求項目ＡとＢのサイズが違うように、項目ごとに粒度が大きく変わる場合や、機能要求項目Ｄのように表現の曖昧な記述が含まれる場合もある。

これは文書フォーマットや、要件定義プロセス内の手順が明確ではない場合に特に発生しやすい。加えて、厳しい締め切り期間によって整合性を取る時間が確保できなく、ばらつきに気がついたとしても修正ができない場合もある。

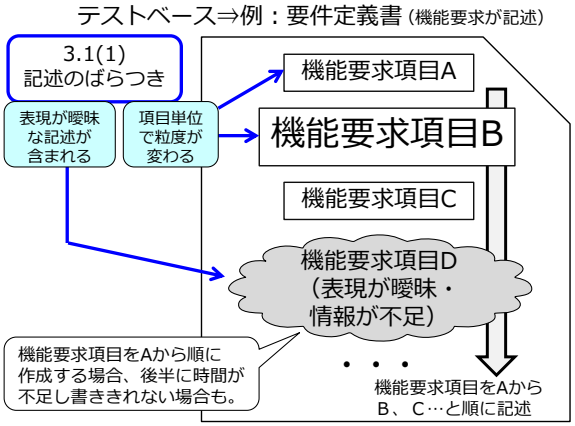

図 3-1. テストベース記述ばらつきの例

(2) テストベースの対象箇所に必要情報が無く、他の場所に散逸して記述される

テストベースにおいて、本来記述される箇所（機能要求項目など）に必要な項目が記述されず、他の場所に散逸する場合がある。次の 3 つの可能性が考えられる。

・テストベースと同一文書の他場所に情報が記述される

図 3-2 のように要件定義書において、機能要求項目を機能要求項目ＡからＢ、Ｃ…と順番に検討する作業を例とする。この際、機能要求項目Ｃを記述している段階で、すでに記述済の（項目Ｃと関連をもつ）機能要求項目Ａにも必要なパラメータ情報に気が付く場合がある。この際に、機能要求項目Ｃ側に機能要求項目Ａ関連情報が記載されてしまうことがある（図 3-2「3.1(2)同一文書の他場所に記述」参照）。

理想的には整合性を取るべきだが、機能要求項目Ａを修正する時間が取れない場合も多く、結果として情報が散逸してしまう。

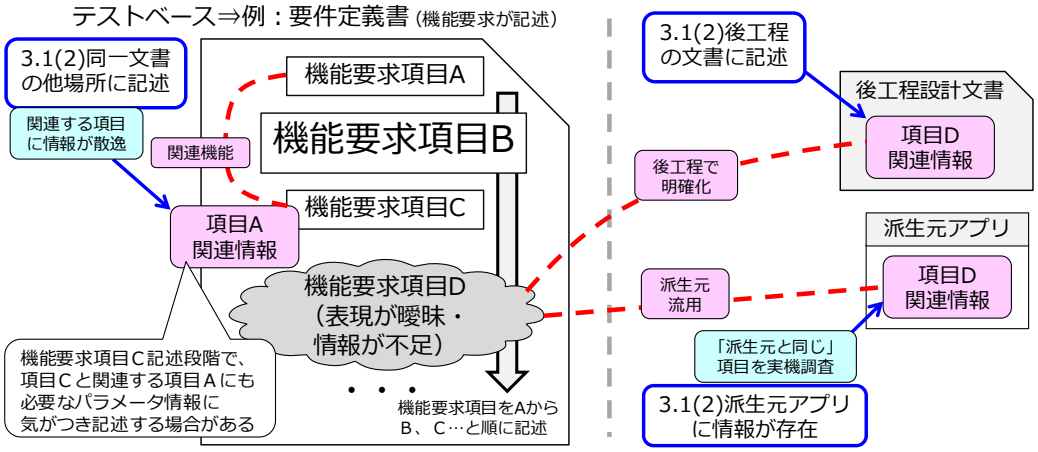

図 3-2. テストベースの対象箇所に必要情報が無く、他の場所に散逸して記述される例

・後工程の設計文書に情報が記述される

情報提供元となる外部ステークホルダが予定を確保できないという外部要因や、後半の時間不足により、要件定義書や設計文書の締め切りまでに十分な情報が記述できないことがある。

このような場合には、不明（もしくは曖昧）な状態で後工程へ進み、後工程にて不明な情報を明らかにする。結果として、必要な情報が後工程の設計文書に散逸する場合がある（図 3-2「3.1 (2)後工程の文書に記述」参照）。

・派生元の文書もしくは実アプリのふるまいとして情報が存在する

派生開発の場合において、派生元から変化のない項目は設計文書に記述されない場合がある。実際には派生元の設計文書にも記述がない場合もあり、実アプリを動かすことでふるまいを確認しなければならない場合もある（図 3-2「3.1(2)派生元アプリに情報が存在」参照）。

(3) 対象テストベース以外を含めてどこにも必要な情報が存在しない

必要な情報が不明（もしくは曖昧）な状態で後工程へ進められ、最終的にプログラム段階で何らかの決定が行われたような場合、担当者の頭の中だけに情報がある場合には、必要な情報は存在しない場合がある。

以上のように、テストベースの対象項目におけるばらつき、情報の散逸、必要な情報が存在しない、といった問題が存在する。

3.2 テスト設計の進め方による問題点

テストを設計する際には、単純にテストベースの項目と対応付けてテスト項目を検討する場合が多い（図 3-3 参照）。

テスト設計担当者にドメイン知識やテストの知見がある場合には、テスト設計時にテストベース記載の情報に加えて必要なパラメータなどの情報を追加してテストケースを作ることができる。しかし、「3.1 テストベースに発生する問題点」記載のテストベースのばらつき、情報の散逸、必要な情報が存在しない状況下にて、テスト設計担当者の知見ですべての対処は不可能である。

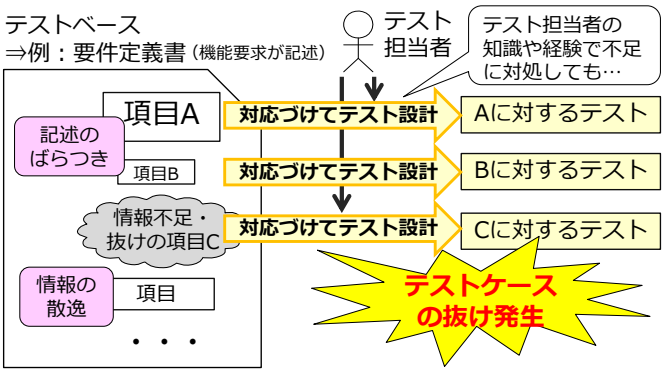

図 3-3. テストベース項目と対応づけてテストを検討

3.3　結果として発生するテストケース抜け問題

3.1 節、3.2 節に記述した問題点の結果として、確認すべきテストパラメータの抜けが発生し、テストケースで確認が不十分（テストケース抜け）となってしまう。そこから、最終的に市場での不具合へとつながり、損害に繋がるリスクも発生してしまう。

3.4　ターゲットとして設定する対象

本論文では、3.1 節記載のテストベースにおける各問題が発生しうるものとして、3.2 節記載の単純にテストベースに対応付けたテスト設計を改善することで、テストケース抜け問題の解決を目指した。

なお、テストベースの問題は設計側で解決することが理想的だが、本論文ではテスト側での問題対処を行った上で、検出した問題を活用して設計側の改善を行う方針で提案を行う。

4.　解決策

テストケース抜けを防止するため「テスト詳細設計プロセスの手順」を提案する。

4.1　TDLC で定義されているテストプロセスとテスト詳細設計

テストプロセス全体は「TDLC（Test Design Life Cycle）」[2]におけるテスト開発プロセスを用いる。TDLC はテストを開発するためのプロセスをテスト要求分析、テストアーキテクチャ設計、テスト詳細設計、テスト実装の 4 つの段階に分ける（図 4-1 参照）。

図 4-1. TDLC におけるテスト開発プロセス

本論文で手順を提案する対象のテスト詳細設計は、分割されたテスト範囲[3][4]におけるテストパラメータを特定し、それを組合せてテストケースに落とし込むプロセスである。

4.2　テスト詳細設計プロセスの手順定義

テスト詳細設計において、テストベースのばらつき、情報の散逸、必要な情報が存在しないという条件下にてテストケースを確実に作成するため、図 4-2 の上部記載(1)～(4)の手順を定めた。

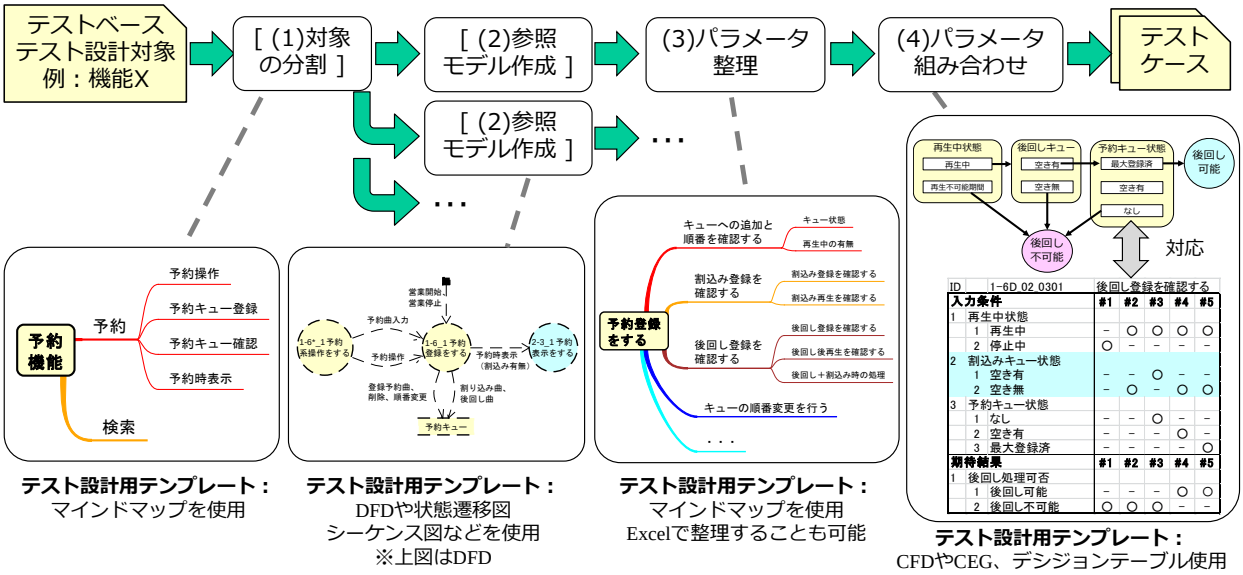

図 4-2. テスト詳細設計手順（上部記載(1)～(4)）と対応したテスト設計用テンプレート

4.3　テスト詳細設計の各手順とテスト設計用テンプレート

図 4-2 にテスト詳細設計の各手順と対応したテスト設計用テンプレートを示す。図 4-2 の(1)～(4)に対応する各手順詳細と、対応したテスト設計用テンプレートについて次に説明する。

(1) 対象の分割（必要に応じて）

テストベース記述のばらつきに対処する。テスト設計対象（例：機能要求項目）がテスト設計にはサイズが大きい場合に、対象の分割を行うことでテスト設計しやすいサイズに分ける。

テスト設計用設計テンプレートとしては主にマインドマップを使用する。

(2) 参照モデルの作成（必要に応じて）

テストベースにおける情報が不足している、もしくは情報が存在しない場合に「参照モデル」を作成することで必要な情報を補完する。必要に応じて DFD や状態遷移図、シーケンス図を参照モデルのテスト設計テンプレートとして活用する。

情報が明らかな場合や、後工程の設計文書に必要な情報が存在して引用可能な場合においてこの手順を実施する必要はない。

(3) パラメータ整理

テストベースの他箇所や他文書に情報が散逸していることを考慮して、パラメータを整理する。テンプレートは主にマインドマップを使用する。Excel 表形式でまとめることも可能である。

テストベースだけでは不足している情報を明らかにするため、必要に応じて成果物となるマインドマップをレビューすることで、不足の有無や情報の存在箇所を確認することができる。

(4) パラメータ組合せ

導出されたパラメータを組合せてテストケースを作成する。設計テンプレートとしては、必要に応じて CFD（Cause Flow Diagram）や CEG（Cause Effect Graph）を用い、結果をデシジョンテーブルにまとめる[5]。有則の組合せではない場合には、CFD や CEG は不要である。

5.　実施結果

本論文で提案するテスト詳細設計の手順を用いてテスト設計を行った結果について示す。同一のテスト設計対象に対して 2 種類の方法でテストケースを設計して比較を行った。

5.1　比較方法

同一のテスト設計対象に対して、単純にテストベースと対応付けてテスト設計を実施する方法、および今回提案する手順を適用した方法の 2 種類でテスト設計を行った。

(1) テスト設計対象

テスト設計対象として、一般公開されているテストベース「ASTER カラオケシステム要件定義書_Ver2」[6]を使用した。

(2) テスト設計方法

テスト設計方法として、次の 2 種類を用いた。共に同一のテストベースから設計を実施した。

- **方法 1：単純にテストベースと対応付けてテスト設計を行う**
- **方法 2：本論文にて提案するテスト詳細設計手順でテスト設計を行う**

なお、方法 1 で設計されたテストケースは、テストケースの粒度が荒いため、作成されたテストケースをさらに追加分割して方法 2 のテストケースと比較を行っている。(図 5-1 参照)

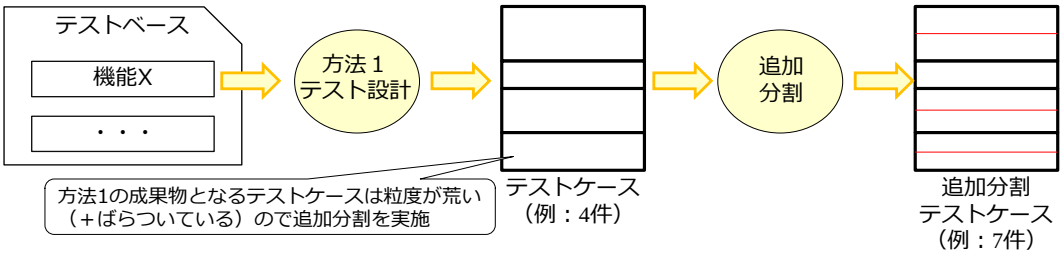

図 5-1 方法 1 で設計したテストケースと追加分割

5.2 比較結果と分析

比較結果および比較結果の分析について記載する。

(1) 比較結果

テストベースの 3 つの機能に対して 2 つの方法でテスト設計を行い、比較した結果を表 5-1 に示す。機能 X は対象テストベースの「予約」機能、機能 Y は「開局」機能、機能 Z は「プログラム更新」機能である。

表 5-1 2 つの方法によるテストケース結果

テストケース作成方法	機能Xでの比較	機能Yでの比較	機能Zでの比較
方法1：単純にテストベースと対応付けてテスト設計を行う（比較での抜けケース数）	31ケース（13ケースの抜け）	5ケース（5ケースの抜け）	3ケース（5ケースの抜け）
方法2：本論文で提案するテスト詳細設計手順でテスト設計を行う（比較での抜けケース数）	41ケース（3ケース：他の機能側でテスト設計実施）	10ケース	8ケース

方法 1 で設計したテストケースは、今回提案する方法 2 で設計したテストケースと比較して、機能 X～機能 Z すべてにおいてテストケース抜けが存在していたことが確認できる。方法 2 のテスト詳細設計手順を用いたテストケースの 30%～60%のテストケースが、要件定義書から直接作成したテストケースの抜けに相当している。

以上から、今回提案するテスト詳細設計プロセス手順による抜け防止の効果があったと考える。

(2) 機能 X における比較結果詳細

効果をより詳細に確認するため、機能 X（予約機能）における詳細の比較を行った。比較結果を図 5-2 にまとめる。

方法 1 と方法 2 で設計を行ったテストケースには、それぞれ存在しているテストケース、存在していないテストケースが含まれていた。これらについて、次の 3 項目に分けて整理を行った。

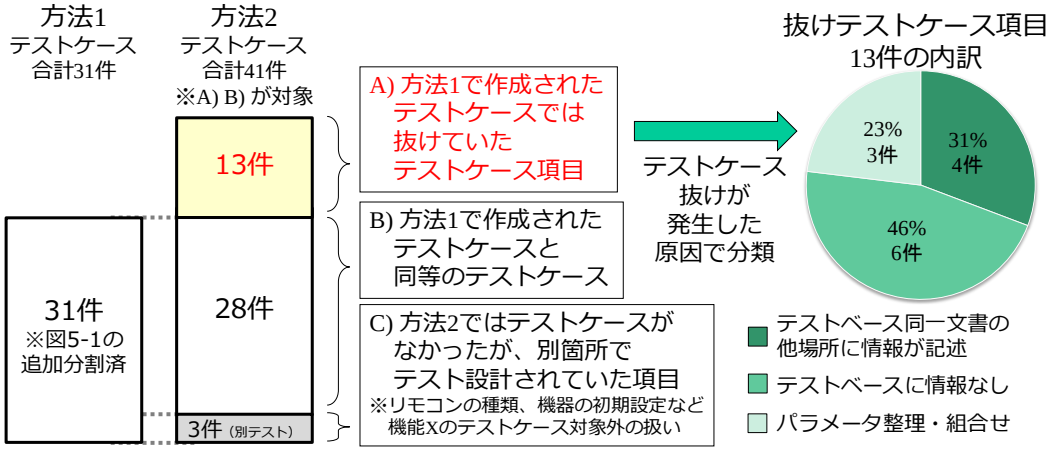

図 5-2 機能 X（予約機能）におけるテストケースの比較詳細

・A) 方法 1 にはテストケースがないが、方法 2 にはテストケースが存在する項目（13 件）

これら 13 件の項目が方法 1 のテストケースにおける「テストケース抜け」に相当する。確認された 13 件をテストケース抜けが発生した原因で分類を行った。

・テストベースと同一文書の他場所に情報が記述　…　4 件
・テストベースに情報なし　…　6 件
・パラメータ整理・組合せの漏れ　…　3 件

「テストベースと同一文書の他場所に情報が記述」項目の例として、機能 X（予約機能）で予約可否を判定するための、楽曲コンテンツの再生可能・不可能の設定が挙げられる。

「テストベースに情報なし」の項目は派生元や後工程の設計文書から情報を探すことで必要な情報を発見できる可能性はあるが、今回は要件定義書以外の情報は存在していなかったため、すべて「テストベースに情報なし」へ分類を行っている。

・B) 方法 1 で作成されたテストケースと同等のテストケース（28 件）

これらは、方法 1 で作成されたテストと同等のテストケース内容である。

・C) 方法 1 にはテストケースが存在するが、方法 2 にはテストケースがなかった項目（3 件）

これら 3 件は、別機能や他テスト設計対象枠でテスト設計されていた項目であった。リモコンの種類に対応するテスト（「機器組合せ」で別途テスト設計)、機器の初期設定に対応するテスト（「機器設定」で別途テスト)、制御タイミングによるふるまいの違いのテスト（「演奏制御」で別途テスト）が対象である。方法 2 においてこれらのテストは、対象の機能 X（予約機能）とは異なる枠でテストすべきと判断された項目である。

6.　考察

得られた結果に対する考察を記載する。

6.1　テストケース抜けの分類とテスト詳細設計手順による効果

まず、テスト設計時にテストケースを作ることができない（例：パラメータ整理や組合せの段階でパラメータの不足や組合せを作ることができない)ことで、情報不足に気付くことができる。これらの情報不足に気づき、補完を効果的に行うためには、曖昧な表現を見つけやすいサイズに分ける「対象の分割」が役に立つ。

情報不足から発生するテストケース抜けの分類とテスト設計手順による効果についてまとめる。

(1) テストベースと同一文書の他場所に情報が記述される場合　に対する効果

テストベースと同一文書の他場所に情報が記述されるような場合には、「パラメータ整理」の段階で必要な情報を確認することで、発見することができる。

(2)テストベースに情報がない場合　に対する効果

上記(1)の作業後には、テストベースに情報が記載されていないことも分かる。その場合には、必要情報が記述されている文書をドメイン知識のある担当者に確認ができる。また、情報が存在しない場合に「参照モデル作成」にて明確化することで不足情報を補完することができる。

(3)パラメータ整理・組合せによる効果

設計段階では詳細なパラメータ組合せまで明らかにする時間は不足していることが多い。そのため、テスト設計時にて不足したパラメータの明確化とその組合せ論理条件を明らかにする作業は、検討不足を発見するために役に立つ。テスト設計の恩恵がもっとも得られる部分といえる。

6.2 抜けの補完方法についての考察

本論文で提案する手順を用いた場合には、テスト詳細設計の各手順において「テストケースが設計できない」ことに気づくことができる。その段階で「6.1 テストケース抜けの分類とテスト詳細設計手順による効果」で記載した情報の探索や、ドメイン知識のある担当者に必要情報を確認するという行動ができる。この行動により、抜けが補完できるようになる。

また、各成果物に対して適切な有識者によるレビューも効果的である。例えば、4.3 節「(1) 対象の分割」の成果物はテスト設計取りまとめ担当が、「(2)参照モデル」や「(3)パラメータ整理」の成果物は各対象(機能 X など)の知識を持つ担当者がレビューすることで抜け防止につながる。

7. 設計側への対処に向けて

本論文で提示している「テストベースに発生する問題」については、本来は設計側で解決することが理想的である。しかし、問題の発生割合がつかめないまま設計、上流工程での対処を行う場合には、どこまで対処すべきか判断できず、過剰な対策を行ってしまう可能性がある。

そのため、本論文で提示しているようなテスト詳細設計の手順での対処を実施し、テストケースの抜けを防止しながら発生しているテストベースの問題傾向を掴み、効果のある部分をピンポイントに改善することが効率的かつ効果的と考えている。

設計側で行うべき対処の案としては、「フォーマットの統一 (USDM[7]の採用など)」、「散逸した情報に対する引用箇所を明示 (もしくは全体の関連図を活用する)」といった対策が考えられる。

8. まとめと今後の目標

本論文では、開発規模の増大や開発期間の短縮、必要な情報を提供する締め切りの厳格さにより発生しやすい、テストベースとなる設計文書にばらつき、情報の散逸、必要な情報が存在しない状況において、テストケース抜けを防止する「テスト詳細設計プロセスの手順」を提案した。提案した方法で作成したテストケースを従来的な方法と比較した結果、テストベースで発生しやすい問題に対応した効果が見られ、テストケース抜け防止の効果が確認できた。

今後は、本論文の手法を適用するプロジェクトを増やすことで、効果の確実性を検証する。適用するプロジェクトが増えることで、規模や複雑度に応じたテスト詳細設計手順のパターン (プロセスパターン) の構築もできると考えている。

また、「7. 設計側への対処に向けて」に記載したように、テストベースで抜けが発生する傾向を捉え、設計側で効率的かつ効果的な改善を行うための仕組みを構築することを目標とする。

9. 参考文献

[1] 清水 吉男、「派生開発」を成功させるプロセス改善の技術と極意、2007 年

[2] Y.Nishi、Viewpoint-based Test Architecture Design、Workshop on Metrics and Standards for Software Testing (MaSST) (in conjunction with SERE/SSIRI 2012)、2012 年

[3] K.Yoshioka、N.Mizuno、Y.Nishi、Stepwise Test Design Method、6th World Congress for Software Quality (6WCSQ)、2014 年

[4] N.Mizuno、M.Nakakuki、Y. Seino、Test Conglomeration - Proposal for Test Design Notation like Class Diagram、4th International Workshop on Software Test Architecture (InSTA2017)、2017 年

[5] 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、2010 年

[6] NPO 法人 ソフトウェアテスト技術振興協会 (ASTER)、ASTER カラオケシステム要件定義書_Ver2、2017 年

[7] 清水 吉男、[改訂第 2 版] [入門+実践]要求を仕様化する技術・表現する技術、2010 年