G-20

# 不具合報告事例分析に基づくテスト設計によるプロセス改善に関する考察

A Case Study of Bug Report Analysis for Test Design based Process Improvement

辻原 拓弥 †　福安 直樹 ‡　満田 成紀 ‡　松延 拓生 ‡　鯵坂 恒夫 ‡
Takumi Tsujihara　Naoki Fukuyasu　Naruki Mitsuda　Takuo Matunobe　Tsuneo Ajisaka

## 1. はじめに

ソフトウェアテストでは開発中のテスト実施工程に向けて，開発初期段階においてテスト設計を行うことがよいと言われている．テスト設計を開発初期段階で行うことによって，効果的なテストの実施によるデバッグ作業の負担軽減や，早期に仕様書や設計書に含まれる不具合の原因や矛盾に気付ける事で設計品質の向上につながるとされる．しかしながら，一般にはシステムそのものの設計に重点が置かれ，テスト設計は後回しにされることが多い．テスト設計を行う場合，過去の類似例を参考できるもののソフトウェアは一点物のため最終的にはケースバイケースでの設計が必要だが，開発初期段階においてはシステム設計と並行してテスト作業を行うためテスト作業に必要なドキュメントは限られていて，かえってテスト設計の実施によるコストや工数の負担が増す可能性があるからである．そのため，低コストでの開発初期段階におけるテスト設計の実施が重要である．

本稿では，開発初期段階においてテスト設計を行った場合の影響や効果を把握することを目的とし，テスト設計を十分行わずに開発を行ったチーム開発事例を対象に，対象事例の不具合の調査および分析を行った．そして，分析結果をもとに対象事例においてテスト設計を開発初期に行っていた場合，テスト設計は不具合の発生にどのように影響したかについて考察を行った．

## 2. テスト設計によるプロセス改善の取り組み

テスト設計とは，ソフトウェアテストの実施に向けて上流工程でのシステム設計と並行してどのようなテストを行うかを考える工程である．従来の V モデルでは開発プロセスの最終工程としてテスト工程が行われるが，後に発表された W モデルにおいて，品質向上やテスト実施工程の負担削減のためテスト対象物が無くても実施可能なテスト工程の一部の作業を上流工程と並行して行うことが提唱されたことでテスト設計という考え方が誕生した．テスト設計で実施可能な作業には，実施するテストケースの洗い出しや各テストケースにおけるテストデータの作成などがある．

† 和歌山大学大学院システム工学研究科, Graduate School of Systems Engineering,Wakayama University
‡ 和歌山大学システム工学部, Faculty of Systems Engineering,Wakayama University

W モデルが提唱されて以降，テスト設計は高品質なソフトウェアを開発する上で非常に効果的とされてきた．一方で，テスト設計は従来の V モデルには存在しなかった工程であり成果物となるテストに関するドキュメントも製品ではないため，テスト設計の行い方次第では開発コストや工数の増加によるデメリットとなりうる．また，技術的な面においてもテスト設計の導入は容易でなく，導入したとはいえテスト設計が必ずしも効果を発揮する訳ではない．こうした背景からテスト設計に関する研究は盛んであり，テスト設計によるプロセス改善に関する研究も行われている．

柚口ら [1] は，システム開発での変更作業における整合性確保を目的とした変更管理の際の変更または障害の判定基準を明確化するための解決策として，従来の V モデルや W モデルを元に新たな開発プロセスとして 3V モデルを提案している．3V モデルでは，上流工程の各工程においてテスト仕様書を作成しテスト設計書を用いてシステム設計書のレビューを行うこととしており，通常のレビューよりテスト設計書を用いる分，詳細にレビューができることから，変更管理基準に役立つ他，設計書の不具合漏れや矛盾を解消し品質向上につながると述べている．町田ら [2] は，W モデルにおけるテスト設計の適用を開発プロセスの一部に限定し，デシジョンテーブルによる設計書のレビューを兼ねたテスト設計方法の検証を行った．この研究では，以前の開発における W モデルの適用効果がコストの面で芳しくなかったことを踏まえ，限定的にテスト設計を適用することでコストを抑えた実施が可能となり，不具合情報から開発の品質について定量的に評価している．開発事例に対して限定的ではあるが一定の効果が得られたと述べている．また，テスト設計の適用による不具合修正の有無によるその後のプロセスへの影響の理解の必要性やドキュメントのレビューにおけるテスト技術による補完の可能性などについても述べている．武澤ら [3] は，ソフトウェアの評価において，通常はテスト段階における不具合検査によって品質評価が行われるが，重要なのは設計段階における品質評価の適切さだと考え，設計段階における不具合検査のための直交表の導入を提案している．ソフトウェア開発では，ある程度機能ごとにモジュール分割を行い役割を分担して開発が行われる．テスト工程で用いられる通常の直交

表よりも小型の直交表を設計段階の各過程に用いて機能ごとに不具合検査を行うことで，設計段階でより細かい不具合の検出が可能となり，不具合を早期に発見できることを述べている．実際に，武澤らが行った開発工程の各段階で直交表を用いた開発事例においては，直交表による不具合検出の結果，不具合による損失を防げたとして一定の効果があったことを報告している．

## 3. 開発事例

本研究では，システム開発においてテスト設計を初期段階で行うことによる開発プロセスへの影響がどのようなものか把握し明確にするため，テスト設計を十分行わなかった開発事例を取り上げ，その事例の不具合に関して分析を行った．本章では，分析対象とした開発事例の概要について述べる．

### 3.1 分析事例

本研究では分析対象としてチーム開発による Web アプリケーション開発の事例を用いた．開発事例の概要は以下の通りである．

- 開発形態：チーム開発（Web アプリケーション開発）
- 開発期間：5 週間 (2017/10/13 〜 2017/11/17)
- 開発者数：6 名

開発スケジュールを表 1 に示す．本事例は，大学院生向けの教育プロジェクトの一環として実施された PBL における開発事例である．アプリケーションの企画段階から実装し運用に至るまでの開発作業がチームで行われた．開発中は，Eclipse*を用いて作業が行われ，GitHub†を用いて成果物であるプログラムが管理された．開発は GitHub Flow ‡にしたがって進められた．開発するアプリケーションの各機能の実装や環境構築などの作業を分担して開発者各自で開発が進められ，随時 GitHub にてチーム内でレビューが行われ最終的に一つにまとめあげられた．また，開発中は LINE§や Skype¶を用いてコミュニケーションがとられた．

### 3.2 アプリケーション概要

本事例で開発された Web アプリケーションの概要を述べる．概要は以下の通りである．

- 開発規模：ファイル総数 23，コード総行数 8169
- 開発言語：Java，HTML/CSS，JavaScript 他

本アプリケーションの機能は，GoogleMap‖API，Flickr**API，Twitter††API を用いて既存の Web サービスを組み合わせることにより開発されている．アプリケーションが提供するサービスの一連の流れは次の通りである．まず始めに，GoogleMap から任意の位置情報を取得し，取得した位置情報を用いて Flickr より風景画像などを取得する．次に，取得した画像と自身のフォルダなどにある人物画像を用いてクロマキー合成より合成を行う．最後に，その合成画像を機械学習よる自動生成，または自由記述によるツイート文を添えて Twitter にて投稿する．

## 4. 不具合報告分析

本研究では不具合分析を行うにあたり，まずは 3 章で述べた開発事例の開発中のログと開発後の報告を対象に不具合の収集を目的とした調査を行った．次に，調査により収集した不具合の詳細内容から原因別と発生工程別の 2 通りによる分類を試みた．そして，分類結果を用いて不具合分析を行った．

### 4.1 調査

調査対象は，以下の 2 つとする．

1. 開発後に開発者から受け取った不具合に関する報告
2. 開発中のログ

それぞれの調査対象から開発中及び開発後に発生した不具合についての検出を行った．本研究で調査した事例で用いられた開発ログは以下の 3 つである．

- GitHub リポジトリ
- LINE トーク履歴
- ミーティング議事録

開発ログにおいて調査した主な項目は，開発中の不具合発生報告日，発生した不具合の内容および原因である．調査の結果，検出できた不具合の一部を例として表 2 に示す．調査より各々の開発者によって報告された不具合の定義や粒度にばらつきが見られた．そこで，本調査では，曖昧な定義による不具合のカウント漏れを防ぐため開発中に起こった全ての不具合および不具合が原因で起こったと見られる事象を不具合としてカウントしている．そのため，開発作業において，その事象が不具合とまでは呼べないものであったとしても，開発を阻害する要因であり不具合であったと判断した開発者の意見を重視し

*https://www.eclipse.org
†https://github.co.jp
‡GitHub 社がシンプルで効率的として実践している開発フロー．近年では，チーム開発におけるスタンダードとなっている．
§https://line.me/ja
¶https://www.eclipse.org
‖https://www.google.com/maps
**https://www.flickr.com
††https://help.twitter.com

表 1: 事例の開発スケジュール

| 日付 | 工程 | 主な作業内容 |
|---|---|---|
| 10/13 | 要求定義 | 企画書の作成，開発環境構築，役割担当決め |
| 10/14〜 | 設計，要求定義 | API の仕様確認，機能設計，画面設計 |
| 10/28〜 | 実装 | 機能実装，サーバ準備 |
| 11/10〜 | 実装，テスト | 機能結合，動作確認 |
| 11/17 | プロダクト提出 | AWS 上にデプロイ |

表 2: 検出された不具合の例

| 発生日 | 内容 | 原因 |
|---|---|---|
| 10/27 | デプロイできない | gradle ファイルの記述ミスによる環境設定の不備 |
| 10/31 | 画像合成機能の挙動が正確でない | 仕様変更前の不要なコードの削除忘れによる挙動の阻害 |
| 11/14 | Flickr から画像を取得できない | クエリパラメータが未定義 |
| 11/16 | クロスドメインの問題 | CORS 制約に従った設計が成されていなかった |
| 11/17 | 結合した機能の挙動が正確でない | 結合するモジュール間のインタフェース仕様が異なっている |

不具合としてカウントした[‡‡]．不具合は，主にシステムに関する不具合，開発環境に関する不具合，その他開発を阻害した不具合などである．調査の結果，検出された不具合の件数を表 3 に示す．表 3 で示す通りいくつかの不具合は重複しており，重複を除いた不具合件数は合計 37 件であった．また，ミーティング議事録からは進捗報告及び計画に関する記録が多く不具合に関する記録は検出されなかった．

### 4.2 分類

調査結果より得た不具合の情報から不具合の傾向や特徴を探るため分類を行った．分類は文献 [4] と文献 [5] を参考に 2 通りの方法で試みた．分類方法は以下の 2 通りである．

- 不具合の原因別分類 [4]
- 不具合の発生工程別分類 [5]

不具合の原因別分類では，検出した不具合の発生原因をもとに開発の観点で構成される大分類ならびに大分類における具体的な不具合原因で項目が構成された小分類に分類を行い，それぞれの分類項目ごとに件数を調べた．不具合の原因別分類を行う目的は，不具合の原因の偏りや傾向を調べることでより明確に課題を把握し，分析や対策を的確に行うことである [4]．不具合の発生工程別分類では，不具合の種類をもとに不具合が発生した（作り込まれた）工程ごとに分類を行い，工程ごとの件数を調べた．不具合の発生工程別分類を行う目的は，開発プロセスにおけるどの時点で不具合が発生したかを明確にすることで不具合発生の防止のための改善ポイントの検討に役立てることである [5]．これらの分類結果を用いて分析することで，本事例では開発プロセスのどこでどのような不具合がなぜ発生したかの把握を試みる．

### 4.3 分析

分類結果を表 4 と表 5 に示す．表 4 は不具合の原因別分類の結果である．原因別分類では，不具合 37 件の原因分析から 9 つの大分類項目と 11 の小分類項目に分類を行った．その結果，2. 機能に関する不具合が不具合報告全体のおよそ 32 %（12 件）と最も比重が大きかった．その他は，3. 制御・論理に関する不具合が全体のおよそ 16 %（6 件），1. 仕様・要件に関する不具合がおよそ 14 %（5 件），5. インプリメント（実装）に関する不具合が全体のおよそ 11 %（4 件），4. データに関する不具合がおよそ 8 %（3 件）であった．最も比重の大きかった 2. 機能の小分類においては，設計の漏れ・重複と処理・組み合わせの不良の 2 項目に分類された．処理・組み合わせの不良の項目では，その全てが機能モジュールやコンポーネント，ファイルの組み合わせに関する不具合であった．設計の漏れ・重複の項目においても，5 件中 4 件が外部ファイルを呼び出す API やモジュールの結合に関する設計漏れであり，機能におけるインタフェースに関する不具合が集中して発生していたことが分かった．次に比重の大きかった 3. 制御・論理では，アルゴリ

[‡‡]ただし，開発者の意見でも明らかに不具合と呼べないものは除いた．

表 3: 検出された不具合の件数

| 不具合件数 | 37 件 (重複除く) |
|---|---|
| GitHub リポジトリ | 14 件（開発中に実行された commit 総数　123） |
| LINE トーク履歴 | 14 件（開発に関する総メッセージ総数　693） |
| メンバーによる報告 | 18 件（報告総件数　19） |

表 4: 不具合の原因別分類

| 大分類 | 小分類 | 小計（31） | 合計（37） |
|---|---|---|---|
| 1. 仕様・要件 | 曖昧な仕様・確認漏れ | 2 | 5 |
| | 仕様変更 | 3 | |
| 2. 機能 | 設計の漏れ・重複 | 5 | 12 |
| | 処理・組み合わせの不良 | 7 | |
| 3. 制御・論理 | アルゴリズムの不良 | 5 | 6 |
| | シーケンスの不良 | 1 | |
| 4. データ | 定義・構造の不良 | 1 | 3 |
| | 外部インタフェースの不良 | 2 | |
| 5. インプリメント (実装) | コーディングの誤り | 2 | 4 |
| | 規約・標準の違反 | 2 | |
| 6. テスト | 確認不足 | 1 | 1 |
| 7. 周辺ソフトウェア（小分類なし） | | | 1 |
| 8. 環境設定（小分類なし） | | | 4 |
| 9. ハードウェア（小分類なし） | | | 1 |

ズムやシーケンスの不良に関する不具合として，処理フローの誤りやこれらの不具合に関して，4. データに関する不具合においても，クエリ発行時のパラメータの定義誤りによる不具合や，外部インタフェースの不具合による外部ライブラリの導入失敗および外部ファイルの環境変数の未反映が見られた．大分類 3，4 から，分類は異なるものの，どちらもシステム外部と関連した挙動における不具合が発生する傾向があると考えられる．また，1. 仕様・要件では，曖昧な仕様・確認漏れや仕様変更に関する不具合が見られた．5. インプリメント（実装）においては，規約・標準の違反といった不具合が見られた．これは，GitHub Flow に反してマスターブランチに不具合の混在したファイルがマージされ，そのまま開発が進められたことで発生した不具合であった．このマージの際に混在した不具合というのが，仕様変更時に生じた不具合である．このことから，機能拡張および機能結合において，変更を加えたことで生じたファイル内における不整合によって生じた不具合を見逃したままマージしてしまったことが問題であったと言える．

分析を行うにあたり原因別分類では，大分類 6.〜9. に該当する不具合を分析対象外とした．6. テストにおける確認不足による不具合はブラウザ対応に関するもので，事例では時間の都合上，意図してブラウザ対応の設計を行っていなかったため分析対象外としている．その他の 7. 周辺ソフトウェア，8. 環境設定，9. ハードウェアに関する不具合についても，開発システムと関連のない不具合と判断したため本研究では分析対象外としている．

表 5 は不具合の発生工程別分類の結果である．発生工程別分類では，要求定義，設計，実装の 3 つの開発工程と特定の開発工程に依存せず開発プロセス全体に共通して起こりうる不具合を分類した工程共通の 4 項目で分類を行った．分類の結果，設計・実装工程に不具合が集中して発生していたこと（27 件中 26 件）が分かった．今回の開発事例では開発初期の要求定義の段階では大まかに仕様を決定するに留め，開発を進めていく中で，詳細を決めていくこととされており要求定義としての期間が短期間であったため，要求定義の工程で発生した不具合が少なかったこと（1 件）が考えられる．そのため，要求定義で仕様の詳細部分が決定されなかったことで仕様や設計の変更が多々生まれたたことから，後の設計や実

表 5: 不具合の発生工程別分類

| 発生工程 | 不具合種類 | 小計（37） | 合計（37） |
|---|---|---|---|
| 要求定義 | 記述誤り | 1 | 1 |
| 設計 | データの誤り | 1 | 12 |
| | アルゴリズム/制御の誤り | 2 | |
| | インタフェースの誤り | 4 | |
| | 記述誤り | 4 | |
| | 機能の欠如 | 1 | |
| 実装 | データの誤り | 2 | 14 |
| | インタフェースの誤り | 2 | |
| | エラーチェックの誤り | 4 | |
| | 機能の実装誤り | 6 | |
| 工程共通 | OS/パッケージソフトウェアの誤り | 4 | 10 |
| | 結合の誤り | 2 | |
| | その他 | 4 | |

装の工程にしわ寄せされた結果，不具合が集中して発生したのではないかと推測できる．不具合の種類に関しては，設計，実装の工程を通してデータ，インタフェース，機能に関する不具合が見られた．これも先に述べた通り，要求定義で仕様を細かく決めるのではなく，開発を進めながら決められたことが要因と思われる．そのため，設計工程以降の仕様変更および仕様の詳細決定などが原因で，実装工程において機能拡張や修正後に発生した不具合原因を見逃すことで生じるエラーチェックの誤りに分類された不具合（4 件）も見られた．

分析を行うにあたって発生工程別分類では．工程共通における OS/パッケージソフトウェアの誤りとその他に分類される不具合の 8 件中 7 件は外部のソフトウェアや本事例の開発ソフトウェアに関係ないところに原因があるもので，開発工程にも依存しないため本事例でのデバッグ対象外であると判断し分析対象外としている．これら対象外とした 7 件は原因別分類においても対象外としたものに一致している．

## 5. 考察

原因別分類による分析結果から，本事例では，機能や制御・論理，データなどの複数の観点にまたがって，インタフェースに関する不具合が多く生じていたことが分かる．例えば，表 2 における 11/17 に発生した不具合では，モジュール間のインタフェースに関する設計が曖昧であり，システムの階層構造に関する設計の見直しが必要であった．調査の結果，設計工程では各モジュールにおける入出力や機能の仕様は決められていたものの，インタフェースの仕様は議論されておらず設計が漏れていた．テスト設計において，各モジュールにおけるスタブやドライバの仕様を議論することで，モジュール間のインタフェースをすり合わせることが可能になり，結合段階での不具合の発生を防ぐことができたのではないかと考えられる．

発生工程別分類による分析結果からも，開発初期において仕様が曖昧なまま開発を進めたことで設計や実装の工程に不具合が集中していたことが分かる．しかしながら，表 4 の原因別分類では，インプリメント（実装）が原因とされる不具合は 4 件のみであるため，多くの不具合がより早期に発見できた可能性がある．例えば，表 2 における 10/31 に発生した不具合では，仕様変更後のファイル内に変更前の不要なコードが残っていたため，本来の挙動に影響を与えていた．仕様変更におけるプログラムの変更範囲が正確に把握できておらず，不要なコード自体も問題なかったため気付くまでに時間と労力を要した．このような仕様変更における不具合の対策としては，テスト実施工程におけるリグレッションテストの実施が挙げられる．しかし，リグレッションテストの場合，プログラムの修正や変更ごとに全てのコードを確認するにはコストが大きくなる恐れがあるためテスト対象の範囲を限定する必要がある．テスト対象の範囲を効率よく限定するには，システムを構造化し容易に変更管理をできるように設計することが重要で，変更管理にはテスト設計書を用いて設計をすることが有効と考えられる [1] ため，テスト容易性を考慮した設計となるようにシステム設計とテスト設計を関連付けて行うことが必要である．例えば，本事例においても，設計工程と並行してテスト設計書を作成しそれを用いて設計に対するレビューを行

うことで，変更管理やリスク分析が可能となり，その結果から判明した各々の変更箇所とその変更箇所との関連度，依存度の大きい部分を中心にリグレッションテストを実施することで不具合に気付くことができ，マスターブランチへの不具合の混入を防げたのではないかと推測される．

以上のように，開発初期段階でテスト設計を行うことで上流工程をレビューやテスト技術によって検査することが可能となり，システム設計およびテスト実施の両工程で相互に補完的な役割を果たす可能性が期待できる．

## 6. おわりに

本稿では，テスト設計を十分行わずに開発を行った事例を対象に開発中に発生した不具合に関する報告の分析を行い，分析結果をもとに対象事例においてテスト設計を開発初期に実施していた場合，テスト設計は不具合の発生にどのように影響したかについて考察した．今回の分析事例においては，開発初期段階でテスト設計を行っていたならば，インタフェースに関する設計漏れへの気付きや仕様変更における変更管理による効果的なテストの実施によって不具合の発生を防止できたことが推測された．その結果，開発初期段階においてテスト設計を行うことで上流工程をテスト仕様書を用いたレビューやテスト技術によって検査することが可能となり，システム設計とテスト実施の両工程で相互に補完的な役割を果たす可能性を述べた．

一方で，テスト設計の実施の際に考慮すべきであるコストに関しては，不具合分析の結果からは議論できておらず，仮に開発初期段階でテスト設計を行っていたとしてもコスト面の問題で考察したような不具合発生の防止には至らなかったかもしれないため議論の余地がある．

今後は，本分析より得た考察を活用してテスト設計によるプロセス改善への実践方法について検討する他，コスト面の問題に関しても検討していく予定である．

## 参考文献


[1] 柚口智史，飯島隆，3V モデルを用いた変更管理基準の明確化，プロジェクトマネジメント学会 2010 年度春季研究発表大会予稿集, pp.339-343, 2010.

[2] 町田欣史，清水誠介，朱峰錦司，伊藤司，テスト技術を用いた第三者による設計書品質の検証と定量化に関する取り組み, プロジェクトマネジメント学会 2013 年度春季研究発表大会予稿集, pp.328-332, 2013.

[3] 武澤泰則，, 天谷浩一，矢野宏，ソフトウェア設計中の直交表導入による開発効率の効果と課題，品質工学会学会誌品質工学，Vol.24，No.2，pp.46-54，2016.

[4] ソフトウェア・エンジニアリング・センター（SEC），独立行政法人　情報処理推進機構（IPA），続　定量的品質予測のススメ〜IT システム開発における定量的品質管理の導入ノウハウ〜と上流工程へのアプローチ，2011，佐伯印刷株式会社，http://www.ipa.go.jp/files/000005143.pdf.

[5] ソフトウェア・エンジニアリング・センター（SEC），独立行政法人　情報処理推進機構（IPA），組込みソフトウェア開発における品質向上の勧め〔バグ管理手法編〕，2013，佐伯印刷株式会社，http://www.ipa.go.jp/files/000027629.pdf.