B-027

受け入れテスト駆動開発による実行可能な仕様に関する検証結果
Evaluating the efficacy of A-TDD for achieving executable specifications

張　嵐† 木村　めぐみ† 藤井　拓†
Lan ZHANG Megumi KIMURA Taku FUJII

1．まえがき

受け入れテスト駆動開発（Acceptance Test Driven Development、以下 A-TDD）は、開発者がソースコードの作成より先に単体レベルでテストケースを作成するというテスト駆動開発（TDD）[1]の手法と類似し、要求定義レベルに TDD を応用したものである。理想的な A-TDD では、要求検討時に、あらかじめ受け入れ側の観点で個々の要求を自然言語で受け入れテストケースとして記述し、自動化する。こうすることによって、受け入れ側のフィードバックが早くなり、要求の不備を早期に発見することができる。また、自動化された受け入れテストとしての要求をメンテナンスしていくため「実行可能な仕様」[2]を実現することを狙う。

当社では、「変化に柔軟に対応するため、短い反復で潜在的に納品可能なシステムを開発する」というアジャイル開発を推進してきた。アジャイル開発の効果を十分に享受するためには、コード品質を保証するための TDD や継続的インテグレーション（CI）などの技術プラクティスが欠かせないが、A-TDD も適切なシステムを開発するために非常に重要なプラクティスとして捉えている。

筆者らは 2012 年から、先行研究、及び 4 つのプロジェクトでの適用を通じ、A-TDD の「実行可能な仕様」とする実用性と効果的適用方法に関する研究を行った。本論では、研究の概要とその結果を説明する。

2．背景と先行研究

近年、アジャイル開発の広がりとともに A-TDD の重要性が認識され、関心が高まっている。欧米のアジャイル開発プロジェクトでは、A-TDD の適用率は 28%に達するが[3]、日本の場合、適用の事例はまだ少数である[4]。その原因としては以下が考えられる。

1) 日本の多くのシステム開発は、開発プロセスと保守プロセスが分断されているため、特に開発側の立場では、開発プロセスの中で長期にわたって繰り返し実施する受け入れテストはコストの視点及び納期の視点から見合わない投資である。
2) 受け入れテスト自動化をサポートするツールに日本語対応のものが少なく、要求のテストケースとしての記述は可読性が低い。

上記の 2 点に着目し、当社の先行研究では、A-TDD のサポートツールの選定と開発プロセスの検討を行い、検証プロジェクトを通じ、A-TDD の適用コストを測定した。

2.1　A-TDD サポートツールの選定

受け入れテスト自動化コード作成の容易さ及び受け入れテストの記述が自然言語との近さに特徴があるオープンソースツール「Cucumber」[5]を選定した。ツールの選定は以下の 3 ステップで行った。

ステップ	選定方法
初期選定 (n = 19)	インターネット上の調査を通じて、ツールの機能、関心度、ユーザコミュニティの活動状況によるふるい分けを行う。
1 次評価 (n = 5)	ツールのマニュアルを利用し、機能特徴、テストケースの記述、日本語対応、実行方法、CI への対応などの観点で評価を実施する。
2 次評価 (n = 3)	1 次評価の結果において上位 3 位のツールをそれぞれインストールし、試用し、評価項目ごとにスコアリングを行う。

2.2　先行研究で顕在化した課題

先行研究では「Cucumber」を駆使し、検証用の Web アプリケーションを A-TDD プロセスで開発し、開発プロセスの確認と A-TDD の実施可能性を考察した。

検証のための開発を通じ、A-TDD のメリットが確認できたものの、要求を自動化できる程度までに具体化したテストケースとして抽出するという技術的なハードルの高さがあると判明した。そのハードルの高さに加えて、A-TDD の導入に伴う 25%の開発工数の増加も、コスト対効果に対する懸念を高めている。逆に言えば、A-TDD が実プロジェクトに「実行可能な仕様」として実用できるように、以下 2 つの課題を解決しなければならない。

1) 要求からどのように効果的に自動化テストケースを抽出するか。
2) A-TDD に伴うコスト増をどのように抑制するか。

3. 研究アプローチ

継続研究では、2 段階に分けて、4 つのプロジェクトで A-TDD を適用し、先行研究で顕在化した課題の対処を目指した。第 1 段階では、2 つの小規模な Web 開発実プロジェクトを通じ、要求から受け入れテストケースを抽出する方法や、自動化の範囲に関する異なる仮説を立て、各プロジェクトで検証した。また、要求としてのテストケースの可読性向上、テストケースの作成及び自動化のコスト低減に貢献するプラクティスについて考察した。第 2 段階では、協力の得られた 2 件の中規模プロジェクトを題材に、今まで得た経験の有効性を検証することとともに、プロジェクトからのフィードバックに基づいて A-TDD の適用効果を高める方法と今後の改善点を考察した。

3.1　工夫した内容

継続研究で工夫した内容の 3 点を以下で紹介する。1 と 2 は A-TDD 適用を阻む課題への対応策の一部である。

1. A-TDD に関する共通認識を持つ

† （株）オージス総研　技術部アジャイル開発センター

A-TDD の作業プロセス、作業方法、作業手順及び自動化に関する考え方がぶれない様に、まず、A-TDD を以下のように定義し、周知した。

定義	目的	手段
受け入れ側、開発側（分析者、開発者、テスト担当者）は共同で、	開発内容に関する共通認識を持つ	役割間の協力
受け入れテストを先に書き、	要求仕様をテストケースとして、文書化し、維持する	自然言語
一部を自動的に実行できるように自動化する	回帰テストによる継続な検証を可能とする	自動化ツール

また、以下 3.2 で示す A-TDD の開発プロセスを明確にし、動くサンプルとともに、プロジェクト関係者にトレーニングの形式で提供した。

2. 使い慣れたツールで A-TDD を行う

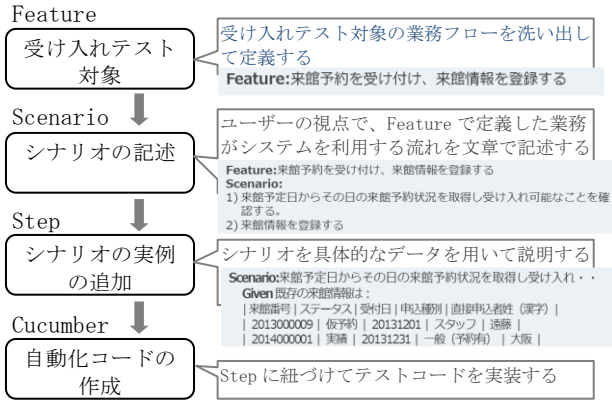
テストケースの一部は受け入れ側も記述できるように、また、ツールの学習コストを低減するため、Excel ベースのテストケースを記述するためのテンプレートを開発した。これを用いることで、テストケースを効率的に作成でき、可読性向上とともに人的バラツキも低減できた。

3. 定量データの収集と定性的振り返りを併用する

コスト対効果を考察するため、各プロジェクトに対し、COSMIC 法[6]に基づく機能規模測定を行い、プロジェクト終了後には、関係者による振り返りを行った。

3.2 適用した開発プロセス

以下は本研究で利用した A-TDD のプロセスである。



1) Feature、Scenario、Step の作成は、当社が開発したテンプレートに自然言語で記述することで行う。記述内容は要求から抽出した受け入れテストケース、シナリオ及びシナリオの実例としてのデータである。これらは開発側、受け入れ側のいずれが記述する。
2) 開発側は記述済みのテンプレートから、Cucumber が識別できる実行用の .feature ファイルを生成する。Cucumber で生成した自動化のための雛形を利用し、自動化コードを作成する。
3) 受け入れテストを実行する。

4. 研究結果

4 つのプロジェクトの定量データと定性的振り返りから以下の結果が観察できた。

1) 粒度が小さいテストケース（ビジネスロジック単位）を網羅なく作成して自動化することで、不具合の検出に効果があった。ただし、この場合、A-TDD に使った工数が全開発工数の 45%に達し、プロジェクトのコストに大きな影響を与える。
2) 重要な業務フロー（一連のビジネスロジックからなる）を識別し、それについて集中的に受け入れテストを作成しながら自動化を行うことで、A-TDD による増加した工数は全開発工数の 8%前後に抑えることができた。また、上記 1)と比べ、受け入れテストのメンテナンスコストも低く抑えられそうと期待できる。

当社はプロジェクトの実施過程及び上記の実施結果に対する考察と分析を通じ、A-TDD の効果とコストのバランスを取れた「業務フローレベルで受け入れテストケースを抽出し、自動化していく」ことが今後の A-TDD の適用の方向性だと考えるに至った。また、要求からテストを抽出する難しさとコストを低減するために、以下のプラクティスは効果があると考えられる。

- 開発側と受け入れ側のコラボレーションを A-TDD の前提条件とする。このコラボレーションを促進するため、A-TDD 担当者を明確にする。
- 受け入れテストの作成は開発より先か、開発と同時進行にすべきである。
- 受け入れ側が受け入れテストの読み書きをできるようにするため、使い慣れたツールを活用する。
- 受け入れテストの自動化はコストがかかるため、自動化対象のコスト対効果を事前に検証すべきである。コスト対効果が十分でない部分には手動テストの併用を考える。
- 受け入れテストのメンテナンス性を向上するため、テストデータと自動化コードを分離する。

5. 今後の取り組み

本研究内容と結果から、A-TDD のための開発プロセス、受け入れテストの抽出と記述方法及び実行環境が揃い、コストが明確になった。これにより、A-TDD は実際のプロジェクトで十分に適用可能な水準に達したと判断した。自然言語で受け入れテストケースを記述し、ツールの活用によってテストの実施を自動化することによって、「実行可能な仕様」の実現へと近づくことができた。

今後、A-TDD のコスト対効果を十分なものにするため、まず、どのタイプの開発に A-TDD を導入すべきかを見極める。そして、A-TDD のトレーニングとともに、要求を発見し、受け入れテストとして記述するテクニック[7]を把握するためのワークショップやトレーニングの開催によって A-TDD の担当者を育成していく。

参考文献

[1] Kent Back, “Extreme Programming Explained: embrace change”, Addison-Wesley, 2000
[2] Gojko Adzic, “Specification by example”, Manning, 2011
[3] VERSIONONE, “ 8th Annual State of Agile Survey”, 2013
[4] IPA, アジャイル型開発における プラクティス活用事例調査 ガイド編, 2013 年 3 月
[5] Cucumber, “https://cucumber.io/”
[6] COSMIC 法, “ http://www.cosmicon.com/”
[7] Ellen Gottesdiener, Mary Gorman, “Discover to Deliver: Agile Product Planning & Analysis”, EBG Consulting, Inc., 2012 (邦訳：発見から納品へ　アジャイルなプロダクトの計画策定と分析)