B-008

未指定機能へのテストケース自動生成によるソフトウェア開発の強化
Enhancement of software development by automatically generated test cases for unspecified fucntions

キム ホンジュン† 島川 博光†
Kim Hongjun Hiromitsu Shimakawa

1. はじめに

ソフトウェア開発工程で注意するべきことの一つは、顧客の要望とソフトウェア結果物の間の乖離を発生させないことである。結果物が、顧客が想定した機能を実行させない、異なった動作をする場合は、新しく機能を追加させるための後処理をすることや、最初からソフトウェアを新しく作らないといけない場合が生じる。その場合には開発期間が長くなり、コストも追加的に必要となるため、開発者に対しての顧客の信頼度と満足度が下がる恐れがある。

本論文では顧客の要望と結果物の間の乖離の問題に対して、顧客の要求を具体化することができなく、その曖昧さに対して開発者個人の想定が入ってしまうことが乖離の主な原因として注目し、そのような開発者の想定を最小限させるため、顧客と作成した開発仕様書を元にテストケースを作成、書かれたテストケースを参考にして顧客と再議論・修正を実行させ、仕様書の水準を向上させる手法を採択し、その一連の過程の名称を「テストケース駆動再設計」とする。

「テストケース駆動再設計」を効率的に行うためには、カバレッジが高いテストケースが必要だと判断したため、完成度が高い設計仕様書とカバレッジが高いテストケースを学習させた生成 AI を用いて顧客と作成した設計仕様書から人間が想定できなかった多数の「未想定テストケース」を生成、開発者が手動で作成したテストケースと紐づけることで簡単にテストケースのカバレッジを高める方法を探すことを目標とする。

2. 前提事項

2.1 テストケース駆動再設計

Bashar Nuseibeh、Steve Easterbrook (2000)はソフトウェア開発工程で成功の主要な判断基準は顧客の要望を把握し、コミュニケーションをとることで進化された要求事項を書くことだと主張する。

本論文ではこの主張を参考にし、顧客の要望と結果物の間の乖離が発生することを防ぐために採択した手法が「テストケース駆動再設計」であり、顧客と作成した仕様設計書を元に、各機能に対して開発者がソフトウェアの動作で起こりえる様々なテストケースを作成、作成されたテストケースを持って顧客と予めソフトウェアに対して議論をして、顧客が想定できなかった機能を追加するなどの修正事項を加え顧客の要望を具体化させたより進化した設計仕様書を書くことを手法である。

「テストケース駆動再設計」は各開発段階でテストの作業をすることで V モデルの形ど同一だが、各段階で顧客と再議論することでソフトウェアへの顧客の要望を具体化させる点ではプロトタイプ型のソフトウェア開発工程の長所を持っているといえる。

2.2 未想定テストケース

「未想定テストケース」とは顧客と開発者が仕様書を書く際に考慮できなかったことに対して作ったテストケースのことで、小川貴史(2023)の「係り受け解析を用いた設計仕様書とテストケースの紐づけ手法」の研究結果によると「テストケース駆動再設計」を利用する場合に、開発者の実力により、テストケースのカバレッジには格差があり、ベテラン開発者の場合は、カバレッジが高いテストケースが作成でき、顧客と議論・修正を通じて顧客の要望を具体化し結果物との乖離を除去することができると思われるが、初心者の開発者が作成したテストケースには漏れが多く、カバレッジが低いため、「テストケース駆動再設計」を実行した後でも、結果物と顧客の要望の間の乖離が発生する可能背が高いことが予想された。

現在の業界では、初心者の開発者が作成したテストケースをベテランの開発者が検討し、漏れを発見・修正する形で問題を解決しているが、本論文では、顧客と開発者が作成した設計仕様書を元に生成 AI が多数のテストケースを作成そこで開発者が「未想定テストケース」を発見し手動で作成したテストケースと紐づけることで、ベテランの役割を生成 AI が代わりに果たすことができ、初心者の開発者も自ら簡単にテストケースのカバレッジを向上させる可能にさせるようにする。

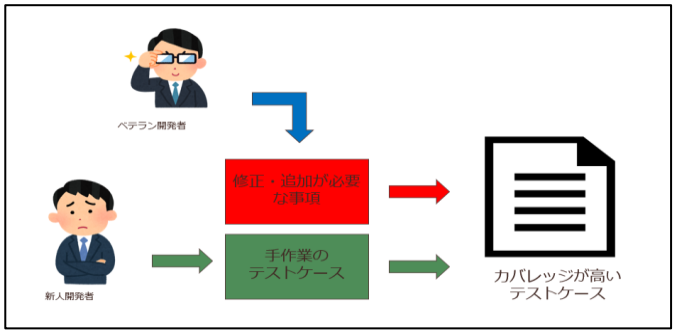

図 1：業界でのテストケース作成の流れ

3. 未想定テストケースの紐づけによるテストケースのカバレッジ向上手法。

3.1 手法概要

本論文では、すでに作られた完成度が高い設計仕様書と、カバレッジが高い検証されたテストケースを学習させて、

新しいソフトウェアの設計仕様書から複数のテストケースを出力するようなモデルを作成する。

仕様書に書かれているソフトウェアの予想入力と予想出力の想定範囲から外れたエラーケースなどの未想定テストケースを生成 AI が作成した複数のテストケースから探し出し、開発者が手動で作成したテストケースに追加させることでテストケースのカバレッジを向上させる。

作られたカバレッジが高いテストケースを参考に顧客と再議論し、顧客の要望の漏れを探し修正・追加などの作業を行う「テストケース駆動再設計」過程で顧客の要望をより具体化し、開発者の想定を最小限にする。

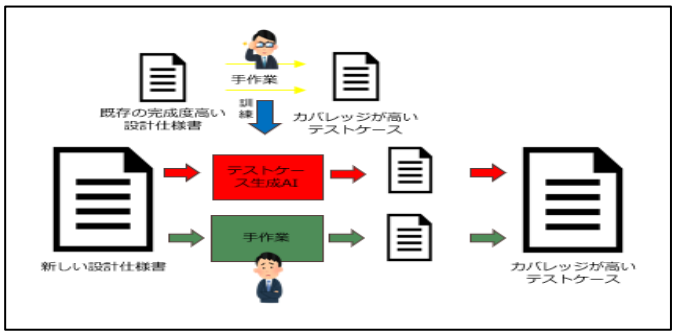

図 2： 生成 AI が作成したテストケースと手動で作成したテストケースを紐づけした、テストケース駆動再設計の

3.2 テストケース自動生成モデルの学習と生成

本論文で使うテストケース自動生成モデルでは日本語の自然言語文で書かれた設計仕様書を形態素分解するために GinZa が使われる、分解された形態素をグラフニューラルネットワークを用いて動作の入力である授与データ、入力に従って予想される出力である確認データ、その 2 つとは関係がない無関係データの 3 つのデータに分割する。

分割されたデータから、授与データと確認データの関係を学習することで、モデルは新しい設計仕様書の自然言語文の 1 つの項目から最初 5 つ以上のテストケースを出力することを目指す。

3.3 未想定テストケースと手動で作成したテストケースの紐づけ

生成 AI が作ったテストケースから、開発者が手動で作ったテストケースに含まれていない、想定できなかったテストケースを追加させる。開発者が手動で作成したテストケースに書かれていない生成 AI が作成したテストケースの数、その中で開発者がテストケースに追加した「未想定テストケース」の数を測定する。

3.4 測定方法

実験に使う新しい設計仕様書は食堂予約ツールのソフトウェアの画面を想定し、顧客と相談することを仮定し、初心者のソフトウェア開発者と設計仕様書を作成、事前にベテランソフトウェア開発者が作成した設計仕様書と比較し、設計仕様書に点数を付ける。

その後、初心者がソフトウェア開発者が設計仕様書から作ったテストケースに対してのカバレッジを測定し、生成 AI が作成した「未想定テストケース」追加させたテストケースにカバレッジと比較する。

最後に、「未想定テストケース」を追加させたテストケースを参考に「テストケース駆動再設計」を実行させた設計仕様書に点数を付け、最初の点数と比較する。

4．終わりに

本研究のように生成 AI を使ったテストケース生成の効率化は顧客の要望と結果物の間の乖離を無くし、ソフトウェア開発にかかるコストと時間を節約、ベテランではない開発者が自ら簡単にテストケースのカバレッジを向上させることができるなどの効果が期待される。

参考文献

[1]Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46).

[2]小川貴史「係り受け解析を用いた設計仕様書とテストケースの紐づけ手法」（2023）

[3]Mäkinen, S., & Münch, J. (2014). Effects of test-driven development: A comparative analysis of empirical studies. In *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering: 6th International Conference, SWQD 2014, Vienna, Austria, January 14-16, 2014. Proceedings 6* (pp. 155-169). Springer International Publishing.

[4] Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A., & Rothermel, G. (2001). An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *10*(2), 184-208.

[5] Myers, J. P. (1992). The complexity of software testing. *Software Engineering Journal*, *7*(1), 13-24.

[6] Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A., & Rothermel, G. (2001). An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *10*(2), 184-208.

[7] Tanizaki, K., Hiruta, Y., Soma, T., Yamao, N., Kato, S., & Iizuka, Y. (2022). A Method for Software Test Design Considering Weakness and Adverse Condition. *Total Quality Science*, *7*(3), 173-189.

[8] 中島毅, & 東基衛. (2008). ソフトウェア開発における品質プロセスのコスト最適化のためのモデルとシミュレーションツール. *電子情報通信学会論文誌 D*, *91*(5), 1216-1230.