ライブラリのバージョン更新支援のための実行トレースからのテストケース生成

嶋利 一真[1,a)]　石尾 隆[2,b)]　井上 克郎[1,c)]

概要：ソフトウェア開発において，ライブラリの利用が盛んに行われている．ライブラリは機能追加やバグ修正のために頻繁に更新されており，その変更がソフトウェアの振舞いを変えてしまう可能性がある．本研究では，利用するライブラリを更新したときの影響の評価を低コストで行うために，ライブラリ更新前後におけるソフトウェアの動作を記録，比較する手法を提案する．

1. はじめに

ソフトウェア開発において，ライブラリの利用が盛んに行われている．ライブラリは機能追加やバグ修正のために頻繁に更新される [1]．しかし，ライブラリの新しいバージョンが後方互換性を持たない場合もあり，開発に利用するライブラリを更新することがソフトウェアの振舞いを変えてしまう可能性がある．Mostafa らの調査 [2] では，ライブラリの新しいバージョンの 76.5%が後方互換性を持っていなかったと報告されている．また，ライブラリ更新時の回帰テストが不十分である場合が多いことも指摘されている．

ライブラリの更新の中には，セキュリティ脆弱性などの問題の修正も含まれるため，利用するライブラリの更新を行わないと，ソフトウェアがライブラリに起因する問題の影響を受ける危険性が高まる．しかし，ライブラリの振舞いの変化の影響を避けるためにライブラリの更新バージョンの利用に消極的な開発者も多い [1]．ライブラリの新しいバージョンが公開されたことを開発者に通知する仕組みは存在しているが，新しいバージョンのライブラリではソフトウェアが動作せず，開発者が以前のバージョンに戻すという事例も観測，報告されている [3]．

本研究では，ライブラリ利用者の新しいバージョンへの移行を支援するために，利用している機能に限定した後方互換性を効率的に検証可能とすることを目標とする．具体的なアプローチとして，ライブラリ更新前後におけるソフトウェアの動作を比較する手法を提案する．ライブラリ更新前の入力と出力の組をテストケースとして，更新後の入力に対しての出力と比較しそれが一致するかどうかを確認することで，ライブラリ更新の影響がソフトウェアにあるかどうかを確認する．

2. 背景

ライブラリの更新がソフトウェアに与える影響を評価する方法としては，回帰テストの実施が一般的である．ソフトウェアに対して用意されたテストスイートをライブラリ更新後のソフトウェアに対して実行し，更新前と同様の動作をすれば影響がないと判断できる．しかし，単体テストのような単純なテストでは，ライブラリの振舞いのバージョン間での差異が十分に検証できない場合があり，連続した多くの API の呼出しを含むような回帰テストの充実が必要である [2]．そのためには結合テストや受け入れテストの実施が考えられるが，ライブラリの更新は頻繁に行われるため，更新の都度，結合テスト等を実施すると実施コストが大きくなるという問題がある．

類似した挙動をするソフトウェアの振舞いの差異を見つける手法としては，Relative Debugging [4] がある．これはソフトウェアの実行環境の変化やマイグレーションの際に，新しいソフトウェアが過去のバージョンと同様の動作をするかの確認のために提案された手法である．両ソフトウェアが各実行ステップにおいて同一のデータを保持しているかを確認していき，差分が生じた段階でそれを開発者に通知する．この手法は，性質上，各ステップごとにすべてのデータを記録，比較する必要があり，大規模ソフトウェ

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University.
[2] 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology.
[a)] k-simari@ist.osaka-u.ac.jp
[b)] ishio@is.naist.jp
[c)] inoue@ist.osaka-u.ac.jp

アで取り扱うデータ量が大きくなると適用が困難となる．

ライブラリの更新前後でのプログラムの実行トレースを比較する手法として POLLUX [5] が提案されている．この手法はライブラリが内部で呼び出すシステムコールの列の変化や，メモリへの書き込みの変化を比較し，動作の変化を検出する．しかし，動作の変化があると判断された場合に，利用ソフトウェアの機能に与える影響を確認することは難しい．

ライブラリの動作の差異を可視化する手法として，DiffGen [6] が提案されている．この手法は Java クラスの 2 つのバージョンを入力として受け取り，それらの振舞いの違いが明らかになるように，同一の入力を 2 つのクラスに与えて出力を比較するようなテストケースを生成する．この手法はクラスの動作の差異を容易に確認することができるが，ソフトウェアでの API の利用方法に対応したテストケースとなるとは限らない．

3. 提案手法

本研究では，利用するライブラリの更新がソフトウェアの振舞いに与える影響を低コストで評価するために，ライブラリの更新前後におけるソフトウェアの動作を記録，比較する手法を提案する．具体的な方法としては，ソフトウェアの実行から現在利用中のライブラリに対する入力と出力の組を収集し，テストケースへと変換することで，新しいバージョンのライブラリに対して同じ入力を行った場合の出力と一致するかどうかを確認する．テスト実行の結果が完全に一致した場合は，ライブラリが後方互換性を維持していると推測できる．

提案手法の動作を図 1 に示すプログラム例を通じて説明する．更新前のライブラリを v_1，更新後のライブラリを v_2 とすると，まず，プログラム中で 4 行目のライブラリ v_1 に対する呼び出し直前の 3 行目の地点 A で，ライブラリへの入力を記録し，次に，ライブラリ v_1 を呼び出した直後の 5 行目の地点 B でライブラリの出力を記録する．ライブラリ v_2 に対しても同様の操作を行い，ライブラリ v_1 における地点 A から B にかけての入出力の情報の変化と，ライブラリ v_2 における地点 A' から B' にかけての入出力の情報の変化を比較する．そして，ソフトウェア側で利用する変数の情報の変化が同一であれば，ライブラリの更新はソフトウェアに影響を全く与えていないことが分かる．ライブラリ呼び出し前後の値のみを比較するため，差異が生じた場合は，その原因はライブラリ側にあり，ソフトウェア側の影響ではないこともわかる．

実行トレースの比較を継続的に利用するために，ソフトウェアに対する単体テスト，結合テストや受け入れテストの実行からトレースを収集し，ライブラリに対する操作のみを抽出してテストケースへと変換する．結合テストや受け入れテストにおける動作を再利用することで，開発者が

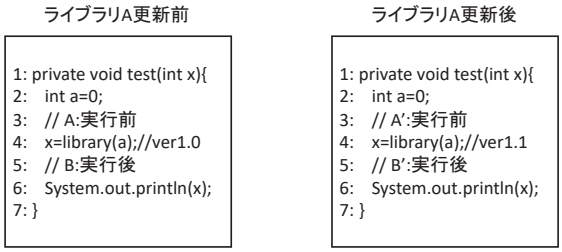

図 1 ライブラリ更新時の振舞いの変化の確認

手作業でライブラリのテストケースを記述することなく，ソフトウェアの何らかの動作に対応したライブラリの振舞いに関するテストを可能とする．

4. まとめと今後の課題

本研究では，ライブラリの新しいバージョンが後方互換性を持つかを検証するために，ライブラリの更新前後におけるソフトウェアの動作を記録，比較するためのテストケースを生成する手法に取り組んでいる．今後の課題として，十分なテストケースの作成のためにソフトウェアにおいてどのような情報を必要とするのか，そして実際のライブラリの更新に対して十分なテストケースを生成できるかどうかを調査していく必要がある．

謝辞 本研究は JSPS 科研費 JP25220003，JP26280021 の助成を受けたものです．

参考文献

[1] Ihara, A., Fujibayashi, D., Suwa, H., Kula, R. G. and Matsumoto, K.: Understanding When to Adopt a Library: A Case Study on ASF Projects, *Open Source Systems: Towards Robust Practices*, pp. 128–138 (2017).

[2] Mostafa, S., Rodriguez, R. and Wang, X.: Experience paper: a study on behavioral backward incompatibilities of Java software, *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 215–225 (2017).

[3] Mirhosseini, S. and Parnin, C.: Can automated pull requests encourage software developers to upgrade out-of-date dependencies?, *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 84–94 (2017).

[4] Abramson, D., Foster, I., Michalakes, J. and Sosi, R.: Relative debugging: a new methodology for debugging scientific applications, *Communications of the ACM*, pp. 69–77 (1996).

[5] Kalra, S., Goel, A., Khanna, D., Dhawan, M., Sharma, S. and Purandare, R.: POLLUX: safely upgrading dependent application libraries, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 290–300 (2016).

[6] Taneja, K. and Xie, T.: DiffGen: Automated Regression Unit-Test Generation, *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 407–410 (2008).