Qt Quality Assurance

WHITEPAPER

自律型テストに向けて

テストケースの設計を自動化するには

Alexej Popovič と Maximilian Blochberger 博士の見解

概要

自動化されたテストの実行は、継続的インテグレーション（CI）プロセスの一部として、開発プロセスの早期にバグを効率的に発見するための確立されたプラクティスです。しかし、テストは通常手動で作成されます。意味のあるテストケースの特定と実装は、テストプロセスの総コストに大きく貢献します。このコストを削減し、完全自動化テストプロセスに一歩近づくためには、テストケースの設計と実装を自動化する必要があります。探索的テストのように、自律型テスターはテスト対象アプリケーション（AUT）と相互作用して、その構造を発見し、最適化されたテストケースを生成します。　本稿では、自律型システムがどのように機能するかについてのアイデアを提示し、そのようなシステムを作成する際の課題について議論します。

ソフトウェア開発が徐々にアジャイル原則や継続的インテグレーション、デリバリー手法を取り入れるようになり、テスト自動化は製品開発ライフサイクルの不可欠な一部となっています。テスト自動化は、人的労力を削減し、実行サイクルを高速化することで、シフト・レフト・テスト、すなわち、可能な限り早期のテストを可能にし、不具合の修正コストを低減します[1]。

しかし、テストの自動化という用語は、通常、テストの実行を指しており、テストケースの定義と実装は、テスターが手作業で行う作業です。これについても、どのように自動化できるかを議論します。

テストケースの作成は、AUTの内部構造と期待される動作に関する利用可能な情報に依存します。ブラックボックステストがシステムの入力と期待される出力のみを考慮するのに対して、ホワイトボックステストはシステムがどのように出力に到達するかも考慮します。一方では、ホワイトボックステスト技法は、すでに自律型テストを可能にしています。　利用可能なツールには、(i)ソースコード[1]に基づいて、指定された関数のユニットテストを生成するもの、(ii)クラッシュ(ファズテスト)などの特定の条件をトリガーする入力データ、または(iii)タイプミスマッチ、ヌルポインタチェックの不足など、一般的に静的に特定できるエラーを識別するツールがあります。これらは、AUTを実行せずに識別できるものです。

一方、ブラックボックステストでは、テストケースは要求事項や仕様に基づいて、あるいは探索的テストを通じて潜在的なテストケースを発見することによって、手作業で作成されます。テストケースの自動作成には、通常、モデルベーステストのためのモデルなど、そのための追加情報が必要です。

テストプロセスのコストをさらに削減するために、GUIアプリケーションのブラックボックステスト生成に注力しています。

[1] CodiumAI (https://www.codium.ai/)
EvoSuite (https://www.evosuite.org/)
Parasoft Jtest (https://www.parasoft.com/jtest)
AgitarOne (http://www.agitar.com/solutions/products/agitarone.html)

アプローチ

GUI アプリケーションのテストケースを作成する場合、AUTの仕様が利用できないことがあります。そのような場合、テスターは通常、学習、テスト設計、テスト実行を同時に行う探索的テストを利用します [1,2]。

テスターは経験と直感を用いてAUTの機能を体系的に探索し、学んだことを仕様として扱い、同時にAUTの動作の正しさを評価します。

自律型テストケースの設計と実装のための我々の提案は、探索的テストとモンキーテストに基づいています。このアプローチは、概念的に4つのフェーズから構成されます：

1. **探索 (Exploration)**
2. **汎化 (Generalization)**
3. **最適化 (Optimization)**
4. **検証 (Validation)**

これらのフェーズは、あらかじめ定義された終了基準が満たされるまで繰り返されます。実際には、図1に示すように、4つのフェーズは絡み合っており、即時のフィードバックがどのアクションを実行するかに影響を与えることができます。

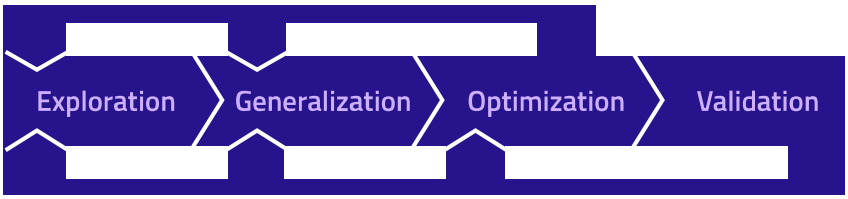

図1：自律型テストケースの設計、実装、実行の4つのフェーズ

AUTと相互作用するためには、AUTを実行する必要があります。探索フェーズでは、自律型テスターはその後(i) AUTの状態のスナップショットを取り、

(ii) 次のアクションを決定し、(iii) そのアクションのための入力データを生成し、(iv) アクションを実行します。図 2 に示すように、状態とアクションはトレースに記録されます。トレースは、後でテストケースとして再生することができます。

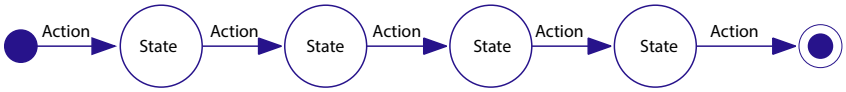

図2：アプリケーションの探索中に収集されたトレース

汎化フェーズでは、収集されたトレースに基づいて、図 3（次ページ参照）に示されるように、状態を抽象的な状態[3] にマージすることによって、AUTのステートマシン （モデル）が推定されます。

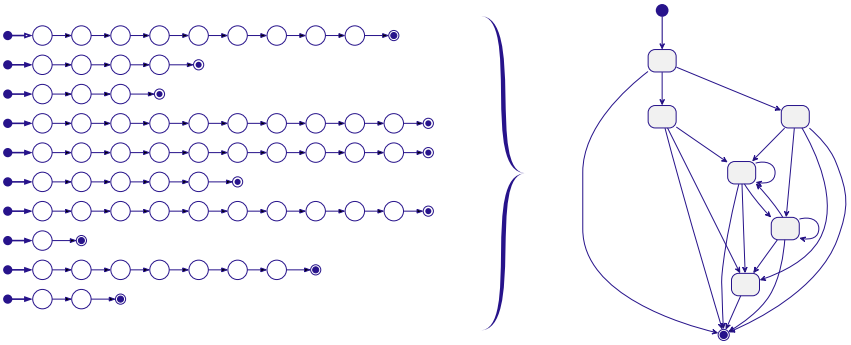

図3：複数のトレースに基づくモデルの汎化

最適化フェーズでは、導出された AUTのモデルを利用して、個々のテストケースの潜在的な改善点を特定します。例えば、特定された不具合を再現するためのより短いパスや、無関係なテストケースの省略などです。

検証段階では、AUTの正しさが評価されます。これは、クラッシュのような普遍的な誤動作、ヒューリスティックス、または非機能テストでは、ユーザビリティの問題などです。機能テストについては、追加入力として仕様を提供する必要があります。さらに、検出されたテストケース自体について、例えば、再現可能かどうか、あるいは、最小化されたテストケースが、最初に検出された不具合をまだ明らかにしているかどうかを識別するために、検証が実行されます。

記述されたアプローチは、既にテストケースの設計と実装をテストの実行と組み合わせています。AUTは検出のために実行される必要があるので、検証のために再度実行する必要はありません。収集されたトレースはオフラインテストを可能にするのに十分な情報を含んでいます。

課題

前で紹介したアプローチは、簡潔にするために多くの詳細を省略しています。このセクションでは、このアプローチを完全に自律的なものにするための選択された課題について議論し、潜在的な解決策を概説します。

探索の際、アプリケーションの内部状態に影響する未知の変数、例えば、実行中のハードウェアの状態、時間、センサー入力など、ブラックボックステストの課題に直面します。さらに、一度に考慮できるのは、よく知られたアクションのセットだけです。特定のテキストを入力したり、タッチジェスチャーを実行したりするような複雑なアクションは、自動検出が困難です。AUTに関する利用可能な情報は、使用するツールキットやオペレーティングシステムの実行時イントロスペクション機能に依存します。

汎化の際、主な課題は、十分に抽象的でありながら、テストケースの生成に役立つモデルを見つけることです。さらに、状態を抽象的な状態にマージするために使用される類似性メトリックによって考慮されるデータは、アプリケーション・ドメインと特定のテスト目標に大きく依存します。

最適化においては、複雑性が最大の課題です。クラッシュが確認された場合、まず、再現性を検証するためにトレースを再生する必要があります。次に、導出されたモデルに基づいてクラッシュに至るすべての短いパスを実行し、最適化の結果がまだクラッシュを明らかにしているかどうかを検証する必要があります。

検証中、主にAUTのビジネスロジックが不明であるという問題に直面します。人間であれば、計算機に対して入力1+1の出力が3であることは欠陥であると直感的に結論づけることができます。ソフトウェアシステムは一般的に、期待値が提供されない限り、出力の正しさを評価することはできません。AI/MLの手法は、期待される結果を予測するために適用することができます。

さらに、自律型テスターは観察できるもののみを検証することができ、つまり、欠落している機能を検出することはできません。ヒューリスティックを適用して一般的な誤りを検出することができます。たとえば、ウィンドウタイトル内の「エラー」という用語を探すこと[4]や、ビジュアルGUIテストで行われるように前回の実行との違いを特定すること[5]が含まれ、偶然に削除された機能を検出するのに役立つ可能性があります。

関連研究

AUTの自律的探索は、通常モンキーテストとして知られていいます [6-9]。これらのテストは、AUTをクロールするためにランダムなアクションを実行したり、新しい状態を発見する確率の高いアクションを優先したりすることができます[10]。これを実現するために、1つのアプローチとしてQ学習があります[3,10-12]。いくつかのツールは、自然言語で記述された製品仕様に基づいてテストケースの形式的記述を生成するために、大規模言語モデル（LLM）を使用しています。LLMは、自然言語の記述に基づいてコードを生成することもできます。したがって、同じ仕様を使用してテストスクリプトを生成することが近い将来現実になるかもしれません。Valdés らは、GUI テスト自動化に関連する研究の包括的な概要を提供しています。

結論

普遍的なオラクルが存在しないため、動作が正しいかどうかを判断することはできず、テストにおいて完全な自律性を達成することは決してできないかもしれません。車両の運転ユニットを自律的にテストすることを考えてみてください。もし自律型テスターが常に車が正しく運転されているかどうかを判断できるのであれば、「正しい」運転方法を常に選択することが、自己運転の課題を解決することになります。不条理に、任意のシステムに汎化すると、自律型テスターはすべてを知っていることになり、正しい情報と誤った情報を区別できるからです。それにもかかわらず、自律型テスターが選択されたユースケースのセットに限定されていても、価値はあります。AUTをクラッシュさせる方法を見つけたり、誤った翻訳や動作やインターフェースの不整合を認識したり、使いやすさやアクセシビリティを検証したり、回帰を見つけたりすることができます。これにより、テスターはビジネスロジックの検証に集中できます。この研究では、テストにおける完全な自律性を達成するために必要なことの一部にのみ焦点を当てました。テストプロセスは、アプリケーションを実行し、結果を収集すること以上のものを含みます。また、自律型テスト計画や結果評価、場合によっては修正提案を含めることを目指すこともできます。

このホワイトペーパーは、ESE Kongress 2023で発表されました。

参考文献

[1] R. Patton, *Software testing*, 2nd ed. Indianapolis, IN: Sams Pub, 2006

[2] J. Bach, 'Exploratory Testing Explained', *Satisfice, Inc.*, 2003

[3] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro, 'Automatic testing of GUI-based applications', *Softw. Test. Verification Reliab.*, vol. 24, no. 5, 2014

[4] S. Bauersfeld, T. E. J. Vos, N. Condori-Fernández, A. Bagnato, and E. Brosse, 'Evaluating the TESTAR tool in an industrial case study', in *ESEM*, ACM, 2014

[5] E. Börjesson and R. Feldt, 'Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry', in *ICST*, IEEE Computer Society, 2012

[6] A. M. Memon, I. Banerjee, and A. Nagarajan, 'GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing', in *WCRE*, IEEE Computer Society, 2003

[7] D. Amalfitano, A. R. Fasolino, and P. Tramontana, 'A GUI Crawling-Based Technique for Android Mobile Application Testing', in ICST Workshops, IEEE Computer Society, 2011

[8] 'UI/Application Exerciser Monkey', *Android Developers*. Accessed: Oct. 12, 2023. Available: https://developer.android.com/studio/test/other-testing-tools monkey

[9] N. Nyman, 'Using monkey test tools', *Software Testing & Quality Engineering*, 2000

[10] S. Bauersfeld and T. E. J. Vos, 'User Interface Level Testing with TESTAR; What about More Sophisticated Action Specification and Selection?', in *SATToSE*, in CEUR Workshop Proceedings, vol. 1354. CEUR-WS.org, 2014

[11] J. Eskonen, J. Kahles, and J. Reijonen, 'Automating GUI Testing with Image-Based Deep Reinforcement Learning', in *ACSOS*, IEEE, 2020

[12] D. Adamo, M. K. Khan, S. Koppula, and R. C. Bryce, 'Reinforcement learning for Android GUI testing', in *A-TEST@ESEC/SIGSOFT FSE*, ACM, 2018

[13] O. R. Valdés, T. E. J. Vos, P. Aho, and B. Marín, '30 Years of Automated GUI Testing: A Bibliometric Analysis', in *QUATIC*, in Communications in Computer and Information Science, vol. 1439. Springer, 2021

著者

Alexej Popovičは、知識表現と推論、さらにはロボティクスに情熱を注ぐソフトウェアエンジニアです。現在、Qt Groupでソフトウェアテストのためのツール開発を支援しています。

Maximilian Blochberger博士は、Qt Groupのソフトウェアエンジニアであり、セキュリティ研究者です。彼は、必要なバックグラウンドがない場合でも、開発者が安全でプライバシーに配慮したアプリケーションを作成できるよう支援することに主に関心を持っています。

詳細については、お問い合わせください。

Qt Quality Assurance

www.qt.io/squish