リスク算出式を変更可能な バグローカリゼーションフレームワーク

坂本一憲[†1], 下條清史[†2], 徳本晋[†3], 上原忠弘[†3], 杉本元気[†2], 本田清[†2], 鷲崎弘宜[†2], 深澤良彰[†2]

アブストラクト
バグローカリゼーションはソースコード中のバグの位置を推定することで, デバッグに必要な労力を削減する有用な手法である. 我々は過去に複数のプログラミング言語に対応して, かつ, ステートメントなどのプログラム要素の実行履歴を元に計算するリスク算出式を変更可能な, バグローカリゼーションフレームワークを提案した. 本論文では既存研究のバグローカリゼーション手法が利用する情報を整理することで, 提案フレームワークがどのような手法に対応可能であるかを議論する.

A Bug Localization Framework Supporting Customizable Risk Calculation

Kazunori Sakamoto,[†1] Kiyofumi Shimojo,[†2] Susumu Tokumoto,[†3] Tadahiro Uehara,[†3] Genki Sugimoto,[†2] Kiyoshi Honda,[†2] Hironori Washizaki[†2] and Yoshiaki Fukazawa[†2]

Abstract
Bug localization is an effective way to reduce debugging efforts by locating bugs in source code. We previously proposed a framework for localizing bugs supporting multiple programming languages and a customization feature for risk formula by utilizing program execution logs. In this paper, we discuss what methods our framework can support by surveying existing bug localization methods.

1. はじめに

バグローカリゼーションはソースコード中のバグの位置を推定する手法であり, デバッグに必要な労力を削減することができる有用な手法である [1].

バグローカリゼーションには様々な種類があり, ペクトラムベース手法が代表的である. スペクトラムベース手法は, ある観点から記録したプログラムの実行履歴を活用することで, バグの位置を推定する. 一般に, 成功テストが頻繁に実行した箇所にバグが含まれている可能性は低く, 失敗テストが頻繁に実行した箇所にバグが含まれている可能性は高い. スペクトラムベース手法は, 各プログラム要素に対して, 失敗テストと成功テストが実行した回数をそれぞれ測定することで, バグが含まれている度合いを示すリスクを計算する. なお, リスクの計算式をリスク計算式と呼ぶ.

†1 国立情報学研究所
National Institute of Informatics
†2 株式会社富士通研究所
Fujitsu Laboratories
†3 早稲田大学
Waseda University

既存研究にて様々なリスク計算式が提案されており, リスク計算式を変えることで, リスクの高い順にデバッグを行った際に, デバッグしなければならないプログラム要素の数が変化する. 実際にバグを含むプログラム要素が最も高いリスクを持つ場合は, デバッグに必要な労力が最小化される. いかにして実際にバグを含むプログラム要素のリスクが高くなるようなリスク計算式を利用するかが, スペクトラムベース手法において重要となる.

我々は, 複数のプログラミング言語に対応したカバレッジ測定フレームワーク Open Code Coverage Framework (OCCF) [2] を利用して, 複数のプログラミング言語に対応して, かつ, リスク算出式を変更可能なバグローカリゼーションフレームワークを提案した.

2. リスク算出式を変更可能なバグローカリゼーションフレームワーク

本節では提案フレームワークを説明する. 図 1 で提案フレームワークの概要とアーキテクチャを示す.

ユーザはソースコードとテストコードを提案フレームワークに入力する. C 言語はマクロによるプリプロセス命令があり, 実行履歴を取得する際は, プリプロセス命令

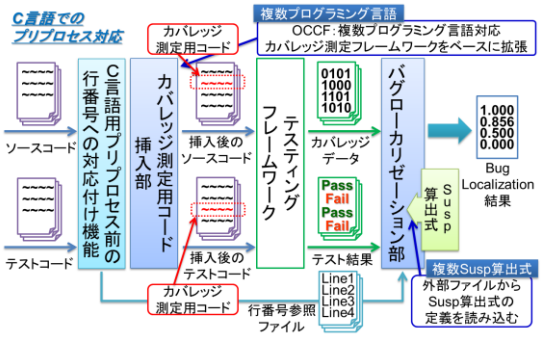

図 1. 提案フレームワークの概要とアーキテクチャ

を全て展開する必要がある．しかし，プリプロセス命令を展開すると，元々のソースコードと内容が変化してしまい，ステートメントの位置が移動して，行番号が変化する場合がある．バグローカリゼーションの結果を表示する際は，ステートメントを指すために行番号を利用するため，プリプロセス命令を展開する前の行番号を表示する必要がある．そこで，展開前と展開後の各ステートメントの位置情報の対応関係を記録するために，C 言語のソースコードに対しては対応付けの処理を行う．

必要に応じて位置情報の対応関係を記録した後，カバレッジ測定用コード挿入部は，OCCF を活用することで，プログラムの実行履歴を取得できるようにソースコードを変形する．その上で，テスティングフレームワークでテストを実行することで，各テストで実行された箇所が取得できる．最後に，ユーザが IronPython 言語で記述したスクリプトからリスク算出式の定義をバグローカリゼーション部が読み込み，定義に基づいて各ステートメントに対するリスクを算出する．C 言語の場合は，対応関係を利用することで，プリプロセス命令を展開する前の位置情報で結果を表示する．

3. 関連研究

Wong らはバグローカリゼーション手法に関する研究の調査を行っており [3]，その結果，既存研究にて提案されている手法を a) スライスベース手法，b) スペクトラムベース手法，c) 統計的手法，d) ステートベース手法，e) マシンラーニングベース手法，f) モデルベース手法，g) データマイニング手法の 7 種類の手法に分類している．また，彼らの調査研究の後に，Papadakis らはミューテーション解析を活用したバグローカリゼーション手法を提案しており [4]，上述の 7 種類のいずれにも当てはまらない新しい手法であると言える．提案フレームワークはプログラムの実行履歴に基づくバグローカリゼーション手法に対応しており，既に述べたように b) スペクトラムベース手法を実装している．しかし，d) ステートベース手法も，プログラムの実行時の情報を活用しているため，単にどのプログラム要素が実行されたかという情報だけではなく，その際のメモリの情報がどうなっていたかも併せて記憶することで，対応可能であると思われる．また，c) 統計的手法と e) マシンラーニングベース手法は，利用する情報について言及してしているわけではないので，実行履歴に基づく手法であれば対応できる可能性がある．

4. まとめと今後の課題

本論文では，複数のプログラミング言語に対応して，かつ，リスク算出式を変更可能なバグローカリゼーションフレームワークの概要について説明して，その上で，既存のバグローカリゼーション手法の分類と，提案フレームワークが対応可能な手法について議論を行った．

現状では，提案フレームワークはスペクトラムベース手法にのみ対応しているが，今後，プログラム中の変数の情報などを記録することで，ステートベース手法についても対応できるよう改良を行っていきたい．

参考文献

[1] Jones, J. A. and Mary J. H.: Empirical evaluation of the tarantula automatic fault-localization technique, 20th IEEE/ACM international Conference on Automated software engineering. ACM, pp. 273-282, 2005.

[2] Sakamoto, K., Shimojo, K., Takasawa, R., Washizaki, H. and Fukazawa, Y.: OCCF: A Framework for Developing Test Coverage Measurement Tools Supporting Multiple Programming Languages, IEEE Sixth International Conference on Software Testing, Verification and Validation, 9 pages, 2013.

[3] Wong, W. Eric and Vidroha Debroy: A survey of software fault localization, University of Texas at Dallas, Tech. Rep. UTDCS-45-09, 2009.

[4] Papadakis, M., and Yves L. T.: Using Mutants to Locate, IEEE Fifth International Conference on Software Testing, Verification and Validation, pp. 691-700, 2012.