テスト戦略の最適化に向けて

門田暁人†

本稿では，バグ予測に基づくテスト戦略の最適化について述べる．

Toward Optimizing Test Strategies

Akito Monden†

This paper discusses about optimizing test strategies based on bug prediction.

1. はじめに

近年，ソフトウェアの大規模化・短納期化に伴い，ソフトウェアテストの効率化がますます重要となっている．従来，テスト資源（人員，テストケースなど）を重点的に配分すべき，バグの多いモジュール（サブシステム，機能）を特定することを目的として，各モジュールのバグの有無やバグ数を予測するバグ予測研究が盛んに行われてきた．

筆者らは，バグ予測に基づくテストの効果（削減可能なコスト）を，シミュレーションにより推定する方法を提案している[1]．バグ予測に基づく 4 つのテスト戦略を比較した結果，テストケース数を予測バグ数×log（モジュール規模）に比例させて配分する戦略が最も優れており，バグ予測を行わない場合と比べて約 25%のテスト工数削減が可能であることを示した[1]．

本稿では，より最適なテスト戦略を追求するために，次の 2 つのアプローチについて検討する．一つ目は，バグ予測結果が 100%正しいという前提に立ち，各モジュールの（予測に基づく）期待バグ発見数の和を最大化することを目的関数とし，各モジュールへのテストケースの配分数を求める「資源配分問題」として定式化する方法である．二つ目は，一つ目の方法の拡張であり，バグ予測に誤差が含まれることを前提とし，予測誤差が大きかった場合のリスクヘッジを行うために，バグ予測に基づかないテストケース配分方法を組み合わせる方法である．

2. シミュレーションモデル

テストのシミュレーションを行うために，テストケースと発見バグ数の関係を表すモデルが必要となる．本稿では，文献[1]に従い，指数型 SRGM にモジュール規模のパラメータを加えた式(1)を用いる．$\hat{H}_i(t_i)$は，テスト対象モジュール m_i に対し，テスト工数（テストケース数）t_i を配分した時のバグ発見数の期待値である．

$$\hat{H}_i(t_i) = a_i[1 - \exp(-b_i t_i)], \qquad b_i = b_0/S_i \qquad (1)$$

ここで，a_i はモジュール m_i に潜在するバグ数，b_i は単位工数（テストケース）あたりのバグ発見率であり，モジュール m_i におけるバグの発見しやすさを表すパラメータとなる．この b_i は b_0 と S_i によって決まり，S_i はモジュール m_i の規模， b_0 は定数である．

式(1)に基づいてテストのシミュレーションを行うためには，定数 b_0 を与えること，及び，各モジュールの潜在バグ数 a_i を与えることが必要となる．a_i については，テスト対象のソフトウェアの次のバージョンで発見されたバグのうち，その混入時期がテスト対象のバージョンであったものを潜在バグとしてカウントする．b_0 については，4.2 節で述べる．

3. テスト戦略

3.1. 戦略1：バグ予測に基づくテストケース配分

総テストケース数 t_{total} の制約のもとで，総バグ発見数の期待値 H_{total} (= $\sum \hat{H}_i(t_i)$)が最大となるように，モジュールに対するテストケース数 t_i （i=1,…,n）を決定する．

本戦略は，$\hat{H}_i(t_i)$が単調増加かつ凹関数（上に凸）であるため，これは分離凸資源配分問題である．この問題は，greedy 法により O(n)で最適解が得られる．

3.2. 戦略2：バグ予測に基づかないテストケース配分

バグ予測を行わない場合のテストケース配分を考えるにあたって，本稿では，全てのモジュールのバグ密度

†奈良先端化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

が等しいと仮定する．この仮定のもとでの最適なテストケース配分は，式(1)から，全モジュールのテストケース密度（規模あたりのテストケース数）を等しくすることが最適解となる．実はこの戦略は，テストケース密度の基準値を決めて網羅的にテストを行うという，従来一般企業で行われている戦略に他ならない．

バグ予測にはある程度の誤差が含まれることが避けられないことから，本稿では，戦略 1 と 2 を組みあわせる方法について検討する．これを戦略 3 と呼ぶことにする．戦略 3 は，総テストケースの m%については戦略 2 を採用し，残りの(100-m)%については戦略 1 を採用する．これはつまり，戦略 1 における greedy 法の最初の $t_{total}\times m\div 100$ ステップを省略し，$t_1=t_2=\cdots t_m= t_{total}\times m\div 100\div n$ を割り当て，以降のステップから進めることを意味する．（ただし，実際には $t_{total}\times m\div 100\div n$ が割り切れるとは限らないため，$t_1,t_2,\cdots t_m$ は高々1 の差を許容して合計が $t_{total}\times m\div 100$ になるようにランダムに割り当てる）

3.3. その他の戦略

文献[1]に倣い，テストケース数を予測バグ数に比例させる戦略(戦略 4)，予測バグ密度に比例させる戦略(戦略 5)，および，予測バグ数×log（モジュール規模）に比例させる戦略(戦略 6)についても比較対象とする．

4. シミュレーション

4.1. バグ予測

オープンソースソフトウェア Mylyn を対象とし，1 ファイルを 1 モジュールとみなしてバグ予測を行う．Mylyn ver.1.0 を予測モデルの構築に用い，ver.2.0 に対してテストのシミュレーションを行う．モジュール数は，ver.1.0 が 1,008（うちバグを含むものは 769），ver.2.0 が 1,230（同 848）である．モデリング手法はランダムフォレストを用い，目的変数をバグ数とする．説明変数は文献[2]と同じものを用いる．

4.2. シミュレーションの条件

本稿では，1 モジュールは 1 ファイルであることから，ファイル単位のテストである単体テストのシミュレーションを行う．

単体テストのテスト基準として，文献[3]において，ソースコード 1 キロステップあたり 100 件のテストケースを割り当てる事例が紹介されている．本稿ではこれに倣い，Mylyn ver.2.0 の総行数が 192224 ステップであることから，総テストケース数 t_{total}=19222 とする．

パラメータ b_0 の決定にあたっては，一般に，単体テストでは残存バグの 50～60%程度しか検出できない（残りは後の結合テスト，総合テスト等で検出される）ことを考慮し，従来のテスト戦略である戦略 2 を用いたときに，総バグ発見率が 50%となるように b_0 を与えることにする．今回のデータにおいては，b_0=6.93 となった．

4.3. 結果

バグ予測精度は，Norm(P_{opt})=0.832 であった．これは悪くない値である．Norm(P_{opt})の定義は[1]を参照されたい．

シミュレーションについては，現在実験を進めているところであるが，総バグ発見率で比較すると，戦略 1 は戦略 2，4，5，6 よりも高い値を示しており，戦略 1 の有効性が示唆された．一方，戦略 3 については，結果が m の値に依存するが，現時点では戦略 1 を上回る結果は得られていない．

5. おわりに

本稿では，バグ予測に基づくより良いテスト戦略を得ることを目的として，(1)総バグ発見数の期待値が最大となるようにテストケースを割り当てる方法，および，(2)バグ予測によらないテスト戦略を組み合わせる方法の 2 つを提案した．現在，Mylyn プロジェクトのデータに基づいてテストのシミュレーションを実施中である．

謝辞

本研究の一部は，文部科学省科学研究費補助金（基盤研究（C）：課題番号 80311786）に基づいて行われた．

参考文献

[1] A. Monden, T. Hayashi, S. Shinoda, K. Shirai, J. Yoshida, M. Barker, K. Matsumoto, "Assessing the cost effectiveness of fault prediction in acceptance testing," *IEEE Transactions on Software Engineering*, Vo.39, No.1, pp.1345-1357, Oct. 2013.

[2] A. Monden, J. Keung, S. Morisaki, Y. Kamei, K. Matsumoto, "A Heuristic Rule Reduction Approach to Software Fault-proneness Prediction," *Proc. Asia-Pacific Software Engineering Conference (APSEC 2012)*, pp.838-849, 2012.

[3] 奈良隆正，"ソフトウェア品質検証，評価技術の勘所", JaSST'09 Kansai, 2009.