サイバーフィジカルシステムのシミュレーションテストの属性を用いたdefect-proneモジュール予測

塚本 悠乃　森崎 修司

サイバーフィジカルシステムのテストではシミュレーションテストが用いられる．通常，シミュレーションテストは「障害物回避」「目標地点での停止」といったフィジカル面での属性を持つ．これまでの defect-prone モジュール予測手法はソースコードメトリクスや更新履歴のようなプロダクトコードの属性やプロダクトの開発プロセスの履歴を入力として使っているが，シミュレーションテストの属性をを使うことで予測精度が高まる可能性がある..．しかし，シミュレーションテストのフィジカル面での属性をプロダクトコードの属性とするためにはシミュレーションテストによって実行されるプロダクトコード (モジュール) を対応づけする必要がある．本研究では，シミュレーションテストから得られる属性もプロダクトコードの defect-prone 予測手法の入力として使うことを目的として，手法を提案する．具体的には，シミュレーションテストの実行時の動的プロファイリングの結果を使ってモジュールの属性とする．

1　はじめに

サイバーフィジカルシステムの検証はシミュレーションテストによって行われる．サイバーフィジカルシステムは，現実世界から得られるデータを入力として動作する．そのため，シミュレーションテストはセンシングから得られる膨大な情報量を模擬的に生成して複雑な計算を行うことから，時間を始め多くのコストがかかる．また，さまざまな環境下でサイバーフィジカルシステムの検証を行うため，たくさんの異なるシミュレーションテストに合格する必要がある．さらに，リグレッションテストを行うシステムであれば，コストのかかるシミュレーションテストを何度も実行しなければならない．シミュレーションテストのコストを削減するには，システムのソフトウェアに含まれている欠陥を検出しやすいシミュレーションテストから実行するべきである．シミュレーションテストがより早い段階で欠陥を検出できれば，実行するテストの数を減らすことが可能である [2].

Defect-prone モジュール予測 [1]，[3]，[4]，[5]，[6] は欠陥を含む可能性が高いモジュールを予測できるため，欠陥を含む可能性が高いモジュールからテストすることにより，テストの数を減らせる可能性がある．しかし，Defect-prone モジュール予測ではソースコードから計測できるソースコードメトリクスやソースコードの編集履歴から得られる情報を使っているため，シミュレーションテストの属性をそのまま使うことはできない．シミュレーションテストの属性とモジュールを結びつける情報がないからである．そこで，本研究では，欠陥を含む可能性が高い Defect-prone モジュールをテストするシミュレーションテストを選ぶ手法を提案する．

2　背景

2.1　サイバーフィジカルシステム (Cyber Physical Systems)

サイバーフィジカルシステムとは現実世界から得られるデータを収集・処理・活用するものであり，あ

---

* Defect-prone module prediction using attributes of simulation tests for Cyber Physical Systems

Yuno Tsukamoto Shuji Morisaki, 名古屋大学 大学院情報学研究科, Graduate School of Informatics, Nagoya University.

らゆる社会システムの効率化，新産業の創出，知的生産性の向上に寄与するものである [cite1]．例えば，自動車の無人運転がサイバーフィジカルシステムにあたる．無人運転では，LiDAR と呼ばれるレーザー光を使用したセンサーで対象物までの距離やその形状を認識したり，カメラから周りの環境を認識したりして，その情報から自車の次の行動・軌道を決定し，その通りに自車が動作するようにアクチュエータに制御の指示を出す．

2.2 シミュレーションテスト

サイバーフィジカルシステムの検証ではハードウェアでテストを行う前にソフトウェア単体でのシミュレーションテストが用いられることが多い．自動車の無人運転を例に取ると，実車を用いたテストでは，人が危険に晒されたり，使用する車やセンサーのために莫大な費用がかかったり，特別な環境の再現が困難であり，システムの検証を現実世界で完結させることは難しい．サイバーフィジカルシステムのシミュレーションテストとは，システムへの入力となるセンサーの値などの現実世界から得られる情報を擬似的に生成し，対象システムのソフトウェアに与え挙動を確認するシステムの検証手法である．

2.3 Defect-prone モジュール予測

Defect-prone モジュール予測はソフトウェアを構成するモジュールのうち，欠陥を含んでいる可能性の高いモジュールを予測する技術である [1]．欠陥が含まれている可能性が高いモジュールをメトリクスをはじめとするソースコードの属性や更新履歴の属性を用いて予測する．用いられるメトリクスにはプロダクトコードメトリクスや，変更にかかった工数や日数のようなプロセスメトリクスなどがある．

2.4 プロファイラ

プロファイラは，対象ソフトウェアを実行しその様子を監視・記録するソフトウェアである．対象ソフトウェアの起動から終了までを計測し，呼び出された関数やメソッド，モジュールなどを記録する．関数やモジュールの呼び出し回数や所要時間をプロファイルとして出力する．

3 提案手法

3.1 前提

3.1.1 シミュレーションテストのフォーマット

実行するシミュレーションテストの属性は，テストコードから知ることができる．テストコードの記述からわかる属性として以下のようなものがある．

1. 登場する物体の種類や数
2. 物体の位置，初期ステータス
3. シミュレーションテスト中の物体の動き
4. シミュレーションテスト中の全体の環境

3.1.2 プロファイラの対象

プログラム全体がプロファイラの対象となる．プロファイラは，ハードウェアの割り込みを利用したり，コードに命令を埋め込んだりしてプログラムの実行を監視する．

3.2 手順

手順を実施する作業者は，開発者，テスト作成者，テスターである．テスターの作業の一部はインテグレーションツールを使えば、自動化できる場合がある．開発者はプロダクトコードを開発し，コードを変更した際はすべてのシミュレーションテストに合格し，デグレードしていないことを確かめる必要がある．テスト作成者はシミュレーションテストのテストコードを書く．テスト作成者は対象のソフトウェアが利用される状況を考え，作成・実行すべきシミュレーションテストを準備する．テスターはテスト作成者が作成したテストコードを実行し，シミュレーションテストでのソフトウェアの振る舞いが意図したものであることを確認する．シミュレーションテストに合格しなかった場合，コードを変更した開発者はコードを修正する必要がある．

1. 対象のソフトウェア，シミュレーションテストに使用するテストコード，プロファイラ，シミュレーターを用意する．開発者は対象のソフトウェアを用意し，テスト作成者はテストコードを用意する．テスターがプロファイラとシミュレーターを用意する．

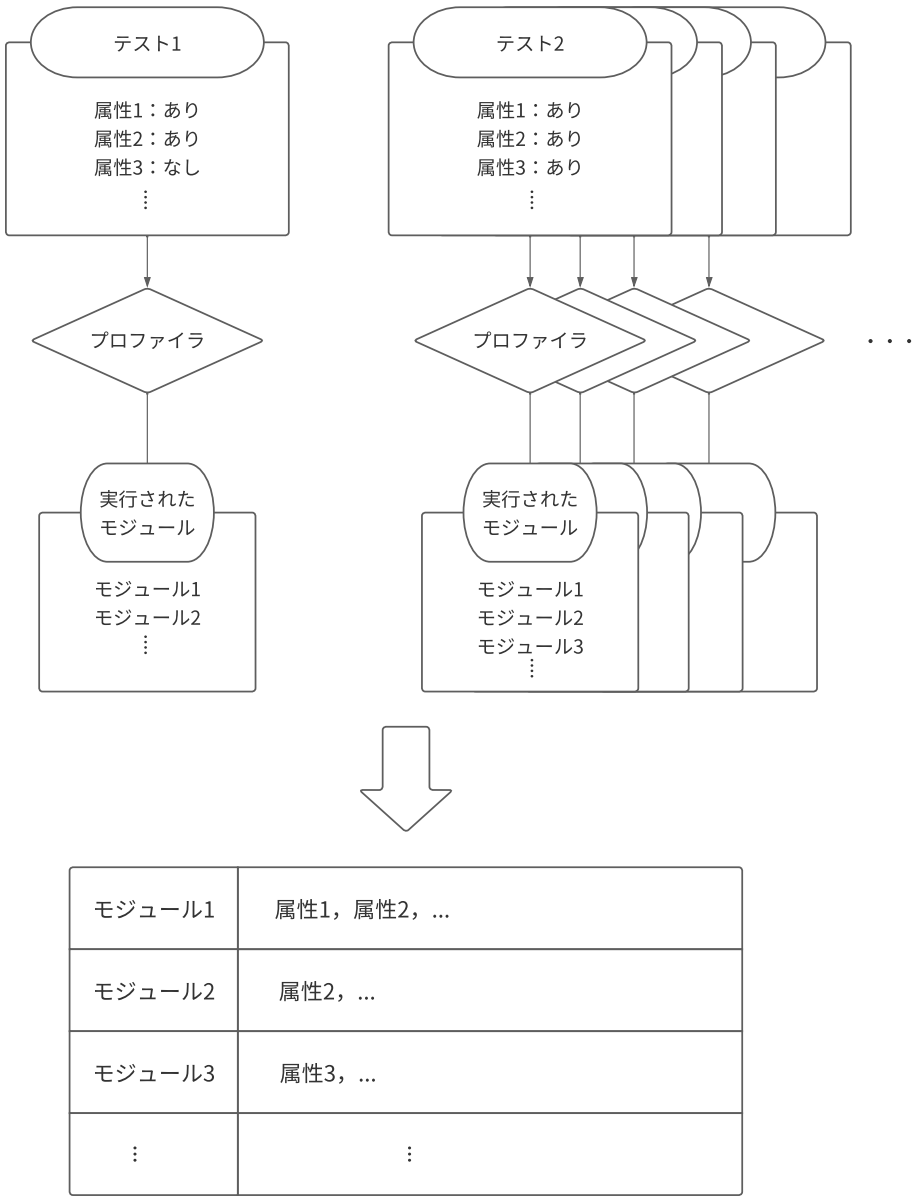

図 1　シミュレーションテストに対してプロファイラを実行し，実行されているモジュールを特定する

2. プロファイラでモニタリングしながらシミュレーションテストを実行する．
   対象のソフトウェアに含まれるモジュールを $m_i(i = 1, 2, 3, \cdots, m)$, 実行するテストコードを $t_j(j = 1, 2, 3, \cdots)$，テストコードの属性を $a_{jk}(k = 1, 2, 3, \cdots)$ とする．テスターはテストコード t_j を用いたシミュレーションテストに対してプロファイラを実行する．各モジュールについて関数の実行時間，呼び出し回数の情報やモジュール間の通信に関する情報がプロファイルとして出力される（図 1）．
3. 実行結果の集計と分析

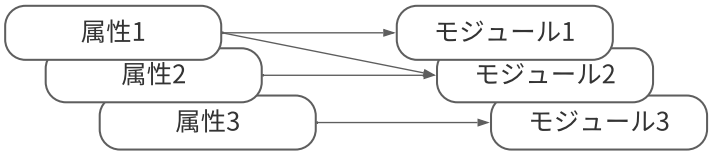

図 2　モジュールの実行とテストコードに含まれる属性の関連性を予測

開発者は，プロファイルからシミュレーションテストで実行されたモジュールを特定し，そのモジュールの集合を M_j とする．モジュール集合 M_j を比較してモジュール m_i の実行に関連しているテストコードの属性 a_{jk} を予測する（図 2）．

4. 属性に応じたシミュレーションテストの実行順序の決定
テスターはシミュレーションテストを行う際に，コードが書き換えられたモジュール m_k を実行させるようなテストコードの属性 a_{jk} を含むテストコード t_j から優先的に実行する（図 3）．

3.3　実装

Defect-prone を予測するモジュールは対象とするシステムにより異なる．サイバーフィジカルシステムであるロボットの自律運転や自動車の無人運転では，ROS がよく用いられる．ROS が用いられる場合は，ノードを Defect-prone モジュール予測の対象であるモジュールの代わりとすることもできる．また，ROS では開発した言語に対するプロファイラを用いることもできるが，ノード間の通信を記録する rosbag を活用することもできる．

3.3.1　実現方法の例

C 言語を用いたプログラムに対して，関数をモジュールとしてプロファイラ gprof を用いた際の手順を例に示す．

1. プログラムのビルド
オプション-pg オプションをつけ，gcc でビルドする．
```
$ gcc main.c -pg -o main
```

2. 実行シミュレーションテストを実行する．プロファイルが出力される．下記の例はシミュレーションテスト t_1 を実行しているとする．またシミュレーションテスト t_1 の属性を a_1 とする．
```
$ ./main test1
```

3. プロファイルの解析
gmon.out がプロファイルである．呼び出された関数のリスト，呼び出し回数，時間，また呼び出し関係が見られる．
```
$ gprof ./main gmon.out
```

実行結果 1 は出力の例である．シミュレーションテスト t_1 では関数 func1，func2 が実行されることがわかる．シミュレーションテスト t_1 で実行されたモジュール M_1 は func1，func2 であることがわかる．

4. モジュールとの対応付け
すべてのシミュレーションテスト $t_j(j = 1, 2, 3, \cdots)$ を実行し，実行したシミュレーションテストの属性 a_{jk} とモジュール m_i の関連を集計する．この情報からモジュール func1，func2 の属性 a_1 が対応することがわかる．

4　まとめと今後の課題

サイバーフィジカルシステムで用いられるシミュレーションテストは莫大なコストがかかる．シミュレーションテストの多くは合格となり，一部が不合格であることが多いため，欠陥を検出する可能性が高いシミュレーションテストから先に実行することでシミュレーションテストのコストを低減できる．そこで，シミュレーションテストの欠陥を検出する可能性を予測するメトリクスとして，Defect-prone モ

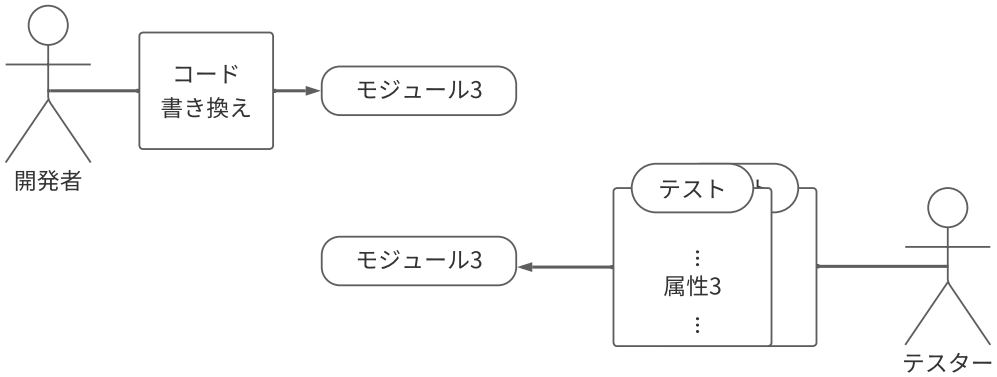

図 3　書き換えたモジュールに関連する属性を含むテストの優先的な実行

実行結果 1　プロファイル gmon.out の一例

```
1
2          Flat profile:
3
4      Each sample counts as 0.01 seconds.
5      %    cumulative self           self    total
6      time seconds    seconds calls s/call s/call name
7      33.86 15.52     15.52    1    15.52  15.52  func1
8      33.29 46.27     15.26    1    15.26  30.75  func2
9      0.07  46.30     0.03                        main
10
11
```

ジュール予測を提案した．具体的には，シミュレーションテストの実行に対してプロファイラを実行し，実行されたソフトウェア内のモジュールを特定する．リグレッションテストにおいて，コードが書き換えられたモジュールとシミュレーションテストで実行されるモジュールが対応するようなテストから実行する．提案したメトリクスを用いて，シミュレーションテストの実行に優先順位を付ける最適な手法の検討，実際のサイバーフィジカルシステムのプログラムを対象としてプロファイラの結果を分析し，どのような属性に対してどのようなモジュールが対応するか調査することが今後の課題である．

参 考 文 献

[1] Denaro, G., Morasca, S., and Pezze, M.: Deriving models of software fault-proneness, *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, 2002, pp. 361–368.

[2] Elsner, D., Wuersching, R., Schnappinger, M., Pretschner, A., Graber, M., Dammer, R., and Reimer, S.: Build System Aware Multi-language Regression Test Selection in Continuous Integration, 2022.

[3] Fenton, N. E. and Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system, *IEEE Transactions on Software engineering*, Vol. 26, No. 8(2000), pp. 797–814.

[4] Kalaichelvan, K. S., Allen, E. B., Goel, N., and Khoshgoftaar, T. M.: Early Quality Prediction: A Case Study in Telecommunications, *IEEE Software*, Vol. 13, No. 01(1996), pp. 65–71.

[5] Menzies, T., Greenwald, J., and Frank, A.: Data mining static code attributes to learn defect pre-

dictors, *IEEE transactions on software engineering*, Vol. 33, No. 1(2006), pp. 2–13.

[6] Porter, A. A. and Selby, R. W.: Empirically guided software development using metric-based classification trees, *IEEE software*, Vol. 7, No. 2(1990), pp. 46–54.