

研究進捗状況

2018年6月11日（月）

発表者 吉富 楓雅

“Silent Horrors”

はSilentを暴けるのか？

テストコードのバグ分類

- 2つに分類することができる[1]

- * Silent Horrors :

- ▶ テストコードに誤りがあるとプロダクトコードの誤りが見つからないことがある

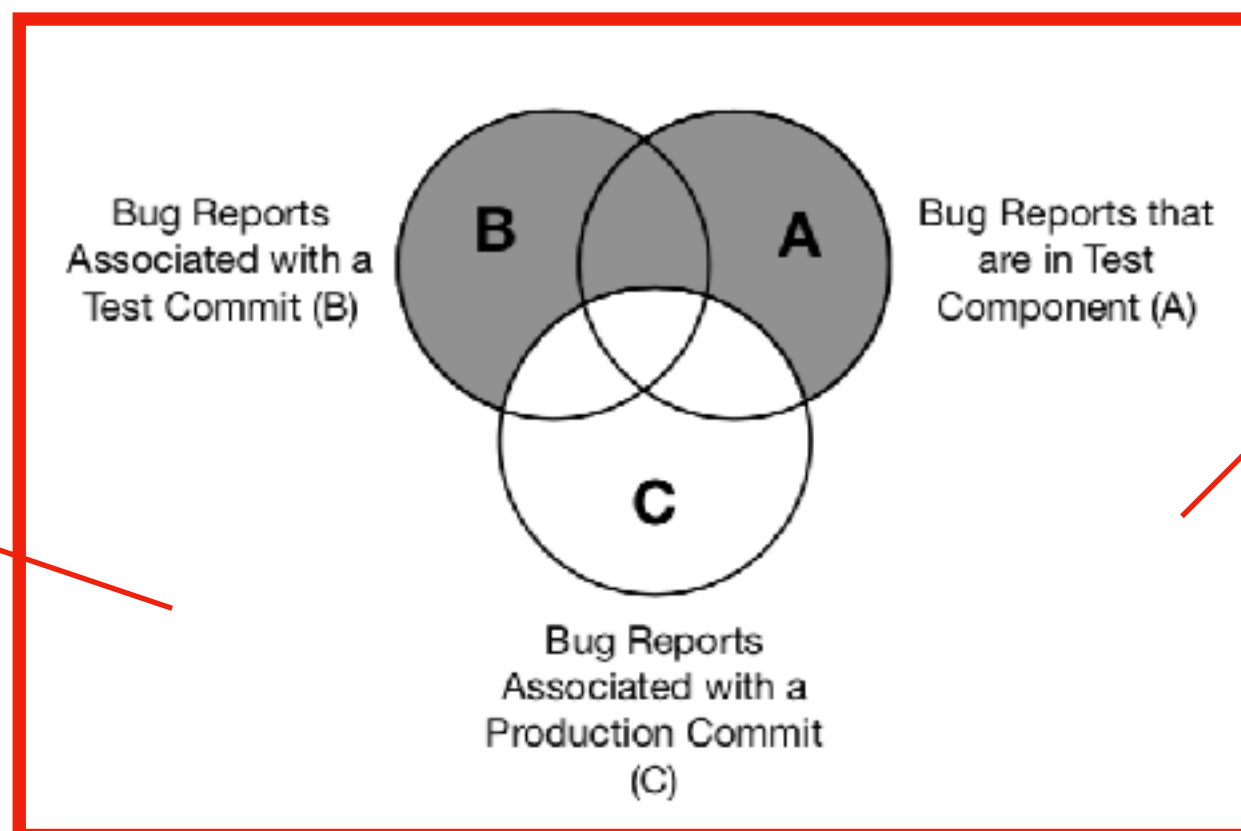
- * False Alarms :

- ▶ プロダクトコードに誤りがないがテストが失敗することがある

Silent Horrorsの抽出

バグ報告されていないものでもテストコードとプロダクトコードに現在でも気づかれていないバグ(Silent Horrors)が混入している可能性は高い

TravisCIでテスト
の
実行結果と紐づいて
いる
コミット
を発見する



バグ報告されておらず
テストコードと
プロダクトコードに
バグが混入している
可能性のある領域

Fig. 3: Bug reports collected from bug repositories and version control systems. $|(A \cup B) - C| = 5,556$ test bug reports in total ($|B - A - C| = 3,849$, $|A - C| = 1,707$).

DONE

JUnitの学習

- テストケースの作成方法と, それに基づいてテストコードを作成する方法を学習した
 - * カバレッジやエラー箇所も確認できた
- バグが混入したものを発見し, 修正する方法も学習した
 - * Eclipseのデバッグ機能も用いて修正を行うことができた

TravisCIとの連携



なぜMSの
傘下になったんだ…



- JUnitのテストをTravisCIで行う方法がわかった
 - * ビルドツール：Ant（とりあえず）
- Antは古いので今後はMavenやGradleなどのビルドツールに変更したい

※YouTubeにTravisCIを日本語で説明した動画を発見しました
（音質は悪いが結構わかりやすい）

<https://www.youtube.com/watch?v=85Z4ctZTYaU&t=424s>

Silent Horrorsの予測指標

- REOPENED(再修正)として登録されたものは, もともと Silent Horrors ということになるのではないか
 - * このケースは $(A \cup B) - C$ から発見されたものであるが, このような例を分析すれば Silent Horrors をコード全体の中に一般化して予測する指標になるのでは?
 - * 「An Empirical Study of Bugs in Test Code」では $(A \cup B) - C$ という具体解のみから発見したから全体の3%しか見つからなかったが, これを一般解にすればより多くの解を得られる可能性はある
 - * ただし, バグIDに紐づけられているものから発見された Silent Horrors が一般的であるかは定かでない(あくまで予想)

TODO

今後の予定

- Silent Horrorsにはどんなものがあるのかを調査
 - * 「An Empirical Study of Bugs in Test Code」の分析結果から、アサーションfail()が欠落しているケースが存在することが分かった
 - * 今日の野口君の発表から、クローンペアの片方がテストされていないケースが存在することがわかった
 - * 今日の岡坂君の発表から、機能性・保守性に問題があるケースが存在することがわかった
- DerbyのREOPENEDに類するものにタグづけられたものを探し分析する

REOPENEDの例

Derby / DERBY-3852

J2EEDataSourceTest may ignore some failures

Details

Type:	Bug	Status:	CLOSED
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	10.5.1.1	Fix Version/s:	10.6.3.1, 10.6.1.0
Component/s:	Test		
Labels:	None		
Issue & fix info:	Newcomer		

Description

I noticed that the method testConnectionErrorEvent() in J2EEDataSourceTest had many occurrences of code that was structured like this:

```
try
{ // do something with a connection }
catch (SQLException e)
{ assertSQLState("...", e); }
```

This code will only fail if an unexpected exception is thrown, not if the code in the try block fails to throw any exception at all.

It may be that we can fix this by adding a call to fail() at the end of each try block. It may also be the case that the code is written like this because it is expected to throw exception with the embedded driver and succeed with the client driver (or the other way around). If the latter is the case, this should be made explicit, for instance by writing it like this:

```
try
{ // do something with a connection
  assertFalse(usingEmbedded(), "should have failed with the embedded driver");
}
catch (SQLException e) {
  if (usingEmbedded())
  { throw e; }
```

try {}catch{}やfail()のような記述で
ミスが起こりやすい…?

Activity

Transition	Time In Source Status	Execution Times
Knut Anders Hatlen made transition - 01/Feb/10 09:43 OPEN → CLOSED	523d 19h 38m	1
Lily Wei made transition - 01/Jul/10 22:14 CLOSED → REOPENED	150d 12h 30m	1
Kathy Marsden made transition - 07/Jul/10 22:17 REOPENED → CLOSED	6d 3m	1

<https://issues.apache.org/jira/browse/DERBY-3852>

疑問点・不安等

- Derbyのテストをどのように行なっていくのか？
 - * Derbyのソースコードを読み込んで一つ一つテストすべきなのか？