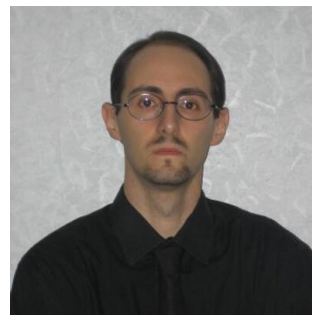
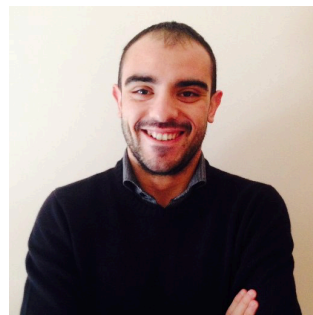


# An Empirical Investigation into the Nature of Test Smells



Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta,  
Rocco Oliveto, Andrea De Lucia, Denys Poshyvanyk

---

Proceeding [ASE 2016](#) Proceedings of the 31st IEEE/ACM International Conference on Automated  
Software Engineering      Pages 4-15

---

発表者 吉富 楓雅

---

# 目次

## 要約

test smellは不適切に設計されたテストとして定義されており、維持を困難にする可能性がある。しかしながらtest smellの識別とリファクタリングのための自動化ツールは存在しない。

本稿ではまず、19人の開発者が、対象としたテストにおいてtest smellの認識を調査する。

その結果、開発者はtest smellを認識せず、そのようなsmellを特定するための自動化されたツールが非常に必要であることを強調した。

最後に、この調査結果を使用して、test smellの検出とリファクタリングのための自動化ツールを構築する方法を提案する。

- 研究の背景と目的
- 開発者に対するtest smellの調査
- 提案手法 History miner
- リサーチクエスチョンと結果
- 結論と所感

# 研究の背景と目的

# 背景

- プロダクトコード生成の際, code smell[1]と呼ばれる劣悪な現象が発生することがある
  - \* code smellに対してはリファクタリングが必要
- code smellの自動抽出ツールが提案されている[2][3]
- テストコードにおいてもcode smellに類似した現象(test smell)が発生する

[1]M. Fowler. Refactoring: improving the design of existing code. Addison-Wesley, 1999.

[2]G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto. Automating extract class refactoring: An improved method and its evaluation. Empirical Softw. Engg., 19(6):1617–1664, Dec. 2014.

[3]N. Tsantalis and A. Chatzigeorgiou. Identification of move method refactoring opportunities. IEEE Transactions on Software Engineering, 35(3):347–367, 2009.

# code smell

- プロダクトコードに深刻な問題が存在することを示す兆候
  - \* Kent Beckが最初に提言
  - \* "Refactoring. Improving the Design of Existing Code"(Martin Fowler著)で取り入れられた
- code smellの例
  - \* 現在必要がないにも関わらず作成された複雑な抽象クラス
  - \* 同一クラスの複数メソッドに同じ式

# test smell

- van Deursenらはtest smellを11種類に分類[5]

| 番号       | 名称                    | 詳細                                                                   |
|----------|-----------------------|----------------------------------------------------------------------|
| Smell 1  | Mystery Guest         | テストが外部リソースを使用しているため、未知の値によってテストの理解が困難になる                             |
| Smell 2  | Resource Optimism     | テストが外部リソースの状態について前提を作成するため、リソースの状態に依存する非確定的な結果を生み出す                  |
| Smell 3  | Test Run War          | テストが、他の人によって使用されるリソースである(tmpファイルなど)ため、同時にテストを実行すると失敗する               |
| Smell 4  | General Fixture       | JUnitフレームワークでは、各テストメソッドの前にsetUpメソッドを記述する。しかしsetUpを汎用すると、可読性が低下する     |
| Smell 5  | Eager Test            | テストの対象となる、プロダクトコードのクラスのメソッドにおいて、複数のメソッドをチェックすることによって、テスト対象の理解が困難になる  |
| Smell 6  | Lazy Test             | 同一のフィクスチャが使用されているテストメソッドであるため、テストの理解とメンテナンスが困難になる                    |
| Smell 7  | Assertion Roulette    | 大量の説明のない例外処理によって、テストが失敗した際どの例外報告でエラーが発生したのか識別が困難になる                  |
| Smell 8  | Indirect Testing      | テストが、テスト中のオブジェクトと他のオブジェクトを介して間接的にやりとりするため、テストの保守とデバッグが困難になる          |
| Smell 9  | For Testers Only      | テストメソッドによってのみ使用されるメソッドが含まれているため、プロダクトコードのメンテナンスと理解が困難になる             |
| Smell 10 | Sensitive Equality    | 例外処理にtoStringメソッドを使用している場合、コンマ・引用符・空白などの比較で使用する文字列によってテストが失敗する可能性がある |
| Smell 11 | Test Code Duplication | 単体テスト内に含まれるコードクローンによって、コードクローンは保守性に悪影響を及ぼす                           |

# 問題点

- 実証研究により， test smellが保守性を損なう可能性があることが示されている
  - \* test smellのリファクタリング法が提案されている[4]
- しかし， 実際に関係者がtest smellをどのように認識しているのかは調査されていない

# 本研究の目的

- 開発者がtest smellにどのような認識を示すのかを調査
- test smellの検出とリファクタリングを行うために、自動化されたツールを構築

# 開発者に対する test smellの調査

# 調査に用いるtest smell

- van Deursenらのtest smellの分類[5]を利用

| 番号      | 名称                 | 詳細                                                                   |
|---------|--------------------|----------------------------------------------------------------------|
| Smell 1 | Mystery Guest      | テストが外部リソースを使用しているため、未知の値によってテストの理解が困難になる                             |
| Smell 2 | General Fixture    | JUnitフレームワークでは、各テストメソッドの前にsetUpメソッドを記述する。しかしsetUpを汎用に使用すると、可読性が低下する  |
| Smell 3 | Eager Test         | テストの対象となる、プロダクトコードのクラスのメソッドにおいて、複数のメソッドをチェックすることによって、テスト対象の理解が困難になる  |
| Smell 4 | Assertion Roulette | 大量の説明のない例外処理によって、テストが失敗した際どの例外報告でエラーが発生したのか識別が困難になる                  |
| Smell 5 | Sensitive Equality | 例外処理にtoStringメソッドを使用している場合、コンマ・引用符・空白などの比較で使用する文字列によってテストが失敗する可能性がある |

# 質問内容

1. このクラスに設計上の問題は存在しますか？

\* 1(全く思わない)～5(非常にそう思う)

2. 設計上の問題が存在する場合、具体的に問題内容を説明してください

3. なぜ設計上の問題が混入したのかを説明してください

4. このクラスはリファクタリングされるべきですか？

\* 1(全く思わない)～5(非常にそう思う)

5. あなたはどのようにリファクタリングしますか？

# 回答の集計

- 質問 1 と 4 について

- \* ほとんどの開発者はtest smellに気づかなかった

- 質問 3 について

- \* ほとんどの開発者は回答できなかった

| Question | Answer |   |   |   |   |
|----------|--------|---|---|---|---|
|          | 1      | 2 | 3 | 4 | 5 |
| #1       | 78     | 1 | 4 | 8 | 4 |
| #4       | 87     | 1 | 1 | 7 | 0 |

- \* 回答例

- ▶ 「テストコードを急いで記述していたため、レビューされなかったのだろう」

- ▶ 「このテストコードは2002年に導入されたもので、注目されていない」

- 結論：test smellについて認識している開発者はほとんどいない

# 提案手法

# History Miner

# データセット

- 190のソフトウェアプロジェクトを使用
  - \* Apache Software Foundation
  - \* Eclipse Foundation
- 190のプロジェクトは全てGitで管理されている

**Table 2: Ecosystems under analysis.**

| Ecosystem      | #Proj.     | #Classes   | KLOC     | #Commits       | Mean Story Length | Min-Max Story Length |
|----------------|------------|------------|----------|----------------|-------------------|----------------------|
| Apache         | 164        | 4-5,052    | 1-1,031  | 207,997        | 6                 | 1-15                 |
| Eclipse        | 26         | 142-16,700 | 26-2,610 | 264,119        | 10                | 1-13                 |
| <b>Overall</b> | <b>190</b> | -          | -        | <b>472,116</b> | <b>6</b>          | <b>1-15</b>          |

# test smellの抽出

## 1. 各プロジェクトのGitリポジトリをクローン

- \* javaファイルのみをコードファイルとし、それ以外は破棄

## 2. 各コミットについて、コードファイルをプロダクトとテストに分類

- \* junit.framework.TestCaseクラスを拡張するファイルは、テストクラスとして分類
- \* テストクラスとして分類しないクラスは全てプロダクトクラスとみなす

## 3. 各テストクラスについて、テストコードの依存性を分析

- \* 関連する(テストにより実行される)プロダクトクラスをリスト化する



# History Minerの出力

## テストファイル(Ftc)のログlogtc

- Ftc.name
  - \* テストファイル名
- ci.id
  - \* テストファイルを変更したi番目のコミットのハッシュ
- ci.time
  - \* i番目のコミットがリポジトリで実行された時のタイムスタンプ
- TStc
  - \* ci.timeによって特定されたtest smellのリスト
- PCtc
  - \* テストファイルと構文的に依存するプロダクトクラスのリスト

## プロダクトファイル(Fpc)のログlogpc

- Fpc.name
  - \* テストファイル (Ftc) 名
- ci.id
  - \* プロダクトファイルを変更したi番目のコミットのハッシュ
- ci.time
  - \* i番目のコミットがリポジトリで実行された時のタイムスタンプ
- CSpc
  - \* ci.timeによって特定されたcode smellのリスト

# test smellの分類

- 抽出されたtest smellを 5 種類に分類
  - \* 分類方法はBavotaらが提案した手法を使用[6]
- code smellの分類は, Mohaらが定義した規則に基づくsmell検出ツールDECORを実装[7]

[6]G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012, pages 56–65, 2012.

[7]N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur. DECOR: A method for the specification and detection of code and design smells. IEEE Trans. on Software Engineering, 36(1):20–36, 2010.

# 調査に用いるtest smell(再掲)

| 番号      | 名称                     | 詳細                                                                   |
|---------|------------------------|----------------------------------------------------------------------|
| Smell 1 | Mystery Guest(MG)      | 外部リソースを使用することによって発生する。外部リソースに対する理解が足りない場合、テストの安定性を維持することが困難になる       |
| Smell 2 | General Fixture(GF)    | JUnitフレームワークでは、各テストメソッドの前にsetUpメソッドを記述する。しかしsetUpを汎用に使用すると、可読性が低下する  |
| Smell 3 | Eager Test(ET)         | テストの対象となる、プロダクトコードのクラスのメソッドにおいて、複数のメソッドをチェックすることによって、テスト対象の理解が困難になる  |
| Smell 4 | Assertion Roulette(AR) | 大量の説明のない例外処理によって、テストが失敗した際どの例外報告でエラーが発生したのか識別が困難になる                  |
| Smell 5 | Sensitive Equality(SE) | 例外処理にtoStringメソッドを使用している場合、コンマ・引用符・空白などの比較で使用する文字列によってテストが失敗する可能性がある |

# 調査に用いるcode smell

| 番号      | 名称                                  | 詳細                          |
|---------|-------------------------------------|-----------------------------|
| Smell 1 | Blob Class(BC)                      | コードの量が多く複雑なクラス              |
| Smell 2 | Class Data Should be Private (CDSP) | privateにすべきクラスがpublicとなっている |
| Smell 3 | Complex Class(CC)                   | 簡潔な設計で良いクラスが、過剰に複雑な設計になっている |
| Smell 4 | Functional Decomposition(FD)        | 継承やポリモーフィズムがあまり使用されていないクラス  |
| Smell 5 | Spaghetti Code(SC)                  | 開発したプログラマ以外に解読不可能なプロダクトコード  |

# リサーチクエスチョン と結果

# リサーチクエスチョン

---

- RQ1 : test smellが混入するのはいつか？
- RQ2 : test smellはどの程度の期間残存するのか？
- RQ3 : test smellは, プロダクトコードに影響を及ぼすcode smellと関連するか？

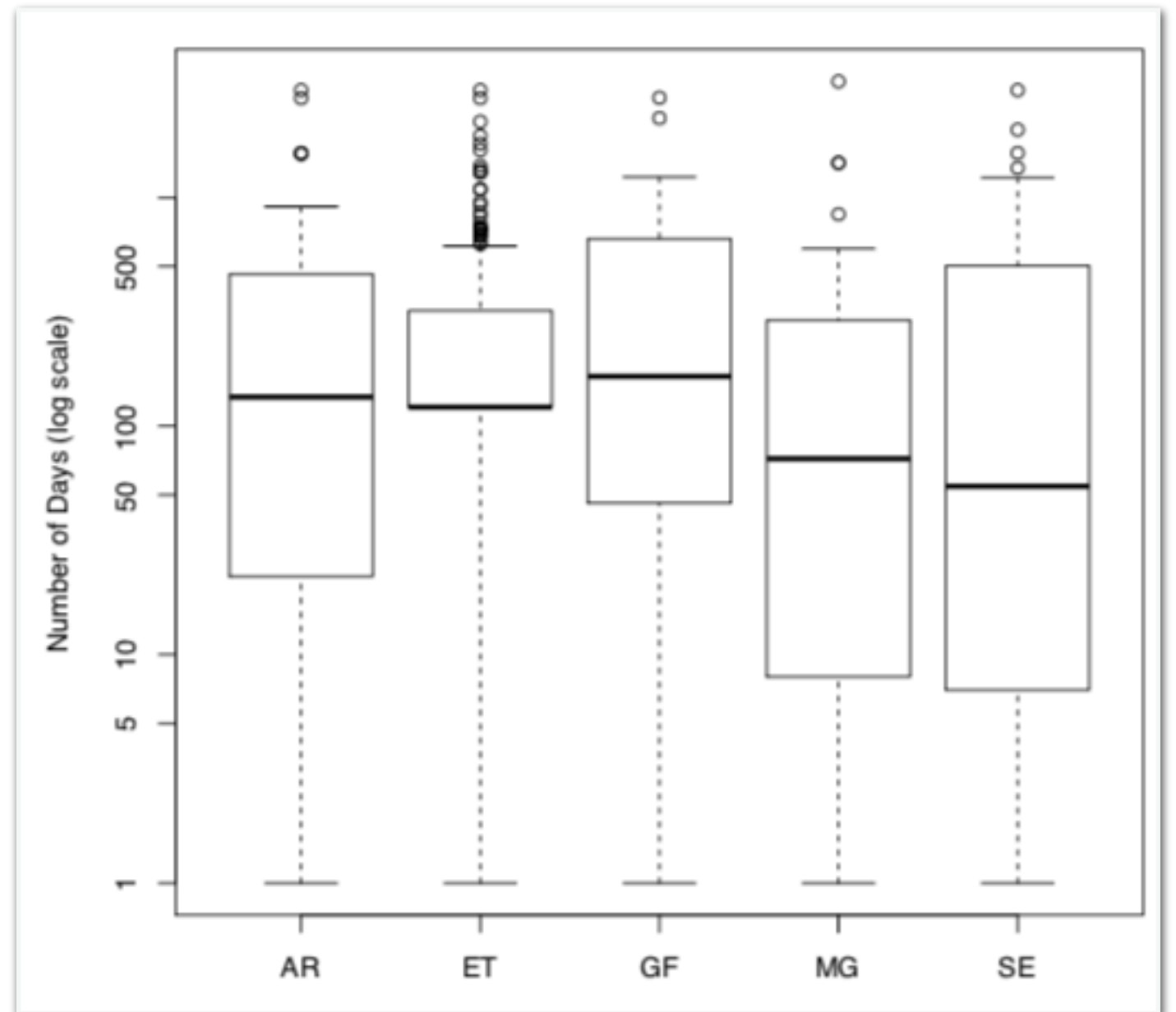
# RQ1

| Ecosystem | Smell | Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max | % Smelly <sub>1st</sub> |
|-----------|-------|------|---------|--------|------|---------|-----|-------------------------|
| Apache    | AR    | 0    | 0       | 0      | 0.13 | 0       | 58  | 96                      |
|           | ET    | 0    | 0       | 0      | 0.12 | 0       | 245 | 97                      |
|           | GF    | 0    | 0       | 0      | 0.27 | 0       | 23  | 93                      |
|           | MG    | 0    | 0       | 0      | 0.49 | 0       | 245 | 93                      |
|           | SE    | 0    | 0       | 0      | 0.59 | 0       | 62  | 90                      |
| Eclipse   | AR    | 0    | 0       | 0      | 0.37 | 0       | 99  | 96                      |
|           | ET    | 0    | 0       | 0      | 0.13 | 0       | 26  | 98                      |
|           | GF    | 0    | 0       | 0      | 0.41 | 0       | 99  | 94                      |
|           | MG    | 0    | 0       | 0      | 1.13 | 0       | 56  | 91                      |
|           | SE    | 0    | 0       | 0      | 1.79 | 0       | 102 | 88                      |

- test smellを混入させているテストクラスは、最初にコミットした際にすでにtest smellを混入してしまっている

# RQ2

- test smell導入から100日後に平均して除去される
- test smellは長期間除去されずに残存する
- 各test smellの種類間に有意な差異は見出されない



Apacheプロジェクトを例に

横軸：test smellの種類

縦軸：除去に要した日数

# RQ3

| id | lhs                | rhs     | support | confidence | lift |
|----|--------------------|---------|---------|------------|------|
| 1  | cleanTC            | cleanPC | 0.05    | 0.96       | 1.24 |
| 2  | AR, SC             | BC      | 0.05    | 0.99       | 4.74 |
| 3  | CDSBP              | ET      | 0.06    | 0.93       | 1.17 |
| 4  | SC                 | ET      | 0.09    | 0.97       | 1.22 |
| 5  | BC                 | ET      | 0.20    | 0.95       | 1.18 |
| 6  | CDSBP, SC          | ET      | 0.02    | 0.98       | 1.22 |
| 7  | AR, CDSBP          | ET      | 0.07    | 0.94       | 1.17 |
| 8  | CDSBP, GF          | BC      | 0.02    | 0.77       | 3.66 |
| 9  | CDSBP, MG          | AR      | 0.01    | 0.72       | 1.14 |
| 10 | AR, CDSBP          | ET      | 0.04    | 0.94       | 1.17 |
| 11 | AR, SC             | CC      | 0.04    | 0.85       | 7.93 |
| 12 | GF, SE             | AR      | 0.02    | 0.76       | 1.22 |
| 13 | AR, CC, ET, GF, SC | BC      | 0.02    | 1.00       | 4.75 |
| 14 | GF, SE             | ET      | 0.02    | 0.87       | 1.08 |
| 15 | MG, SE             | AR      | 0.01    | 0.72       | 1.15 |
| 16 | BC, SE             | AR      | 0.02    | 0.79       | 1.25 |
| 17 | BC, SE             | ET      | 0.02    | 0.97       | 1.21 |

# 結論と所感

# 結論と所感

- 結論

- \* test smellを認識している開発者は少ない
- \* テストクラスを作成した後にtest smellが混入する可能性が高い
- \* test smellは非常に高い生存性を持っている

- 所感

- \* code smellとtest smellの関連づけについて分析を行っており, smellの影響度が理解しやすかった