Grigore Mihai-Catalin
Grupa 342

Raport - Test Smell Detection Tools: A Systematic Mapping Study

Introducere

În acest raport am sumarizat cele mai importante idei din articolul "Test Smell Detection Tools: A Systematic Mapping Study", publicat în ACM Digital Library pe 3 mai 2021.

Concepte

Test Smells = teste automate de slabă calitate care predispun codul la defecte şi la dificultăţi de mentenanţă

Test Smell Detection Tool = software care detectează test smells într-un proiect

Obiectivele Studiului

O1 - Formarea unei liste de Test Smell Detection Tools

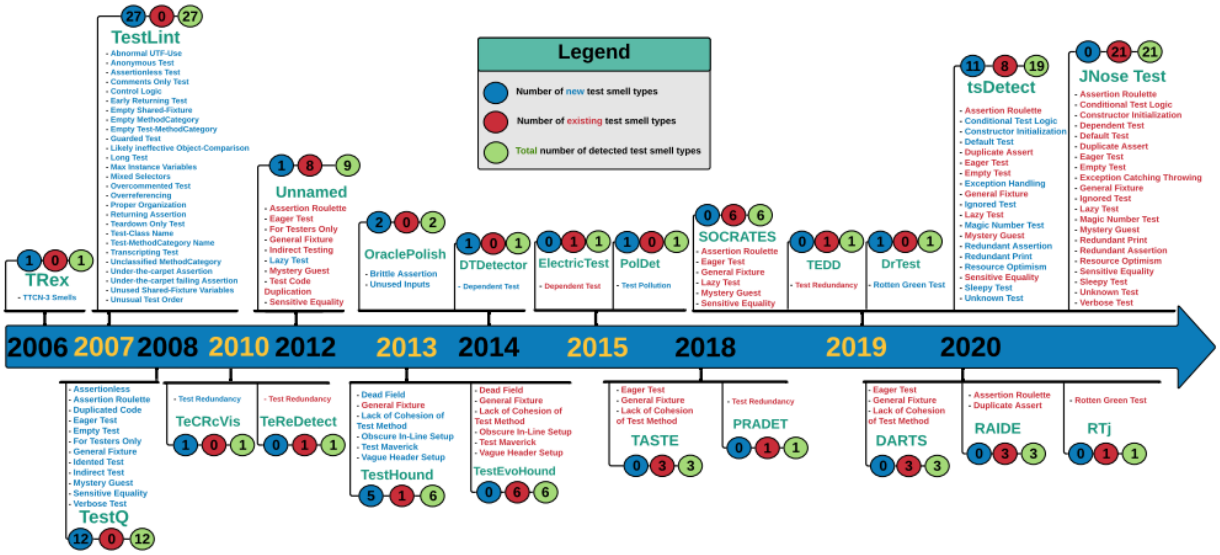
Metode folosite:

- căutarea în metadatele articolelor cu următorul pattern

```
Title:("tool*" OR "detect*" OR "test smell" OR
"test smells") AND Abstract:("test smell" OR "test
smells" OR "test code" OR "unit test smell")
```

- analiza individuală a articolelor pentru a asigura un standard de calitate

Grigore Mihai-Catalin
Grupa 342



O2 - Determinarea caracteristicilor principale ale acestor tool-uri

Metode folosite:

- comparaţii din punct de vedere al:

a) limbajului de programare
b) framework-ului de testare
c) corectitudine
d) tehnică de detecţie
e) interfaţă
f) usage guide disponibil
g) popularitate în comunitatea de research
h) website-ul tool-ului

Grigore Mihai-Catalin
Grupa 342

Table 6: Characteristics of test smell detection tools.

Tool	Programming Language Implemented	Language Analyzed	Supported Test Framework	Correctness	Detection Technique	Interface	Usage Guide	Adoption in Studies	Tool Website
DARTS ‡ [53]	Java	Java	JUnit	F-Measure: 62%-76%	Information Retrieval	IntelliJ plugin	Yes	–	[3]
DrTest [36]	Smalltalk	Pharo ▽	SUnit	UNK	Rule Dynamic Tainting	Pharo plugin	Yes	–	[4]
DTDetector ⋆ ◇ [91]	Java	Java	JUnit	UNK	Dynamic Tainting	Command-line	Yes	–	[5]
ElectricTest [29]	Java	Java	JUnit	UNK	Dynamic Tainting	Command-line	No	–	UNK
JNose Test [85]	Java	Java	JUnit	UNK	Rule	Local web application	Yes	[86, 87]	[6]
OraclePolish ⋆ [48]	Java	Java	JUnit	UNK	Dynamic Tainting	Command-line	Yes	–	[7]
PolDet [47]	Java	Java	JUnit	UNK	Dynamic Tainting	UNK	No	–	UNK
PraDeT [39]	Java	Java	JUnit	UNK	Dynamic Tainting	Command-line	Yes	–	[10]
RAIDE ‡ [71]	Java	Java	JUnit	UNK	Rule	Eclipse plugin	Yes	–	[11]
RTj ‡ [55]	Java	Java	JUnit	UNK	Rule Dynamic Tainting	Command-line	Yes	–	[12]
SoCRATES [35]	Scala	Scala	ScalaTest	Precision: 98.94% Recall: 89.59%	Rule	IntelliJ plugin	Yes	[34]	[13]
Taste [60]	UNK	Java	JUnit	Precision: 57%-75% Recall: 60%-80%	Information Retrieval	UNK	No	[63]	UNK
TeCReVis ⋆ [50]	Java	Java	JUnit	UNK	Metrics Dynamic Tainting	Eclipse plugin †	Yes	–	[15]
TEDD [30]	Java	Java	JUnit	Precision: 80% Recall: 94%	Information Retrieval	Command-line	Yes	[31]	[14]
TeReDetect ⋆ [51]	Java	Java	JUnit	UNK	Metrics Dynamic Tainting	Eclipse plugin †	Yes	–	[15]
TestEvoHound [46]	Java	Java	JUnit, TestNG	UNK	Metrics	UNK	No	–	UNK
TestHound ‡ ⋆ [45]	Java	Java	JUnit, TestNG	UNK	Metrics	Desktop application	No	–	[16]
TestLint ⋆ [70]	Smalltalk	Smalltalk	Sunit	UNK	Rule Dynamic Tainting	UNK	Yes	–	[17]
TestQ ⋆ [32]	Python	C++, Java	CppUnit, JUnit, Qtest	UNK	Metrics	Desktop application	Yes	–	[18]
TRex ‡ § ⋆ [25]	Java	Java	TTCN-3	UNK	Rule	Eclipse plugin	Yes	[57, 58, 88, 90]	[19]
tsDetect [65]	Java	Java	JUnit	Precision: 85%-100% Recall: 90%-100%	Rule	Command-line	Yes	[49, 64, 72, 79] [38, 61, 66, 77]	[20]
Unnamed [27]	UNK	Java	JUnit	Precision: 88% Recall: 100%	Rule	Command-line	No	[28, 59, 80, 81] [43, 44, 61, 69, 78]	UNK

Concepte:

- tehnici de detecţie

a) metrici – sunt măsurate diverse caracteristici structurale şi semantice, iar dacă se depăşeşte un anumit prag, se consideră că testul “suferă” de un smell. Codul sursă este convertit într-un Abstract Syntax Tree (AST), care este folosit pentru analiza metricilor.
b) Reguli / euristici – diverse pattern-uri în cod e.g. assertion roulette
c) Contaminare dinamică / dynamic tainting – monitorizează codul sursă în timpul execuţiei. Foloseşte 2 paşi: execută codul sursă cu diverse input-uri şi apoi decide dacă unele execuţii sunt afectate.
d) Extragere de informaţii / information retrieval – normalizarea codului testelor (multiple procesări precum stemming, descompunerea numelor identificatorilor, ştergerea keyword-urilor şi a stop words). Într-un final se aplică algoritmi de machine learning pentru a extrage feature-uri care disting între clase (reprezentate de test smells).

Concluzii:

- Este importantă standardizarea numelor / definiţiilor pentru test smells
- Este necesar un suport îmbunătăţit pentru alte limbaje de programare în afară de Java
- Putem îmbunătăţi tool-urile existente în loc să creăm unele de la zero
- Este utilă publicarea măsurătorilor de corectitudine a tool-urilor
- Este utilă nu doar detectarea, ci şi refactorizarea interactivă

Grigore Mihai-Catalin
Grupa 342