DO Automatically Generated Unit Tests Find Real Faults?

—

Ερρίκος Καλτσόπουλος

Ηρακλής Θεοφανίδης

Εισαγωγη(½)

Το unit testing είναι μια συνήθης πρακτική στον oop για να βρίσκουμε λάθη στον κώδικα μας.

Ωστόσο γράφοντας tests πολλές φορές μπορεί να είναι κουραστικό και για να γράψουμε καλά unit tests μπορεί να αποδειχθεί πιο πολύ τέχνη παρά επιστήμη.

Εισαγωγη (2/2)

Για την υποστήριξη των developers στο Unit Testing, οι ερευνητές έχουν ερευνήσει διάφορες πτυχές στα αυτόματα unit test.

Τα οποία φαίνεται να καλύπτουν αρκετί μέρος του κώδικα και να βρίσκουν αρκετά λάθη.

Αλλά όμως είναι αρκετά αποτελεσματικά?

Μεθοδολογια

Για να απαντήσουμε σε αυτο ερωτημα σχεδιαστηκε το παρακατω πειραμα:

- Dataset:
  Defects4j(357)
- Projects:
  Chart(26)
  Closure(133)
  math(106)
  Lang(65)
  Time(27)

Εργαλεια για αυτοματο Unit testing

Τα εργαλεια που χρησιμοποιηθηκαν ειναι τα εξης:

- RANDOOP
- EVOSUITE
- AGITARONE

Διαδικασια του πειραματος

- Test Generation:Randoop και Evosuite μπορούν να παράγουν διαφορετικά αποτελέσματα μετά από κάθε πέρασμα για αυτό το λόγο εκτελέστηκαν 10 διαφορετικά τεστ.
- Flaky Tests:όταν ένα τεστ αποτυγχάνει,αλλά αν προσπαθήσεις αρκετές φορές μπορεί
  να περάσει.
- False Positives:Όταν δεν περνάει το τεστ αλλά στην πραγματικότητα δεν υπάρχει κάποιο λάθος και η μέθοδος π.χ λειτουργεί κανονικά.
- Fault Detection
- Coverage Analysis:Για να μελετηθεί κατα πόσο η κάλυψη όλου του κώδικα κάθε κλάσης συμβάλλει στην εύρεση λαθών.

Διαδικασία του πειράματος

Project	Tool	Compilable	Tests	Flaky	False Pos.	Coverage	Max Bugs	Avg. Bugs	Assertion	Exception	Timeout
Chart	AGITARONE	100.0%	131.2	0.2%	30.6%	84.7%	17	17.0	10.0	11.0	0.0
	EVOSUITE	100.0%	45.9	3.5%	0.0%	68.1%	18	9.7	5.4	5.2	0.3
	RANDOOP	100.0%	4874.9	36.8%	0.0%	54.8%	18	14.1	7.5	9.1	0.0
	Manual	100.0%	230.6	0.0%	0.0%	70.5%	26	26.0	17.0	12.0	0.0
Closure	AGITARONE	100.0%	199.4	0.4%	79.3%	79.1%	25	25.0	16.0	10.0	0.0
	EVOSUITE	100.0%	34.9	1.7%	0.0%	34.5%	27	11.8	10.5	1.4	0.0
	RANDOOP	98.4%	5518.4	19.8%	15.8%	9.8%	9	2.2	0.5	1.7	0.0
	Manual	100.0%	3511.1	0.0%	0.0%	90.9%	133	133.0	103.0	42.0	0.0
Lang	AGITARONE	100.0%	127.7	1.0%	23.5%	50.9%	22	22.0	10.0	14.0	0.0
	EVOSUITE	79.5%	48.6	5.4%	0.0%	55.4%	18	9.2	5.5	3.3	0.9
	RANDOOP	68.3%	11450.7	5.7%	0.0%	50.7%	10	7.0	1.7	6.3	0.0
	Manual	100.0%	169.2	0.0%	0.0%	91.4%	65	65.0	31.0	36.0	0.0
Math	AGITARONE	100.0%	105.8	0.1%	8.9%	83.5%	53	53.0	34.0	25.0	0.0
	EVOSUITE	99.8%	29.7	0.2%	0.0%	77.9%	66	42.9	26.1	17.7	0.3
	RANDOOP	97.8%	7371.4	15.6%	0.0%	43.4%	41	26.0	17.8	10.8	0.0
	Manual	100.0%	167.8	0.0%	0.0%	91.1%	106	106.0	76.0	31.0	0.0
Time	AGITARONE	100.0%	187.2	3.3%	30.9%	86.7%	13	13.0	10.0	8.0	0.0
	EVOSUITE	100.0%	58.0	2.8%	0.0%	86.7%	16	8.5	4.9	4.0	0.0
	RANDOOP	81.1%	2807.1	25.3%	0.0%	43.0%	15	4.5	3.8	1.1	0.0
	Manual	100.0%	2532.7	0.0%	0.0%	91.8%	27	27.0	13.0	17.0	0.0

Παρατηρήσεις Αποτελεσμάτων(1/2)

- Το EvoSuite και Randoop δημιούργησαν test τα οποία δεν γινόταν compile.
- AgitarOne είχε τη χαμηλότερη αναλογία flaky test με μέγιστο 3.3% για το Project time
- Randoop ήταν ανάμεσα σε 5,7%-36,8% flaky tests
- Το Evosuite και το Randoop δυσκολεύτηκαν να πετύχουν υψηλή κάλυψη κώδικα στο project Closure λόγω του ότι είχαν πολλές private μεθοδους
- Απο τα τρία εργαλεια το Agiratone πετυχε τον υψηλότερο ποσοστο κάλυψης κώδικα.

Παρατηρήσεις Αποτελεσμάτων(2/2)

1)Τα generated test suites βρηκαν 199 απο τα 357 σφάλματα(55,7%),αλλά κανένα απο τα τρία εργαλεία δεν ξεπέρασε το 40,6 %.

2) Μόνο το 19 % όλων των εκτελέσεων ανίχνευσε ένα σφάλμα.

Πιό συγκεγκριμένα

Evosuite = 145 sfalamta

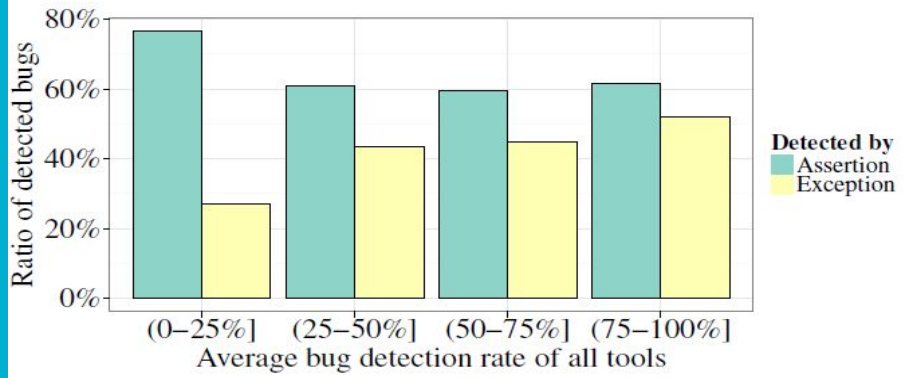
Agitarone =130

Randoop = 93

Πως βρεθηκαν τα σφαλματα ?

- Βρέθηκαν περισσότερα σφάλματα με assertions(146).
- Από την άλλη 109 βρέθηκαν με Exceptions.
- Και 56 και με τα δυο.

Project	Tool	Assertions	Exceptions	Both
Chart	AGITARONE	64.3%	55.6%	100.0%
	EVOSUITE	57.1%	77.8%	100.0%
	RANDOOP	57.1%	77.8%	100.0%
Closure	AGITARONE	18.7%	16.7%	25.0%
	EVOSUITE	17.6%	30.0%	16.7%
	RANDOOP	3.3%	16.7%	8.3%
Lang	AGITARONE	31.0%	35.3%	50.0%
	EVOSUITE	20.7%	29.4%	100.0%
	RANDOOP	6.9%	23.5%	0.0%
Math	AGITARONE	42.7%	70.0%	0.0%
	EVOSUITE	56.0%	80.0%	0.0%
	RANDOOP	34.7%	50.0%	0.0%
Time	AGITARONE	30.0%	71.4%	0.0%
	EVOSUITE	80.0%	42.9%	66.7%
	RANDOOP	40.0%	64.3%	66.7%



Προβλήματα που εμποδίζουν τα test generation tools(1/2)

Υπαρχουν 4 προβλήματα που εμποδίζουν τα test generation tools να επιτύχουν τόσο υψηλή κάλυψη κώδικα.

α)Creation of Complex Objects

b)String Optimization

c)Complex Conditions

d)private methods/fields

Προβλήματα που εμποδίζουν τα test generation tools(2/2)

Τα Flaky Tests προκαλούνται συχνά από

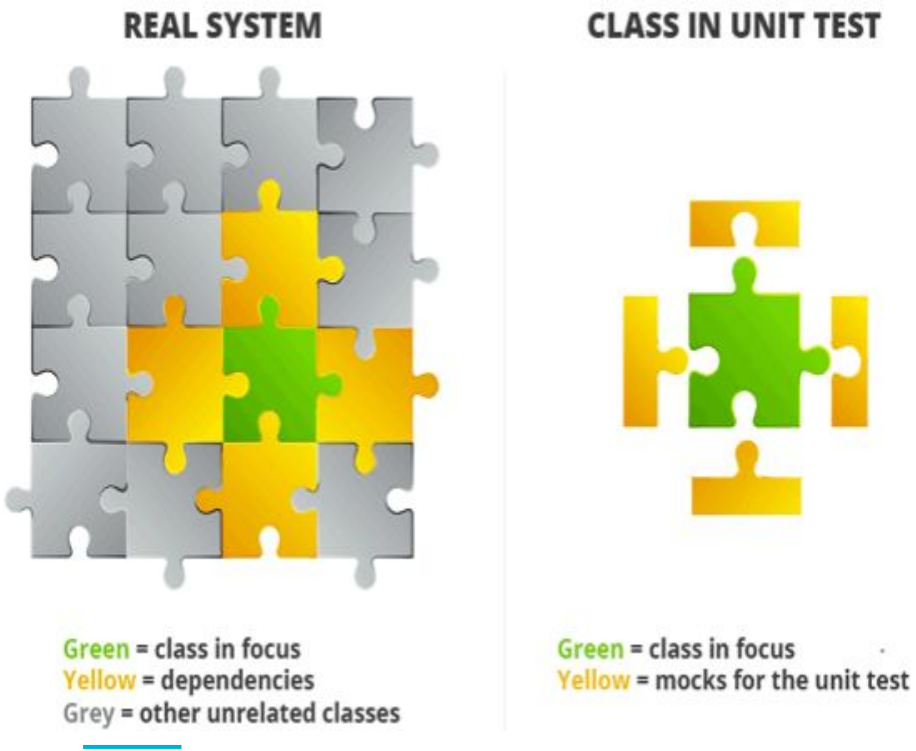
- Environment Dependencies όπου συνήθως είναι:
  - η τρέχουσα ώρα και η ημερομηνία του συστήματος.
- Static δηλώσεις.

False positives:

- Για να μειωθούν τα false positives και να αυξηθει η καλυψη του κωδικα πρέπει να έχει πρόσβαση στα private methods/fields.

AgitarOne’s mocking

Ένα αντικείμενο το οποίο είναι υπό test ενδεχομένως να έχει εξαρτήσεις σε άλλα (συνθετα)αντικείμενα. Για να απομονώσουμε την συμπεριφορά του αντικειμένου πού θέλουμε να αντικαταστήσουμε τα άλλα αντικείμενα προσομοιώνουν την συμπεριφορα των αληθινων αντικειμενων. Αυτό είναι χρήσιμο εάν τα πραγματικά αντικείμενα δεν μπορούν να ενσωματωθούν στο unit test.Εν συντομία, Το mocking δημιουργεί αντικείμενα τα οποία προσομοιώνουν την συμπεριφορά των αληθινών αντικειμένων.



Συνοψίζοντας

1. Βρέθηκαν το 55,7 % των λαθών αλλά κανένα εργαλείο δεν πέτυχε από μόνο του πάνω από 40% των λαθών.
2. Το 63,3% των οποίων δεν βρέθηκαν , καλύφθηκαν αυτόματα από το autommated generated test.
3. Το 16,2 % των λαθών δεν εκτελέστηκαν απο τα tests,
4. Το 15,2 % από όλα τα Test ήταν Flaky

ΤΕΛΟΣ