Προτεραιοποίηση Περιπτώσεων Ελέγχου σε Συστήματα Λογισμικού

Γιώργος Βλαχάβας

Πανεπιστήμιο Μακεδονίας
Πρόγραμμα Μεταπτυχιακών Σπουδών
Τμήματος Εφαρμοσμένης Πληροφορικής

21 Ιουνίου 2019

Σκοπός

Σκοπός της παρούσας εργασίας είναι η διερεύνηση της επίδρασης διαφορετικών τεχνικών προτεραιοποίησης στο ρυθμό ανίχνευσης σφαλμάτων έργων λογισμικού υλοποιημένων με τη χρήση της γλώσσας Python.

- Ιδανικά, έλεγχοι παλινδρόμησης θα πρέπει να εκτελούνται με κάθε αλλαγή στον κώδικα
- Πλήρης εκτέλεση όλων των περιπτώσεων ελέγχου απαιτεί συχνά χρόνο που δεν είναι διαθέσιμος
- Απαιτούνται τεχνικές αποδοτικής προτεραιοποίησης περιπτώσεων ελέγχου ώστε τα σφάλματα να εμφανίζονται το συντομότερο δυνατό

Τεχνικές για την αποδοτικότερη διενέργεια ελέγχων

- Επιλογή των περιπτώσεων ελέγχου (regression test selection)
- Ελαχιστοποίηση της σουίτας ελέγχου (test case elimination)
- Προτεραιοποίηση των περιπτώσεων ελέγχου (test case prioritization)

Προτεραιοποίηση περιπτώσεων ελέγχου

«Η Προτεραιοποίηση Περιπτώσεων Ελέγχου είναι το πρόβλημα της διάταξης των Περιπτώσεων Ελέγχου που περιλαμβάνονται σε μία Σουίτα Περιπτώσεων Ελέγχου, με στόχο τη μεγιστοποίηση κάποιου κριτηρίου, όπως ο ρυθμός ανίχνευσης σφαλμάτων των περιπτώσεων ελέγχου. Στην περίπτωση που η εκτέλεση του συνόλου της Σουίτας Περιπτώσεων Ελέγχου διακοπεί ή σταματήσει για οποιοδήποτε λόγο, οι σημαντικότερες περιπτώσεις ελέγχου (όσον αφορά στο κριτήριο που χρησιμοποιείται), θα έχουν εκτελεστεί πρώτες».

Κριτήριο προτεραιοποίησης

Κάλυψη κώδικα: περιγράφει το βαθμό στον οποίο εκτελείται ο πηγαίος κώδικας ενός προγράμματος, όταν εκτελείται μια σουίτα ελέγχου πάνω στον ίδιο κώδικα

- Στατική ανάλυση
- Δυναμική ανάλυση

- Ολική κάλυψη
- Επιπρόσθετη κάλυψη

Παράδειγμα εντοπισμού σφαλμάτων σε σουίτα πέντε περιπτώσεων ελέγχου

Περίπτωση	Σφάλμα									
Ελέγχου	1	2	3	4	5	6	7	8	9	10
A	✓				✓					
B	✓				✓	✓	✓			
C	✓	✓	✓	✓	✓	✓	✓			
D					✓					
E								✓	✓	✓

Αξιολόγηση εκτέλεσης διατάξεων Περιπτώσεων Ελέγχου

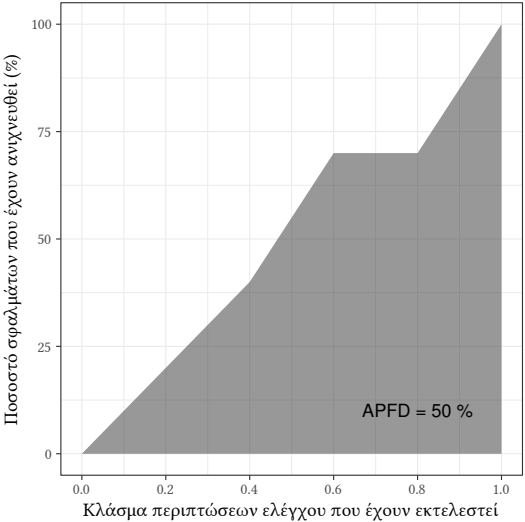
- APFD: Average Percentage of Faults Detected

$$APFD = 100 \times \left(1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n}\right)$$

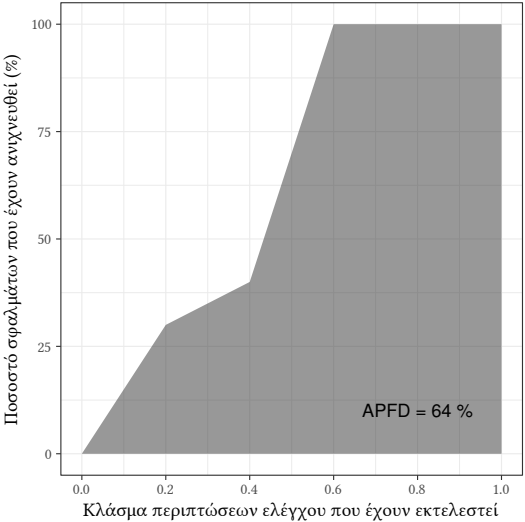
όπου:

- n, ο αριθμός των περιπτώσεων ελέγχου
- m, ο αριθμός των σφαλμάτων που εντοπίστηκαν
- TF_i, ο αριθμός των περιπτώσεων ελέγχου που πρέπει να εκτελεστούν ώστε να εντοπιστεί το i-στο σφάλμα.

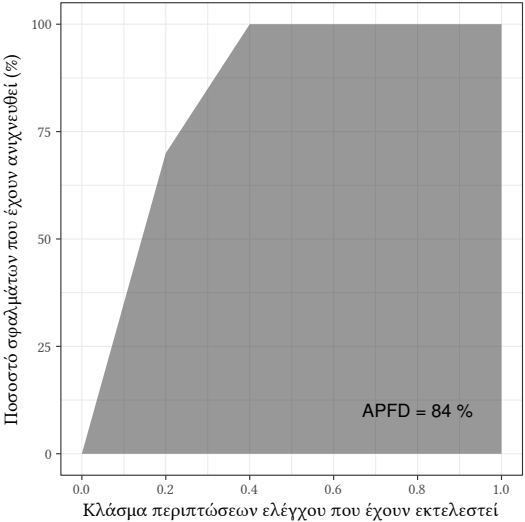
Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου [A, B, C, D, E]



Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου [E, D, C, B, A]



Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου [C, E, B, A, D]



Τεχνικές προτεραιοποίησης περιπτώσεων ελέγχου

Κωδικός	Τίτλος	Περιγραφή
ΤΠ1	Optimal	Βέλτιστος ρυθμός ανίχνευσης σφαλμάτων
ΤΠ2	Untreated	Αρχική σειρά εκτέλεσης
ΤΠ3	Random	Τυχαία σειρά εκτέλεσης
ΤΠ4	Inverse	Αντίστροφη σειρά εκτέλεσης
ΤΠ5	File-total	Ολική κάλυψη σε επίπεδο αρχείου
ΤΠ6	File-addtl	Επιπρόσθετη κάλυψη σε επίπεδο αρχείου
ΤΠ7	Method-total	Ολική κάλυψη σε επίπεδο συνάρτησης
ΤΠ8	Method-addtl	Επιπρόσθετη κάλυψη σε επίπεδο συνάρτησης
ΤΠ9	Branch-total	Ολική κάλυψη σε επίπεδο διακλάδωσης
ΤΠ10	Branch-addtl	Επιπρόσθετη κάλυψη σε επίπεδο διακλάδωσης
ΤΠ11	Statement-total	Ολική κάλυψη σε επίπεδο εντολής
ΤΠ12	Statement-addtl	Επιπρόσθετη κάλυψη σε επίπεδο εντολής

Σφάλματα

Δημιουργήθηκαν τεχνητά σφάλματα στις σουίτες ελέγχου των έργων που μελετήθηκαν:

- μεταφορά περιπτώσεων ελέγχου από επόμενες εκδόσεις
- εμφύτευση σφαλμάτων
- συγγραφή περιπτώσεων ελέγχου

Μεταφορά περιπτώσεων ελέγχου

- Μεταξύ διαδοχικών εκδόσεων, επιδιορθώνονται σφάλματα
- Ταυτόχρονα δημιουργούνται νέες περιπτώσεις ελέγχου γι'αυτά
- Οι νέες περιπτώσεις ελέγχου, μεταφέρονται στην προηγούμενη έκδοση του έργου
- Χωρίς όμως να μεταφέρονται οι αντίστοιχες διορθώσεις στον κώδικα
- Οι περιπτώσεις ελέγχου θα εμφανίζουν σφάλματα

Μεταφορά περιπτώσεων ελέγχου (παράδειγμα)

```
--- pypy3-v5.10.0-src/pypy/interpreter/test/test_timeutils.py
1970-01-01 02:00:00.000000000 +0200
+++ pypy3-v5.10.1-src/pypy/interpreter/test/test_timeutils.py
2018-01-10 16:58:49.000000000 +0200
@@ -0,0 +1,13 @@
+import pytest
+from rpython.rlib.rarithmetic import r_longlong
+from pypy.interpreter.error import OperationError
+from pypy.interpreter.timeutils import timestamp_w
+
+def test_timestamp_w(space):
+    w_1_year = space.newint(365 * 24 * 3600)
+    result = timestamp_w(space, w_1_year)
+    assert isinstance(result, r_longlong)
+    assert result // 10 ** 9 == space.int_w(w_1_year)
+    w_millenium = space.mul(w_1_year, space.newint(1000))
+    with pytest.raises(OperationError):  # timestamps overflow after
    ~300 years
+        timestamp_w(space, w_millenium)
```

Εμφύτευση σφαλμάτων

- Τυχαία επιλογή μιας περίπτωσης ελέγχου
- Εντοπισμός του κομματιού του πηγαίου κώδικα που ελέγχει
- Αν είναι ρεαλιστικό, δημιουργία κάποιου «συνηθισμένου» σφάλματος, πχ:
  - `TypeError: 'NoneType'` exceptions
  - off-by-one
  - λάθος συγκρίσεις
- Αν δεν είναι ρεαλιστικό, τυχαία αναζήτηση κάποιας άλλης περίπτωσης ελέγχου

Συγγραφή περιπτώσεων ελέγχου

- Διορθώσεις στον κώδικα, κάποιες φορές δεν συνοδεύονται από αντίστοιχες περιπτώσεις ελέγχου
- Συγγραφή των περιπτώσεων ελέγχου
- Εφαρμογή στην προηγούμενο έκδοση του εξεταζόμενου έργου

Λογισμικό

Τα έργα λογισμικού που μελετήθηκαν ήταν τα εξής:

- PyPy
- Django
- MoinMoin
- Flask
- Six

Κριτήρια επιλογής

- Ανοικτός κώδικας
- Υλοποίηση εξ'ολοκλήρου σε Python
- Μέγεθος
- Αριθμός περιπτώσεων ελέγχου

Εκδόσεις του PyPy που μελετήθηκαν

Έκδοση	Άριθμός αρχείων	Γραμμές κώδικα	Περιπτώσεις ελέγχου	Σφάλματα
PyPy2 5.7.0	4100	1140997	26471	18
PyPy2 5.7.1	4100	1141014	26472	15
PyPy2 5.8.0	4103	1143258	26534	25
PyPy2 5.9.0	4107	1146197	26615	22
PyPy3 5.10.0	5587	1589079	43874	17
PyPy3 5.10.1	5588	1589145	43875	18
PyPy3.6 7.1.0	4210	1383197	34293	20
PyPy3.6 7.1.1	4210	1384179	34311	20

Εκδόσεις του Django που μελετήθηκαν

Έκδοση	Άριθμός αρχείων	Γραμμές κώδικα	Περιπτώσεις ελέγχου	Σφάλματα
Django 2.0	1912	220346	10940	17
Django 2.0.1	1913	220575	10957	15
Django 2.0.2	1913	220673	10965	14
Django 2.0.3	1913	220756	10974	15
Django 2.0.4	1913	220806	10979	12
Django 2.0.5	1913	220915	10985	13
Django 2.0.6	1913	221062	10990	12
Django 2.0.7	1913	221111	10994	12

Εκδόσεις του MoinMoin που μελετήθηκαν

Έκδοση	Άριθμός αρχείων	Γραμμές κώδικα	Περιπτώσεις ελέγχου	Σφάλματα
MoinMoin 1.9.2	642	83572	142	7
MoinMoin 1.9.3	648	87085	149	12
MoinMoin 1.9.4	652	90966	161	7
MoinMoin 1.9.9	836	139623	161	6

Εκδόσεις του Flask που μελετήθηκαν

Έκδοση	Άριθμός αρχείων	Γραμμές κώδικα	Περιπτώσεις ελέγχου	Σφάλματα
Flask 0.11	62	8035	292	7
Flask 0.12	65	8311	309	9
Flask 0.12.3	64	8355	312	7
Flask 1.0	67	8911	386	9
Flask 1.0.2	67	8962	387	6

Εκδόσεις του Six που μελετήθηκαν

Έκδοση	Άριθμός αρχείων	Γραμμές κώδικα	Περιπτώσεις ελέγχου	Σφάλματα
six 1.8.0	4	1191	61	7
six 1.9.0	4	1328	67	5
six 1.10.0	4	1351	68	5
six 1.11.0	4	1395	70	5

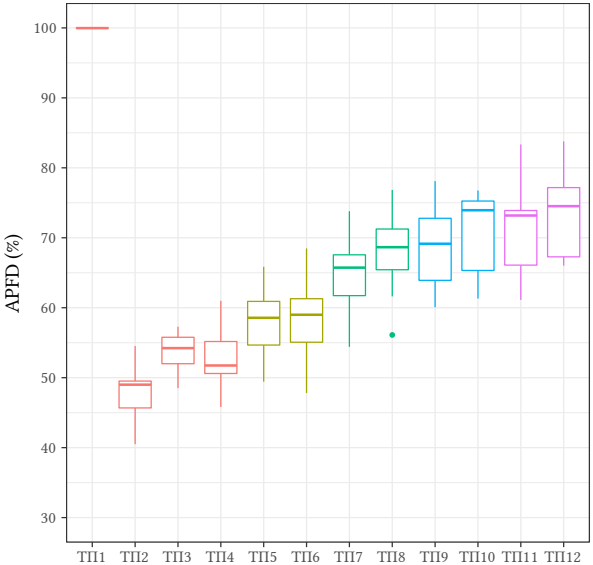
Εργαλεία

- Χρησιμοποιήθηκε το πλαίσιο ελέγχων pytest
- Έγινε ανάπτυξη ενός προσθέτου, του *pytest-prioritize*
  - `pytest --pytest-prioritize=branch-total`
  - `pytest --prioritization-gather-cov`
  - `pytest --prioritization-gather-cov-all`
- Μεταχειρίζεται τις μεθόδους του pytest
  - `pytest_sessionstart`
  - `pytest_collection_modifyitems`
- Χρησιμοποιήθηκε κατά περίπτωση διαφορετική έκδοση της Python

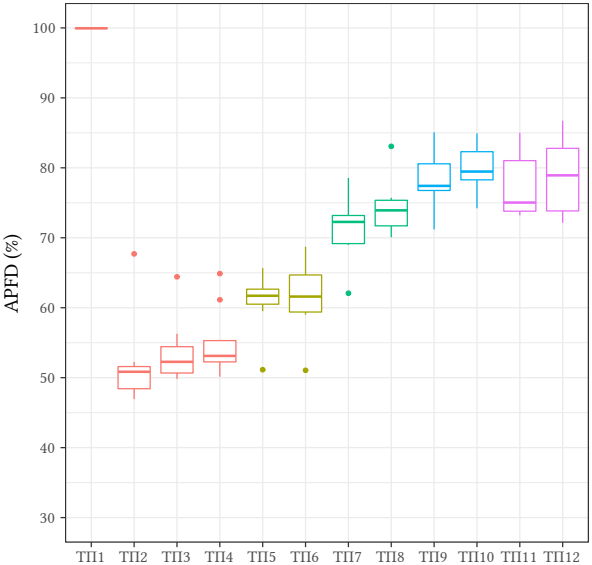
Ερευνητικά ερωτήματα

1. Μπορεί η προτεραιοποίηση περιπτώσεων ελέγχου, με βάση την πληροφορία κάλυψης κώδικα, σε έργα λογισμικού υλοποιημένα με τη γλώσσα Python, να βελτιώσει το ρυθμό ανίχνευσης σφαλμάτων, όταν εκτελείται μια σουίτα ελέγχων;
2. Ποια είναι η επίδραση της χρήσης διαφορετικών επιπέδων λεπτομέρειας της κάλυψης κώδικα στον ρυθμό ανίχνευσης σφαλμάτων;
3. Υπάρχουν διαφορές ως προς το ρυθμό ανίχνευσης σφαλμάτων όταν χρησιμοποιείται επιπρόσθετη πληροφορία κάλυψης κώδικα σε σχέση με την ολική κάλυψη κώδικα;
4. Υπάρχουν διαφορές μεταξύ των έργων λογισμικού σε γλώσσα Python, όσον αφορά στο ρυθμό ανίχνευσης σφαλμάτων, ανάλογα με το μέγεθος του έργου;
5. Υπάρχουν διαφορές μεταξύ των έργων λογισμικού σε γλώσσα Python, όσον αφορά στο ρυθμό ανίχνευσης σφαλμάτων, σε σχέση με τα αντίστοιχα αποτελέσματα, που παρουσιάζονται στη βιβλιογραφία με τη χρήση άλλων γλωσσών;

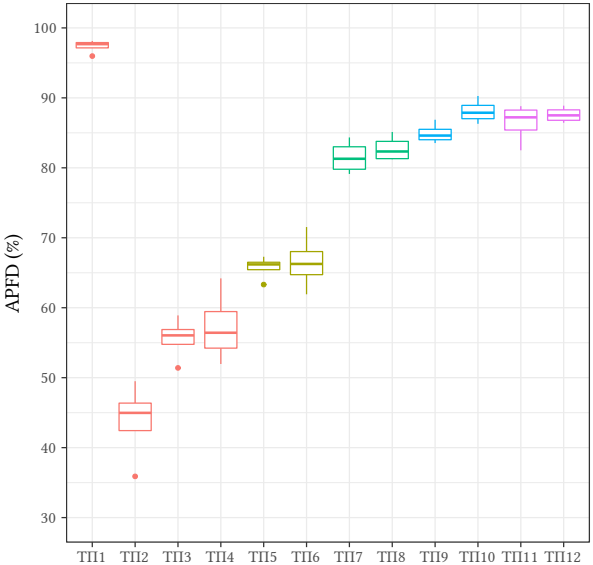
Αποτελέσματα - PyPy



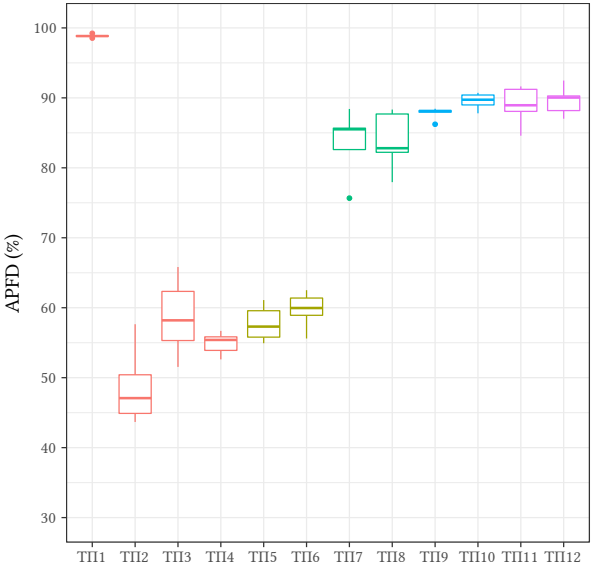
Αποτελέσματα - Django



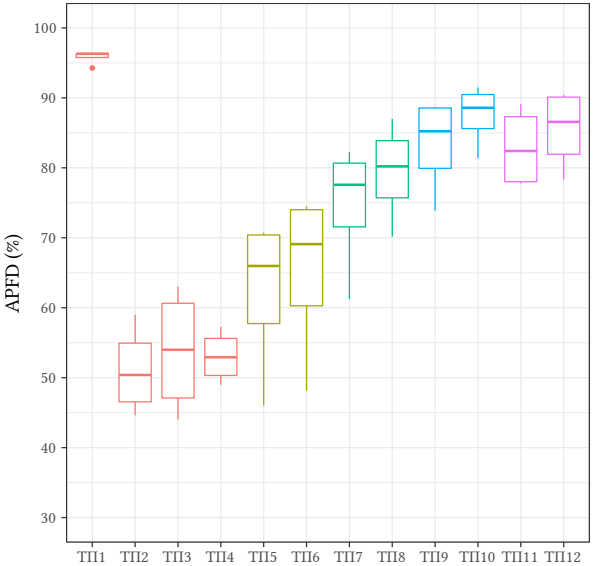
Αποτελέσματα - MoinMoin



Αποτελέσματα - Flask



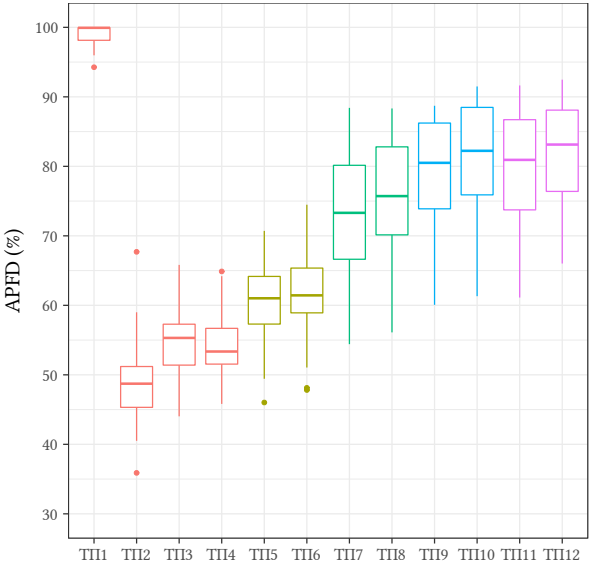
Αποτελέσματα - Six



Ανάλυση Διακύμανσης

	DF	Sum Sq	Mean Sq	F value	Pr(>F)
ΤΠ	11	68846.33	6258.76	243.66	<0.001
Έργο	4	4639.04	1159.76	45.15	<0.001
ΤΠ × Έργο	44	4102.40	93.24	3.63	<0.001
Υπολείμματα	288	7397.56	25.69		

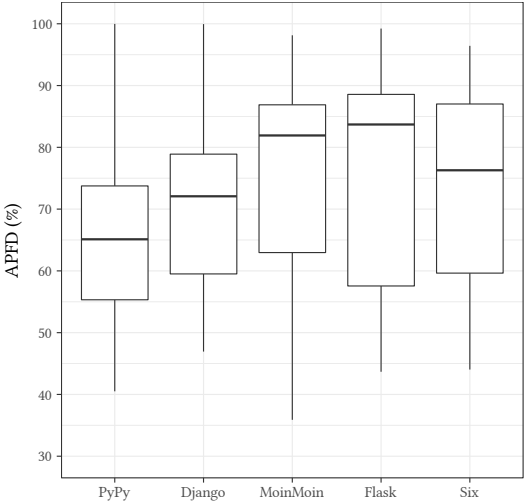
Συνολικά Αποτελέσματα



Έλεγχος Tukey - Τεχνικές Προτεραιοποίησης

ΤΠ	Τίτλος	Μέσος Όρος	Ομάδα						
ΤΠ2	Untreated	49.03	Α						
ΤΠ4	Inverse	54.49		Β					
ΤΠ3	Random	54.80		Β					
ΤΠ5	File-total	60.39			Γ				
ΤΠ6	File-addtl	61.53			Γ				
ΤΠ7	Method-total	73.50				Δ			
ΤΠ8	Method-addtl	76.07				Δ	Ε		
ΤΠ9	Branch-total	78.82					Ε	Ζ	
ΤΠ11	Statement-total	79.68						Ζ	
ΤΠ10	Branch-addtl	81.22						Ζ	
ΤΠ12	Statement-addtl	81.31						Ζ	
ΤΠ1	Optimal	98.84							Η

Σύγκριση Έργων - Θηκόγραμμα και έλεγχος Tukey



Έργο	Μέσος Όρος	Ομάδα		
PyPy	65.7	A		
Django	70.2		B	
Six	72.9			Γ
MoinMoin	74.8			Γ
Flask	75.1			Γ

Απαντήσεις (1)

Μπορεί η προτεραιοποίηση περιπτώσεων ελέγχου, με βάση την πληροφορία κάλυψης κώδικα, σε έργα λογισμικού υλοποιημένα με τη γλώσσα Python, να βελτιώσει το ρυθμό ανίχνευσης σφαλμάτων, όταν εκτελείται μια σουίτα ελέγχων;

- Ναι, η χρήση της πληροφορίας κάλυψης κώδικα αυξάνει σημαντικά το ρυθμό ανίχνευσης σφαλμάτων

Απαντήσεις (2)

Ποια είναι η επίδραση της χρήσης διαφορετικών επιπέδων λεπτομέρειας της κάλυψης κώδικα στον ρυθμό ανίχνευσης σφαλμάτων;

- Όσο μεγαλύτερη η λεπτομέρεια στην κάλυψη κώδικα, τόσο μεγαλύτερος ο ρυθμός ανίχνευσης σφαλμάτων.

Απαντήσεις (3)

Υπάρχουν διαφορές ως προς το ρυθμό ανίχνευσης σφαλμάτων όταν χρησιμοποιείται επιπρόσθετη πληροφορία κάλυψης κώδικα σε σχέση με την ολική κάλυψη κώδικα;

- Η χρήση επιπρόσθετης πληροφορίας δίνει ελάχιστα καλύτερα αποτελέσματα, τα οποία όμως δεν είναι σημαντικά. Δεδομένης της αυξημένης πολυπλοκότητας, μάλλον θα πρέπει να προτιμάται η χρήση της πληροφορίας ολικής κάλυψης

Απαντήσεις (4)

Υπάρχουν διαφορές μεταξύ των έργων λογισμικού σε γλώσσα Python, όσον αφορά στο ρυθμό ανίχνευσης σφαλμάτων, ανάλογα με το μέγεθος του έργου;

- Ναι. Φαίνεται πως μεγαλύτερα έργα έχουν μικρότερους ρυθμούς ανίχνευσης σφαλμάτων.

Απαντήσεις (5)

Υπάρχουν διαφορές μεταξύ των έργων λογισμικού σε γλώσσα Python, όσον αφορά στο ρυθμό ανίχνευσης σφαλμάτων, σε σχέση με τα αντίστοιχα αποτελέσματα, που παρουσιάζονται στη βιβλιογραφία με τη χρήση άλλων γλωσσών;

- Οι τιμές της μετρικής APFD που επιτυγχάνονται στη βιβλιογραφία για άλλες γλώσσες είναι αρκετά υψηλότερες. Πιθανόν να οφείλεται στο μικρότερο μέγεθος των έργων που μελετούνται.

Bonus

Φαίνεται πως η χρήση της τυχαίας ή καλύτερα της αντίστροφης διάταξης των περιπτώσεων ελέγχου, οδηγεί σε μικρές αλλά σημαντικές βελτιώσεις του ρυθμού ανίχνευσης σφαλμάτων.

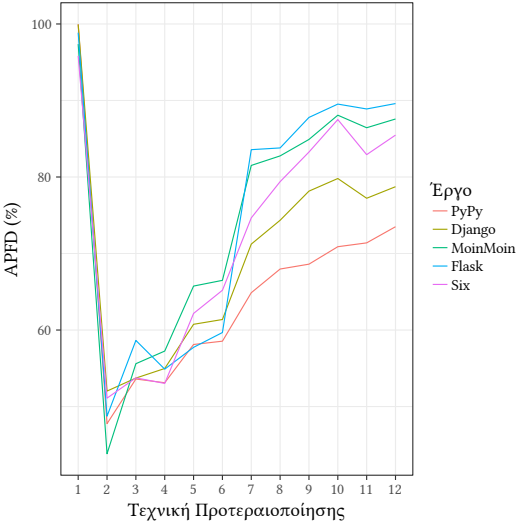
Μελλοντικές επεκτάσεις

- Μελέτη περισσότερων έργων λογισμικού
- Περισσότερες στρατηγικές βελτιστοποίησης, πέρα από τον άπληστο αλγόριθμο
- Αξιοποίηση πληροφορίας κόστους (π.χ. APFDc)
- Περισσότερες μετρικές για την επιβεβαίωση της καταλληλότητας της μετρικής APFD
- Χρήση πληροφορίας αλλαγών
- Επέκταση σε άλλες δημοφιλείς γλώσσες προγραμματισμού (π.χ. Javascript, PHP)

Σας ευχαριστώ!

Ερωτήσεις;

Όρια και περιορισμοί

- Η μετρική APFD
  - δεν είναι ο μόνος τρόπος ώστε να εκτιμηθεί ο ρυθμός ανίχνευσης σφαλμάτων.
  - δεν εκτιμά το κόστος εκτέλεσης
  - αντιπαράθεση με άλλες μετρικές
- Σωστή λειτουργία των εργαλείων
  - έγινε διασταύρωση των αποτελεσμάτων για το Six και το docopt με το χέρι
- Διαφορές μεταξύ των έργων λογισμικού, τοπικότητα των αλλαγών, σύνθεση της κάθε σουίτας ελέγχου
  - όλες οι τεχνικές προτεραιοποίησης εφαρμόστηκαν σε όλα τα έργα
- Είναι τα έργα που μελετήθηκαν αντιπροσωπευτικά;
  - χρήση (επί το πλείστον) πραγματικών σφαλμάτων
  - θα πρέπει να μελετηθούν περισσότερα έργα

Αλληλεπιδράσεις