ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΤΕΡΑΙΟΠΟΙΗΣΗ ΠΕΡΙΠΤΩΣΕΩΝ ΕΛΕΓΧΟΥ ΣΕ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Διπλωματική Εργασία

του

Γεωργίου Βλαχάβα

Θεσσαλονίκη , 21/6/2019

ΠΡΟΤΕΡΑΙΟΠΟΙΗΣΗ ΠΕΡΙΠΤΩΣΕΩΝ ΕΛΕΓΧΟΥ ΣΕ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Γεώργιος Βλαχάβας

Πτυχίο Πληροφορικής, ΕΑΠ, 2015
MSc Food Technology – Quality Assurance, University of Reading, 2003
Πτυχίο Τεχνολογίας Τροφίμων, ΑΤΕΙΘ, 2002

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Αλέξανδρος Χατζηγεωργίου

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ηη/μμ/εεεε

Αλέξανδρος Χατζηγεωργίου	Ελευθέριος Μαμάτας	Παναγιώτης Παπαδημητρίου
.....	.....	.....

Γεώργιος Βλαχάβας

.....

Περίληψη

Σε σύγχρονα συστήματα μεγάλου μεγέθους, με υψηλή πολυπλοκότητα και ταχύτατη ανάπτυξη, εφαρμόζονται έλεγχοι παλινδρόμησης, ώστε να διασφαλιστεί η σωστή λειτουργία του υπάρχοντος κώδικα. Αυτοί όμως οι έλεγχοι αποτελούν μια χρονοβόρα και κοστοβόρα διαδικασία και είναι πιθανό να μην υπάρχουν οι πόροι για την εκτέλεσή τους με κάθε αλλαγή που πραγματοποιείται στον κώδικα. Η Προτεραιοποίηση Περιπτώσεων Ελέγχου είναι η διαδικασία με την οποία μεταβάλλεται η σειρά εκτέλεσης των περιπτώσεων ελέγχου ενός έργου λογισμικού, με σκοπό την ανίχνευση των σφαλμάτων που πιθανόν υπάρχουν, νωρίτερα.

Στην παρούσα έρευνα χρησιμοποιήθηκαν έργα λογισμικού, μεσαίου και μεγάλου μεγέθους, υλοποιημένα με τη χρήση της γλώσσας Python. Τα πειράματα πραγματοποιήθηκαν σε πολλαπλές εκδόσεις των έργων με τη χρήση του πλαισίου ελέγχων pytest. Σφάλματα προστέθηκαν με τεχνητό τρόπο στις σουίτες περιπτώσεων ελέγχου των έργων αυτών.

Αξιοποιήθηκε η πληροφορία κάλυψης των περιπτώσεων ελέγχου σε επίπεδα αρχείου, μεθόδου, διακλάδωσης και εντολής υπό τη μορφή τόσο της ολικής κάλυψης, όσο και της επιπρόσθετης κάλυψης κώδικα, για τη δημιουργία και την αξιολόγηση διαφορετικών διατάξεων των περιπτώσεων ελέγχου του κάθε έργου. Εξετάστηκαν επίσης, πέρα από την αρχική διάταξη των περιπτώσεων ελέγχου, η χρήση της τυχαίας διάταξης και η αντίστροφη διάταξη της αρχικής. Η αξιολόγηση των αποτελεσμάτων πραγματοποιήθηκε με τη χρήση της μετρικής APFD (Average Percentage of Faults Detected), η οποία εκτιμά το ρυθμό ανίχνευσης σφαλμάτων σε μία σουίτα ελέγχου.

Τα αποτελέσματα έδειξαν ότι η χρήση της πληροφορίας κάλυψης για την προτεραιοποίηση περιπτώσεων ελέγχου, βελτιώνει σημαντικά τους ρυθμούς ανίχνευσης σφαλμάτων. Μάλιστα, η βελτίωση είναι τόσο μεγαλύτερη, όσο πιο λεπτομερής είναι η χρήση της πληροφορίας κάλυψης. Δεν υπήρχαν σημαντικές διαφορές μεταξύ της χρήσης πληροφορίας ολικής και επιπρόσθετης κάλυψης. Αρκετά μικρότερες, αλλά επίσης σημαντικές βελτιώσεις στο ρυθμό ανίχνευσης σφαλμάτων επιτυγχάνονται και από τη χρήση της τυχαίας και της αντίστροφης διάταξης των περιπτώσεων ελέγχου.

Λέξεις Κλειδιά: Προτεραιοποίηση Περιπτώσεων Ελέγχου, έλεγχοι παλινδρόμησης, Python, pytest, κάλυψη κώδικα, APFD, εμπειρική μελέτη

Abstract

In modern software systems of considerable size, with high complexity and rapid development cycles, regression tests are performed, ensuring that software functionality remains intact. However, these regression tests often induce considerable costs, both in terms of time and money, resources that might be limited. Therefore, the execution of all test cases after every small code change, might not be an affordable practice. Test case prioritization is the process by which the execution order of test cases is altered, with the purpose of detecting faults sooner during the execution of the test suite.

In this study, software projects of medium and large size, written using Python, were used as subjects. Experiments were conducted for multiple version of said projects using the pytest testing framework. Faults were artificially added in the projects' test.

Code coverage information at file, method, branch and statement granularity levels was gathered and leveraged, in order to construct and assess different test case orderings for each project. Total coverage information as well as additional coverage information was used. Additionally to the original ordering of test cases, a random ordering as well as the reverse ordering with respect to the original were evaluated. Results were recorded in the form of the APFD (Average Percentage of Faults Detected) metric, which estimates the fault detection rate of a test suite.

Results showed that using code coverage information for test case prioritization improves fault detection rates considerably. The improvement is more pronounced for increased granularity of coverage information. No significant differences were discovered between total and additional coverage methods. Quite smaller, but significant improvements compared to the original ordering, were also detected when using random or inverse orderings of test cases.

Keywords: Test case prioritization, regression testing, Python, pytest, code coverage, APFD, empirical study

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Συμβολισμοί

ΠΕ Περίπτωση Ελέγχου

ΠΠΕ Προτεραιοποίηση Περιπτώσεων Ελέγχου

ΣΠΕ Σουίτα Περιπτώσεων Ελέγχου

ΤΠ Τεχνική Προτεραιοποίησης

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Η διασφάλιση ποιότητας λογισμικού (Software Quality Assurance) περιλαμβάνει το σύνολο των διαδικασιών που εξασφαλίζουν ότι το παραγόμενο λογισμικό συμμορφώνεται με καθορισμένες προδιαγραφές ποιότητας, τόσο σε σχέση με τις λειτουργικές όσο και σε σχέση με τις μη-λειτουργικές απαιτήσεις. Η διασφάλιση ποιότητας λογισμικού είναι μια συνεχής δραστηριότητα κατά τη διάρκεια του κύκλου ανάπτυξης λογισμικού με στόχο το συστηματικό και διαρκή έλεγχο των χαρακτηριστικών ποιότητας. Στα σύγχρονα συστήματα λογισμικού που χαρακτηρίζονται από υψηλή πολυπλοκότητα, ταχύτατη ανάπτυξη, συνεχή συντήρηση και εξέλιξη και αποτελούν τον πυρήνα προϊόντων και υπηρεσιών η διασφάλιση της ποιότητας αποτελεί παράγοντα μείζονος σημασίας για την «επιβίωση» σε περιβάλλον έντονο οικονομικού ανταγωνισμού. Για τον λόγο αυτό, οι εταιρείες ανάπτυξης λογισμικού συνεργάζονται σε συστηματική βάση με εταιρείες διασφάλισης ποιότητας που αναπτύσσουν ενδελεχείς ελέγχους των εξεταζόμενων συστημάτων.

Παράλληλα, η υιοθέτηση ευέλικτων μεθοδολογιών ανάπτυξης (agile development methodologies) όπου τα συστήματα αναπτύσσονται σε πολλαπλές και σύντομες επαναλήψεις στο τέλος των οποίων παράγεται πλήρως λειτουργικό λογισμικό, έχει επιβάλλει τη συνεχή ολοκλήρωση (continuous integration) και συνεχή παράδοση (continuous delivery). Η συνεχής ολοκλήρωση αφορά στη δυνατότητα συγχώνευσης όλων των αλλαγών που πραγματοποιούνται από τα μέλη της ομάδας ανάπτυξης (που σε σύγχρονα συστήματα ενδέχεται να εργάζονται σε διαφορετικές θέσεις ή χρονικές ζώνες) επί ενός κοινού αποθετηρίου (ακόμα και πολλές φορές στην ίδια ημέρα). Η συνεχής παράδοση αφορά στη δυνατότητα κατασκευής λογισμικού που μπορεί να τεθεί σε λειτουργία ανά πάσα χρονική στιγμή, και εξασφαλίζεται με εργαλεία κατασκευής (build automation tools) που ενσωματώνουν στο παραγόμενο λογισμικό νέες λειτουργίες, διορθώσεις σφαλμάτων και ρυθμίσεις. Συστατικό στοιχείο για τη συνεχή παράδοση λογισμικού είναι η πραγματοποίηση αυτοματοποιημένων ελέγχων, συνήθως υπό τη μορφή ελέγχων παλινδρόμησης (regression tests) που εξασφαλίζουν την εκτέλεση ενός ικανού αριθμού περιπτώσεων ελέγχου. Οι έλεγχοι παλινδρόμησης αποτελούν μια χρονοβόρα και κοστοβόρα διαδικασία. Εκτιμάται ότι η διενέργεια των ελέγχων παλινδρόμησης αντιστοιχεί

περίπου στο μισό του κόστους συντήρησης του λογισμικού [34, 27, 3]. Η επιτυχής εκτέλεση περιπτώσεων ελέγχου για τις λειτουργίες που πρέπει να παρέχει ένα σύστημα λογισμικού εγγυάται σε μεγάλο βαθμό, ότι κάθε νέα έκδοση του συστήματος δεν προκαλεί σφάλματα σε ήδη υπάρχουσες αλλά και σε νέες λειτουργίες.

Η πρόκληση ωστόσο που αντιμετωπίζεται σε καθημερινή βάση από τις ομάδες ή εταιρείες διασφάλισης ποιότητας λογισμικού είναι η αδυναμία πραγματοποίησης όλων των ελέγχων παλινδρόμησης μέσα στα περιορισμένα χρονικά όρια που διατίθενται για το σκοπό αυτό, εντός των επαναλήψεων μιας ευέλικτης μεθοδολογίας ανάπτυξης. Για παράδειγμα, το 2017 η Google αναφέρεται ότι είχε 150 εκατομμύρια εκτελέσεις ελέγχων και 800.000 μεταγλωττίσεις κώδικα (builds) ανά ημέρα [31]. Πλήρης εκτέλεση όλων των περιπτώσεων ελέγχου απαιτεί συχνά πλήθος ωρών ή και ημερών που δεν είναι διαθέσιμες. Ανεπαρκής εκτέλεση περιπτώσεων ελέγχου ενέχει τον κίνδυνο μη εντοπισμού σφαλμάτων στο παραγόμενο λογισμικό. Κατά συνέπεια, απαιτούνται τεχνικές αποδοτικής προτεραιοποίησης περιπτώσεων ελέγχου (test case prioritization) ώστε με την εκτέλεση ενός υποσυνόλου των διαθέσιμων ελέγχων να εξασφαλίζεται επαρκής κάλυψη του εξεταζόμενου κώδικα καθώς και υψηλή πιθανότητα αποκάλυψης σφαλμάτων.

1.2 Σκοπός – Στόχοι

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

Για το σκοπό αυτό, θα χρησιμοποιηθούν έργα λογισμικού ανοιχτού κώδικα, διαφορετικών μεγεθών. Σε αυτά, θα δημιουργηθούν τεχνητά σφάλματα στις σουίτες περιπτώσεων ελέγχων τους, τα οποία θα χρησιμοποιηθούν για την αξιολόγηση των διαφορετικών τεχνικών προτεραιοποίησης.

Πέρα από την αρχική διάταξη εκτέλεσης των περιπτώσεων ελέγχου, θα εξεταστεί η χρήση της τυχαίας διάταξής τους και της αντίστροφης διάταξης από την αρχική, ως τεχνικές σύγκρισης. Θα εξεταστούν επίσης τεχνικές προτεραιοποίησης βασισμένες στην κάλυψη κώδικα. Αυτές κατηγοριοποιούνται ανάλογα με τη λεπτομέρεια με την οποία γίνεται η ανάλυση της κάλυψης κώδικα, σε επίπεδα αρχείου, μεθόδου, διακλάδωσης και εντολής. Επίσης για κάθε μία από αυτές τις κατηγορίες, θα εξεταστούν παραλλαγές τους οι οποίες θα χρησιμοποιούν πληροφορία ολικής κάλυψης κώδικα και πληροφορία

επιπρόσθετης κάλυψης κώδικα.

1.3 Ερωτήματα

Τα ερευνητικά ερωτήματα στα οποία θα προσπαθήσει να δώσει απαντήσεις η παρούσα εργασία είναι τα παρακάτω:

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;

1.4 Συνεισφορά

Η εργασία αυτή αποτελεί την πρώτη προσπάθεια εκτίμησης της επίδρασης διαφορετικών τεχνικών προτεραιοποίησης σε έργα λογισμικού υλοποιημένα με τη χρήση της γλώσσας Python. Υπάρχουσες μελέτες αναφέρονται αποκλειστικά σε έργα λογισμικού υλοποιημένα με τη χρήση των γλωσσών C και Java. Επίσης, στη βιβλιογραφία, δεν εμφανίζονται έργα λογισμικού του μεγέθους που χρησιμοποιήθηκαν στην παρούσα εργασία. Χρησιμοποιούνται κυρίως συγκεκριμένα έργα λογισμικού τα οποία αποτελούνται συνήθως από μερικές εκατοντάδες ή το πολύ δεκάδες χιλιάδες γραμμές κώδικα.

1.5 Βασική Ορολογία

Στη συνέχεια παρατίθεται μια σύντομη περιγραφή για κάποιους από τους βασικούς όρους που θα χρησιμοποιούνται στο κείμενο. Αναλυτικότερη περιγραφή για κάθε ένα

από αυτούς υπάρχει στο κεφάλαιο 2.

Περίπτωση ελέγχου: Κώδικας, αποκλειστικός στόχος του οποίου είναι ο έλεγχος της ορθής λειτουργίας κάποιου άλλου τμήματος του κώδικα

Σουίτα περιπτώσεων ελέγχου: Το σύνολο των περιπτώσεων ελέγχου σε ένα έργο λογισμικού

Έλεγχοι παλινδρόμησης: Η εκτέλεση των περιπτώσεων ελέγχου που περιλαμβάνονται στη σουίτα ελέγχου, μετά από μία ή περισσότερες αλλαγές στον κώδικα του έργου, με σκοπό την ανίχνευση ανεπιθύμητων σφαλμάτων

Ρυθμός ανίχνευσης σφαλμάτων: Η ταχύτητα με την οποία μία σουίτα ελέγχου ανιχνεύει σφάλματα. Όσο πιο νωρίς στην εκτέλεση της σουίτας ελέγχου ανιχνεύονται τα σφάλματα, τόσο πιο μεγάλος ο ρυθμός ανίχνευσης σφαλμάτων

Κάλυψη κώδικα: Αποτελεί μέτρο του βαθμού στον οποίο καλύπτεται (εκτελείται) ο κώδικας ενός έργου, από τη σουίτα ελέγχου

Ολική κάλυψη κώδικα: Για κάθε περίπτωση ελέγχου, το ποσοστό του κώδικα που καλύπτεται από αυτή, στο σύνολο του έργου

Επιπρόσθετη κάλυψη κώδικα: Για κάθε περίπτωση ελέγχου, το ποσοστό του κώδικα που καλύπτεται από αυτή, το οποίο δεν έχει καλυφθεί από κάποια άλλη περίπτωση ελέγχου που έχει εκτελεστεί νωρίτερα.

1.6 Διάρθρωση της μελέτης

Η διάρθρωση της παρούσας εργασίας είναι αυτή που περιγράφεται παρακάτω.

Στο κεφάλαιο 2, γίνεται παρουσίαση του προβλήματος της Προτεραιοποίησης Περιπτώσεων Ελέγχου (ΠΠΕ). Συγκεκριμένα, παρουσιάζεται ο τυπικός ορισμός της ΠΠΕ, ενώ γίνεται περιγραφή των βασικών τεχνικών που υπάρχουν για την προτεραιοποίηση. Σε σχέση με τις τεχνικές ανάλυσης κάλυψης κώδικα, αναλύεται η διαφορά μεταξύ της δυναμικής και στατικής ανάλυσης κώδικα, τα πλεονεκτήματα και μειονεκτήματά τους καθώς και ο τρόπος που αξιοποιούνται τα αποτελέσματα της ανάλυσης αυτής. Στη συνέχεια, παρουσιάζονται οι διαφορετικές στρατηγικές βελτιστοποίησης της προτεραιοποίησης των περιπτώσεων ελέγχου σε σχέση με την κάλυψη κώδικα και τρόποι που μπορούν να

χρησιμοποιηθούν για την εκτίμηση και τον εντοπισμό της βέλτιστης διάταξης των περιπτώσεων ελέγχου. Παρουσιάζεται η έννοια της «περιοχής κάτω από την καμπύλη» (Area Under the Curve – AUC), συνοδευόμενη από παραδείγματα, καθώς και των μεθόδων που υπάρχουν για την αξιολόγηση διαφορετικών διατάξεων των περιπτώσεων ελέγχου. Τέλος, παρουσιάζεται συνοπτικά η γλώσσα προγραμματισμού Python, και το πλαίσιο ελέγχων pytest που χρησιμοποιήθηκε για το πειραματικό μέρος της εργασίας.

Το κεφάλαιο 3, αναφέρεται στη μεθοδολογία που χρησιμοποιήθηκε για τη διενέργεια του πειραματικού μέρους της εργασίας, καθώς και για την αξιολόγηση των αποτελεσμάτων που προέκυψαν. Συγκεκριμένα, παρουσιάζονται αναλυτικά οι τεχνικές προτεραιοποίησης που χρησιμοποιήθηκαν για τη δημιουργία των διαφορετικών διατάξεων των περιπτώσεων ελέγχου, κατηγοριοποιημένες σύμφωνα με το είδος της πληροφορίας που χρησιμοποιούν. Στη συνέχεια, αναφέρονται οι τρόποι με τους οποίους δημιουργήθηκαν σφάλματα για την αξιολόγηση των διαφορετικών τεχνικών προτεραιοποίησης. Ακολούθως, παρουσιάζονται συνοπτικά τα έργα λογισμικού, ο κώδικας των οποίων χρησιμοποιήθηκε για τη διενέργεια των ελέγχων, οι εκδόσεις τους που αναλύθηκαν, μαζί με τα χαρακτηριστικά τους, όπως ο αριθμός των γραμμών κώδικα και το μέγεθος της σουίτας ελέγχου τους. Παρουσιάζονται επίσης τα εργαλεία που χρησιμοποιήθηκαν για τη διενέργεια του πειραματικού μέρους. Το κεφάλαιο κλείνει με την αναφορά των στατιστικών μεθόδων και εργαλείων που χρησιμοποιήθηκαν για την αξιολόγηση των αποτελεσμάτων.

Τα αποτελέσματα του πειραματικού μέρους, εμφανίζονται στο κεφάλαιο 4. Αυτά παρουσιάζονται αναλυτικά για κάθε έργο λογισμικού που μελετήθηκε, ενώ επίσης αξιολογούνται με τη χρήση κατάλληλων τεχνικών στατιστικής ανάλυσης. Γίνεται συζήτηση των αποτελεσμάτων και σύγκριση με τα αντίστοιχα που εμφανίζονται στη βιβλιογραφία.

Ο επίλογος της εργασίας βρίσκεται στο κεφάλαιο 5. Εκεί αναφέρονται συνοπτικά τα αποτελέσματα και τα συμπεράσματα της έρευνας, αναλύονται τα όρια και οι περιορισμοί της, ενώ γίνεται αναφορά και σε μελλοντικές επεκτάσεις που θα μπορούσε να έχει η έρευνα.

Μετά την παρουσίαση των βιβλιογραφικών αναφορών, ακολουθεί Παράρτημα, στο οποίο περιλαμβάνονται αποτελέσματα της έρευνας που λόγω του όγκου τους δεν κρίθηκε δόκιμο να περιληφθούν στο βασικό μέρος του κειμένου.

2 Βιβλιογραφική Επισκόπηση

2.1 Έλεγχοι παλινδρόμησης

Οι έλεγχοι παλινδρόμησης (regression tests) εκτελούνται όταν πραγματοποιούνται μεταβολές σε υπάρχον λογισμικό. Ο στόχος των ελέγχων παλινδρόμησης είναι η αύξηση της εμπιστοσύνης στο ότι προσθήκες ή αλλαγές στον υπάρχων κώδικα, δεν μεταβάλλουν τη συμπεριφορά του μέρους του κώδικα που δεν έχει υποστεί μεταβολές [50]. Αυτή είναι μια περίπλοκη διαδικασία η οποία γίνεται περισσότερο περίπλοκη, λόγω των σύγχρονων μεθόδων ανάπτυξης λογισμικού. Για παράδειγμα, γίνεται συχνή χρήση εξωτερικών βιβλιοθηκών οι οποίες έχουν αναπτυχθεί από τρίτους. Οποιαδήποτε μεταβολή στον κώδικα αυτών των εξωτερικών βιβλιοθηκών, ενδέχεται να προκαλέσει μεταβολές στο υπόλοιπο λογισμικό. Επίσης, οι συντομότεροι κύκλοι ανάπτυξης λογισμικού, όπως περιγράφονται από ευέλικτες (agile) μεθοδολογίες ανάπτυξης, επιβάλλουν περιορισμούς στον τρόπο που οι έλεγχοι παλινδρόμησης μπορούν να εκτελεστούν με τη χρήση περιορισμένων πόρων [15].

Ο πιο άμεσος τρόπος για την επίλυση του προβλήματος της εκτέλεσης των ελέγχων παλινδρόμησης είναι απλά η εκτέλεση όλων των περιπτώσεων ελέγχου που περιλαμβάνει το λογισμικό. Η προσέγγιση αυτή ονομάζεται *retest-all*. Ωστόσο, ενώ ένα λογισμικό αναπτύσσεται, η σουίτα ελέγχου του τείνει να μεγαλώσει σε μέγεθος, το οποίο σημαίνει ότι ενδεχομένως να είναι αδύνατη η εκτέλεση ολόκληρης της σουίτας ελέγχου, είτε λόγω έλλειψης χρόνου, είτε λόγω κόστους [50]. Οι περιορισμοί αυτοί οδηγούν στην αναζήτηση εναλλακτικών τεχνικών οι οποίες έχουν ως στόχο να μειώσουν το έργο που απαιτείται για την πραγματοποίηση των ελέγχων παλινδρόμησης.

Οι τρεις βασικές μέθοδοι που χρησιμοποιούνται για την υποβοήθηση των ελέγχων παλινδρόμησης είναι η επιλογή μεταξύ των περιπτώσεων ελέγχου (test case selection), η ελαχιστοποίηση της σουίτας ελέγχου (test suite minimization) και η προτεραιοποίηση των περιπτώσεων ελέγχου (test case prioritization) [35].

2.2 Επιλογή των περιπτώσεων ελέγχου

Η εκτέλεση όλων των περιπτώσεων ελέγχου δεν είναι πάντα εφικτή, αφού ενδέχεται να απαιτεί περισσότερους πόρους από όσους είναι διαθέσιμοι. Οι τεχνικές επιλογής των περιπτώσεων ελέγχου (regression test selection) [4, 8, 36, 44, 28] χρησιμοποιούν πληρο-

φορία σχετικά με την προηγούμενη και την τρέχουσα έκδοση του λογισμικού αλλά και τη σουίτα ελέγχου, για να επιλέξουν το υποσύνολο των περιπτώσεων ελέγχου που θα εκτελεστούν. Εμπειρικές μελέτες που χρησιμοποιούν τέτοιες τεχνικές [8, 18, 38, 39] έχουν δείξει ότι παρουσιάζουν σημαντικά οφέλη ως προς το κόστος εκτέλεσης των περιπτώσεων ελέγχου.

Ένας συνήθης συμβιβασμός των τεχνικών επιλογής των περιπτώσεων ελέγχου γίνεται μεταξύ της ασφάλειας και της αποδοτικότητας. Ασφαλείς τεχνικές επιλογής περιπτώσεων ελέγχου (π.χ. [8, 36, 28]) εγγυώνται ότι, κάτω από συγκεκριμένες συνθήκες, οι περιπτώσεις ελέγχου που δεν έχουν επιλεχθεί, δεν θα είχαν εκθέσει σφάλματα στη νέα έκδοση του λογισμικού [37]. Μη ασφαλείς τεχνικές (π.χ. [16, 19, 28]) θυσιάζουν την ασφάλεια προς όφελος της αποδοτικότητας. Αυτές, επιλέγουν περιπτώσεις ελέγχου, οι οποίες θεωρούνται πιο «χρήσιμες» από τις υπόλοιπες [35].

2.3 Ελαχιστοποίηση της σουίτας ελέγχου

Η ελαχιστοποίηση της σουίτας ελέγχου, αναφέρεται στη διαδικασία με την οποία αναζητούνται οι περιπτώσεις ελέγχου αυτές, οι οποίες είναι παρωχημένες και οι οποίες θα πρέπει να απομακρυνθούν από τη σουίτα ελέγχου [23]. Τεχνικές που αντιμετωπίζουν το πρόβλημα αυτό [6, 20, 1, 33], το κάνουν αναλύοντας τις περιπτώσεις ελέγχου σε σχέση με τον υπάρχον κώδικα του λογισμικού. Στόχος τους είναι η μόνιμη απομάκρυνση των περιπτώσεων ελέγχου που κατηγοριοποιούνται ως παρωχημένες, από τη σουίτα ελέγχου. Αυτή είναι και η βασικότερη διαφορά της τεχνικής ελαχιστοποίησης της σουίτας ελέγχου σε σχέση με την τεχνική επιλογής των περιπτώσεων ελέγχου, αφού στην τελευταία, δεν απομακρύνεται καμία περίπτωση ελέγχου από τη σουίτα ελέγχου, απλά δεν επιλέγονται να εκτελεστούν.

Ελαττώνοντας τον αριθμό των περιπτώσεων ελέγχου που περιλαμβάνει η σουίτα ελέγχου, μειώνεται το κόστος εκτέλεσης, επικύρωσης και διαχείρισης της σουίτας ελέγχου για όλες τις μελλοντικές εκδόσεις του λογισμικού. Ένα πιθανό μειονέκτημα της ελαχιστοποίησης της σουίτας ελέγχου, είναι ότι με την αφαίρεση κάποιων περιπτώσεων ελέγχου, ενδέχεται να μειωθεί η ικανότητα εντοπισμού σφαλμάτων της σουίτας ελέγχου [35]. Ωστόσο, μελέτες έχουν δείξει ότι η ελαχιστοποίηση της σουίτας ελέγχου, έχει τη δυνατότητα να περιορίσει σημαντικά το κόστος, χωρίς ταυτόχρονα να επηρεάσει σημαντικά την ικανότητα εντοπισμού σφαλμάτων [47]. Υπάρχουν όμως και μελέτες,

οι οποίες έχουν δείξει το αντίθετο, ότι δηλαδή η ελαχιστοποίηση της σουίτας ελέγχου μειώνει σημαντικά την ικανότητα εντοπισμού σφαλμάτων [40].

2.4 Προτεραιοποίηση Περιπτώσεων Ελέγχου

Η Προτεραιοποίηση των Περιπτώσεων Ελέγχου (ΠΠΕ), είναι η διαδικασία με την οποία οι περιπτώσεις ελέγχου που ανήκουν σε μία σουίτα ελέγχου, κατατάσσονται σε μια σειρά σημαντικότητας, σύμφωνα με κάποιο κριτήριο. Στη συνέχεια, αυτές εκτελούνται σύμφωνα με αυτή τη σειρά, με σκοπό τον εντοπισμό των σφαλμάτων το συντομότερο δυνατό κατά την εκτέλεση της σουίτας ελέγχου [35].

Εμπειρικές μελέτες έχουν δείξει ότι υπάρχουν απλές τεχνικές προτεραιοποίησης, οι οποίες έχουν τη δυνατότητα να αυξήσουν σημαντικά τον ρυθμό εντοπισμού των σφαλμάτων, δηλαδή να εκτελέσουν νωρίτερα τις περιπτώσεις ελέγχου αυτές οι οποίες θα αποτύχουν να εκτελεστούν [12, 34, 48]. Όταν συμβαίνει αυτό, είναι επόμενο ότι η ανατροφοδότηση των σημαντικών αποτελεσμάτων της σουίτας ελέγχου γίνεται νωρίτερα, επιτρέποντας στους προγραμματιστές την αντιμετώπιση των σφαλμάτων που θα προκύψουν, νωρίτερα.

Ακόμα κι αν η εκτέλεση της σουίτας ελέγχου διακοπεί πριν την εκτέλεση όλων των περιπτώσεων ελέγχου, η προτεραιοποίηση των περιπτώσεων ελέγχου διαφέρει από την επιλογή των περιπτώσεων ελέγχου. Αυτό συμβαίνει γιατί στην τελευταία, οι περιπτώσεις ελέγχου που επιλέγονται να εκτελεστούν, εκτελούνται όλες με την αρχική τους σειρά.

Το πρόβλημα της ΠΠΕ μπορεί να οριστεί γενικά ως το εξής [43]: «Η Προτεραιοποίηση Περιπτώσεων Ελέγχου είναι το πρόβλημα της διάταξης των Περιπτώσεων Ελέγχου (ΠΕ) που περιλαμβάνονται σε μία Σουίτα Περιπτώσεων Ελέγχου (ΣΠΕ), με στόχο τη μεγιστοποίηση κάποιου κριτηρίου, όπως ο ρυθμός ανίχνευσης σφαλμάτων των περιπτώσεων ελέγχου [48]. Στην περίπτωση που η εκτέλεση του συνόλου της ΣΠΕ διακοπεί ή σταματήσει για οποιοδήποτε λόγο, οι σημαντικότερες περιπτώσεις ελέγχου (όσον αφορά στο κριτήριο που χρησιμοποιείται), θα έχουν εκτελεστεί πρώτες».

Ένας πιο τυπικός ορισμός είναι ο παρακάτω [34]: Έστω T μία σουίτα περιπτώσεων ελέγχου, PT, το σύνολο των μεταθέσεων του T και f μία συνάρτηση από το PT στους πραγματικούς αριθμούς. Να βρεθεί:

$$T' \in PTs.t(\forall T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$$

όπου το PT είναι το σύνολο των πιθανών μεταθέσεων του T και f είναι μία συνάρτηση η οποία καθορίζει την απόδοση της κάθε μετάθεσης των περιπτώσεων ελέγχου. Ο ορισμός της απόδοσης ενδέχεται να ποικίλλει, αφού διαφορετικοί προγραμματιστές θα έχουν διαφορετικούς στόχους σε διαφορετικές χρονικές στιγμές [34].

Οι διαθέσιμες τεχνικές προτεραιοποίησης περιπτώσεων ελέγχου είναι πολυάριθμες [50]. Κάθε τεχνική έχει πρόσβαση σε πληροφορία διαφορετικού τύπου και να χρησιμοποιεί διαφορετικές στρατηγικές για την επίτευξη του στόχου της προτεραιοποίησης. Οι τεχνικές μπορούν να κατηγοριοποιηθούν τόσο σύμφωνα με τους πόρους που χρησιμοποιούνται ως πληροφορία εισόδου, όσο και με τις ευρετικές μεθόδους και στρατηγικές βελτιστοποίησής τους.

Για την πραγματοποίηση της ΠΠΕ, έχουν προταθεί αρκετές τεχνικές [11, 12, 34, 42, 48]. Η πιο διαδεδομένη τεχνική στη βιβλιογραφία, αλλά και στην πράξη είναι η χρήση πληροφορίας κάλυψης κώδικα [35, 50, 34]. Αυτή μπορεί να βελτιωθεί αν είναι γνωστές λεπτομέρειες σχετικά με το ποια κομμάτια κώδικα έχουν μεταβληθεί. Η πρακτική αυτή έχει αποδειχθεί ιδιαίτερα αποδοτική σε συστήματα πολύ μεγάλου μεγέθους στη Microsoft [42]. Ωστόσο, η αποδοτικότητα της μεθόδου δεν είναι σταθερή και εξαρτάται από διάφορους παράγοντες, όπως τα χαρακτηριστικά της σουίτας ελέγχου [14].

2.5 Κάλυψη κώδικα

Η κάλυψη κώδικα, είναι μια ευρέως διαδεδομένη μετρική, η οποία χρησιμοποιείται για να περιγράψει το βαθμό στον οποίο εκτελείται ο πηγαίος κώδικας ενός προγράμματος, όταν εκτελείται μια σουίτα ελέγχου πάνω στον ίδιο κώδικα. Η πρώτη αναφορά σε αυτή τη μετρική έχει πραγματοποιηθεί στη βιβλιογραφία ήδη από το 1963 [32].

2.5.1 Δυναμική και στατική ανάλυση κώδικα

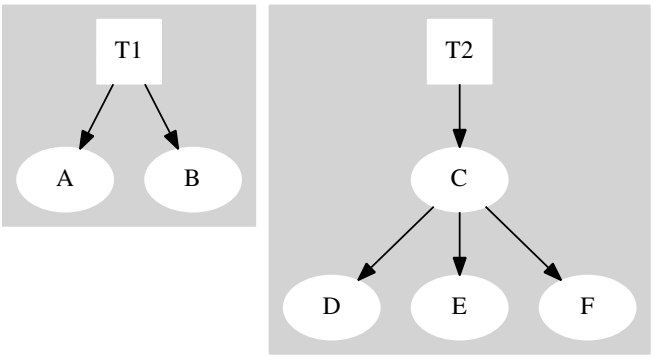
Η κάλυψη κώδικα μπορεί να μετρηθεί σε διαφορετικά επίπεδα, όπως σε επίπεδο κλάσης, μεθόδου, διακλάδωσης κώδικα ή ακόμα και σε επίπεδο εντολής. Η πληροφορία για την κάλυψη κώδικα είναι δυνατό να εξαχθεί είτε δυναμικά, δηλαδή με την εκτέλεση του κώδικα, είτε στατικά, χρησιμοποιώντας δηλαδή τον πηγαίο κώδικα του λογισμικού και των περιπτώσεων ελέγχου. Η δυναμική ανάλυση κώδικα είναι περισσότερο ακριβής, αλλά απαιτεί την εκτέλεση των περιπτώσεων ελέγχου. Από τη φύση του προβλήματος της προτεραιοποίησης περιπτώσεων ελέγχου όμως, αυτό δεν είναι δυνατό, οπότε πραγματοποιείται μόνο αν η πληροφορία της κάλυψης είναι ήδη διαθέσιμη από προηγούμενες

εκτελέσεις του κώδικα [23]. Προφανώς αν η πληροφορία κάλυψης είναι διαθέσιμη από προηγούμενες εκτελέσεις του κώδικα, αυτή ίσως να είναι ανακριβής, αφού για νέες περιπτώσεις ελέγχου δεν υπάρχει καμία πληροφορία, ενώ για προϋπάρχουσες περιπτώσεις ελέγχου ενδεχομένως να έχει καταστεί παρωχημένη.

Όταν δεν είναι δυνατό, για οποιοδήποτε λόγο, να πραγματοποιηθεί δυναμική ανάλυση του κώδικα, τότε η πληροφορία της κάλυψης κώδικα μπορεί μόνο να εξαχθεί από τη στατική ανάλυσή του. Για παράδειγμα, η κάλυψη κώδικα των περιπτώσεων ελέγχου μπορεί να υπολογιστεί εξάγοντας την ακολουθία των κλήσεων των μεθόδων του πηγαίου κώδικα, που αντιστοιχούν στις συγκεκριμένες περιπτώσεις ελέγχου [30]. Σε αυτή την περίπτωση, η διαθεσιμότητα σεναρίων ελέγχου, τα οποία εκτελούν συγκεκριμένες μεθόδους, είναι απαραίτητη. Αυτό, πιθανόν να μην είναι αληθές πάντα, όπως για περιπτώσεις που τα σενάρια ελέγχου γράφονται σε φυσική γλώσσα [21]. Επιπρόσθετα, όπως αναφέρθηκε προηγουμένως, η στατική ανάλυση κώδικα δεν είναι τόσο ακριβής όσο η δυναμική εκτέλεσή του, όσον αφορά στον υπολογισμό της κάλυψης κώδικα.

Υπάρχουν δύο βασικοί τρόποι για τη διενέργεια στατικής ανάλυσης κώδικα και για την εξαγωγή της πληροφορίας της κάλυψης. Ο ταχύτερος, αλλά λιγότερο ακριβής τρόπος, είναι η ανάλυση των σεναρίων ελέγχου και η εξαγωγή μόνο του πρώτου επιπέδου των μεθόδων που καλούνται. Ο περισσότερο ακριβής τρόπος, που είναι όμως και λιγότερο ταχύς, ακολουθεί την εκτέλεση των μεθόδων του πρώτου επιπέδου στον πηγαίο κώδικα και εξάγει όλες τις εμφωλευμένες κλήσεις μεθόδων από αυτόν [21]. Για παράδειγμα, μία περίπτωση ελέγχου T1, μπορεί να καλεί άμεσα τις μεθόδους A και B, ενώ μια περίπτωση ελέγχου T2, μπορεί να καλεί άμεσα μια μέθοδο C (σχ. 1). Αν μία στατική ανάλυση η οποία πραγματοποιείται σε επίπεδο μεθόδου, φτάνει μόνο μέχρι το πρώτο επίπεδο κλήσεων, τότε η περίπτωση ελέγχου T1 θα πρέπει να τοποθετείται πριν από την T2, αφού καλύπτει περισσότερες μεθόδους. Αν όμως η μέθοδος C, καλεί με τη σειρά της, τις μεθόδους D, E και F, ενώ οι μέθοδοι A και B δεν καλούν καμία επιπρόσθετη μέθοδο, τότε αν η στατική ανάλυση κώδικα έφτανε στα δύο επίπεδα, η περίπτωση ελέγχου T2 θα έπρεπε να τοποθετείται πριν από την T1, αφού τελικά αυτή καλύπτει περισσότερες μεθόδους.

Η πληροφορία κάλυψης κώδικα που λαμβάνεται με οποιοδήποτε τρόπο, για κάθε περίπτωση ελέγχου, χρησιμοποιείται στη συνέχεια για την ταξινόμηση των περιπτώσεων ελέγχου. Ο στόχος είναι οι περιπτώσεις ελέγχου να ταξινομηθούν, ώστε η κάλυψη κώ-



Σχήμα 1: Παράδειγμα εκτέλεσης μεθόδων από δύο περιπτώσεις ελέγχου (T1 και T2)

δικα που παρέχουν οι εξεταζόμενες περιπτώσεις ελέγχου, να καλύπτει μεγαλύτερο ποσοστό του συνολικού κώδικα της εφαρμογής το συντομότερο δυνατό, εκτελώντας δηλαδή λιγότερες περιπτώσεις ελέγχου. Έτσι, αφού καλύπτεται μεγαλύτερο ποσοστό του εξεταζόμενου κώδικα νωρίτερα, αυξάνονται και οι πιθανότητες να εντοπιστούν περισσότερα σφάλματα, επίσης νωρίτερα [26].

2.5.2 Στρατηγικές βελτιστοποίησης

Το πρόβλημα της μεγιστοποίησης της κάλυψης κώδικα από τις εκτελούμενες περιπτώσεις ελέγχου είναι στην πραγματικότητα ένα πρόβλημα βελτιστοποίησης. Για την επίλυση του προβλήματος αυτού, υπάρχουν διαφορετικές τεχνικές και προσεγγίσεις.

Άπληστος αλγόριθμος

Η απλούστερη από αυτές τις τεχνικές είναι η χρήση ενός άπληστου αλγόριθμου [13]. Ο άπληστος αλγόριθμος εκτελείται σύμφωνα με τα παρακάτω βήματα:

1. Εξέταση όλων των περιπτώσεων ελέγχου που βρίσκονται στη σουίτα ελέγχου και υπολογισμός της κάλυψης κώδικα για κάθε μία από αυτές.

2. Ταυτοποίηση της περίπτωσης ελέγχου με τη μεγαλύτερη κάλυψη κώδικα. Σε περίπτωση ισοτήτων μπορεί να χρησιμοποιηθεί κάποια στρατηγική επίλυσής τους, όπως η τυχαία επιλογή.

3. Αφαίρεση της περίπτωσης ελέγχου από τη σουίτα ελέγχου και τοποθέτησή της στη επόμενη θέση της λίστας των περιπτώσεων ελέγχου

4. Αν η σουίτα ελέγχου δεν είναι άδεια, επιστροφή στο βήμα 2.

Πίνακας 1: Παράδειγμα κάλυψης κώδικα τριών περιπτώσεων ελέγχου σε επίπεδο μεθόδου (το σύμβολο ✓δείχνει ότι η περίπτωση ελέγχου Τ εκτελεί τη μέθοδο Μ)

	T1	T2	T3
M1	✓	✓	
M2	✓	✓	
M3	✓	✓	
M4	✓		
M5			✓

Αν και ο άπληστος αλγόριθμος είναι απλός και πρακτικός, έχει κάποιες αδυναμίες. Το κυριότερο πρόβλημά του είναι ότι δεν λαμβάνει υπόψη το ποια κομμάτια κώδικα καλύπτονται από τις περιπτώσεις ελέγχου που έχουν ήδη επιλεχθεί να εκτελεστούν πρώτες.

Ένα παράδειγμα που δείχνει το πρόβλημα αυτό είναι το παρακάτω: έστω ότι υπάρχει μια σουίτα ελέγχου η οποία περιλαμβάνει τρεις περιπτώσεις ελέγχου, τις T1, T2 και T3. Έστω επίσης ότι ο πηγαίος κώδικας περιλαμβάνει πέντε μεθόδους, τις M1, M2, M3, M4 και M5. Ας υποθέσουμε επίσης ότι η κάλυψη κώδικα μετριέται σε επίπεδο μεθόδου. Η T1 εκτελεί τις μεθόδους M1, M2, M3 και M4. Η T2 εκτελεί τις μεθόδους M1, M2 και M3, ενώ η T3 εκτελεί μόνο τη M5 (πίνακας 1).

Σε αυτή την περίπτωση, η άπληστος αλγόριθμος θα επιλέξει για εκτέλεση πρώτη την περίπτωση ελέγχου T1, αφού αυτή έχει συνολικά τη μεγαλύτερη κάλυψη, αφού καλύπτει τέσσερις από τις πέντε μεθόδους. Στη συνέχεια, θα επιλέξει την περίπτωση ελέγχου T2, ενώ τελευταία θα αφήσει την περίπτωση ελέγχου T3, καταλήγοντας στη σειρά κατάταξης [T1, T2, T3]. Εύκολα φαίνεται πως, αν και με την εκτέλεση μίας και μόνο περίπτωσης ελέγχου, της T1, καλύπτεται το 80% των διαθέσιμων μεθόδων, για να καλυφθεί το 100%, θα πρέπει να εκτελεστούν και οι τρεις περιπτώσεις ελέγχου. Μια εναλλακτική κατάταξη θα μπορούσε να είναι η [T1, T3, T2]. Με αυτό τον τρόπο, το 100% της κάλυψης επιτυγχάνεται μόνο με την εκτέλεση των δύο εκ των τριών περιπτώσεων ελέγχου.

Έχοντας υπόψη το παραπάνω παράδειγμα, μια παραλλαγή του άπληστου αλγορίθμου για την κατάταξη των περιπτώσεων ελέγχου είναι η χρήση της «επιπρόσθετης κάλυψης» [34]. Σε αυτή την περίπτωση, δεν λαμβάνεται υπόψη η συνολική κάλυψη κώδικα της κάθε περίπτωσης ελέγχου, αλλά μόνο αυτή που δεν έχει καλυφθεί από τις περιπτώσεις ελέγχου που έχουν επιλεγεί ήδη. Έτσι, για το παράδειγμα του πίνακα 1, επιλέγεται αρχικά η T1, αφού έχει κάλυψη 80%. Στη συνέχεια όμως, η επιπρόσθετη κάλυψη της T2 είναι 0%, αφού όλες οι μέθοδοι που εκτελεί, περιλαμβάνονται σε αυτές που θα εκτε-

Πίνακας 2: Παράδειγμα αποτυχίας εύρεσης βέλτιστης λύσης από τον άπληστο αλγόριθμο (το σύμβολο ✓δείχνει ότι η περίπτωση ελέγχου Τ εκτελεί τη μέθοδο Μ)

	T1	T2	T3	T4
M1	✓	✓		
M2		✓		
M3	✓		✓	
M4			✓	
M5	✓			✓
M6				✓

λεστούν από την Τ1. Αντίθετα, η Τ3 έχει επιπρόσθετη κάλυψη ίση με 20%, οπότε θα τοποθετηθεί δεύτερη στη σειρά. Με αυτό τον τρόπο, επιτυγχάνεται συνολική κάλυψη κώδικα 100% μόνο με την εκτέλεση δύο περιπτώσεων ελέγχου.

Ολικές και τοπικές βέλτιστες τιμές

Όμως, και οι δύο παραλλαγές του άπληστου αλγορίθμου παρουσιάζουν το ίδιο πρόβλημα, αυτό της τοπικής βέλτιστης τιμής. Ως γνωστό, στον τομέα της βελτιστοποίησης, μια τοπική βέλτιστη τιμή είναι μια λύση, η οποία είναι η βέλτιστη, είτε αυτό σημαίνει η ελάχιστη ή η μέγιστη, σε ένα δεδομένο εύρος του πεδίου ορισμού μιας συνάρτησης βελτιστοποίησης. Αντίθετα, μία ολική βέλτιστη τιμή είναι η βέλτιστη μεταξύ όλων των πιθανών λύσεων της συνάρτησης βελτιστοποίησης. Ένας άπληστος αλγόριθμος επιλέγει τη τοπικά βέλτιστη λύση σε κάθε βήμα του, οπότε σε αρκετά προβλήματα, δεν κατορθώνει να εντοπίσει την ολική βέλτιστη τιμή [9].

Ως παράδειγμα δίνεται αυτό του πίνακα 2. Ο άπληστος αλγόριθμος, στο πρώτο βήμα του, θα επιλέγει την περίπτωση ελέγχου Τ1, αφού σε οποιαδήποτε περίπτωση, αυτή καλύπτει μεγαλύτερο μέρος του κώδικα από οποιαδήποτε άλλη, ίσο με το 50% της συνολικής κάλυψης. Στη συνέχεια, και σε κάθε βήμα, η επιλογή μεταξύ των περιπτώσεων ελέγχου Τ2, Τ3 και Τ4 γίνεται με τυχαίο τρόπο, αφού η κάθε μία από αυτές προσθέτουν μία επιπλέον μέθοδο στον κώδικα που έχει καλυφθεί. Σε οποιαδήποτε περίπτωση, με την επιλογή της περίπτωσης ελέγχου Τ1 στο αρχικό βήμα, για να επιτευχθεί ποσοστό κάλυψης ίσο με 100%, απαιτείται να εκτελεστούν και οι τέσσερις περιπτώσεις ελέγχου. Εύκολα φαίνεται πως συνολική κάλυψη του ελεγχόμενου κώδικα θα μπορούσε επίσης να επιτευχθεί μόνο με την εκτέλεση των περιπτώσεων ελέγχου Τ2, Τ3 και Τ4, με οποιαδήποτε σειρά. Η αθροιστική κάλυψη κώδικα για τις δύο αναφερθείσες διατάξεις φαίνεται στον πίνακα 3.

Πίνακας 3: Αθροιστική κάλυψη κώδικα για δύο διαφορετικές διατάξεις περιπτώσεων ελέγχου

Διάταξη 1		Διάταξη 2	
Περίπτωση ελέγχου	Αθροιστική κάλυψη	Περίπτωση ελέγχου	Αθροιστική κάλυψη
T1	50%	T2	33.33%
T2	66.67%	T3	66.67%
T3	83.34%	T4	100%
T4	100%	T1	100%

2.5.3 Τεχνικές αναζήτησης

Η λύση στο πρόβλημα του εντοπισμού της βέλτιστης διάταξης των περιπτώσεων ελέγχου είναι η χρήση τεχνικών καθολικής αναζήτησης [2]. Όμως το πρόβλημα του εντοπισμού της διάταξης των περιπτώσεων ελέγχου που θα οδηγήσουν στη μεγιστοποίηση της κάλυψης κώδικα το ταχύτερο δυνατό, κατηγοριοποιείται ως NP-σκληρό [29]. Οπότε η χρήση κάποιας τεχνικής εξαντλητικής αναζήτησης για τη βέλτιστη διάταξη, δοκιμάζοντας δηλαδή όλες τις δυνατές διατάξεις των περιπτώσεων ελέγχου, δεν είναι εφικτή. Ο χώρος αναζήτησης σε ένα τέτοιο πρόβλημα είναι ίσος με τον αριθμό των δυνατών διατάξεων, οι οποίες για n περιπτώσεις ελέγχου, είναι ίσες με $n!$.

Για την αξιολόγηση των διαφορετικών διατάξεων χρησιμοποιείται μια συνάρτηση προσαρμογής. Μέσω αυτής, ανατίθεται σε κάθε διάταξη μία τιμή. Θα πρέπει να βρεθεί η διάταξη αυτή που μεταξύ όλων των υπολοίπων καταλήγει στη βέλτιστη τιμή. Ένας αλγόριθμος αναζήτησης μπορεί να εκτελείται για οποιοδήποτε χρονικό διάστημα, ενώ όσο περισσότερο χρησιμοποιείται, τόσο μεγαλύτερο μέρος του χώρου αναζήτησης μπορεί να καλύψει. Δυστυχώς, δεν είναι εφικτό να είναι γνωστό αν κάποιο σημείο αποτελεί ολικό βέλτιστο, γιατί κάτι τέτοιο θα είχε ως προϋπόθεση την εξάντληση του χώρου αναζήτησης. Οπότε, θα πρέπει να οριστούν κάποια κριτήρια, σύμφωνα με τα οποία η αναζήτηση θα πρέπει να σταματήσει. Αυτά μπορεί να είναι είτε χρονικοί περιορισμοί, είτε η αξιολόγηση κάποιου ορισμένου από την αρχή αριθμού διατάξεων.

Υπάρχουν αρκετοί αλγόριθμοι που έχουν τη δυνατότητα να πραγματοποιήσουν καθολική αναζήτηση, όπως για παράδειγμα, γενετικοί αλγόριθμοι (Genetic Algorithms), η προσομοιωμένη ανόπτηση (Simulated Annealing), και βελτιστοποίηση αποικίας μυρμηγκιών (Ant Colony Optimization) [2]. Κατά μέσο όρο, σε όλα τα προβλήματα, όλοι οι αλγόριθμοι αναζήτησης αποδίδουν το ίδιο, κάτι το οποίο έχει αποδειχθεί σε θεωρητικό επίπεδο [46]. Παρ' όλα αυτά, για συγκεκριμένα προβλήματα, ενδέχεται να παρουσιάζον-

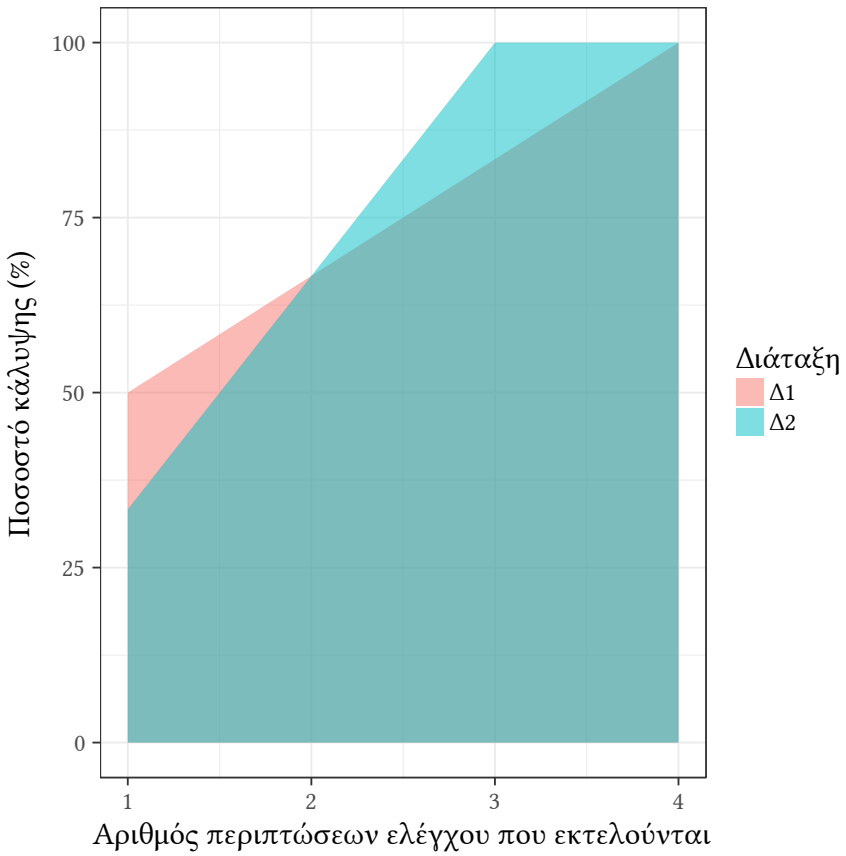
ται σημαντικές αποκλίσεις στις επιδόσεις των αλγορίθμων αναζήτησης. Έχουν πραγματοποιηθεί αρκετές συγκρίσεις στη βιβλιογραφία, όσον αφορά στην απόδοση των αλγορίθμων αναζήτησης σε προβλήματα ελέγχου λογισμικού [50, 22, 7]. Συγκεκριμένα, για την περίπτωση των προβλημάτων προτεραιοποίησης περιπτώσεων ελέγχου, απαιτείται ένας αλγόριθμος καθολικής αναζήτησης, ο οποίος δεν εγκλωβίζεται σε τοπικές βέλτιστες λύσεις. Μία τέτοια κατηγορία αλγορίθμων είναι οι γενετικοί αλγόριθμοι. Κατά μέσο όρο, στα περισσότερες μελέτες προτεραιοποίησης στη βιβλιογραφία, οι αλγόριθμοι αυτοί δίνουν καλύτερα αποτελέσματα σε σχέση με άπληστους αλγορίθμους, ή με αλγορίθμους τοπικής αναζήτησης όπως αυτός της αναρρίχησης λόφων (Hill Climbing) [21].

Περιοχή κάτω από την καμπύλη

Η περισσότερο διαδεδομένη επιλογή για τη συνάρτηση προσαρμογής που μπορεί να χρησιμοποιηθεί για την αξιολόγηση διαφορετικών τεχνικών προτεραιοποίησης περιπτώσεων ελέγχου είναι αυτή της περιοχής κάτω από την καμπύλη (Area Under the Curve – AUC). Αυτή αναφέρεται στο εμβαδό της περιοχής που βρίσκεται κάτω από την καμπύλη, η οποία ορίζεται από το ποσοστό κάλυψης κάθε περίπτωσης ελέγχου που εκτελείται στη σειρά [13]. Όπως αναφέρθηκε, ένας τρόπος για την αξιολόγηση μιας διάταξης περιπτώσεων ελέγχου είναι η επίτευξη ποσοστού κάλυψης 100%, ή σε οποιαδήποτε περίπτωση του μέγιστου ποσοστού που μπορεί να επιτευχθεί από τη συγκεκριμένη σουίτα περιπτώσεων ελέγχου, το συντομότερο δυνατό, δηλαδή με την εκτέλεση του ελάχιστου αριθμού περιπτώσεων ελέγχου. Μια άλλη πρακτική όμως είναι η επιλογή και εκτέλεση ενός υποσυνόλου των διαθέσιμων περιπτώσεων ελέγχου. Με τη χρήση της AUC, είναι δυνατό να επιλεχθεί η βέλτιστη διάταξη περιπτώσεων ελέγχου για οποιοδήποτε πλήθος περιπτώσεων ελέγχου που μπορεί να εκτελεστούν.

Έτσι, για το παράδειγμα των περιπτώσεων ελέγχου του πίνακα 2, μπορούμε να έχουμε τις διατάξεις Δ1: [T1, T2, T3, T4] και Δ2: [T2, T3, T4, T1]. Τα αθροιστικά ποσοστά κάλυψης μετά από την εκτέλεση κάθε μίας από τις περιπτώσεις ελέγχου εμφανίζονται όπως είδαμε στον πίνακα 3. Αν αυτά σχεδιαστούν, προκύπτει το σχήμα 2.

Η AUC μπορεί εύκολα να υπολογιστεί σε αυτή την περίπτωση, χρησιμοποιώντας το



Σχήμα 2: Παράδειγμα υπολογισμού της AUC για δύο διαφορετικές διατάξεις της ίδιας σουίτας ελέγχου

γεωμετρικό τύπο για τον υπολογισμό του εμβαδού ενός τραπεζίου [80]:

$$E = \frac{(a+b)\cdot h}{2}$$

όπου:

- a το μήκος της μικρής βάσης
- b το μήκος της μεγάλης βάσης
- h το ύψος του τραπεζίου

Αυτό είναι δυνατό, γιατί η AUC αποτελεί ουσιαστικά το συνδυασμό ενός ή περισσότερων τραπεζίων.

Οι τιμές της AUC, όπως υπολογίζονται για τις δύο διατάξεις, Δ1 και Δ2, είναι αντίστοιχα ίσες με 225 και 233.34. Οπότε σύμφωνα με την AUC η διάταξη Δ2 έχει μεγαλύτερο εμβαδό και είναι προτιμότερη από τη Δ1.

2.5.4 Αξιολόγηση εκτέλεσης διατάξεων περιπτώσεων ελέγχου

Για την αξιολόγηση των διαφορετικών διατάξεων, ως προς την ικανότητά τους να εντοπίζουν σε μια σουίτα ελέγχου, όσο περισσότερα σφάλματα το συντομότερο δυνάτό, η περισσότερο κοινή μετρική που εμφανίζεται στη βιβλιογραφία, είναι η *APFD* (Average Percentage of Faults Detected) [11]. Αυτή υπολογίζει το μέσο αριθμό των σφαλμάτων που ανιχνεύονται με μια διάταξη των περιπτώσεων ελέγχου. Για τον υπολογισμό της APFD χρησιμοποιείται ο εξής μαθηματικός τύπος:

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

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.

Η μετρική APFD λαμβάνει τιμές μεταξύ του 0% και του 100%. Υψηλότερες τιμές συνεπάγονται γρηγορότερους, άρα και καλύτερους, ρυθμούς ανίχνευσης σφαλμάτων [34], αφού αυτά εντοπίζονται στις πρώτες περιπτώσεις ελέγχου που εκτελούνται. Αντίθετα, μικρότερες τιμές του APFD συνεπάγονται πιο αργούς ρυθμούς ανίχνευσης σφαλμάτων, αφού αυτά εντοπίζονται κυρίως στις τελευταίες περιπτώσεις ελέγχου που εκτελούνται. Για την εύκολη οπτικοποίηση των αποτελεσμάτων χρησιμοποιούνται συνήθως γραφήματα, όπου στον άξονα X εμφανίζεται το κλάσμα της σουίτας ελέγχου που έχει εκτελεστεί, ενώ στον άξονα Y εμφανίζεται το ποσοστό των σφαλμάτων που έχει εκτελεστεί. Η τιμή της μετρικής APFD, σε αυτή την περίπτωση αντιστοιχεί στο εμβαδό της περιοχής κάτω από την καμπύλη που σχηματίζεται.

Ως παράδειγμα χρήσης της μετρικής APFD παρουσιάζεται αυτό του πίνακα 4 [34]. Σε αυτόν, υπάρχουν πέντε διαφορετικές περιπτώσεις ελέγχου, οι A–E, καθεμία από τις οποίες εντοπίζει τα σημασμένα σφάλματα.

Αν εκτελεστούν οι περιπτώσεις ελέγχου σύμφωνα με τη διάταξη Δ1: [A, B, C, D, E], τότε προκύπτουν τα αποτελέσματα που εμφανίζονται στο σχήμα 3. Αυτά δείχνουν, ότι μετά την εκτέλεση της περίπτωσης ελέγχου A, ανιχνεύονται δύο από τα δέκα σφάλματα

Πίνακας 4: 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								✓	✓	✓

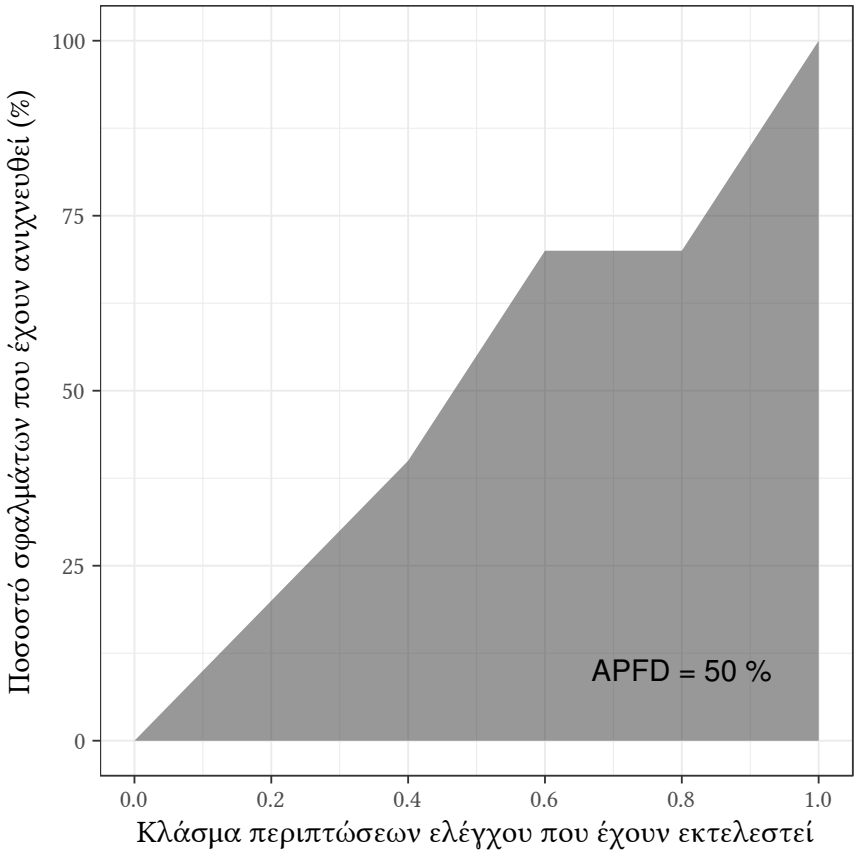
που υπάρχουν. Δηλαδή το 20% των σφαλμάτων έχει ανιχνευθεί αφού εκτελέστηκε κλάσμα της σουίτας ελέγχου ίσου με 0.2. Μετά την εκτέλεση της περίπτωσης ελέγχου B, ανιχνεύονται ακόμα δύο σφάλματα, οπότε το 40% των σφαλμάτων έχει ανιχνευθεί αφού εκτελέστηκε κλάσμα της σουίτας ελέγχου ίσο με 0.4 κ.ο.κ. Στο σχήμα 3 η σκιασμένη επιφάνεια, η οποία αντιστοιχεί στην περιοχή κάτω από την καμπύλη, αντιπροσωπεύει το σταθμισμένο μέσο του ποσοστού των σφαλμάτων που ανιχνεύονται καθ' όλη τη διάρκεια εκτέλεσης της σουίτας ελέγχου, η οποία είναι και η τιμή της μετρικής APFD, που σε αυτή την περίπτωση ισούται με 50%.

Αντίστοιχα, για τις διατάξεις Δ2: [E, D, C, B, A] και Δ3: [C, E, B, A, D] προκύπτουν τα αποτελέσματα που εμφανίζονται στα σχήματα 4 και 5, με τιμές για τη μετρική APFD ίσες με 64% και 84%. Είναι φανερό πως η τελευταία διάταξη είναι αυτή που ανιχνεύει τα σφάλματα νωρίτερα από τις υπόλοιπες δύο.

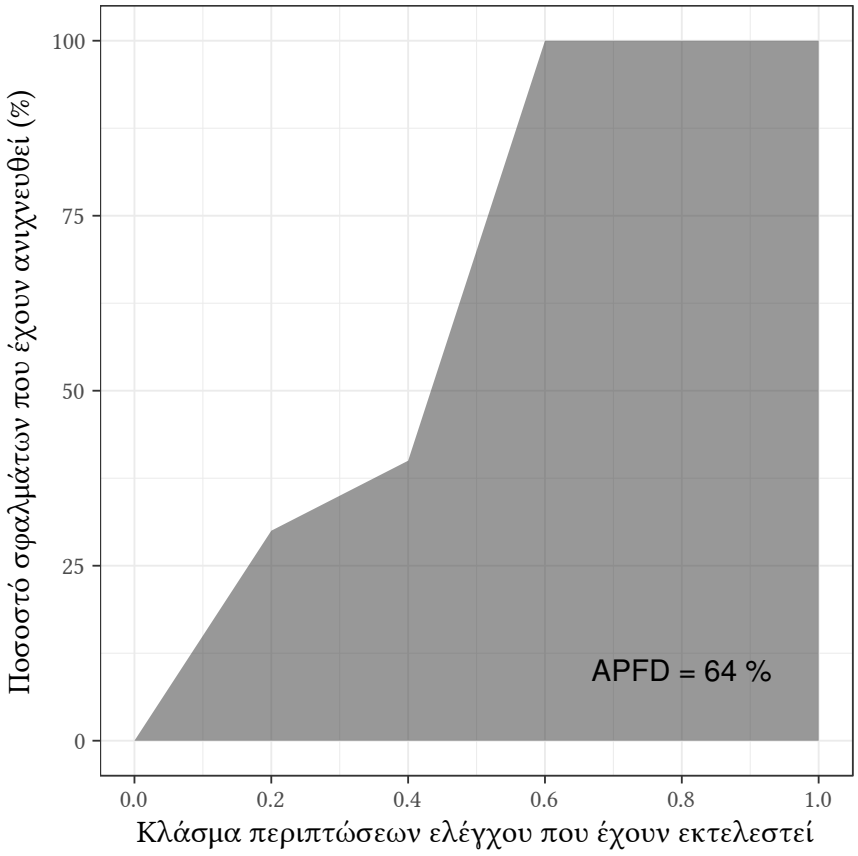
Υπάρχουν δύο παράγοντες που δεν λαμβάνονται υπόψιν με τη χρήση της μετρικής APFD. Ο πρώτος είναι το κόστος εκτέλεσης των διαφορετικών περιπτώσεων ελέγχου, το οποίο εκφράζεται συνήθως σε μονάδες χρόνου. Μία περίπτωση ελέγχου που απαιτεί πολλαπλάσιο χρόνο εκτέλεσης σε σχέση με κάποια άλλη, θα πρέπει να έχει αντίστοιχα μεγαλύτερη βαρύτητα από αυτή [21]. Άλλες μετρήσεις κόστους ενδέχεται να αναφέρονται στο σχετικό μέγεθος της κάθε περίπτωσης ελέγχου σε σχέση με το συνολικό μέγεθος της σουίτας ελέγχου ή το χρόνο ανίχνευσης σφαλμάτων [25].

Ο δεύτερος παράγοντας είναι η δριμύτητα των σφαλμάτων που εκθέτουν οι περιπτώσεις ελέγχου. Κάποια σφάλματα θεωρούνται αναμφισβήτητα περισσότερο σοβαρά σε σχέση με κάποια άλλα. Η πληροφορία της δριμύτητας θα πρέπει βέβαια να προέρχεται από κάποιο εξωτερικό παράγοντα και μπορεί να αναφέρεται στην πολυπλοκότητα, την ασφάλεια ή και το μέγεθος των απαιτήσεων του λογισμικού [24].

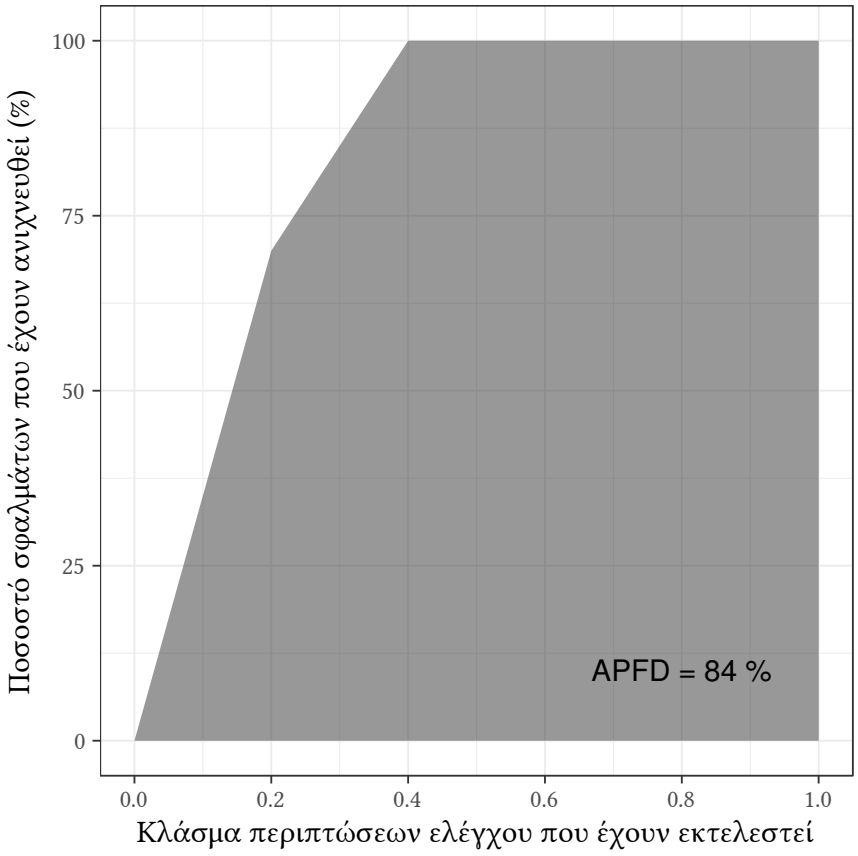
Η προτεινόμενη λύση, ώστε να ληφθούν αυτοί οι παράγοντες υπόψιν κατά την προ-



Σχήμα 3: Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου Δ1: [A, B, C, D, E] του πίνακα 4



Σχήμα 4: Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου Δ2: [E, D, C, B, A] του πίνακα 4



Σχήμα 5: Υπολογισμός APFD για τη διάταξη περιπτώσεων ελέγχου Δ3: [C, E, B, A, D] του πίνακα 4

τεραιοποίηση των περιπτώσεων ελέγχου είναι απλή: μπορεί να χρησιμοποιηθεί μια σταθμισμένη παραλλαγή της μετρικής APFD. Μία τέτοια παραλλαγή είναι η μετρική APFDc (c για cost) [11]. Αυτή βέβαια, χρησιμοποιείται κυρίως για την αξιολόγηση διαφορετικών τεχνικών προτεραιοποίησης, μετά την εκτέλεσή τους, οπότε και γίνονται γνωστά τα κόστη. Για να χρησιμοποιηθεί για την πρόβλεψη των αποτελεσμάτων ως προς τους ρυθμούς ανίχνευσης σφαλμάτων που μπορούν να επιτευχθούν, θα πρέπει αυτά τα κόστη να είναι δυνατό να προβλεφθούν. Βέβαια, η πληροφορία αυτή θα μπορούσε να είναι διαθέσιμη από προηγούμενες εκτελέσεις των περιπτώσεων ελέγχου.

2.6 Η γλώσσα προγραμματισμού Python

Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου και γενικού σκοπού. Δημιουργήθηκε αρχικά το 1991 και έκτοτε έχει γίνει ιδιαίτερα δημοφιλής. Τον Ιούνιο του 2019, αποτελεί την τρίτη πιο δημοφιλή γλώσσα παγκοσμίως, σύμφωνα με τον δείκτη TIOBE [75]. Ο δείκτης TIOBE (The Importance Of Being Earnest) χρησιμοποιείται ευρύτατα για την εκτίμηση της δημοτικότητας όλων των γλωσσών προγραμματισμού παγκοσμίως και για το σκοπό αυτό χρησιμοποιεί τον αριθμό των αναζητήσεων που πραγματοποιούνται σε διάφορες μηχανές αναζήτησης, οι οποίες περιλαμβάνουν την ονομασία της κάθε γλώσσας προγραμματισμού, τον αριθμό των έμπειρων προγραμματιστών που υπάρχουν σε όλο τον κόσμο, τον αριθμό των διαθέσιμων μαθημάτων, διαδικτυακών και μη, κ.ά.. Η γλώσσα προγραμματισμού Python, βρίσκεται σήμερα στην υψηλότερη θέση που έχει βρεθεί ποτέ, με μόνες πιο δημοφιλείς γλώσσες από αυτή να είναι η Java και η C [75]. Αναφέρεται πως, αν ο ρυθμός εξάπλωσης της Python συνεχιστεί, αναμένεται να ξεπεράσει σε δημοτικότητα και τις δύο αυτές πολύ σημαντικές γλώσσες προγραμματισμού σε 3 με 4 έτη.

Παρόμοιες αναφορές δημοσιοποιούνται και από την ιδιαίτερα δημοφιλή ιστοσελίδα επίλυσης προβλημάτων προγραμματισμού Stack Overflow [72]. Η Python εμφανίζεται κι εκεί ως η πλέον αναπτυσσόμενη γλώσσα προγραμματισμού, εκτοπίζοντας τη Java, ενώ ταυτόχρονα είναι η δεύτερη πιο αγαπημένη γλώσσα μεταξύ των προγραμματιστών, πίσω μόνο από τη Rust.

Ο κύριος λόγος της ιδιαίτερα αυξημένης δημοτικότητας της Python, είναι ότι με την ανάπτυξη του χώρου της πληροφορικής, αυτός ελκύει μεγάλους αριθμούς νέων προγραμματιστών. Η Python θεωρείται μια ιδιαίτερα προσιτή γλώσσα, πολύ πιο εύκολη

στην εκμάθηση των βασικών εννοιών του προγραμματισμού σε σχέση με την Java ή τη C. Ο χρόνος που απαιτείται για ένα νέο χρήστη να νιώσει ότι δημιουργεί κάτι χρήσιμο είναι ελάχιστος, οπότε είναι ένας προφανής προορισμός για όλους αυτούς τους νέους προγραμματιστές.

Ταυτόχρονα, η ιδιαίτερα πλούσια τυπική βιβλιοθήκη της, διευκολύνει ιδιαίτερα πολλές συνηθισμένες εργασίες, ενώ χιλιάδες ακόμα βιβλιοθήκες, σχεδόν για οποιοδήποτε σκοπό, είναι διαθέσιμες μέσω του αποθετηρίου PyPI (Python Package Index) [63].

Η γλώσσα Python, πέρα από τον μεγάλο αριθμό έργων λογισμικού ανοιχτού κώδικα, χρησιμοποιείται επίσης ευρύτατα από μεγάλες εταιρίες ανάπτυξης λογισμικού. Ενδεικτικά αναφέρεται πως το μέγεθος του υλοποιημένου κώδικα σε γλώσσα Python σε εταιρείες όπως η Facebook ή η Dropbox ανέρχεται σε εκατομμύρια γραμμές κώδικα [56, 55].

Η βασική υλοποίηση της γλώσσας Python, είναι η CPython, η οποία είναι υλοποιημένη σε γλώσσα C. Η CPython παρέχει ένα διερμηνευτή (interpreter) για τη γλώσσα. Αποτελεί ελεύθερο λογισμικό και αναπτύσσεται από πλήθος προγραμματιστών σε όλο τον κόσμο. Βιβλιοθήκες επέκτασης της λειτουργικότητας της γλώσσας είναι δυνατό να γραφτούν με τη χρήση της ίδιας της γλώσσας Python, αλλά και με τη χρήση γλωσσών όπως η C, η C++ και η Fortran, σε περιπτώσεις που απαιτούνται ταχύτεροι χρόνοι υπολογισμού, όπως είναι π.χ. εφαρμογές που απαιτούν τη διαχείριση μεγάλου όγκου δεδομένων.

Πέρα από την υλοποίηση αναφοράς CPython, υπάρχουν αρκετές εναλλακτικές υλοποιήσεις της Python. Ενδεικτικά αναφέρονται:

- η Jython [73], η οποία είναι μια υλοποίηση της γλώσσας Python στην εικονική μηχανή της Java (Java Virtual Machine - JVM) και η οποία επιτρέπει την άμεση επικοινωνία μεταξύ κώδικα γραμμένου σε Python και κώδικα γραμμένου σε Java,
- η IronPython [59], μια αντίστοιχη υλοποίηση στο .NET,
- η PyPy [64], η οποία είναι μια υλοποίηση της γλώσσας Python γραμμένη σε Python και η οποία μάλλον αποτελεί τη γρηγορότερη υλοποίηση της γλώσσας χάρη στον Just-In-Time compiler που περιλαμβάνει
- η MicroPython [60], η οποία είναι μια υλοποίηση της γλώσσας, προορισμένη και

βελτιστοποιημένη να εκτελείται σε μικροελεγκτές (microcontrollers) όπως το Arduino και το ESP8266.

Βασικά στοιχεία της γλώσσας Python, τα οποία έχουν συμβάλει στην δημοτικότητά της, είναι η δυναμική αναγνώριση των τύπων των μεταβλητών, η σημαντικότητα των εσοχών (indentation) και των «λευκών» χαρακτήρων (κενό, tab), οι οποίοι καθορίζουν την ομαδοποίηση (block) του κώδικα, η αυξημένη αναγνωσιμότητα του κώδικα που προκύπτει, η οποία σε μεγάλο βαθμό οφείλεται στη σημαντικότητα των εσοχών και στο συντακτικό της, το οποίο επιτρέπει την καταγραφή σε κώδικα των εννοιών σε λιγότερες γραμμές κώδικα.

Ανήκει στις γλώσσες προστακτικού προγραμματισμού (imperative programming), ενώ υποστηρίζει τόσο διαδικαστικό προγραμματισμό (procedular programming), παρόμοιο με γλώσσες όπως η C, αλλά και αντικειμενοστρεφή προγραμματισμό (object-oriented programming), όπως η Java, με δυνατότητα ανάμιξης των διαφορετικών τρόπων προγραμματισμού, χωρίς κανένα περιορισμό.

2.7 Το πλαίσιο ελέγχων pytest

Το pytest [65] είναι ένα πλαίσιο εκτέλεσης περιπτώσεων ελέγχου για τη γλώσσα Python. Δεν περιλαμβάνεται στη βασική βιβλιοθήκη της γλώσσας Python, αλλά αποτελεί εξωτερική βιβλιοθήκη που μπορεί να εγκατασταθεί επιπρόσθετα. Ωστόσο, μάλλον αποτελεί το πλέον δημοφιλές πλαίσιο ελέγχων για την Python. Αυτό συμβαίνει κυρίως λόγω των βασικών χαρακτηριστικών του, τα οποία είναι τα παρακάτω:

- η παροχή πληροφοριών με μεγάλη λεπτομέρεια σε σχέση με τις εντολές `assert` που αποτυγχάνουν να εκτελεστούν,
- η αυτόματη ανακάλυψη των περιπτώσεων ελέγχου, σε όποιο αρχείο πηγαίου κώδικα κι αν έχουν τοποθετηθεί αυτές,
- ο υψηλός βαθμός παραμετροποίησης για τη διαχείριση της εκτέλεσης των περιπτώσεων ελέγχου,
- η δυνατότητα εκτέλεσης περιπτώσεων ελέγχου που έχουν γραφτεί για τη βιβλιοθήκη unittest, η οποία περιλαμβάνεται στη βασική βιβλιοθήκη της Python, αλλά και της nose [61], ενός ακόμα δημοφιλούς (στο παρελθόν περισσότερο) πλαισίου ελέγχων για την Python,

Πίνακας 5: Ένα απλό παράδειγμα περίπτωσης ελέγχου με τη χρήση του pytest

```
# content of test_sample.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```

Πίνακας 6: Παράδειγμα εκτέλεσης του pytest

```
$ pytest
===== test session starts =====
collected 1 items

test_sample.py F

===== FAILURES =====
_____ test_answer _____

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:5: AssertionError
===== 1 failed in 0.04 seconds =====
```

- η υποστήριξη για νεότερες εκδόσεις της γλώσσας, CPython 3.5+ και PyPy3 και
- η δυνατότητα επέκτασης της λειτουργικότητάς του μέσω προσθέτων και της πλούσιας συλλογής προσθέτων που έχουν δημιουργηθεί λόγω αυτής.

Ο απλός τρόπος με τον οποίο μπορούν να γραφτούν περιπτώσεις ελέγχου χρησιμοποιώντας το pytest, φαίνεται στο παράδειγμα του πίνακα 5 [65].

Η εκτέλεση όλων των περιπτώσεων ελέγχου που ορίζονται για το κάθε έργο, γίνεται μόνο με την εκτέλεση της εντολής `pytest`. Έτσι, για το παράδειγμα του πίνακα 5, η έξοδος που θα προκύψει από την εκτέλεση του pytest είναι αυτή που φαίνεται στον πίνακα 6 [65].

3 Μεθοδολογία

Στο κεφάλαιο αυτό, θα παρουσιαστούν αναλυτικά οι τεχνικές προτεραιοποίησης που χρησιμοποιήθηκαν για τη διενέργεια των πειραμάτων και τον εντοπισμό των σφαλμάτων. Στη συνέχεια, θα περιγραφούν οι τεχνικές που χρησιμοποιήθηκαν για τη εμφύτευση των σφαλμάτων στα έργα λογισμικού, ενώ θα παρουσιαστούν συνοπτικά και τα έργα λογισμικού που μελετήθηκαν.

Τέλος, θα παρουσιαστούν τα εργαλεία που αναπτύχθηκαν ή/και χρησιμοποιήθηκαν για τη διενέργεια των πειραμάτων και την εξαγωγή των τελικών αποτελεσμάτων της έρευνας και θα αναφερθούν οι στατιστικοί έλεγχοι και τα εργαλεία στατιστικής ανάλυσης που χρησιμοποιήθηκαν για την εξαγωγή των συμπερασμάτων.

3.1 Τεχνικές προτεραιοποίησης

Στον πίνακα 7 παρουσιάζονται οι τεχνικές προτεραιοποίησης που χρησιμοποιήθηκαν. Αυτές ομαδοποιούνται σε πέντε ομάδες. Η πρώτη από αυτές είναι η ομάδα σύγκρισης, η οποία περιλαμβάνει τέσσερις τεχνικές οι οποίες μπορούν να χρησιμοποιηθούν ως σημεία αναφοράς για την εκτίμηση της απόδοσης των υπολοίπων τεχνικών. Σε αυτή την περίπτωση, ο όρος «τεχνική προτεραιοποίησης» χρησιμοποιείται χαλαρά. Στην πραγματικότητα, η ομάδα σύγκρισης δεν χρησιμοποιεί καμία ευρετική προτεραιοποίησης. Απλά καθορίζει κάποιες διατάξεις περιπτώσεων ελέγχου με τις οποίες μπορούν να συγκριθούν οι διατάξεις που θα προκύψουν από τις υπόλοιπες τεχνικές. Οι υπόλοιπες τέσσερις ομάδες αναφέρονται στις τεχνικές τις οποίες επιθυμούμε να εξεταστούν. Σε όλες τις τεχνικές προτεραιοποίησης, χρησιμοποιήθηκε ο άπληστος αλγόριθμος για τον καθορισμό της διάταξης των περιπτώσεων ελέγχου. Οι τεχνικές προτεραιοποίησης παρουσιάζονται στη συνέχεια.

Τεχνικές σύγκρισης

ΤΠ1: Βέλτιστη προτεραιοποίηση (Optimal). Οι περιπτώσεις ελέγχου εκτελούνται με τη σειρά με την οποία εξασφαλίζεται ο βέλτιστος ρυθμός ανίχνευσης σφαλμάτων. Δηλαδή με τη σειρά με την οποία τα σφάλματα εμφανίζονται το συντομότερο δυνατό. Προφανώς αυτή η τεχνική προτεραιοποίησης δεν είναι πρακτική, αφού προϋποθέτει γνώση των περιπτώσεων ελέγχου που θα παρουσιάσουν σφάλμα.

Πίνακας 7: Τεχνικές προτεραιοποίησης περιπτώσεων ελέγχου

Κωδικός	Τίτλος	Περιγραφή
ΤΠ1	Optimal	Σειρά εκτέλεσης με το βέλτιστο ρυθμό ανίχνευσης σφαλμάτων
ΤΠ2	Untreated	Αρχική σειρά εκτέλεσης
ΤΠ3	Random	Τυχαία σειρά εκτέλεσης
ΤΠ4	Inverse	Αντίστροφη σειρά εκτέλεσης σε σχέση με την αρχική
ΤΠ5	File-total	Σειρά εκτέλεσης σε σχέση με την κάλυψη σε επίπεδο αρχείου
ΤΠ6	File-addtl	Σειρά εκτέλεσης σε σχέση με την επιπρόσθετη κάλυψη σε επίπεδο αρχείου
ΤΠ7	Method-total	Σειρά εκτέλεσης σε σχέση με την κάλυψη σε επίπεδο μεθόδου/συνάρτησης
ΤΠ8	Method-addtl	Σειρά εκτέλεσης σε σχέση με την επιπρόσθετη κάλυψη σε επίπεδο μεθόδου/συνάρτησης
ΤΠ9	Branch-total	Σειρά εκτέλεσης σε σχέση με την κάλυψη σε επίπεδο διακλάδωσης
ΤΠ10	Branch-addtl	Σειρά εκτέλεσης σε σχέση με την επιπρόσθετη κάλυψη σε επίπεδο διακλάδωσης
ΤΠ11	Statement-total	Σειρά εκτέλεσης σε σχέση με την κάλυψη σε επίπεδο εντολής
ΤΠ12	Statement-addtl	Σειρά εκτέλεσης σε σχέση με την επιπρόσθετη κάλυψη σε επίπεδο εντολής

Χρησιμοποιείται μόνο ως δείκτης του άνω ορίου της αποτελεσματικότητας των υπόλοιπων τεχνικών προτεραιοποίησης.

ΤΠ2: Καμία προτεραιοποίηση (Untreated). Η πρώτη τεχνική που χρησιμοποιείται ως μέτρο σύγκρισης είναι η μη εφαρμογή οποιασδήποτε τεχνικής προτεραιοποίησης. Οι περιπτώσεις ελέγχου εκτελούνται ακριβώς με την ίδια σειρά με την οποία εμφανίζονται στα σενάρια ελέγχου.

ΤΠ3: Τυχαία προτεραιοποίηση (Random). Οι περιπτώσεις ελέγχου εκτελούνται με τυχαίο τρόπο.

ΤΠ4: Αντίστροφη προτεραιοποίηση (Inverse). Οι περιπτώσεις ελέγχου εκτελούνται με την αντίστροφη σειρά από την οποία εμφανίζονται στα σενάρια ελέγχου. Πιθανόν να παρουσιάζει ενδιαφέρον, λόγω του ότι συνήθως οι προγραμματιστές προσθέτουν νέες περιπτώσεις ελέγχου στο τέλος των σεναρίων ελέγχου. Αυτό έχει ως άμεσο αποτέλεσμα, όταν δεν υπάρχει καμία προτεραιοποίηση, οι παλαιότερες περιπτώσεις ελέγχου, οι οποίες αφορούν επίσης σε παλαιότερο κώδικα που συνήθως δεν έχει αλλάξει, να εκτελούνται πρώτες. Αντίθετα, οι νεότερες περιπτώσεις ελέγχου, οι οποίες αφορούν σε νεότερο κώδικα, ο οποίος πιθανόν να μην έχει ελεγχθεί στο ίδιο επίπεδο και πιθανόν παρουσιάζουν περισσότερα σφάλματα, εκτελούνται τελευταίες.

Τεχνικές προτεραιοποίησης σε επίπεδο αρχείου

ΤΠ5: Κάλυψη σε επίπεδο αρχείου (File-total). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα κάλυψη σε επίπεδο αρχείου. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν, ανεξάρτητα η μία από την άλλη, τα περισσότερα αρχεία πηγαίου κώδικα.

ΤΠ6: Επιπρόσθετη κάλυψη σε επίπεδο αρχείου (File-addtl). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα επιπρόσθετη κάλυψη σε επίπεδο αρχείου. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν τα περισσότερα αρχεία πηγαίου κώδικα, τα οποία δεν έχουν καλυφθεί από περιπτώσεις ελέγχου που έχουν εκτελεστεί ήδη.

Τεχνικές προτεραιοποίησης σε επίπεδο μεθόδου

ΤΠ7: Κάλυψη σε επίπεδο μεθόδου/συνάρτησης (Method-total). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα κάλυψη σε επίπεδο μεθόδου. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν, ανεξάρτητα η μία από την άλλη, τις περισσότερες μεθόδους/συναρτήσεις.

ΤΠ8: Επιπρόσθετη κάλυψη σε επίπεδο μεθόδου (Method-addtl). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα επιπρόσθετη κάλυψη σε επίπεδο μεθόδου. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν τις περισσότερες μεθόδους/συναρτήσεις, οι οποίες δεν έχουν καλυφθεί από περιπτώσεις ελέγχου που έχουν εκτελεστεί ήδη.

Τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης

ΤΠ9: Κάλυψη σε επίπεδο διακλάδωσης (Branch-total). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα κάλυψη σε επίπεδο διακλάδωσης. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν, ανεξάρτητα η μία από την άλλη, τις περισσότερες διακλαδώσεις που εμφανίζονται στον πηγαίο κώδικα.

ΤΠ10: Επιπρόσθετη κάλυψη σε επίπεδο διακλάδωσης (Branch-addtl). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα επιπρόσθετη κάλυψη σε επίπεδο διακλάδωσης. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν τις περισσότερες διακλαδώσεις, οι οποίες δεν έχουν καλυφθεί από περιπτώσεις ελέγχου που έχουν εκτελεστεί ήδη.

Τεχνικές προτεραιοποίησης σε επίπεδο εντολής

ΤΠ11: Κάλυψη σε επίπεδο εντολής (Statement-total). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα κάλυψη σε επίπεδο εντολής. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν, ανεξάρτητα η μία από την άλλη, τις περισσότερες εντολές που εμφανίζονται μεμονωμένα στον πηγαίο κώδικα..

ΤΠ12: Επιπρόσθετη κάλυψη σε επίπεδο εντολής (Statement-addtl). Οι περιπτώσεις ελέγχου εκτελούνται σε σχέση με τη φθίνουσα επιπρόσθετη κάλυψη σε επίπεδο εντο-

λής. Δηλαδή εκτελούνται πρώτες οι περιπτώσεις ελέγχου, οι οποίες καλύπτουν τις περισσότερες εντολές, οι οποίες δεν έχουν καλυφθεί από περιπτώσεις ελέγχου που έχουν εκτελεστεί ήδη.

Για όλες τις παραπάνω τεχνικές προτεραιοποίησης, σε περίπτωση που υπήρχαν ισοψηφίες, αυτές επιλύθηκαν με τυχαία επιλογή μεταξύ των περιπτώσεων ελέγχου που συμμετείχαν στην ισοψηφία.

3.2 Σφάλματα

Για τα έργα λογισμικού που μελετήθηκαν, και για τις εκδόσεις τους που χρησιμοποιήθηκαν για τη διενέργεια των πειραμάτων, δεν υπάρχουν σφάλματα που να εμφανίζονται με την εκτέλεση της σουίτας ελέγχου. Αυτό είναι φυσιολογικό, αφού οι προγραμματιστές τους δεν θα τα είχαν διαθέσει πριν διορθώσουν τα οποιαδήποτε σφάλματα ήταν γνωστά μέχρι εκείνο το σημείο. Έτσι, θα έπρεπε για τη διενέργεια των πειραμάτων, με κάποιο τρόπο είτε να προστεθούν (τεχνητά) σφάλματα στον πηγαίο κώδικα, είτε να προστεθούν νέες περιπτώσεις ελέγχου, οι οποίες εμφανίζουν τα οποιαδήποτε σφάλματα υπήρχαν, χωρίς τη γνώση των προγραμματιστών.

Τα οποιαδήποτε σφάλματα έχει ένα έργο λογισμικού, αποκαλύπτονται κατά την ανάπτυξη των επόμενων εκδόσεών του. Εκεί, τα σφάλματα διορθώνονται, και αν το έργο έχει πολιτική που ενθαρρύνει τη συγγραφή περιπτώσεων ελέγχου, ταυτόχρονα δημιουργείται και μια περίπτωση ελέγχου, η οποία θα ελέγχει τον κώδικα, ώστε το σφάλμα να μην ξαναπαρουσιαστεί μελλοντικά.

Έτσι, μια ευκαιρία που προκύπτει για τη δημιουργία περιπτώσεων ελέγχου, που είναι ρεαλιστικές και των οποίων η εκτέλεση θα αποτυγχάνει, είναι ο «δανεισμός» τους από κάποια επόμενη έκδοση του λογισμικού, χωρίς όμως ταυτόχρονα να μεταφέρεται και ο κώδικας που επιδιορθώθηκε και τον οποίο αυτές οι περιπτώσεις ελέγχου εξετάζουν. Φυσικά, θα πρέπει οι περιπτώσεις ελέγχου να αναφέρονται σε κάποια επιδιόρθωση ήδη υπάρχοντα κώδικα, και όχι σε κώδικα που αντιστοιχεί σε νέα λειτουργικότητα του λογισμικού. Γι' αυτό το λόγο στην παρούσα έρευνα χρησιμοποιούνται κυρίως διαδοχικές ελάχιστες (minor) εκδόσεις των έργων λογισμικού, οι οποίες συνηθίζεται να διορθώνουν τέτοιου είδους προβλήματα στον πηγαίο κώδικα. Νέα λειτουργικότητα, είναι σύνηθες να συμβολίζεται με την αύξηση της κύριας (major) έκδοσης του λογισμικού.

Για παράδειγμα, μεταξύ των εκδόσεων του PyPy 5.10.0 και 5.10.1, και με την εκτέ-

λεση του εργαλείου `diff` [76] που υπάρχει εγκατεστημένο σε οποιαδήποτε διανομή του λειτουργικού συστήματος Linux, μπορεί να βρεθεί πως, μεταξύ άλλων, διορθώθηκε ο κώδικας στο αρχείο `timeutils.py` όπως φαίνεται στον πίνακα 8. Ταυτόχρονα, προστέθηκε στη σουίτα ελέγχου η περίπτωση ελέγχου `test_timestamp`, όπως φαίνεται στον πίνακα 9. Η περίπτωση ελέγχου αυτή, αν εκτελεστεί στη βάση του κώδικα της έκδοσης 5.10.0 του PyPy, θα αποτύχει.

Λόγω του ότι δεν ήταν εφικτό να δημιουργηθεί ικανός αριθμός σφαλμάτων μόνο με τη χρήση της προαναφερόμενης τεχνικής, μία ακόμα τεχνική που χρησιμοποιήθηκε ήταν αυτή της εμφύτευσης σφαλμάτων στον πηγαίο κώδικα. Για τη δημιουργία αυτών των σφαλμάτων, επιλεγόταν με τυχαίο τρόπο κάποια περίπτωση ελέγχου και εξεταζόταν το κομμάτι του πηγαίου κώδικα που αυτή ήλεγχε. Σε αυτό το κομμάτι κώδικα, εφαρμοζόταν αλλαγές οι οποίες τελικά θα οδηγούσαν την εκτέλεση της περίπτωσης ελέγχου να αποτύχει. Οι αλλαγές ήταν τέτοιες, ώστε να είναι όσο το δυνατό ρεαλιστικές και να δημιουργούν σφάλματα τα οποία είναι συχνά στην Python, σύμφωνα με την εμπειρία του γράφοντα. Παραδείγματα τέτοιων σφαλμάτων είναι τα σφάλματα off-by-one [69], οι αποτυχημένες αρχικοποιήσεις κάποιας μεταβλητής, που τελικά καταλήγουν σε `TypeError: 'NoneType'` σφάλμα [68] ή σφάλματα που προκύπτουν από λάθος συγκρίσεις, όπως για παράδειγμα η αντικατάσταση ενός ελέγχου `if val >= limit` με τον `if val > limit`. Αν για κάποια περίπτωση ελέγχου δεν ήταν εφικτό να εμφυτευθεί κάποιο σφάλμα το οποίο θα ήταν ρεαλιστικό, δεν γινόταν καμία αλλαγή και γινόταν επιλογή κάποιας άλλης περίπτωσης ελέγχου για την εφαρμογή της ίδιας διαδικασίας.

Σε κάποιες περιπτώσεις, εξετάζοντας τις αλλαγές μεταξύ διαδοχικών εκδόσεων των έργων λογισμικού, υπήρχαν διορθώσεις στον πηγαίο κώδικα, οι οποίες δεν συνοδευόταν από τη συγγραφή αντίστοιχων περιπτώσεων ελέγχου. Οπότε, ένας ακόμα τρόπος για τη δημιουργία σφαλμάτων, ήταν η συγγραφή των αντίστοιχων ελέγχων και η εφαρμογή τους στην προηγούμενη έκδοση του έργου.

3.3 Λογισμικό

Τα έργα λογισμικού που μελετήθηκαν στην παρούσα έρευνα ήταν τα εξής:

- PyPy [64]
- Django [53]

Πίνακας 8: Διόρθωση στον κώδικα του PyPy μεταξύ των εκδόσεων 5.10.0 και 5.10.1 (diff)

```
--- pypy3-v5.10.0-src/pypy/interpreter/timeutils.py     2017-12-22
    12:09:25.000000000 +0200
+++ pypy3-v5.10.1-src/pypy/interpreter/timeutils.py     2018-01-10
    16:58:49.000000000 +0200
@@ -3,7 +3,8 @@
 """
 import math
 from rpython.rlib.rarithmetic import (
-    r_longlong, ovfcheck, ovfcheck_float_to_longlong)
+    r_longlong, ovfcheck_float_to_longlong)
+from rpython.rlib import rfloat
 from pypy.interpreter.error import oefmt

 SECS_TO_NS = 10 ** 9
@@ -21,6 +22,8 @@
 def timestamp_w(space, w_secs):
     if space.isinstance_w(w_secs, space.w_float):
         secs = space.float_w(w_secs)
+        if rfloat.isnan(secs):
+            raise oefmt(space.w_ValueError, "timestamp is nan")
         result_float = math.ceil(secs * SECS_TO_NS)
         try:
             return ovfcheck_float_to_longlong(result_float)
@@ -28,10 +31,10 @@
             raise oefmt(space.w_OverflowError,
                 "timestamp %R too large to convert to C _PyTime_t",
                    w_secs)
     else:
-        sec = space.int_w(w_secs)
         try:
-            result = ovfcheck(sec * SECS_TO_NS)
+            sec = space.bigint_w(w_secs).tolonglong()
+            result = sec * r_longlong(SECS_TO_NS)
         except OverflowError:
             raise oefmt(space.w_OverflowError,
-                "timestamp too large to convert to C _PyTime_t")
-        return r_longlong(result)
+                "timestamp %R too large to convert to C _PyTime_t",
    w_secs)
+        return result
```

Πίνακας 9: Περίπτωση ελέγχου που προστέθηκε μεταξύ των εκδόσεων 5.10.0 και 5.10.1 του Pypy (diff)

```
--- 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)
```

- MoinMoin [74]
- Flask [57]
- Six [71]

Η επιλογή τους έγινε με κύριο γνώμονα κάποια βασικά χαρακτηριστικά:

- τα έργα έπρεπε να είναι ανοικτού κώδικα, ώστε αυτός να είναι ευρέως διαθέσιμος για διαφορετικές εκδόσεις του λογισμικού,
- τα έργα έπρεπε να είναι υλοποιημένα εξ' ολοκλήρου σε γλώσσα Python, ώστε οι περιπτώσεις ελέγχου να μην εξαρτώνται σε καμία περίπτωση από κώδικα που είναι υλοποιημένος σε κάποια άλλη γλώσσα προγραμματισμού,
- τα έργα έπρεπε να είναι αρκετά μεγάλα σε όγκο, ώστε η προτεραιοποίηση των περιπτώσεων ελέγχου να έχει απτά αποτελέσματα. Σε ένα μικρό έργο λογισμικού, για το οποίο η σουίτα περιπτώσεων ελέγχου εκτελείται σε ελάχιστο χρόνο, η προτεραιοποίηση των περιπτώσεων ελέγχου δεν είναι σημαντική,
- οι σουίτες περιπτώσεων ελέγχου των έργων θα έπρεπε να περιλαμβάνουν ικανό αριθμό περιπτώσεων ελέγχου, οι οποίες θα κάλυπταν σημαντικό κομμάτι του πηγαίου κώδικα

Πίνακας 10: Εκδόσεις του PyPy που μελετήθηκαν

Έκδοση	Αριθμός 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

Μοναδική εξαίρεση στα παραπάνω ήταν το Six, το οποίο χρησιμοποιήθηκε ως αντιπρόσωπος των πολυάριθμων βιβλιοθηκών μικρού μεγέθους που υπάρχουν για τη γλώσσα Python. Επιπρόσθετα, η διενέργεια του πειραματικού μέρους της εργασίας ξεκίνησε από αυτή τη βιβλιοθήκη, αφού τα μικρότερα έργα λογισμικού είναι πιο εύκολα κατανοητά και τα αποτελέσματα σε οποιαδήποτε περίπτωση λαμβάνονται ταχύτερα. Ακόμα κι έτσι, το Six διαθέτει μια αρκετά μεγάλη σουίτα περιπτώσεων ελέγχου, σε σχέση πάντα με το μέγεθος του έργου.

Η PyPy είναι μια εναλλακτική υλοποίησης της γλώσσας Python, γραμμένη πλήρως σε γλώσσα Python. Σε αντίθεση με την Pypy, η CPython [67], η οποία αποτελεί την εξ' ορισμού υλοποίησης της Python, αναπτύσσεται με τη χρήση της γλώσσας προγραμματισμού C. Η PyPy εμφανίζει αρκετά πλεονεκτήματα σε σχέση με τη CPython, όπως η ταχύτερη εκτέλεση του κώδικα, χάρη στον Just-in-time compiler που διαθέτει και η απουσία του GIL (Global Interpreter Lock) [58], ο οποίος είναι υπεύθυνος για τα πολλά προβλήματα που έχει η CPython σε πολυνηματικές εφαρμογές. Κατά τα άλλα, η PyPy έχει αρκετά μεγάλο βαθμό συμβατότητας με τη CPython και με τις πολυάριθμες βιβλιοθήκες που υπάρχουν γι' αυτή, με εξαιρέσεις τις βιβλιοθήκες που χρησιμοποιούν για την υλοποίησή τους κάποια άλλη γλώσσα πέρα από την Python. Ακόμα και σε αυτές τις περιπτώσεις, για δημοφιλείς βιβλιοθήκες, όπως η NumPy [62], αναπτύσσονται υλοποιήσεις με στόχο να είναι συμβατές με την PyPy. Η PyPy, πιθανόν είναι το μεγαλύτερο σε όγκο έργο λογισμικού ανοικτού κώδικα που είναι υλοποιημένο αποκλειστικά με τη χρήση της γλώσσας Python. Οι εκδόσεις του έργου PyPy που μελετήθηκαν φαίνονται στον πίνακα 10.

Το Django είναι ένα πλαίσιο ανάπτυξης εφαρμογών ιστού ανοικτού λογισμικού, το οποίο ακολουθεί το πρότυπο σχεδίασης Model-Template-View (MTV). Αποτελεί ένα από

Πίνακας 11: Εκδόσεις του Django που μελετήθηκαν

Έκδοση	Αριθμός 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

Πίνακας 12: Εκδόσεις του MoinMoin που μελετήθηκαν

Έκδοση	Αριθμός 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

τα πλέον δημοφιλή πλαίσια ανάπτυξης εφαρμογών ιστού ανεξαρτήτως γλώσσας προγραμματισμού και με τη βοήθειά του μπορούν να δημιουργηθούν εφαρμογές οποιασδήποτε πολυπλοκότητας. Σύμφωνα με τη Wikipedia [77], πολλές δημοφιλείς ιστοσελίδες αναπτύσσονται με τη χρήση του Django, όπως το Instagram, η Mozilla, οι Washington Times και το Bitbucket. Οι εκδόσεις του Django που μελετήθηκαν στην παρούσα εργασία, μαζί με τα χαρακτηριστικά τους, φαίνονται στον πίνακα 11.

Το MoinMoin είναι μια δημοφιλής διαδικτυακή εφαρμογή Wiki, το οποίο είναι υλοποιημένο πλήρως με τη χρήση της γλώσσας Python, Το MoinMoin χρησιμοποιείται ευρύτατα στον χώρο του ανοικτού λογισμικού από διάφορους οργανισμούς και με τη χρήση του υλοποιούνται οι σελίδες Wiki αρκετών έργων, όπως το Debian, το Ubuntu, το Apache και το FreeBSD [79]. Σημαντικό χαρακτηριστικό του MoinMoin, είναι ότι για την αποθήκευση των σελίδων του wiki, αλλά και οποιασδήποτε πληροφορίας, δεν χρησιμοποιείται κάποια βάση δεδομένων, αλλά απλά αρχεία κειμένου. Οι εκδόσεις του MoinMoin που μελετήθηκαν στην παρούσα εργασία, εμφανίζονται στον πίνακα 12.

Το Flask είναι ένα πλαίσιο ανάπτυξης εφαρμογών ιστού, το οποίο είναι μικρότερο σε όγκο σε σχέση με το Django, ενώ ταυτόχρονα είναι ευκολότερα προσβάσιμο από νέους χρήστες. Δεν απαιτεί καμία επιπρόσθετη βιβλιοθήκη ή εργαλείο πέρα από την ίδια τη γλώσσα Python και αν και το μέγεθός του είναι σχετικά μικρό, μπορεί να επεκταθεί σε πολύ μεγάλο βαθμό με τη βοήθεια προσθέτων. Χρησιμοποιείται επίσης εκτενέστατα

Πίνακας 13: Εκδόσεις του Flask που μελετήθηκαν

Έκδοση	Αριθμός 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

Πίνακας 14: Εκδόσεις του Six που μελετήθηκαν

Έκδοση	Αριθμός 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

από δημοφιλείς ιστοσελίδες, όπως το Pinterest και το LinkedIn [78]. Οι εκδόσεις και τα χαρακτηριστικά του Flask που μελετήθηκαν στην παρούσα έρευνα παρουσιάζονται στον πίνακα 13.

Το Six είναι μια μικρή βιβλιοθήκη, η οποία όμως χρησιμοποιείται ευρύτατα από πολλούς προγραμματιστές της γλώσσας Python. Η βιβλιοθήκη παρέχει ένα επίπεδο συμβατότητας μεταξύ των εκδόσεων 2 και 3 της Python, και βοηθάει τους προγραμματιστές να γράψουν κώδικα ο οποίος θα εκτελείται χωρίς προβλήματα και χωρίς καμία αλλαγή σε οποιαδήποτε από τις δύο κύριες, και ασύμβατες μεταξύ τους, εκδόσεις της γλώσσας. Οι εκδόσεις του Six που μελετήθηκαν μαζί με τα χαρακτηριστικά τους, εμφανίζονται στον πίνακα 14.

3.4 Εργαλεία

Για την αυτοματοποίηση της διενέργειας των πειραμάτων, χρησιμοποιώντας τις διαφορετικές τεχνικές προτεραιοποίησης, αναπτύχθηκε ένα πρόσθετο για το πλαίσιο ελέγχων pytest, σε γλώσσα Python με την ονομασία *pytest-prioritize*.

Το πρόσθετο, υλοποιεί την παράμετρο γραμμής εντολών `--pytest-prioritize`, το οποίο μπορεί να λάβει τις τιμές `inverse`, `random`, `file-total`, `file-addtl`, `method-total`, `method-addtl`, `branch-total`, `branch-addtl`, `statement-total`, `statement-addtl` και `none`, η οποία είναι και η εξ' ορισμού επιλογή. Αυτές μπορούν να οριστούν κατά την εκτέλεση του pytest, όπως στο παρακάτω παράδειγμα:

```
$ pytest --pytest-prioritize=branch-total
```

Το πρόσθετο, χρησιμοποιεί τη μέθοδο `pytest_sessionstart` που παρέχει το πλαίσιο ελέγχων pytest, ώστε να αρχικοποιήσει το περιβάλλον ελέγχου, πριν εκτελεστούν οι περιπτώσεις ελέγχου και να επιλέξει την τεχνική προτεραιοποίησης που θα χρησιμοποιηθεί.

Χρησιμοποιείται επίσης το πρόσθετο pytest-cov [66], για τη συλλογή πληροφοριών κάλυψης για κάθε μία από τις περιπτώσεις ελέγχου που περιλαμβάνει η σουίτα ελέγχου. Το πρόσθετο επιστρέφει πληροφορίες κάλυψης, τόσο σε επίπεδο αρχείου, μεθόδου, διακλάδωσης ή εντολής.

Το πρόσθετο pytest-prioritize, χρησιμοποιεί την πληροφορία κάλυψης κώδικα, αν αυτή υπάρχει αποθηκευμένη στο δίσκο για το έργο λογισμικού. Σε περίπτωση που δεν υπάρχει καμία αποθηκευμένη πληροφορία, αυτή μπορεί να δημιουργηθεί εκτελώντας το pytest με τη χρήση της παραμέτρου `--prioritization-gather-cov` ή με τη χρήση της παραμέτρου `--prioritization-gather-cov-all`, π.χ. ως εξής:

```
$ pytest --prioritization-gather-cov
```

Υπενθυμίζεται ότι η κάλυψη κώδικα μετριέται με δυναμική ανάλυση του κώδικα. Όταν χρησιμοποιείται η παράμετρος `--prioritization-gather-gov`, θα εκτελεστούν οι περιπτώσεις ελέγχου για τις οποίες δεν υπάρχει καταγεγραμμένη πληροφορία κάλυψης. Όταν χρησιμοποιείται η παράμετρος `--prioritization-gather-gov-all`, εκτελούνται όλες οι περιπτώσεις ελέγχου που περιλαμβάνονται στη σουίτα ελέγχου και η πληροφορία κάλυψης κώδικα αποθηκεύεται εκ νέου, για όλες τις περιπτώσεις ελέγχου. Η πληροφορία κάλυψης κώδικα αποθηκεύεται και προσπελάζεται μέσω του αρθρώματος `pytest.config` που παρέχει το πλαίσιο ελέγχων pytest.

Αφού το πρόσθετο pytest-prioritize αρχικοποιήσει το περιβάλλον ελέγχου, χρησιμοποιεί τη μέθοδο `pytest_collection_modifyitems` που παρέχει το πλαίσιο ελέγχου pytest και με την οποία υπάρχει η δυνατότητα να αναδιαταχθούν οι περιπτώσεις ελέγχου κατά το δοκούν. Όταν το πρόσθετο pytest-prioritize εκτελείται με τη χρήση κάποιας τεχνικής προτεραιοποίησης που χρησιμοποιεί πληροφορία κάλυψης, και για κάποια περίπτωση ελέγχου αυτή η πληροφορία δεν υπάρχει ήδη αποθηκευμένη στο δίσκο, η περίπτωση ελέγχου τοποθετείται στην αρχή της διάταξης, ώστε να εκτελεστεί πρώτη.

Όλοι οι έλεγχοι πραγματοποιήθηκαν χρησιμοποιώντας διαφορετικές εκδόσεις της γλώσσας Python και συγκεκριμένα της υλοποίησης CPython, για όλα τα έργα λογισμικού. Η χρήση διαφορετικών εκδόσεων της CPython, ήταν αναγκαία, αφού διαφορετικές

εκδόσεις των έργων λογισμικού που μελετήθηκαν, ήταν συμβατές μόνο με συγκεκριμένες εκδόσεις της CPython. Μοναδική εξαίρεση στα παραπάνω, ήταν το έργο PyPy, για το οποίο χρησιμοποιήθηκε η ίδια έκδοση του PyPy με αυτή που εξεταζόταν κατά περίπτωση.

3.5 Στατιστικές αναλύσεις

Για την αξιολόγηση των αποτελεσμάτων όσον αφορά τις τιμές της μετρικής APFD, χρησιμοποιήθηκε η γλώσσα GNU R 3.3.1 [70], ενώ για το σχεδιασμό των θηκογραμμάτων, αλλά και των σχημάτων του υπολογισμού της μετρικής APFD, χρησιμοποιήθηκε η βιβλιοθήκη γραφικών ggplot2 [45] του GNU R.

Για τη αξιολόγηση της εγκυρότητας του ελέγχου της ανάλυσης διακύμανσης, χρησιμοποιήθηκε ο έλεγχος κανονικότητας των Shapiro-Wilk [41] στα υπολείμματα του ελέγχου της ανάλυσης διακύμανσης, όπως επίσης και ο έλεγχος ομοιογένειας των διακυμάνσεων του Levene [17].

Πραγματοποιήθηκε έλεγχος της ανάλυσης διακύμανσης δύο παραγόντων, με αλληλεπίδραση [5], όπου οι εξεταζόμενοι παράγοντες ήταν η τεχνική παραμετροποίησης που χρησιμοποιήθηκε και το έργο λογισμικού που μελετήθηκε.

Μετά τη διενέργεια του ελέγχου της ανάλυσης διακύμανσης, πραγματοποιήθηκαν κατά ζεύγη συγκρίσεις των μέσων όρων για τους δύο κύριους παράγοντες, χρησιμοποιώντας τον έλεγχο πολλαπλών συγκρίσεων του Tukey [49].

Σε κάθε περίπτωση, ως στατιστικά σημαντικά θεωρήθηκαν τα αποτελέσματα για τα οποία προέκυψε $P < 0.05$.

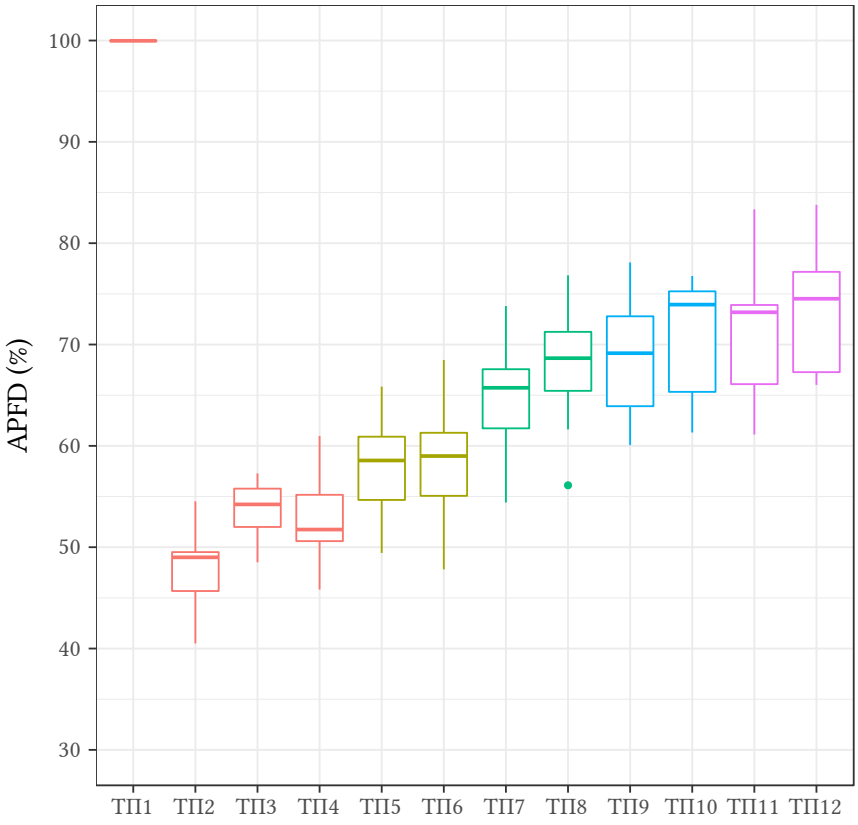
4 Αποτελέσματα

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των τιμών APFD όπως αυτά υπολογίστηκαν για τα διαφορετικά έργα λογισμικού που μελετήθηκαν και για κάθε τεχνική προτεραιοποίησης. Τα αποτελέσματα παρατίθενται τόσο με τη μορφή θηκογραμμάτων, όσο και πινάκων περιγραφικής στατιστικής για κάθε έργο λογισμικού.

Τα γραφήματα υπολογισμού της μετρικής APFD για κάθε έργο λογισμικού και κάθε τεχνική προτεραιοποίησης, λόγω του αυξημένου όγκου τους, βρίσκονται στο Παράρτημα (σελ. 69).

Λεπτομερέστερα, τα αποτελέσματα της μετρικής APFD για το PyPy εμφανίζονται στο σχήμα 6 και τον πίνακα 15. Είναι εμφανές πως η αρχική διάταξη των περιπτώσεων ελέγχου (ΤΠ2), επιτυγχάνει τις μικρότερες τιμές της μετρικής APFD σε σχέση με οποιαδήποτε άλλη τεχνική προτεραιοποίησης. Η τυχαία διάταξη (ΤΠ3), αλλά και η αντίστροφη διάταξη (ΤΠ4), δίνουν καλύτερα αποτελέσματα σε σχέση με την αρχική διάταξη. Σε οποιαδήποτε περίπτωση, οι τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία κάλυψης κώδικα, επιτυγχάνουν υψηλότερες τιμές της μετρικής APFD σε σχέση με τις τεχνικές σύγκρισης. Οι τεχνικές προτεραιοποίησης με τα καλύτερα αποτελέσματα είναι αυτές που χρησιμοποιούν μεγαλύτερη λεπτομέρεια ως προς την καταγραφή της κάλυψης κώδικα. Έτσι, οι υψηλότερες τιμές της μετρικής APFD παρουσιάζονται για τις τεχνικές προτεραιοποίησης με χρήση της κάλυψης κώδικα σε επίπεδο εντολής και διακλάδωσης. Ακολουθούν οι τεχνικές προτεραιοποίησης με χρήση της κάλυψης κώδικα σε επίπεδο μεθόδου, ενώ οι τεχνικές προτεραιοποίησης με χρήση της κάλυψης κώδικα σε επίπεδο αρχείου, παρουσιάζουν τις χαμηλότερες τιμές της μετρικής APFD. Οι τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία επιπρόσθετης κάλυψης κώδικα, εμφανίζουν ελαφρώς υψηλότερες τιμές της μετρικής APFD σε σχέση με τις τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία ολικής κάλυψης κώδικα. Οι διαφορές αυτές όμως δεν φαίνεται να είναι σημαντικές, αφού τα αντίστοιχα θηκογράμματα παρουσιάζουν αρκετά μεγάλες επικαλύψεις.

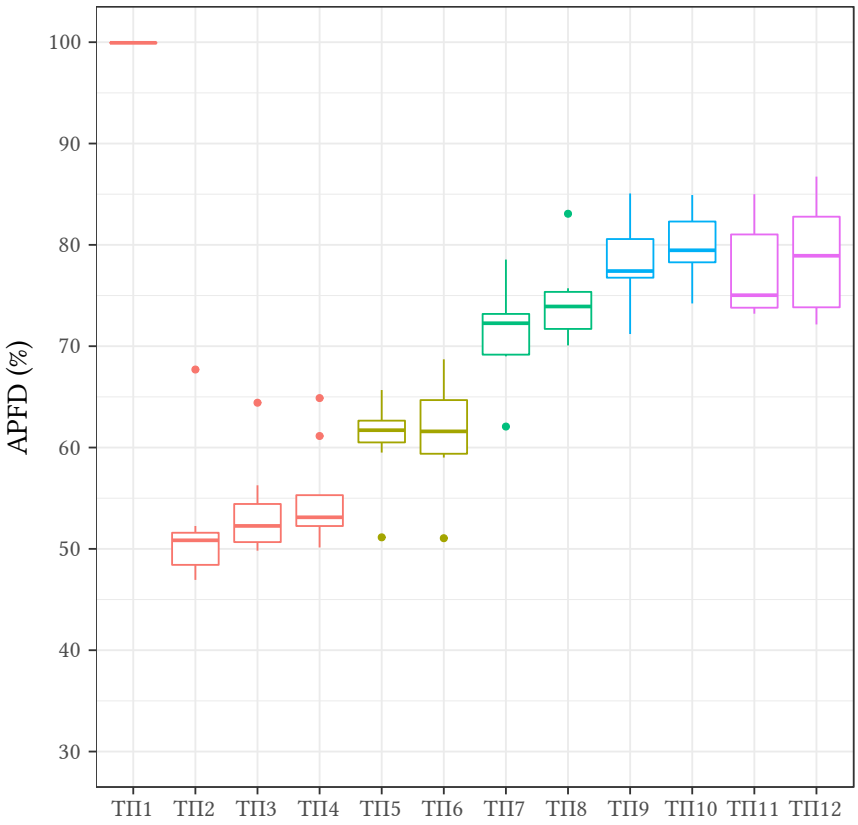
Για το Django, τα αποτελέσματα εμφανίζονται στο σχήμα 7 και τον πίνακα 16. Οι διαφορές μεταξύ των τεχνικών προτεραιοποίησης είναι περισσότερο εμφανείς σε αυτή την περίπτωση, ενώ υπάρχουν και κάποιες εμφανίσεις απομακρυσμένων τιμών, κυρίως όσον αφορά στις τεχνικές σύγκρισης. Και για το Django, φαίνεται πως η αρχική διάταξη



Σχήμα 6: Θηκόγραμμα τιμών APFD του PyPy για κάθε τεχνική προτεραιοποίησης (ΤΠ). ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

Πίνακας 15: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) του PyPy

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	8	99.97	0.01	99.95	99.96	99.97	99.98	99.98
ΤΠ2	8	47.77	4.27	40.50	44.28	49.00	49.63	54.53
ΤΠ3	8	53.60	3.20	48.51	50.45	54.23	56.31	57.29
ΤΠ4	8	53.11	5.03	45.82	50.57	51.74	58.27	60.98
ΤΠ5	8	58.11	5.45	49.43	53.85	58.56	63.08	65.85
ΤΠ6	8	58.54	6.57	47.82	53.69	59.00	64.00	68.48
ΤΠ7	8	64.88	6.03	54.41	60.60	65.73	69.43	73.80
ΤΠ8	8	67.97	6.76	56.11	62.89	68.65	73.97	76.83
ΤΠ9	8	68.61	6.36	60.07	61.95	69.14	73.76	78.10
ΤΠ10	8	70.89	6.03	61.32	64.89	73.94	75.68	76.76
ΤΠ11	8	71.39	6.82	61.12	66.08	73.18	74.23	83.34
ΤΠ12	8	73.50	6.36	66.02	67.05	74.52	77.70	83.78



Σχήμα 7: Θηκόγραμμα τιμών APFD του Django για κάθε τεχνική προτεραιοποίησης (ΤΠ). ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

(ΤΠ2) είναι αυτή που παρουσιάζει τις μικρότερες τιμές της μετρικής APFD. Η τυχαία διάταξη (ΤΠ3) και η αντίστροφη διάταξη (ΤΠ4) παρουσιάζουν επίσης καλύτερα αποτελέσματα σε σχέση με την αρχική. Οι τεχνικές προτεραιοποίησης με κάλυψη κώδικα σε επίπεδο εντολής (ΤΠ11-12), καθώς και αυτές με κάλυψη κώδικα σε επίπεδο διακλάδωσης (ΤΠ9-10), εμφανίζουν κι εδώ τα καλύτερα αποτελέσματα. Ακολουθούν οι τεχνικές προτεραιοποίησης με κάλυψη κώδικα σε επίπεδο μεθόδου (ΤΠ7-8), ενώ οι τεχνικές προτεραιοποίησης με κάλυψη κώδικα σε επίπεδο αρχείου (ΤΠ5-6) παρουσιάζουν τις μικρότερες τιμές της μετρικής APFD, οι οποίες όμως είναι σημαντικά αυξημένες σε σχέση με τις τεχνικές σύγκρισης. Η χρήση της επιπρόσθετης πληροφορίας κάλυψης φαίνεται πως εμφανίζει υψηλότερες τιμές για την κάλυψη κώδικα σε επίπεδο διακλάδωσης και μεθόδου.

Για το MoinMoin, τα αποτελέσματα εμφανίζονται στο σχήμα 8 και στον πίνακα 17. Σε αυτή την περίπτωση, η αρχική διάταξη (ΤΠ2) έχει ακόμα περισσότερο εμφανείς διαφορές σε σχέση με οποιαδήποτε άλλη τεχνική προτεραιοποίησης, αφού παρουσιάζει

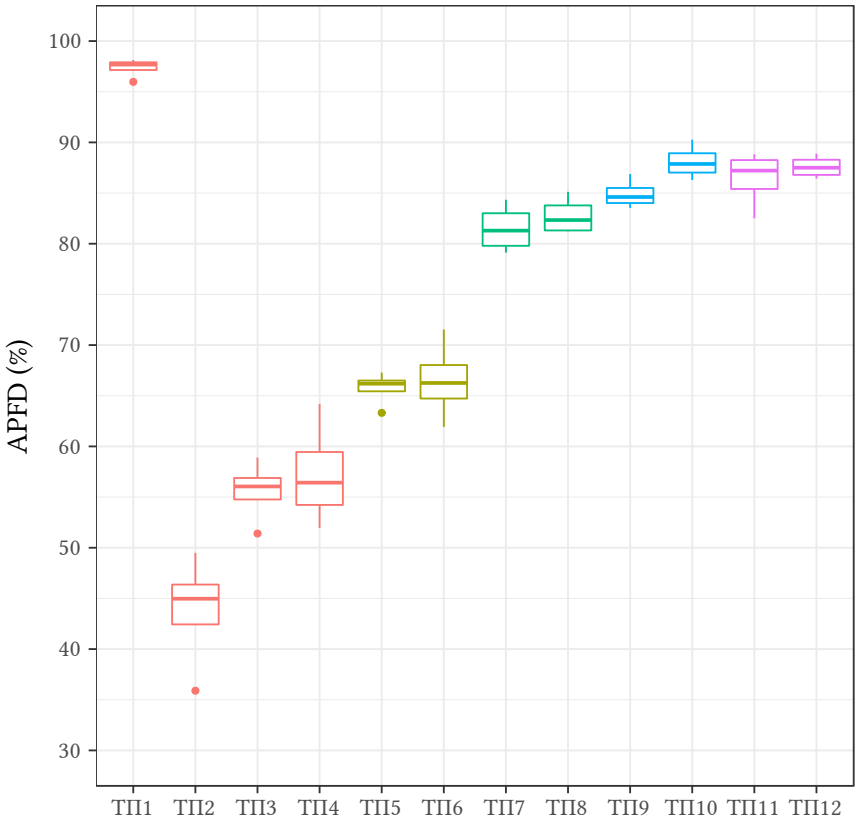
Πίνακας 16: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) του Django

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	8	99.94	0.01	99.92	99.93	99.94	99.95	99.95
ΤΠ2	8	52.02	6.62	46.94	47.81	50.85	52.04	67.70
ΤΠ3	8	53.72	4.83	49.82	50.24	52.26	55.66	64.42
ΤΠ4	8	54.97	5.17	50.14	51.78	53.12	59.19	64.88
ΤΠ5	8	60.75	4.29	51.14	59.84	61.71	62.77	65.67
ΤΠ6	8	61.38	5.25	51.05	59.14	61.59	64.85	68.71
ΤΠ7	8	71.23	4.75	62.07	69.06	72.27	73.27	78.55
ΤΠ8	8	74.34	4.12	70.08	71.11	73.92	75.60	83.07
ΤΠ9	8	78.15	4.08	71.20	76.06	77.42	80.72	85.07
ΤΠ10	8	79.80	3.40	74.22	77.33	79.47	82.45	84.92
ΤΠ11	8	77.22	4.53	73.21	73.45	75.03	81.23	85.00
ΤΠ12	8	78.73	5.35	72.15	73.25	78.93	83.25	86.74

Πίνακας 17: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) του MoinMoin

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	4	97.37	0.96	95.97	96.36	97.68	98.06	98.14
ΤΠ2	4	43.83	5.71	35.89	38.07	44.97	48.45	49.50
ΤΠ3	4	55.60	3.11	51.40	52.52	56.05	58.23	58.90
ΤΠ4	4	57.25	5.22	51.96	52.71	56.42	62.61	64.20
ΤΠ5	4	65.75	1.70	63.31	64.02	66.19	67.03	67.29
ΤΠ6	4	66.49	3.96	61.92	62.86	66.26	70.36	71.53
ΤΠ7	4	81.51	2.38	79.12	79.35	81.29	83.90	84.34
ΤΠ8	4	82.76	1.85	81.23	81.26	82.34	84.68	85.12
ΤΠ9	4	84.91	1.45	83.54	83.70	84.61	86.42	86.87
ΤΠ10	4	88.08	1.71	86.29	86.54	87.87	89.82	90.27
ΤΠ11	4	86.44	2.81	82.51	83.47	87.22	88.63	88.81
ΤΠ12	4	87.58	1.11	86.44	86.56	87.50	88.68	88.88

σημαντικά χαμηλότερες τιμές της μετρικής APFD. Η τυχαία διάταξη (ΤΠ3) και η αντίστροφη διάταξη (ΤΠ4), δίνουν σε κάθε περίπτωση καλύτερα αποτελέσματα σε σχέση με την αρχική. Επίσης, οι τεχνικές προτεραιοποίησης με χρήση πληροφορίας κάλυψης κώδικα, δίνουν εμφανώς καλύτερα αποτελέσματα για τις τιμές της μετρικής APFD, που σε κάποιες περιπτώσεις πλησιάζουν στα επίπεδα του 90%. Όπως και στα προηγούμενα έργα, οι τεχνικές με χρήση μεγαλύτερης λεπτομέρειας για την καταγραφή της κάλυψης κώδικα, εμφανίζουν τα καλύτερα αποτελέσματα. Φαίνεται πως οι τεχνικές που χρησιμοποιούν πληροφορία επιπρόσθετης κάλυψης κώδικα, δίνουν υψηλότερες τιμές τις μετρικής APFD, ειδικά για τις τεχνικές προτεραιοποίησης με χρήση κάλυψης σε επίπεδο διακλάδωσης.



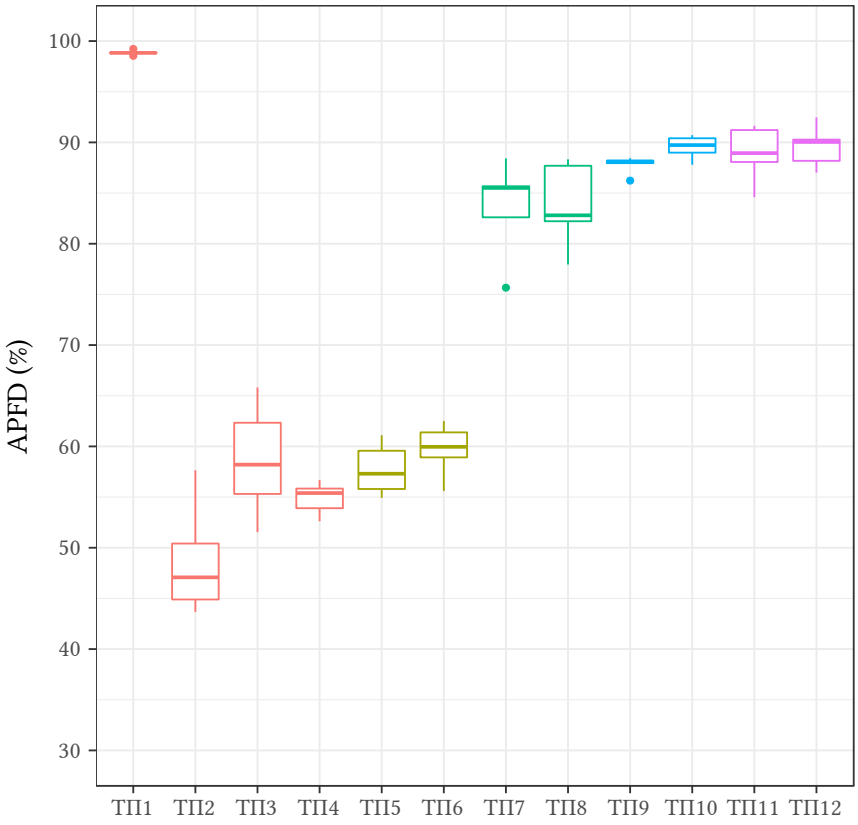
Σχήμα 8: Θηκόγραμμα τιμών APFD του MoinMoin για κάθε τεχνική προτεραιοποίησης (ΤΠ). ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

Πίνακας 18: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) του Flask

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	5	98.86	0.24	98.54	98.67	98.83	99.05	99.22
ΤΠ2	5	48.74	5.60	43.67	44.28	47.08	54.03	57.64
ΤΠ3	5	58.64	5.63	51.54	53.43	58.20	64.08	65.82
ΤΠ4	5	54.89	1.62	52.62	53.26	55.40	56.26	56.69
ΤΠ5	5	57.74	2.57	54.93	55.36	57.30	60.33	61.10
ΤΠ6	5	59.67	2.65	55.60	57.26	59.96	61.95	62.51
ΤΠ7	5	83.57	4.88	75.66	79.14	85.51	87.04	88.41
ΤΠ8	5	83.80	4.28	77.95	80.09	82.80	88.01	88.33
ΤΠ9	5	87.79	0.89	86.23	87.09	88.12	88.32	88.44
ΤΠ10	5	89.53	1.18	87.79	88.39	89.73	90.56	90.72
ΤΠ11	5	88.89	2.83	84.59	86.33	88.94	91.43	91.64
ΤΠ12	5	89.60	2.10	87.02	87.60	90.04	91.37	92.47

Για το Flask, τα αποτελέσματα εμφανίζονται στο σχήμα 9 και στον πίνακα 18. Και στην περίπτωση του Flask, η αρχική διάταξη των περιπτώσεων ελέγχου (ΤΠ2) είναι αυτή που παρουσιάζει τις χαμηλότερες τιμές της μετρικής APFD. Οι τεχνικές προτεραιοποίησης με χρήση κάλυψης κώδικα σε επίπεδο αρχείου (ΤΠ5-6) δίνουν αποτελέσματα που κυμαίνονται μεταξύ του 55% και 65%, χαμηλότερα σε σχέση με τις ίδιες τεχνικές στα προηγούμενα έργα. Για το Flask, αυτές δεν παρουσιάζουν σημαντικές διαφορές με την τυχαία διάταξη (ΤΠ3), παρά μόνο με την αντίστροφη διάταξη (ΤΠ4). Σε αντίθεση πάλι με τα προηγούμενα έργα, η αντίστροφη διάταξη εμφανίζει χαμηλότερες τιμές σε σχέση με την τυχαία διάταξη. Όμως, οι τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία κάλυψης με περισσότερη λεπτομέρεια, δίνουν αρκετά υψηλότερες τιμές της μετρικής APFD, οι οποίες σε περιπτώσεις ξεπερνούν το 90%. Και στην περίπτωση του Flask, οι τεχνικές προτεραιοποίησης με χρήση πληροφορίας κάλυψης κώδικα σε επίπεδο διακλάδωσης (ΤΠ9-10) και εντολής (ΤΠ11-12) παρουσιάζουν τις υψηλότερες τιμές της μετρικής APFD. Οι τεχνικές προτεραιοποίησης με χρήση επιπρόσθετης κάλυψης κώδικα εμφανίζουν ελαφρώς υψηλότερες τιμές της μετρικής APFD σε σχέση με αυτές που χρησιμοποιούν πληροφορία ολικής κάλυψης κώδικα, ειδικά για την περίπτωση της κάλυψης κώδικα σε επίπεδο διακλάδωσης (ΤΠ10 σε σχέση με ΤΠ9).

Τέλος, για το Six, τα αποτελέσματα εμφανίζονται στο σχήμα 10 και στον πίνακα 19. Καθώς το Six είναι το έργο με το μικρότερο μέγεθος, η μεταβλητότητα των τιμών της μετρικής APFD μεταξύ των διαφορετικών εκδόσεών του για κάθε τεχνική προτεραιοποίησης, είναι μεγαλύτερη. Αυτό συμβαίνει, γιατί με τόσο μικρό αριθμό περιπτώσεων



Σχήμα 9: Θηκόγραμμα τιμών APFD του Flask για κάθε τεχνική προτεραιοποίησης (ΤΠ). ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

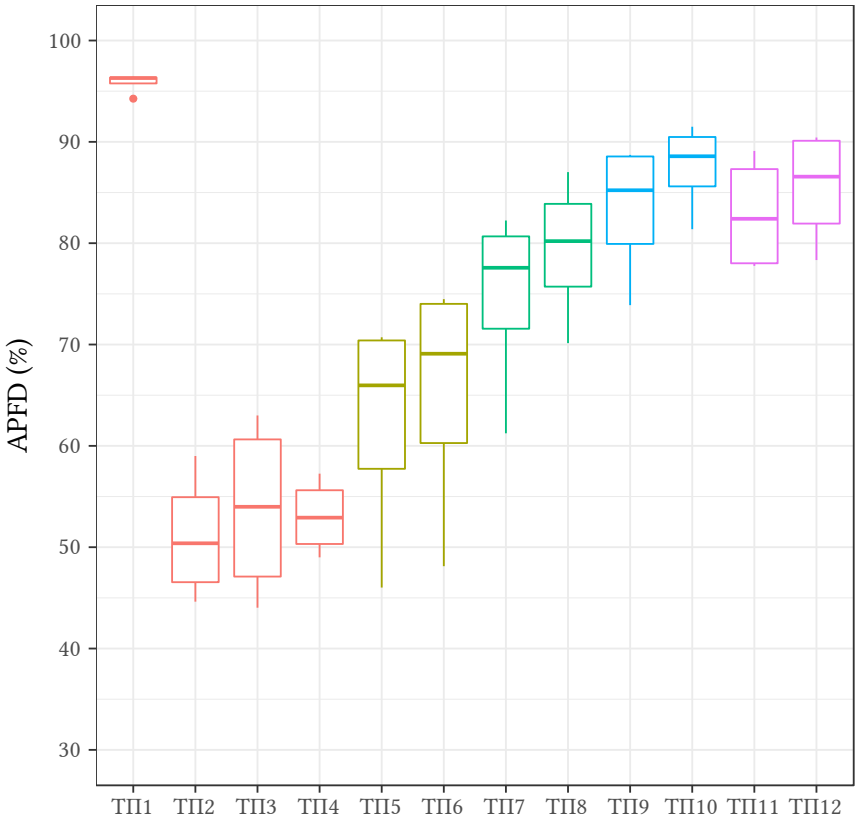
Πίνακας 19: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) του Six

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	4	95.82	1.04	94.26	94.76	96.30	96.40	96.43
ΤΠ2	4	51.10	6.47	44.63	45.27	50.39	57.65	59.00
ΤΠ3	4	53.75	9.11	44.03	45.05	53.99	62.21	63.00
ΤΠ4	4	53.02	3.81	49.00	49.44	52.92	56.71	57.26
ΤΠ5	4	62.17	11.55	46.02	49.92	65.97	70.61	70.71
ΤΠ6	4	65.20	12.29	48.13	52.18	69.09	74.32	74.48
ΤΠ7	4	74.66	9.45	61.24	64.68	77.57	81.72	82.24
ΤΠ8	4	79.39	7.28	70.14	72.00	80.20	85.97	87.01
ΤΠ9	4	83.26	7.00	73.89	75.90	85.22	88.66	88.71
ΤΠ10	4	87.51	4.49	81.38	82.79	88.58	91.16	91.49
ΤΠ11	4	82.92	5.84	77.76	77.85	82.41	88.51	89.10
ΤΠ12	4	85.48	5.82	78.34	79.54	86.57	90.32	90.43

ελέγχου, η εύρεση ενός σφάλματος ακόμα και με διαφορά μίας περίπτωσης ελέγχου, οδηγεί σε μεγαλύτερες ποσοστιαίες διαφορές των τιμών APFD. Παρόλα αυτά, το μοτίβο που εμφανίστηκε σε όλα τα προηγούμενα έργα, σε σχέση με τις τιμές της μετρικής APFD, είναι παρόμοιο. Οι τεχνικές σύγκρισης παρουσιάζουν τις μικρότερες τιμές APFD, αν και μεταξύ τους δεν εμφανίζονται σημαντικές διαφορές. Σε σχέση με αυτές, οι τεχνικές προτεραιοποίησης με χρήση πληροφορίας κάλυψης κώδικα σε επίπεδο αρχείου (ΤΠ5-6), δίνουν καλύτερα αποτελέσματα. Οι τιμές της μετρικής APFD είναι ακόμα υψηλότερες για τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία κάλυψης σε επίπεδο μεθόδου (ΤΠ7-8), ενώ για ακόμα μία φορά, οι υψηλότερες τιμές της μετρικής APFD εμφανίζονται για τις τεχνικές προτεραιοποίησης με χρήση πληροφορίας κάλυψης κώδικα σε επίπεδο διακλάδωσης (ΤΠ9-10) και εντολής (ΤΠ11-12). Η χρήση της πληροφορίας επιπρόσθετης κάλυψης, εμφανίζει γενικά υψηλότερες τιμές της μετρικής APFD σε σχέση με τις τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία ολικής κάλυψης, κάτι που είναι περισσότερο εμφανές για τις τεχνικές που χρησιμοποιούν πληροφορία κάλυψης σε επίπεδο διακλάδωσης.

Επιπρόσθετα, στο σχήμα 11 και στον πίνακα 20 εμφανίζονται συγκεντρωτικά τα αποτελέσματα των τιμών της μετρικής APFD για όλα τα έργα λογισμικού που μελετήθηκαν.

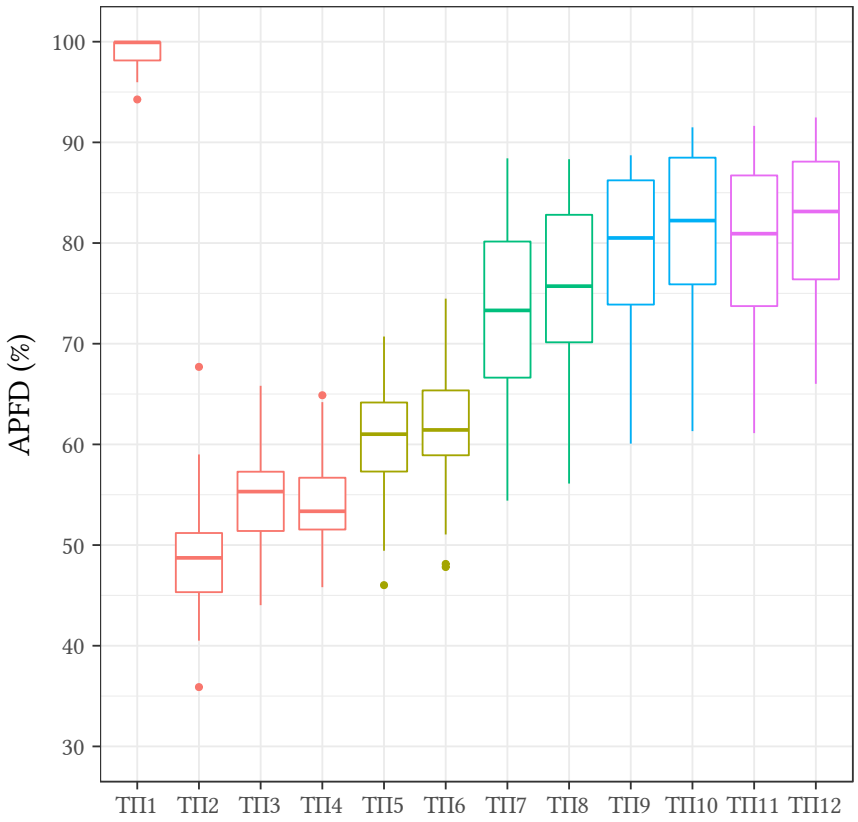
Τα αποτελέσματα του ελέγχου της ανάλυσης διακύμανσης δύο παραγόντων (two-way ANOVA) που πραγματοποιήθηκε για τη σύγκριση των διαφορετικών τεχνικών προτεραιοποίησης και των έργων λογισμικού εμφανίζονται στον πίνακα 21. Σε αυτά, φαί-



Σχήμα 10: Θηκόγραμμα τιμών APFD του Six για κάθε τεχνική προτεραιοποίησης (ΤΠ). ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

Πίνακας 20: Περιγραφική στατιστική των τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) και όλα τα έργα λογισμικού που μελετήθηκαν

ΤΠ	N	Mean	SD	Min	Q1	Median	Q3	Max
ΤΠ1	29	98.84	1.59	94.26	97.98	99.93	99.96	99.98
ΤΠ2	29	49.03	5.96	35.89	45.11	48.73	51.28	67.70
ΤΠ3	29	54.80	5.15	44.03	51.14	55.31	57.74	65.82
ΤΠ4	29	54.49	4.46	45.82	51.51	53.36	56.97	64.88
ΤΠ5	29	60.39	5.91	46.02	56.55	61.01	64.92	70.71
ΤΠ6	29	61.53	6.72	47.82	58.53	61.44	65.51	74.48
ΤΠ7	29	73.50	8.86	54.41	66.38	73.31	81.19	88.41
ΤΠ8	29	76.07	7.91	56.11	70.11	75.72	82.82	88.33
ΤΠ9	29	78.82	8.50	60.07	73.09	80.50	86.55	88.71
ΤΠ10	29	81.22	8.39	61.32	75.47	82.23	88.74	91.49
ΤΠ11	29	79.68	8.18	61.12	73.58	80.93	87.39	91.64
ΤΠ12	29	81.31	7.83	66.02	75.26	83.13	88.13	92.47



Σχήμα 11: Θηκόγραμμα τιμών APFD για κάθε τεχνική προτεραιοποίησης (ΤΠ) και όλα τα έργα λογισμικού που μελετήθηκαν. ●: Τεχνικές σύγκρισης, ●: ΤΠ σε επίπεδο αρχείου, ●: ΤΠ σε επίπεδο μεθόδου, ●: ΤΠ σε επίπεδο διακλάδωσης, ●: ΤΠ σε επίπεδο εντολής.

Πίνακας 21: Αποτελέσματα ελέγχου ανάλυσης διακύμανσης δύο παραγόντων για τη σύγκριση μεταξύ των τεχνικών προτεραιοποίησης (ΤΠ) και των έργων λογισμικού

	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		

νεται πως υπάρχει στατιστικά σημαντική διαφορά μεταξύ των τεχνικών παραμετροποίησης αλλά και μεταξύ των έργων λογισμικού που μελετήθηκαν. Στατιστικά σημαντική είναι επίσης και η αλληλεπίδραση των δύο παραγόντων, κάτι που σημαίνει ότι η επίδραση της χρήσης των διαφορετικών τεχνικών προτεραιοποίησης μεταβάλλεται για τα διαφορετικά έργα λογισμικού.

Τα αποτελέσματα του ελέγχου των πολλαπλών συγκρίσεων κατά ζεύγη του Tukey που ακολουθεί τον έλεγχο της ανάλυσης διακύμανσης, όσον αφορά στη σύγκριση των τεχνικών προτεραιοποίησης, εμφανίζονται σε συνοπτική μορφή στον πίνακα 22. Συγκεκριμένα, παρουσιάζονται οι διαφορετικές Τεχνικές Προτεραιοποίησης, σε αύξουσα σειρά μέσων όρων, καθώς και ο τρόπος που ομαδοποιούνται οι Τεχνικές Προτεραιοποίησης σύμφωνα με τον έλεγχο του Tukey. Μέσοι όροι που ανήκουν στην ίδια ομάδα, δεν παρουσιάζουν στατιστικά σημαντικές διαφορές. Τα πλήρη αποτελέσματα των συγκρίσεων κατά ζεύγη, λόγω του όγκου τους, βρίσκονται στο παράρτημα (πίν. 24, σελ. 156). Στο σύνολο των έργων λογισμικού που μελετήθηκαν, προκύπτει ότι η εκτέλεση των περιπτώσεων ελέγχου με την αρχική τους διάταξη, είναι αυτή που δίνει τους μικρότερους ρυθμούς εντοπισμού σφαλμάτων. Η αντίστροφη διάταξη, όπως και η τυχαία διάταξη, εμφανίζουν στατιστικά σημαντικές βελτιώσεις της μετρικής APFD σε σχέση με την αρχική διάταξη. Οι διαφορές μεταξύ της τυχαίας διάταξης και της αντίστροφης διάταξης δεν είναι στατιστικά σημαντικές. Εφόσον η χρήση της αντίστροφης διάταξης ή της τυχαίας διάταξης για την εκτέλεση των περιπτώσεων ελέγχου δεν απαιτεί αυξημένη πολυπλοκότητα και γενικά δεν έχει κανένα επιπλέον κόστος σε σχέση με την αρχική διάταξη, πιθανόν να είναι μια καλή επιλογή αν ο στόχος είναι μια μικρή, αλλά σημαντική βελτίωση του ρυθμού ανίχνευσης σφαλμάτων από τη σουίτα ελέγχου.

Σε ότι αφορά στις τεχνικές προτεραιοποίησης που χρησιμοποιούν την κάλυψη κώδικα ως μέτρο πρόβλεψης του ρυθμού εντοπισμού σφαλμάτων, όλες τους εμφανίζουν στατιστικά σημαντικά αυξημένους ρυθμούς σε σχέση με την αρχική διάταξη, την αντί-

Πίνακας 22: Ομαδοποίηση των μέσων όρων των τιμών APFD των τεχνικών προτεραιοποίησης σύμφωνα με τον έλεγχο πολλαπλών συγκρίσεων κατά ζεύγη του Tukey. Τεχνικές που ανήκουν στην ίδια ομάδα, δεν παρουσιάζουν στατιστικά σημαντικές διαφορές

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

στροφη διάταξη αλλά και με την τυχαία διάταξη.

Σημαντικές διαφορές όμως εντοπίζονται και μεταξύ των τεχνικών που βασίζονται στην κάλυψη κώδικα. Φαίνεται πως όσο πιο λεπτομερές είναι το επίπεδο της ανάλυσης της κάλυψης κώδικα, τόσο πιο αυξημένοι είναι οι ρυθμοί εντοπισμού των σφαλμάτων.

Πιο συγκεκριμένα, οι μέθοδοι που βασίζονται στην κάλυψη κώδικα σε επίπεδο μεθόδου, εμφανίζουν καλύτερα αποτελέσματα από αυτές που βασίζονται στην κάλυψη κώδικα σε επίπεδο αρχείου. Μάλιστα, οι διαφορές αυτές είναι περισσότερο έντονες για έργα λογισμικού που οργανώνουν τον κώδικά τους σε λιγότερα αρχεία πηγαίου κώδικα, σε σχέση με τον συνολικό αριθμό γραμμών κώδικα.

Ακόμα καλύτερα αποτελέσματα λαμβάνονται σε όλες τις περιπτώσεις όταν χρησιμοποιείται ακόμα πιο λεπτομερές επίπεδο της ανάλυσης της κάλυψης κώδικα. Αν χρησιμοποιείται ανάλυση κώδικα σε επίπεδο διακλάδωσης ή εντολής, οι ρυθμοί εντοπισμού σφαλμάτων είναι αυξημένοι σε σχέση με οποιαδήποτε άλλη τεχνική προτεραιοποίησης. Μεταξύ τους όμως, οι τεχνικές προτεραιοποίησης που βασίζονται στην κάλυψη σε επίπεδο διακλάδωσης και σε επίπεδο εντολής δεν εμφανίζουν στατιστικά σημαντικές διαφορές.

Μικρές διαφορές, οι οποίες όμως δεν είναι στατιστικά σημαντικές, εντοπίζονται και μεταξύ όλων των τεχνικών προτεραιοποίησης που χρησιμοποιούν πληροφορία ολικής κάλυψης κώδικα και επιπρόσθετης κάλυψης κώδικα. Σε όλες τις περιπτώσεις (επίπεδο αρχείο, μεθόδου, διακλάδωσης, εντολής), οι τεχνικές που χρησιμοποιούν την πληροφο-

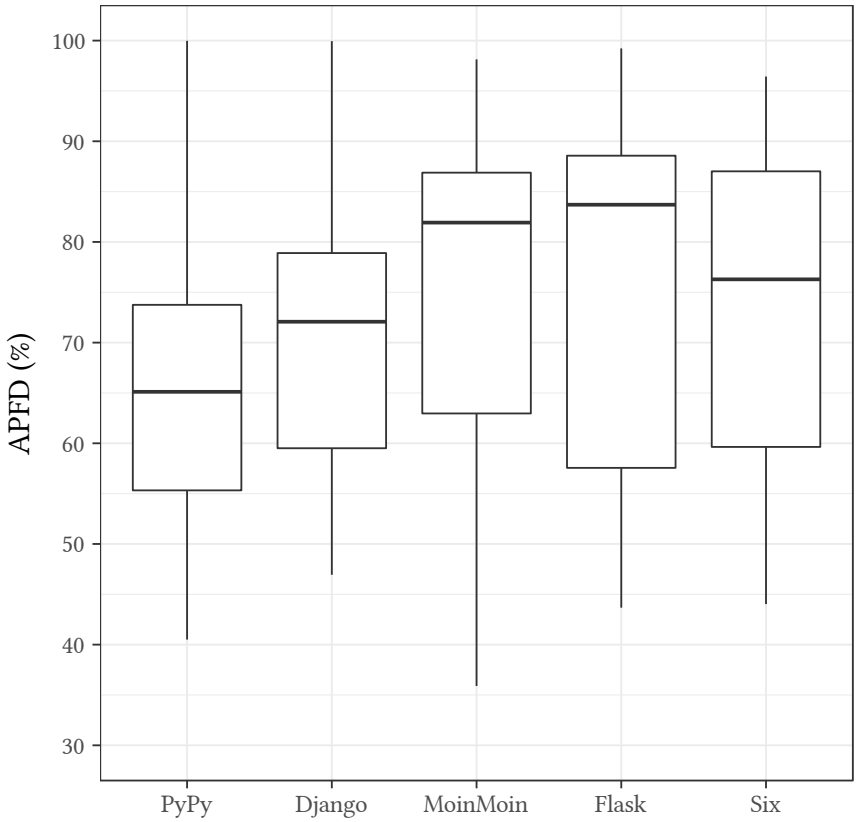
ρία της επιπρόσθετης κάλυψης εμφανίζουν ελαφρώς καλύτερους ρυθμούς εντοπισμού σφαλμάτων σε σχέση με αυτές που χρησιμοποιούν πληροφορία ολικής κάλυψης.

Βέβαια, σε κανένα έργο λογισμικού, καμία από τις τεχνικές προτεραιοποίησης δεν πλησιάζει τα επίπεδα εντοπισμού σφαλμάτων που θεωρητικά υπολογίζονται για τη βέλτιστη διάταξη. Παρόλα αυτά, τα κέρδη είναι σημαντικά όταν χρησιμοποιούνται τεχνικές που χρησιμοποιούν μεγάλη λεπτομέρεια στο επίπεδο ανάλυσης της κάλυψης κώδικα. Εξετάζοντας τους υπολογισμούς της μετρικής APFD (Παράρτημα σελ. 69–155) είναι εμφανές πως, ειδικά για τις τεχνικές που χρησιμοποιούν πληροφορία κάλυψης σε επίπεδο διακλάδωσης ή εντολής, σχεδόν σε όλες τις περιπτώσεις, όλα τα σφάλματα έχουν εντοπιστεί όταν έχει εκτελεστεί το 20% με 50% των περιπτώσεων ελέγχου, ανάλογα με το έργο λογισμικού.

Αν και στη βιβλιογραφία δεν έχουν χρησιμοποιηθεί ταυτόχρονα όλες οι παραπάνω τεχνικές προτεραιοποίησης στην ίδια έρευνα, ενώ και το μέγεθος των έργων λογισμικού που χρησιμοποιείται είναι συνήθως σχετικά μικρό, στα επίπεδα των μερικών χιλιάδων γραμμών κώδικα το πολύ, τα αποτελέσματα που παρουσιάζονται εκεί σε διαφορετικές έρευνες [13, 34, 30, 26, 10], και αναφέρονται σε σουίτες ελέγχου που εκτελούνται σε έργα λογισμικού γραμμένα σε γλώσσες προγραμματισμού C και Java, συμφωνούν με τα αυτά της παρούσας εργασίας.

Στο σχήμα 12 φαίνεται η μεταβολή των τιμών της μετρικής APFD μεταξύ των έργων που μελετήθηκαν. Ταυτόχρονα, στον πίνακα 23 εμφανίζονται οι τιμές των μέσων όρων των τιμών APFD για κάθε έργο, καθώς και η ομαδοποίησή τους σύμφωνα με τον έλεγχο του Tukey. Τα πλήρη αποτελέσματα του ελέγχου του Tukey σε ότι αφορά στη σύγκριση μεταξύ των έργων βρίσκονται στο παράρτημα (σελ. 159). Επιπρόσθετα, το γράφημα των αλληλεπιδράσεων μεταξύ των δύο εξεταζόμενων παραγόντων (τεχνική προτεραιοποίησης και έργο λογισμικού) φαίνεται στο σχήμα 13.

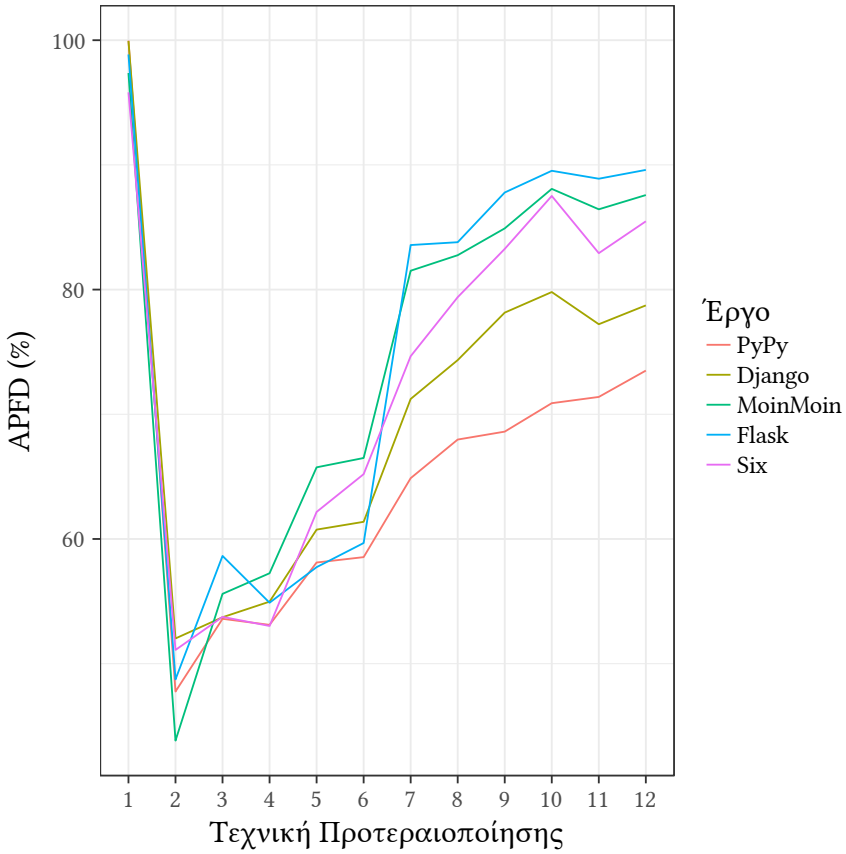
Το Six είναι το μόνο έργο με αρκετά μικρό μέγεθος που μελετήθηκε. Ωστόσο, με την εξαίρεσή του, τουλάχιστον για τις μεθόδους προτεραιοποίησης που βασίζονται στην κάλυψη κώδικα σε επίπεδο μεθόδου, διακλάδωσης και εντολής, φαίνεται πως υπάρχει μια σχέση μεταξύ του μεγέθους του κώδικα ενός έργου και του ρυθμού εντοπισμού σφαλμάτων. Συγκεκριμένα, φαίνεται πως για μεγαλύτερα έργα λογισμικού οι τεχνικές προτεραιοποίησης παρουσιάζουν μικρότερες βελτιώσεις ως προς την ανίχνευση των σφαλμάτων νωρίτερα. Αυτό πιθανόν να οφείλεται στον πολύ μεγάλο όγκο τόσο των γραμμών



Σχήμα 12: Θηκόγραμμα τιμών APFD για τα έργα λογισμικού που μελετήθηκαν

Πίνακας 23: Ομαδοποίηση των μέσων όρων των τιμών APFD των έργων λογισμικού που μελετήθηκαν σύμφωνα με τον έλεγχο πολλαπλών συγκρίσεων κατά ζεύγη του Tukey. Τεχνικές που ανήκουν στην ίδια ομάδα, δεν παρουσιάζουν στατιστικά σημαντικές διαφορές

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



Σχήμα 13: Γράφημα αλληλεπιδράσεων μεταξύ των έργων λογισμικού και των τεχνικών προτεραιοποίησης

του πηγαίου κώδικα, όσο και των περιπτώσεων ελέγχου που περιλαμβάνονται στις σουίτες περιπτώσεων ελέγχου των έργων λογισμικού. Σε σχέση με τον συνολικό αριθμό των περιπτώσεων ελέγχου που περιλαμβάνονται, οι περιπτώσεις ελέγχου που παρουσιάζουν σφάλματα είναι σε κάθε περίπτωση πολύ λίγες. Οπότε είναι και δυσκολότερο μέσα σε αυτό το μεγάλο πλήθος των περιπτώσεων ελέγχου να εντοπιστούν αυτές που εμφανίζουν σφάλματα.

Αν και η αλληλεπίδραση των δύο εξεταζόμενων παραγόντων είναι σημαντική, από το γράφημα των αλληλεπιδράσεων (σχ. 13), φαίνεται πως, με μικρές εξαιρέσεις, η συμπεριφορά όλων των έργων λογισμικού για τις διαφορετικές τεχνικές προτεραιοποίησης είναι παρόμοια. Οι εξαιρέσεις είναι προφανώς και αυτές στις οποίες οφείλεται η στατιστική σημαντικότητα. Φαίνεται για παράδειγμα, πως μεταξύ των ΤΠ10 (Branch-addtl) και ΤΠ11 (Statement-total) για όλα τα έργα λογισμικού ισχύει ότι η τιμή του APFD ελαττώνεται, εκτός από την περίπτωση του PyPy, για το οποίο αυτή αυξάνεται, έστω και ελάχιστα.

Όπως προαναφέρθηκε, στη βιβλιογραφία δεν υπάρχουν έρευνες για έργα λογισμικού του μεγέθους του PyPy ή του Django, με εκατοντάδες χιλιάδες ή εκατομμύρια γραμμών κώδικα. Τα μεγαλύτερα έργα λογισμικού που έχουν μελετηθεί είναι το Apache Ant [51] και το Apache JMeter [52] τα οποία είχαν αντίστοιχα περίπου 40.000 και 80.000 γραμμές κώδικα [30]. Λαμβάνοντας υπόψη ότι τα έργα αυτά είναι γραμμένα σε γλώσσα προγραμματισμού Java, η οποία, κατά γενική ομολογία, έχει σύνταξη που δεν είναι όσο λιτή όσο αυτή της γλώσσας Python, συμπεραίνεται πως τα έργα αυτά υπολείπονται πολυπλοκότητας σε σχέση με κάποια από αυτά που μελετήθηκαν στην παρούσα εργασία. Αν γίνει σύγκριση των αποτελεσμάτων της παρούσας εργασίας με τα αντίστοιχα που εμφανίζονται στη βιβλιογραφία, είναι εμφανές πως οι τιμές της μετρικής APFD στη βιβλιογραφία είναι υψηλότερες. Αυτό πιθανόν επιβεβαιώνει περαιτέρω το συμπέρασμα ότι σε μικρότερα έργα λογισμικού είναι δυνατό να επιτευχθούν υψηλότεροι ρυθμοί εντοπισμού σφαλμάτων.

5 Επίλογος

Στο κεφάλαιο αυτό, παρουσιάζονται συνοπτικά τα αποτελέσματα και τα συμπεράσματα της έρευνας, οι προβληματισμοί που προκύπτουν λόγω των ορίων και των περιορισμών της έρευνας καθώς και ιδέες για μελλοντικές επεκτάσεις της έρευνας.

5.1 Σύνοψη και συμπεράσματα

Στην παρούσα εργασία αξιολογήθηκαν τεχνικές προτεραιοποίησης περιπτώσεων ελέγχου σε έργα ανοικτού κώδικα, υλοποιημένα με τη χρήση της γλώσσας Python και με τη χρήση του πλαισίου ελέγχου pytest.

Για το σκοπό αυτό, χρησιμοποιήθηκαν τα έργα λογισμικού PyPy, Django, MoinMoin, Flask και Six. Σφάλματα εμφυτεύθηκαν στις σουίτες ελέγχου των έργων και σε διαφορετικές εκδόσεις αυτών.

Οι τεχνικές προτεραιοποίησης που εξετάστηκαν βασίστηκαν στην πληροφορία κάλυψης κώδικα που καταγράφηκε για κάθε μία από τις περιπτώσεις ελέγχου. Εξετάστηκε η χρήση της πληροφορίας κάλυψης κώδικα σε διαφορετικά επίπεδα λεπτομέρειας, σε επίπεδο αρχείου, μεθόδου, διακλάδωσης και εντολής. Για κάθε μία από αυτές αξιολογήθηκαν παραλλαγές που χρησιμοποίησαν πληροφορία ολικής κάλυψης και επιπρόσθετης κάλυψης. Επιπρόσθετα, αξιολογήθηκαν εναλλακτικές διατάξεις της σουίτας ελέγχου, η χρήση τυχαίας διάταξης και η χρήση της αντίστροφης διάταξης σε σχέση με την αρχική.

Η αξιολόγηση των διατάξεων που προέκυψαν, έγινε χρησιμοποιώντας την μετρική APFD (Average Percentage Faults Detected), η οποία εκτιμά το ρυθμό ανίχνευσης σφαλμάτων σε μία σουίτα ελέγχου.

Τα αποτελέσματα της έρευνας έδειξαν ότι η χρήση της πληροφορίας κάλυψης για την προτεραιοποίηση των περιπτώσεων ελέγχου, οδηγεί σε σημαντικές βελτιώσεις του ρυθμού ανίχνευσης σφαλμάτων. Μάλιστα, η βελτίωση είναι τόσο μεγαλύτερη όσο περισσότερη η λεπτομέρεια υπολογισμού της πληροφορίας κάλυψης. Η χρήση της πληροφορίας κάλυψης σε επίπεδο εντολής ή διακλάδωσης έδωσε σε κάθε περίπτωση τα καλύτερα αποτελέσματα.

Ακόμα και η χρήση μια τυχαίας διάταξης των περιπτώσεων ελέγχου ή της αντίστροφης διάταξης σε σχέση με την αρχική μπορεί να βελτιώσει σημαντικά τους ρυθμούς

ανίχνευσης σφαλμάτων, όχι όμως στα επίπεδα που επιτυγχάνεται με τη χρήση πληροφορίας κάλυψης.

Αν και οι τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία επιπρόσθετης κάλυψης, παρουσιάζουν σχεδόν πάντα καλύτερες τιμές της μετρικής APFD σε απόλυτες τιμές, σε σχέση με τις τεχνικές προτεραιοποίησης που χρησιμοποιούν πληροφορία ολικής κάλυψης, οι διαφορές μεταξύ τους δεν είναι στατιστικά σημαντικές. Αν ληφθεί υπόψιν ότι η καταγραφή της επιπρόσθετης κάλυψης κώδικα είναι μια διαδικασία σαφώς περισσότερο περίπλοκη από την καταγραφή της ολικής κάλυψης κώδικα, συμπεραίνεται πως το κέρδος δεν είναι αρκετό για να δικαιολογήσει την επιλογή της πρώτης. Η πληροφορία επιπρόσθετης κάλυψης κώδικα απαιτεί την αποθήκευση της κάθε γραμμής κώδικα (ή της κάθε διακλάδωσης, ή της κάθε μεθόδου, ή του κάθε αρχείου) που εκτελεί η κάθε περίπτωση ελέγχου και αξιολόγηση αυτής της πληροφορίας σε κάθε βήμα επιλογής της επόμενης περίπτωσης ελέγχου που πρόκειται να εκτελεστεί. Αντίθετα, η πληροφορία ολικής κάλυψης κώδικα απαιτεί την αποθήκευση μόνο μίας τιμής για κάθε περίπτωση ελέγχου, του ποσοστού κάλυψης κώδικα της περίπτωσης ελέγχου στο σύνολο του κώδικα του έργου.

Τα αποτελέσματα δείχνουν πως το μέγεθος των έργων λογισμικού παίζει ρόλο στο ρυθμό ανίχνευσης σφαλμάτων που μπορεί να επιτευχθεί με τη χρήση των τεχνικών προτεραιοποίησης που χρησιμοποιούν πληροφορίας κάλυψης κώδικα. Συγκεκριμένα, και με την εξαίρεση ενός έργου, φαίνεται πως, σε μεγαλύτερα έργα δεν επιτυγχάνονται τόσο αυξημένοι ρυθμοί ανίχνευσης σφαλμάτων όπως συμβαίνει σε μικρότερα.

Τέλος τα αποτελέσματα της παρούσας έρευνας, δείχνουν τις ίδιες τάσεις με τα αντίστοιχα που έχουν αναφερθεί σε άλλες γλώσσες προγραμματισμού, όχι όμως στο ίδιο επίπεδο. Σε καμία περίπτωση δεν εντοπίστηκαν ρυθμοί ανίχνευσης σφαλμάτων τόσο υψηλοί, όσο αυτοί που αναφέρονται στη βιβλιογραφία για τις γλώσσες C και Java. Το γεγονός αυτό όμως, πιθανόν να οφείλεται στο ότι τα έργα αυτά έχουν σημαντικά μικρότερο μέγεθος από αυτά που χρησιμοποιήθηκαν στην παρούσα εργασία.

5.2 Όρια και περιορισμοί της έρευνας

Η παράγραφος αυτή πραγματεύεται τα όρια και τους περιορισμούς της έρευνας σε σχέση με την εγκυρότητά των αποτελεσμάτων.

- Όρια σε σχέση με τη *δομική εγκυρότητα* (construct validity). Αυτά αφορούν στις

ίδιες τις μετρικές που ήταν αντικείμενο της μελέτης και τη δυνατότητά τους να εκτιμήσουν πραγματικά αυτό για το οποίο προορίζονται.

- Όρια σε σχέση με την *εσωτερική εγκυρότητα* (internal validity). Αυτά αφορούν στις υποθέσεις που πραγματοποιήθηκαν σχετικά με την ύπαρξη σχέσης μεταξύ των εξεταζόμενων ανεξαρτήτων και εξαρτημένων μεταβλητών.

- Όρια σε σχέση με την *εξωτερική εγκυρότητα* (external validity). Αυτά αφορούν στη δυνατότητα να γενικευτούν τα αποτελέσματα της έρευνας.

5.2.1 Δομική εγκυρότητα

Στην παρούσα έρευνα χρησιμοποιήθηκε η μετρική APFD για την εκτίμηση του ρυθμού ανίχνευσης σφαλμάτων σε μια σουίτα περιπτώσεων ελέγχου. Οι ίδιες οι μετρήσεις είναι απόλυτα ακριβείς, αλλά η μετρική APFD δεν είναι ο μόνος τρόπος με τον οποίο μπορεί να εκτιμηθεί ο ρυθμός ανίχνευσης σφαλμάτων. Για παράδειγμα, δεν χρησιμοποιείται σε κανένα σημείο πληροφορία σχετική με το αν κάποια επόμενη περίπτωση ελέγχου ανιχνεύει πάλι ένα σφάλμα το οποίο έχει ανιχνευθεί ήδη. Πληροφορία όπως αυτή, ενδεχομένως μπορεί να χρησιμοποιηθεί από debuggers ώστε να εντοπιστεί ευκολότερα ένα πρόβλημα.

Επιπρόσθετα, η μετρική APFD δεν λαμβάνει υπόψιν την πιθανότητα διαφορετικά σφάλματα και περιπτώσεις ελέγχου να επισύρουν διαφορετικά κόστη. Ενδεχομένως μια άλλη πρακτική να μπορούσε να είναι, αντί της εκτίμησης του ρυθμού ανίχνευσης σφαλμάτων, του ποσοστού των περιπτώσεων ελέγχου που πρέπει να εκτελεστούν ώστε να αποκαλυφθούν όλα τα σφάλματα που υπάρχουν.

Τέλος, λόγω του ότι η μετρική APFD μετρά την αποδοτικότητα της προτεραιοποίησης μονοδιάστατα, θα πρέπει να γίνει αντιπαράθεση με άλλες μετρικές, για τη διασταύρωση των αποτελεσμάτων.

5.2.2 Εσωτερική εγκυρότητα

Ίσως η σημαντικότερη ανησυχία, σε ότι αφορά στην εσωτερική εγκυρότητα, σε τέτοιου είδους μελέτες είναι ότι τα εργαλεία και οι μέθοδοι που χρησιμοποιήθηκαν ενδεχομένως να επηρεάζουν τα αποτελέσματα που λαμβάνονται.

Μια πηγή τέτοιας επίδρασης είναι η σωστή λειτουργία των εργαλείων που χρησιμοποιήθηκαν. Για την άμβλυνση της πιθανότητας ύπαρξης τέτοιων σφαλμάτων, τα αποτελέσματα διασταυρώθηκαν για το Six 1.8.0, το οποίο είναι το μικρότερο από τα έργα που

μελετήθηκαν, αλλά και για ένα ακόμα μικρότερο έργο, το docopt (386 γραμμές κώδικα) [54], το οποίο δεν ήταν αντικείμενο της μελέτης, με αποτελέσματα που λήφθηκαν για το ίδιο έργο χειροκίνητα.

Μια ακόμα πηγή αυτής της επίδρασης είναι οι διαφορές μεταξύ των εισόδων που χρησιμοποιήθηκαν, δηλαδή οι διαφορές μεταξύ των έργων λογισμικού που μελετήθηκαν, η τοπικότητα των αλλαγών ή η σύνθεση της κάθε σουίτας ελέγχου. Δεν υπάρχει κάποιος τρόπος να εκλείψουν οι διαφορές αυτές, αλλά για τον περιορισμό των προβλημάτων αυτού του είδους, όλες οι τεχνικές προτεραιοποίησης εφαρμόστηκαν σε όλες τις σουίτες ελέγχου, σε κάθε ένα από τα έργα λογισμικού και τις εκδόσεις τους που χρησιμοποιήθηκαν.

5.2.3 Εξωτερική εγκυρότητα

Ο μεγαλύτερος κίνδυνος ως προς την εξωτερική εγκυρότητα είναι το πόσο αντιπροσωπευτικά δείγματα σε σχέση με τον πληθυσμό των έργων λογισμικού υλοποιημένων σε Python, είναι τα έργα που μελετήθηκαν. Στην παρούσα μελέτη τα έργα λογισμικού που μελετήθηκαν θα μπορούσαν κυρίως να κατηγοριοποιηθούν από μεσαίου μεγέθους έως μεγάλα. Η πρόσβαση σε ακόμα μεγαλύτερα έργα λογισμικού είναι πολύ δύσκολη, αν όχι αδύνατη, αφού αυτά δεν είναι έργα ανοικτού κώδικα και ανήκουν σε μεγάλες εταιρίες πληροφορικής. Η επέκταση σε μικρότερα έργα, για τα οποία ούτως ή άλλως υπήρχε ένας αντιπρόσωπος στην παρούσα μελέτη, ίσως να μην έχει αξία, αφού σε μικρότερα έργα, το πρόβλημα της προτεραιοποίησης των περιπτώσεων ελέγχου δεν είναι σημαντικό.

Προφανώς υπάρχουν επίσης κίνδυνοι σχετικά με τα σφάλματα που τεχνητά δημιουργήθηκαν για την αξιολόγηση των τεχνικών προτεραιοποίησης. Για τον περιορισμό τους, χρησιμοποιήθηκαν, όπου ήταν δυνατό, πραγματικά σφάλματα τα οποία έχουν αποκαλυφθεί σε νεότερες εκδόσεις των έργων λογισμικού. Σχετικά με τα σφάλματα που εμφυτεύθηκαν στον κώδικα των έργων, αυτά δημιουργήθηκαν βασιζόμενα μόνο στην εμπειρία του συγγραφέα. Σε αντίστοιχες μελέτες στη βιβλιογραφία, μια συνηθισμένη πρακτική ήταν η χρησιμοποίηση ομάδων σχετικά έμπειρων (μεταπτυχιακών) φοιτητών στην Τεχνολογία Λογισμικού και την ανάπτυξη κώδικα, ώστε να εξαλειφθεί η όποια προσωπική επιρροή. Αυτό δεν ήταν εφικτό να συμβεί στο επίπεδο της παρούσας έρευνας και ενδεχομένως τα σφάλματα που εμφυτεύθηκαν να μην είναι αρκετά μεγάλης ποικιλίας, αντίστοιχης με αυτή που εμφανίζεται στην πράξη κατά την ανάπτυξη έργων λογισμικού.

Γενικά, όποιες ανησυχίες σχετικά με την εξωτερική εγκυρότητα, μπορούν να αντιμετωπιστούν μόνο με τη διενέργεια περισσότερων μελετών, σε περισσότερα έργα λογισμικού.

5.3 Μελλοντικές Επεκτάσεις

Μια προφανής επέκταση της έρευνας είναι η μελέτη περισσότερων έργων λογισμικού. Για μεγαλύτερα έργα λογισμικού, αυτό απαιτεί τη συνεργασία με κάποια μεγάλη εταιρία πληροφορικής, η οποία χρησιμοποιεί σε μεγάλο βαθμό τη γλώσσα Python για την ανάπτυξη του λογισμικού. Περισσότερα έργα του ίδιου ή μικρότερου μεγέθους πιθανόν να μπορούν να βρεθούν ευκολότερα στο χώρο του λογισμικού ανοικτού κώδικα.

Επίσης, θα μπορούσαν να μελετηθούν περισσότερες στρατηγικές βελτιστοποίησης, πέρα από αυτή του άπληστου αλγορίθμου που χρησιμοποιήθηκε στην παρούσα εργασία και η σύγκριση μεταξύ τους ως προς την ικανότητα πρόβλεψης μεγαλύτερων ρυθμών ανίχνευσης σφαλμάτων.

Ακόμα μία επέκταση, θα μπορούσε να είναι αξιοποίηση πληροφορίας κόστους εκτέλεσης των περιπτώσεων ελέγχου. Στην πιο απλή περίπτωση, το κόστος θα μπορούσε να αναφέρεται στο χρόνο εκτέλεσης της κάθε περίπτωσης ελέγχου, μέσω της χρήσης μιας μετρικής όπως η APFDc. Σε άλλες περιπτώσεις, το κόστος θα μπορούσε να αναφέρεται σε κάτι πιο περίπλοκο, όπως τον απαιτούμενο χρόνο εργασίας, άρα και το εργατικό κόστος, των προγραμματιστών για την επιδιόρθωση των σφαλμάτων που εντοπίζονται.

Εναλλακτικά και επιπρόσθετα στη χρήση της μετρικής APFD, θα μπορούσαν να χρησιμοποιηθούν περισσότερες μετρικές, με σκοπό την επιβεβαίωση της καταλληλότητας της μετρικής APFD για τον επιδιωκόμενο σκοπό. Όπως αναφέρθηκε, ένα παράδειγμα μιας άλλης μετρικής θα μπορούσε να είναι ο χρόνος εκτέλεσης των περιπτώσεων ελέγχου, ή ο αριθμός των περιπτώσεων ελέγχου που έχουν εκτελεστεί, μέχρι να ανιχνευθούν όλα τα σφάλματα που υπάρχουν.

Η χρήση της πληροφορίας αλλαγών στο πηγαίο κώδικα των έργων κατά την ανάπτυξή τους θα μπορούσε να δώσει ένα παραπάνω στοιχείο που θα βοηθούσε περαιτέρω την αποδοτικότερη προτεραιοποίηση των περιπτώσεων ελέγχου.

Τέλος, ενδιαφέρουσα θα ήταν η επέκταση της μελέτης σε άλλες δημοφιλείς γλώσσες προγραμματισμού, όπως η Javascript ή η PHP, οι οποίες χρησιμοποιούνται επίσης ευρύτατα σε μεγάλα έργα λογισμικού και για τις οποίες δεν έχουν πραγματοποιηθεί αν-

τίστοιχες μελέτες.

Βιβλιογραφικές Αναφορές

[1] A. Jones, James, Harrold, Mary & Computer Society, Ieee. "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage". *IEEE Transactions on Software Engineering* 29 (June 2003). DOI: 10.1109/TSE.2003.1183927.

[2] Ali, S., Briand, L. C., Hemmati, H. & Panesar-Walawege, R. K. "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation". *IEEE Transactions on Software Engineering* 36.6 (Nov. 2010), pp. 742–762. ISSN: 0098-5589. DOI: 10.1109/TSE.2009.52.

[3] Beizer, Boris. *Software Testing Techniques*. Van Nostrand Reinhold, 1983.

[4] Binkley, David. "Semantics Guided Regression Test Cost Reduction". *IEEE Trans. Softw. Eng.* 23.8 (Aug. 1997), pp. 498–516. ISSN: 0098-5589. DOI: 10.1109/32.624306.

[5] Chambers, J. M., Freeny, A & Heiberger, R. M. *Analysis of variance; designed experiments. Chapter 5 of Statistical Models in S*. Ed. by J.M., Chambers & T.J., Hastie. Wadsworth and Brooks/Cole, 1992.

[6] Chen, Tsong Yueh & Lau, Man Fai. "Dividing strategies for the optimization of a test suite". *Information Processing Letters* 60.3 (1996), pp. 135–141. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(96)00135-4.

[7] Chen, Yanping, L. Probert, Robert & Ural, Hasan. "Regression test suite reduction based on SDL models of system requirements". *Journal of Software Maintenance* 21 (Nov. 2009), pp. 379–405. DOI: 10.1002/smr.415.

[8] Chen, Yih-Farn, Rosenblum, David & Vo, Kiem-phong. "TESTTUBE: a system for selective regression testing". June 1994, pp. 211–220. ISBN: 0-8186-5855-X. DOI: 10.1109/ICSE.1994.296780.

[9] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. & Stein, Clifford. *Introduction to Algorithms, 3rd edition*. MIT Press, 2009.

[10] Do, Hyunsook, Rothermel, Gregg & Kinneer, Alex. "Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis". *Empirical Software Engineering* 11.1 (Mar. 2006), pp. 33–70. ISSN: 1573-7616. DOI: 10.1007/s10664-006-5965-8.

[11] Elbaum, S., Malishevsky, A. & Rothermel, G. “Incorporating varying test costs and fault severities into test case prioritization”. *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. May 2001, pp. 329–338. DOI: 10.1109/ICSE.2001.919106.

[12] Elbaum, S., Malishevsky, A.G. & Rothermel, G. “Test Case Prioritization: A Family of Empirical Studies”. *IEEE Transactions on Software Engineering* 28 (2002), pp. 159–182. ISSN: 0098-5589. DOI: doi.ieeecomputersociety.org/10.1109/32.988497.

[13] Elbaum, Sebastian, Malishevsky, Alexey G. & Rothermel, Gregg. “Prioritizing Test Cases for Regression Testing”. *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA ’00. Portland, Oregon, USA: ACM, 2000, pp. 102–112. ISBN: 1-58113-266-2. DOI: 10.1145/347324.348910.

[14] Elbaum, Sebastian, Rothermel, Gregg, Kanduri, Satya & Malishevsky, Alexey G. “Selecting a Cost-Effective Test Case Prioritization Technique”. *Software Quality Journal* 12.3 (Sept. 2004), pp. 185–210. ISSN: 1573-1367. DOI: 10.1023/B:SQJO.0000034708.84524.22.

[15] Elbaum, Sebastian, Rothermel, Gregg & Penix, John. “Techniques for Improving Regression Testing in Continuous Integration Development Environments”. *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: ACM, 2014, pp. 235–245. ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635910.

[16] Fischer, K.F., Raji, F. & A., Chruscicki. “A methodology for retesting modified software”. *Proceedings of the National Telecommunications Conference*. Vol. B-6-3j. Oct. 1981, pp. 1–6.

[17] Fox, John. *Applied Regression Analysis and Generalized Linear Models*. 3rd. SAGE Publications, Inc, 2015.

[18] Graves, L. Todd, Harrold, Mary, Kim, J, Porter, Adam & Rothermel, Gregg. “An Empirical Study of Regression Test Selection Techniques.” May 1998, pp. 188–197. DOI: 10.1109/ICSE.1998.671115.

[19] Gupta, R., Harrold, M. J. & Soffa, M. L. “An approach to regression testing using slicing”. *Proceedings Conference on Software Maintenance 1992.* Nov. 1992, pp. 299–308. DOI: 10.1109/ICSM.1992.242531.

[20] Harrold, Mary, Gupta, Rajiv & Soffa, Mary. “A Methodology for Controlling the Size of a Test Suite.” *ACM Transactions on Software Engineering and Methodology* 2 (July 1993), pp. 270–285. DOI: 10.1145/152388.152391.

[21] Hemmati, Hadi. “Advances in Techniques for Test Prioritization”. Ed. by Memon, Atif M. Vol. 112. Advances in Computers. Elsevier, 2019, pp. 185–221. DOI: https://doi.org/10.1016/bs.adcom.2017.12.004.

[22] Hemmati, Hadi, Arcuri, Andrea & Briand, Lionel. “Achieving Scalable Model-based Testing Through Test Case Diversity”. *ACM Trans. Softw. Eng. Methodol.* 22.1 (Mar. 2013), 6:1–6:42. ISSN: 1049-331X. DOI: 10.1145/2430536.2430540.

[23] Hemmati, Hadi, Fang, Zhihan, Mäntylä, Mika V. & Adams, Bram. “Prioritizing manual test cases in rapid release environments”. *Software Testing, Verification and Reliability* 27.6 (2017). e1609 stvr.1609, e1609. DOI: 10.1002/stvr.1609.

[24] Hettiarachchi, Charitha, Do, Hyunsook & Choi, Byoungju. “Risk-based test case prioritization using a fuzzy expert system”. *Information and Software Technology* 69 (2016), pp. 1–15. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2015.08.008.

[25] Kazmi, Rafaqut, Jawawi, Dayang N. A., Mohamad, Radziah & Ghani, Imran. “Effective Regression Test Case Selection: A Systematic Literature Review”. *ACM Comput. Surv.* 50.2 (May 2017), 29:1–29:32. ISSN: 0360-0300. DOI: 10.1145/3057269.

[26] Leon, D. & Podgurski, A. “A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases”. *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* Nov. 2003, pp. 442–453. DOI: 10.1109/ISSRE.2003.1251065.

[27] Leung, H. K. N. & White, L. “Insights into regression testing (software testing)”. *Proceedings. Conference on Software Maintenance - 1989.* Oct. 1989, pp. 60–69. DOI: 10.1109/ICSM.1989.65194.

[28] Leung, Hareton & White, LJ. “A study of integration testing and regression testing at the integration level”. *Proceedings of the Conference on Software Maintenance*. Dec. 1990, pp. 290–301. ISBN: 0-8186-2091-9. DOI: 10.1109/ICSM.1990.131377.

[29] Mathur, Aditya P. *Foundations of Software Testing*. Addison-Wesley Professional, 2008.

[30] Mei, H., Hao, D., Zhang, L., Zhang, L., Zhou, J. & Rothermel, G. “A Static Approach to Prioritizing JUnit Test Cases”. *IEEE Transactions on Software Engineering* 38.6 (Nov. 2012), pp. 1258–1275. ISSN: 0098-5589. DOI: 10.1109/TSE.2011.106.

[31] Memon, Atif, Gao, Zebao, Nguyen, Bao, Dhanda, Sanjeev, Nickell, Eric, Siemborski, Rob & Micco, John. “Taming Google-scale Continuous Testing”. *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*. ICSE-SEIP ’17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 233–242. ISBN: 978-1-5386-2717-4. DOI: 10.1109/ICSE-SEIP.2017.16.

[32] Miller, Joan C. & Maloney, Clifford J. “Systematic Mistake Analysis of Digital Computer Programs”. *Communications of the ACM* 6.2 (Feb. 1963), pp. 58–63. ISSN: 0001-0782. DOI: 10.1145/366246.366248.

[33] Offutt, A. Jefferson, Pan, Jie & Voas, Jeffrey M. “Procedures for reducing the size of coverage-based test sets”. *Proceedings of the Twelfth International Conference on Testing Computer Software*. 1995, pp. 111–123.

[34] Rothermel, G., Untch, R. H., Chu, Chengyun & Harrold, M. J. “Prioritizing Test Cases For Regression Testing”. *IEEE Transactions on Software Engineering* 27.10 (Oct. 2001), pp. 929–948. ISSN: 0098-5589. DOI: 10.1109/32.962562.

[35] Rothermel, Gregg, Elbaum, Sebastian, Malishevsky, Alexey & Kallakuri, Praveen. “On test suite composition and cost-effective regression testing”. *ACM Transactions on Software Engineering and Methodology* 13 (2004), pp. 277–331.

[36] Rothermel, Gregg & Harrold, Mary. “A Safe, Efficient Regression Test Selection Technique”. *ACM Transactions on Software Engineering and Methodology* 6 (Feb. 2000). DOI: 10.1145/248233.248262.

[37] Rothermel, Gregg & Harrold, Mary. “Analyzing regression test selection techniques”. *EEE Transactions on Software Engineering* 22.8 (Aug. 1996), pp. 529–551.

[38] Rothermel, Gregg & Harrold, Mary. "Empirical Studies of a Safe Regression Test Selection Technique". *IEEE Transactions on Software Engineering* 24.6 (June 1998), pp. 401–419.

[39] Rothermel, Gregg, Harrold, Mary Jean & Dedhia, Jeinay. "Regression Test Selection for C++ Software". *Software Testing, Verification & Reliability* 10 (1999), p. 2000.

[40] Rothermel, Gregg, Harrold, Mary, Ostrin, Jeffery & Hong, Christie. "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites". *Proceedings of the International Conference on Software Maintenance*. July 2000, pp. 34–43.

[41] Royston, J. P. "An Extension of Shapiro and Wilk's W Test for Normality to Large Samples". *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 31.2 (1982), pp. 115–124. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2347973.

[42] Srivastava, Amitabh & Thiagarajan, Jay. *Effectively Prioritizing Tests in Development Environment*. Tech. rep. Feb. 2002, p. 11. URL: https://www.microsoft.com/en-us/research/publication/effectively-prioritizing-tests-in-development-environment/.

[43] Thomas, Stephen W., Hemmati, Hadi, Hassan, Ahmed E. & Blostein, Dorothea. "Static test case prioritization using topic models". *Empirical Software Engineering* 19.1 (Feb. 2014), pp. 182–212. ISSN: 1573-7616. DOI: 10.1007/s10664-012-9219-7.

[44] Vokolos, Filippos & Frankl, P.G. "Pythia: A regression test selection tool based on textual differencing". *Proceedings of the 3rd International Conference on Rel., Quality & Safety of Software-Intensive Sys. (ENCRESS '97)*. Aug. 2000. DOI: 10.1007/978-0-387-35097-4_1.

[45] Wickham, Hadley. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org.

[46] Wolpert, D. H. & Macready, W. G. "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1089-778X. DOI: 10.1109/4235.585893.

[47] Wong, W. Eric, Horgan, Joseph R., London, Saul & Mathur, Aditya P. “Effect of test set minimization on fault detection effectiveness”. *Software: Practice and Experience* 28.4 (1998), pp. 347–369. DOI: 10.1002/(SICI)1097-024X(19980410)28:4<347::AID-SPE145>3.0.CO;2-L.

[48] Wong, W. Eric, Horgan, Joseph Robert, London, Saul & Agrawal, Hiralal. “A Study Of Effective Regression Testing In Practice”. *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering*. 1997.

[49] Yandell, B.S. *Practical Data Analysis for Designed Experiments*. Chapman and Hall, 1997.

[50] Yoo, S. & Harman, M. “Regression testing minimization, selection and prioritization: a survey”. *Software Testing, Verification and Reliability* 22.2 (2012), pp. 67–120. DOI: 10.1002/stvr.430.

Διαδικτυακές Αναφορές

[51] *Apache Ant*. URL: https://ant.apache.org/ (ημερομηνία προσπέλασης: 25/5/2019).

[52] *Apache JMeter*. URL: https://jmeter.apache.org/ (ημερομηνία προσπέλασης: 25/5/2019).

[53] *django: The web framework for perfectionists with deadlines*. URL: https://www.djangoproject.com/ (ημερομηνία προσπέλασης: 22/2/2019).

[54] *docopt: Command-line interface description language*. URL: http://docopt.org/ (ημερομηνία προσπέλασης: 2/6/2019).

[55] *Dropbox: Incrementally migrating over one million lines of code from Python 2 to Python 3*. Feb. 2019. URL: https://blogs.dropbox.com/tech/2019/02/incrementally-migrating-over-one-million-lines-of-code-from-python-2-to-python-3/ (ημερομηνία προσπέλασης: 12/6/2019).

[56] *Facebook Code: Python in production engineering*. May 2016. URL: https://code.fb.com/production-engineering/python-in-production-engineering/ (ημερομηνία προσπέλασης: 12/6/2019).

[57] *Flask: The Python micro framework for building web applications.* URL: https://github.com/pallets/flask (ημερομηνία προσπέλασης: 19/2/2019).

[58] *Global Interpreter Lock.* URL: https://wiki.python.org/moin/GlobalInterpreterLock (ημερομηνία προσπέλασης: 25/5/2019).

[59] *IronPython: The Python Programming Language for the .NET Framework.* URL: https://ironpython.net/ (ημερομηνία προσπέλασης: 9/6/2019).

[60] *MicroPython: Python for Microcontrollers.* URL: https://micropython.org/ (ημερομηνία προσπέλασης: 9/6/2019).

[61] *Nose: Is Nicer Testing for Python.* URL: https://nose.readthedocs.io (ημερομηνία προσπέλασης: 10/6/2019).

[62] *NumPy.* URL: https://www.numpy.org/ (ημερομηνία προσπέλασης: 26/5/2019).

[63] *PyPi: The Python Package Index.* URL: https://pypi.org/ (ημερομηνία προσπέλασης: 9/6/2019).

[64] *PyPy: a fast, compliant alternative implementation of the Python language.* URL: https://pypy.org/ (ημερομηνία προσπέλασης: 1/2/2019).

[65] *pytest: helps you write better programs.* URL: https://docs.pytest.org/en/latest/ (ημερομηνία προσπέλασης: 19/2/2019).

[66] *pytest-cov.* URL: https://pypi.org/project/pytest-cov/ (ημερομηνία προσπέλασης: 25/5/2019).

[67] *Python.* URL: https://www.python.org/ (ημερομηνία προσπέλασης: 25/5/2019).

[68] *Python Built-in Exceptions.* URL: https://docs.python.org/3/library/exceptions.html#TypeError (ημερομηνία προσπέλασης: 25/5/2019).

[69] *Python "Off by one" error.* URL: https://pythondebugging.com/articles/python-off-by-one-error-range (ημερομηνία προσπέλασης: 25/5/2019).

[70] R Core Team. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing, 2016. URL: https://www.R-project.org/.

[71] *Six: Python 2 and 3 compatibility library.* URL: https://github.com/benjaminp/six (ημερομηνία προσπέλασης: 19/2/2019).

[72] *Stack Overflow Developer Survey 2019*. URL: https://insights.stackoverflow.com/survey/2019 (ημερομηνία προσπέλασης: 9/6/2019).

[73] *The Jython Project*. URL: https://jython.org/ (ημερομηνία προσπέλασης: 9/6/2019).

[74] *The MoinMoin Wiki Engine*. URL: http://moinmo.in/ (ημερομηνία προσπέλασης: 19/2/2019).

[75] *TIOBE Index for June 2019. June Headline: Python continues to soar in the TIOBE index*. URL: https://www.tiobe.com/tiobe-index/ (ημερομηνία προσπέλασης: 9/6/2019).

[76] *Wikipedia: diff*. URL: https://en.wikipedia.org/wiki/Diff (ημερομηνία προσπέλασης: 25/5/2019).

[77] *Wikipedia: Django*. URL: https://en.wikipedia.org/wiki/Django_(web_framework) (ημερομηνία προσπέλασης: 26/5/2019).

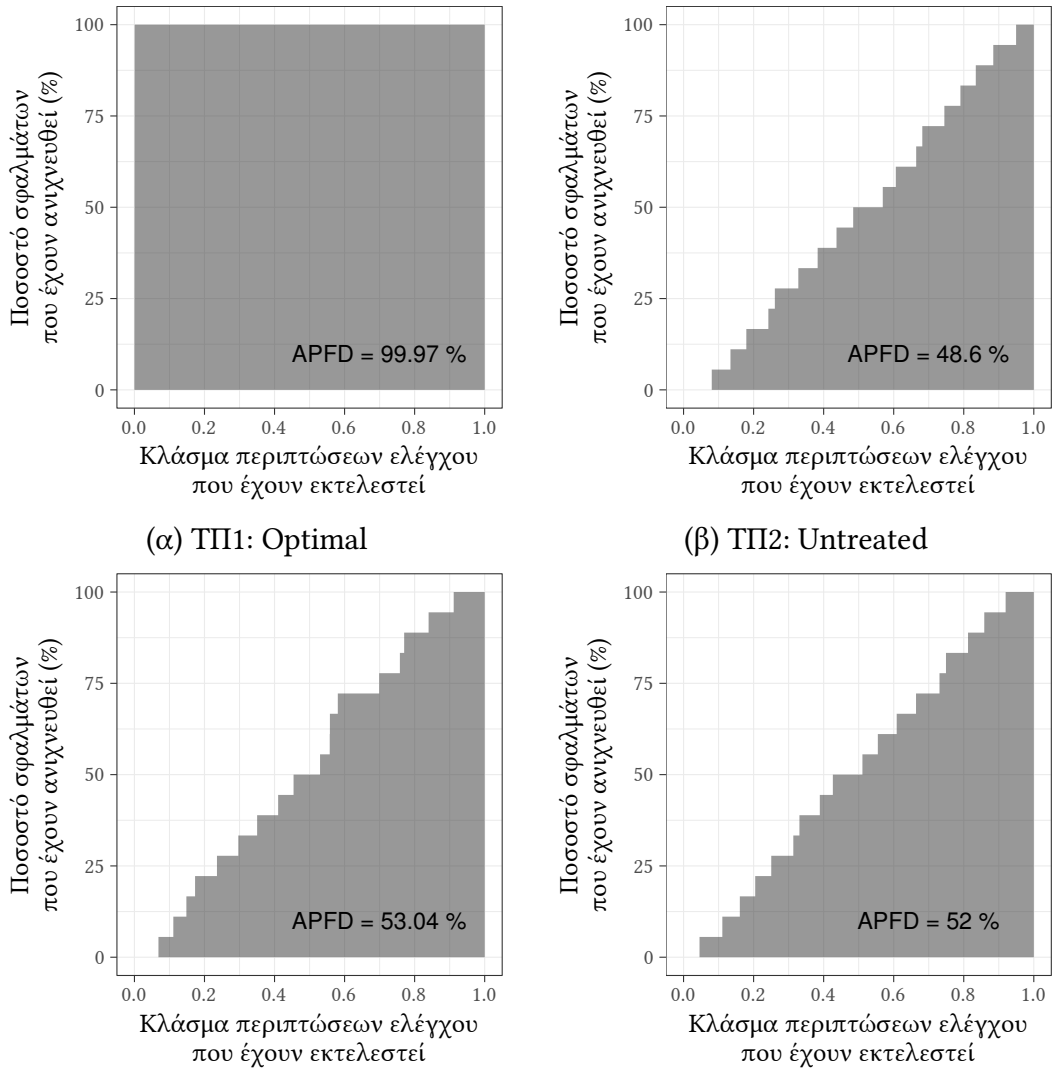
[78] *Wikipedia: Flask*. URL: https://en.wikipedia.org/wiki/Flask_%28web_framework%29 (ημερομηνία προσπέλασης: 26/5/2019).

[79] *Wikipedia: MoinMoin*. URL: https://en.wikipedia.org/wiki/MoinMoin (ημερομηνία προσπέλασης: 26/5/2019).

[80] *Εμβαδό τραπεζίου*. URL: http://www.calcfun.com/calc-9-emvadon-trapeziou.html (ημερομηνία προσπέλασης: 13/2/2019).

Παράρτημα

Μετρήσεις APFD για το PyPy



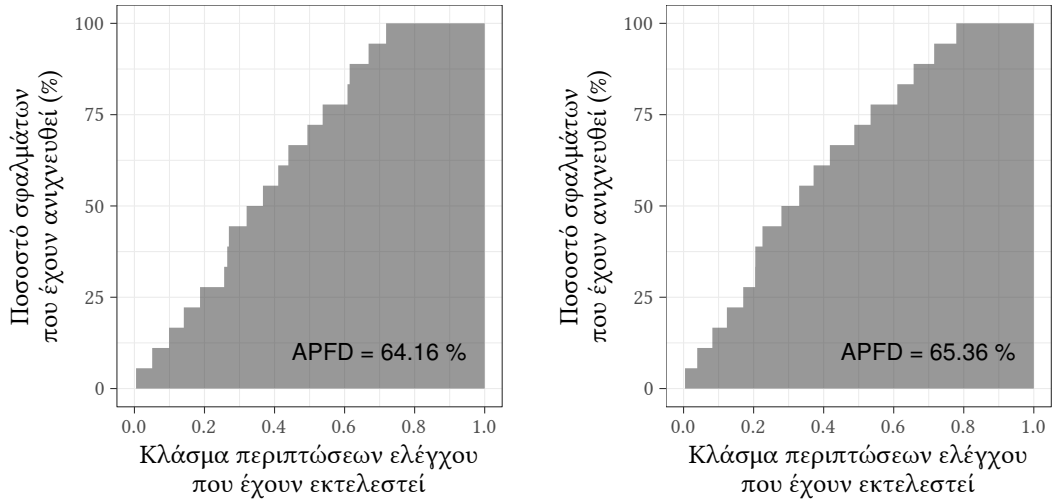
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

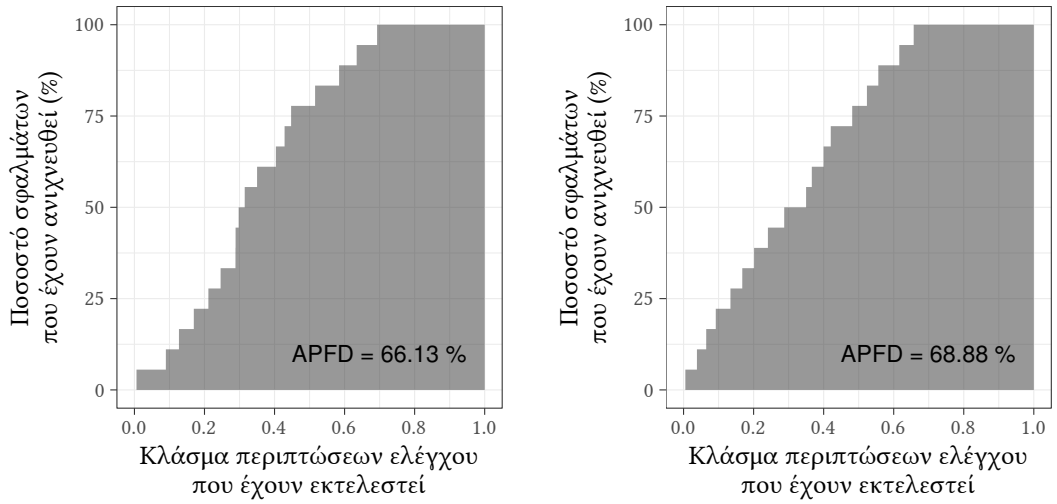
(δ) ΤΠ4: Inverse

Σχήμα 14: Υπολογισμοί APFD για το PyPy2 5.7.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

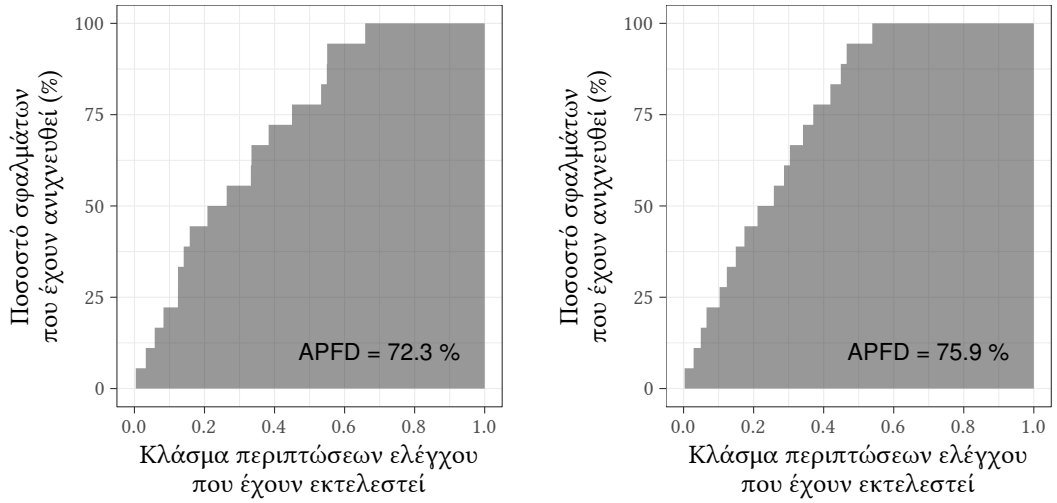
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

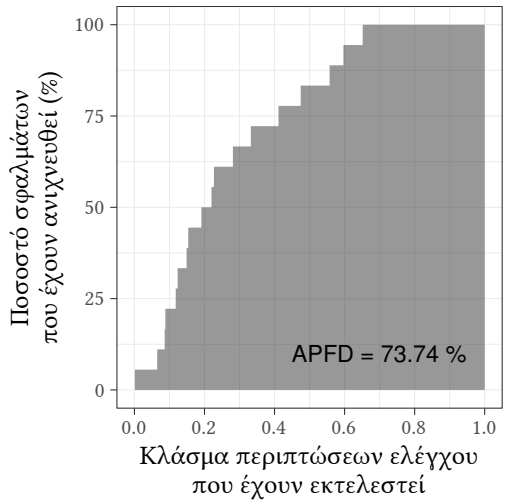
(δ) ΤΠ8: Method-addtl

Σχήμα 15: Υπολογισμοί APFD για το PyPy2 5.7.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)

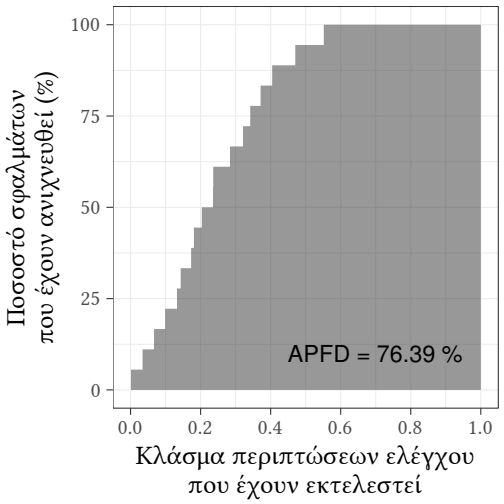


(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

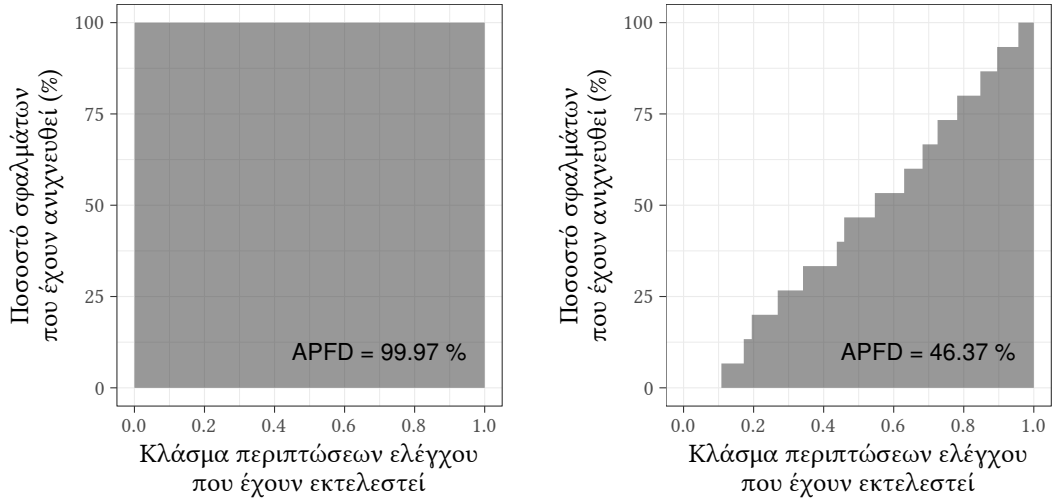


(γ) ΤΠ11: Statement-total



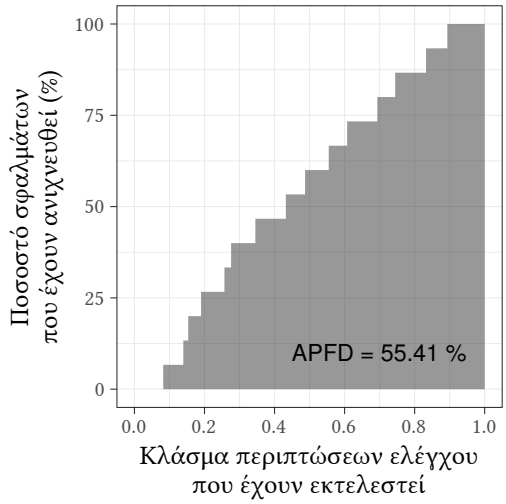
(δ) ΤΠ12: Statement-addtl

Σχήμα 16: Υπολογισμοί APFD για το PyPy2 5.7.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



(α) ΤΠ1: Optimal

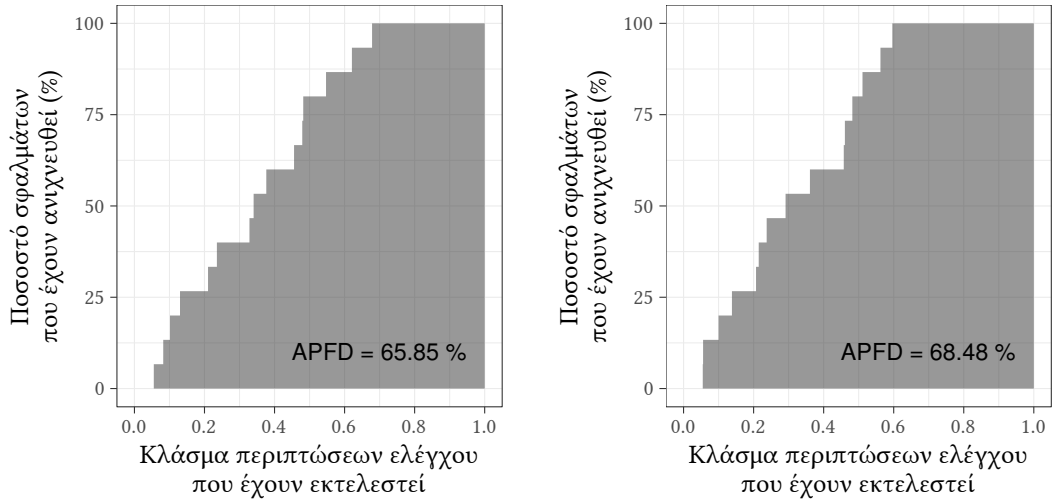
(β) ΤΠ2: Untreated



(γ) ΤΠ3: Random

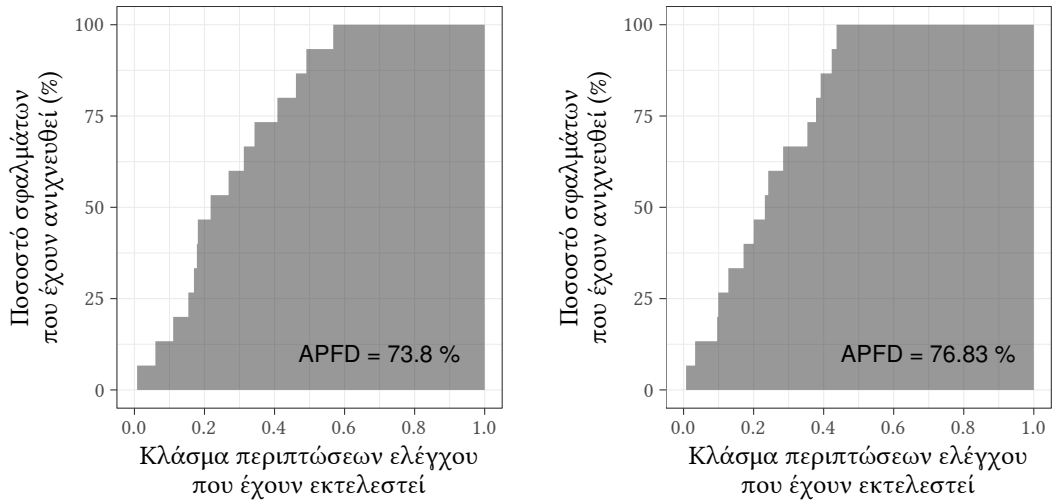
(δ) ΤΠ4: Inverse

Σχήμα 17: Υπολογισμοί APFD για το PyPy2 5.7.1 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

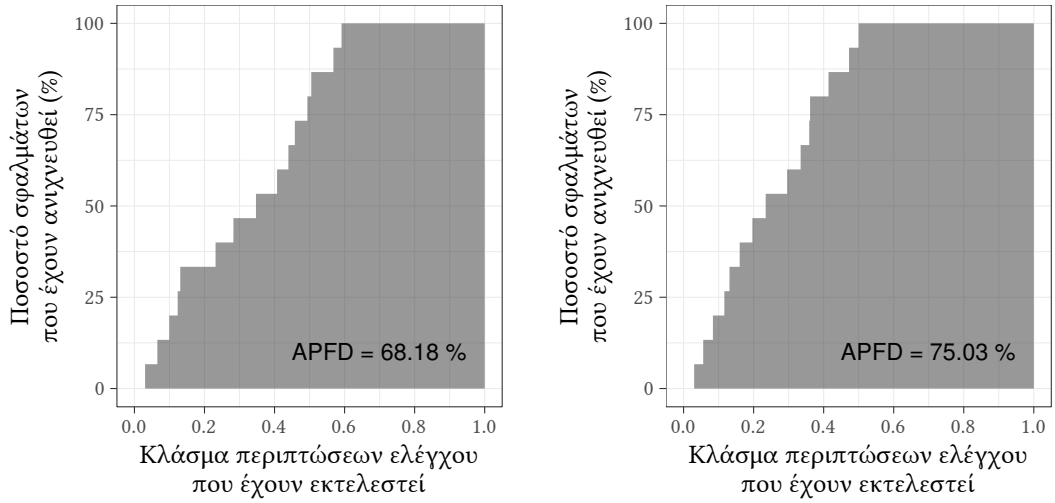
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

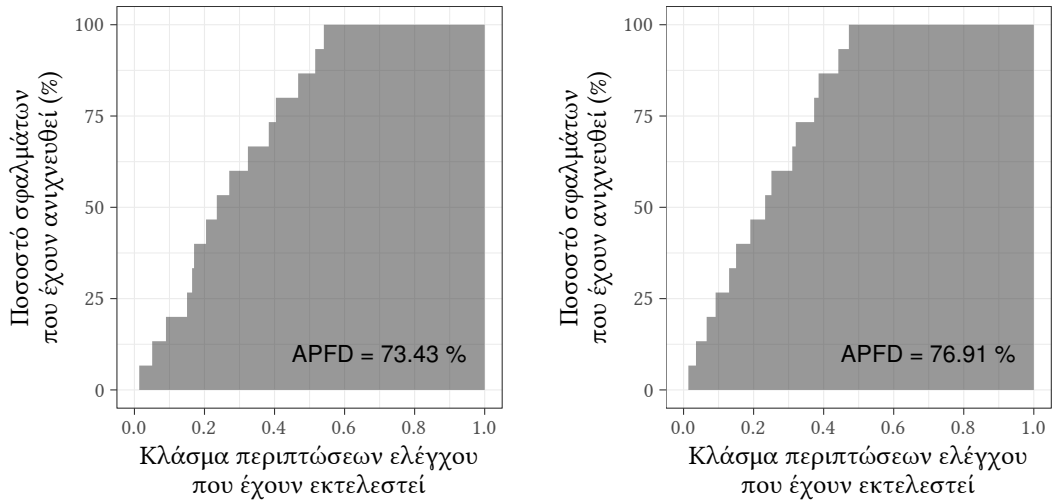
(δ) ΤΠ8: Method-addtl

Σχήμα 18: Υπολογισμοί APFD για το PyPy2 5.7.1 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

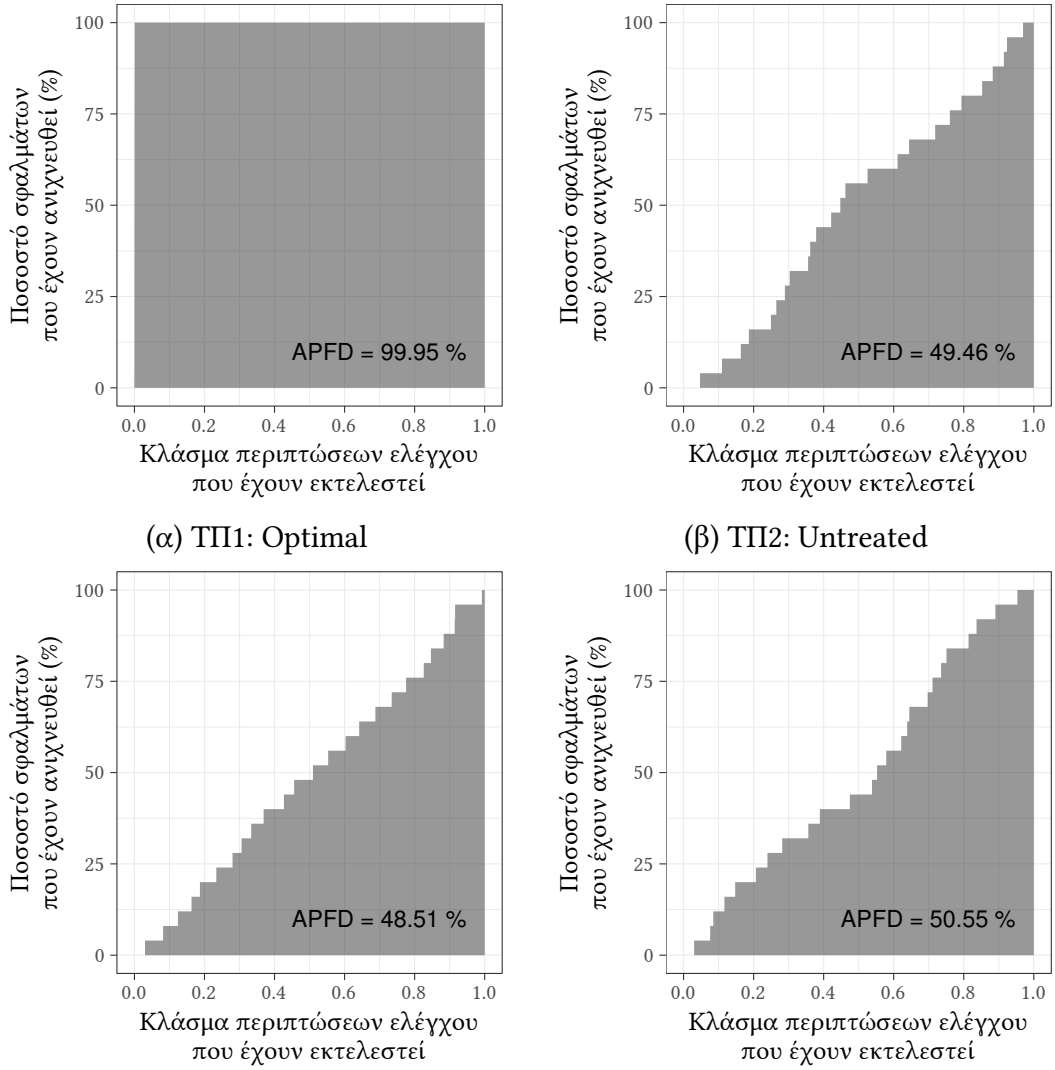
(β) ΤΠ10: Branch-addtl



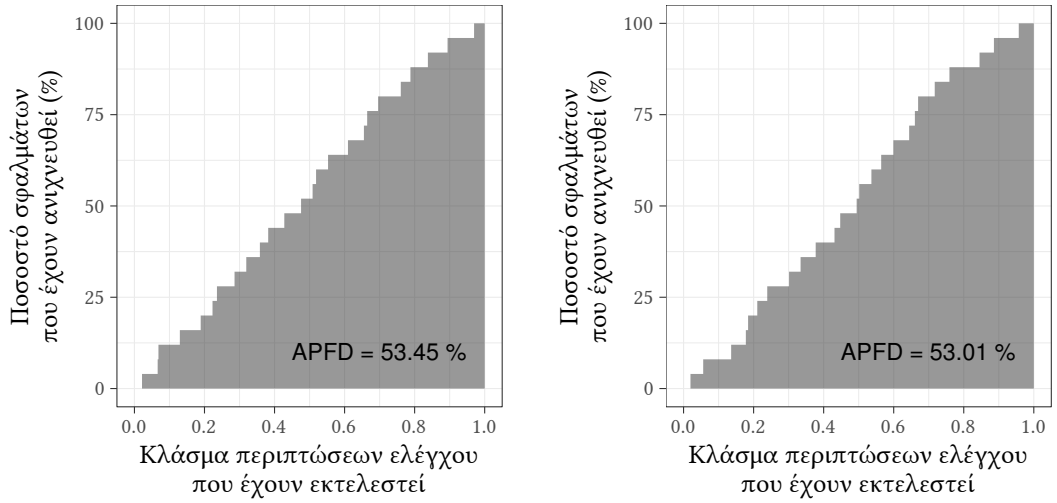
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 19: Υπολογισμοί APFD για το PyPy2 5.7.1 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



Σχήμα 20: Υπολογισμοί APFD για το PyPy2 5.8.0 (τεχνικές σύγκρισης)



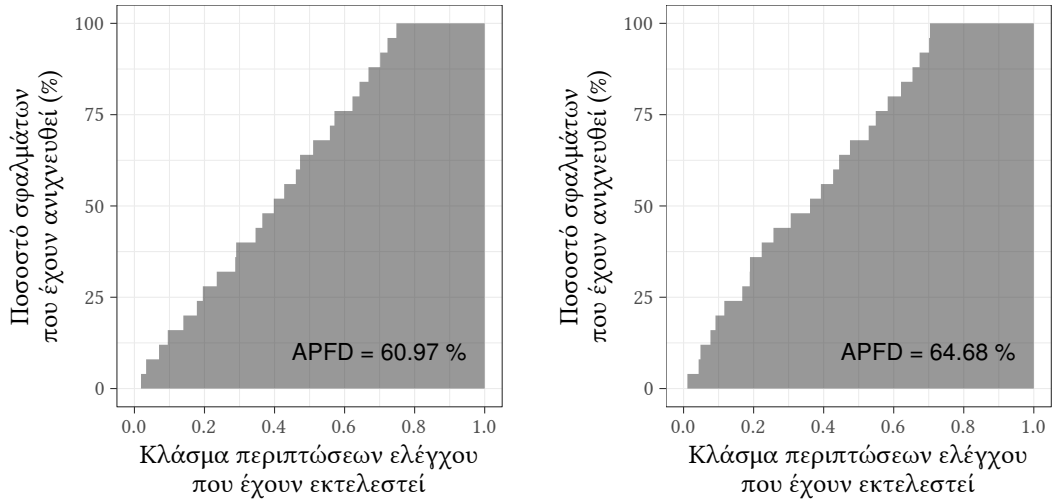
(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

(γ) ΤΠ7: Method-total

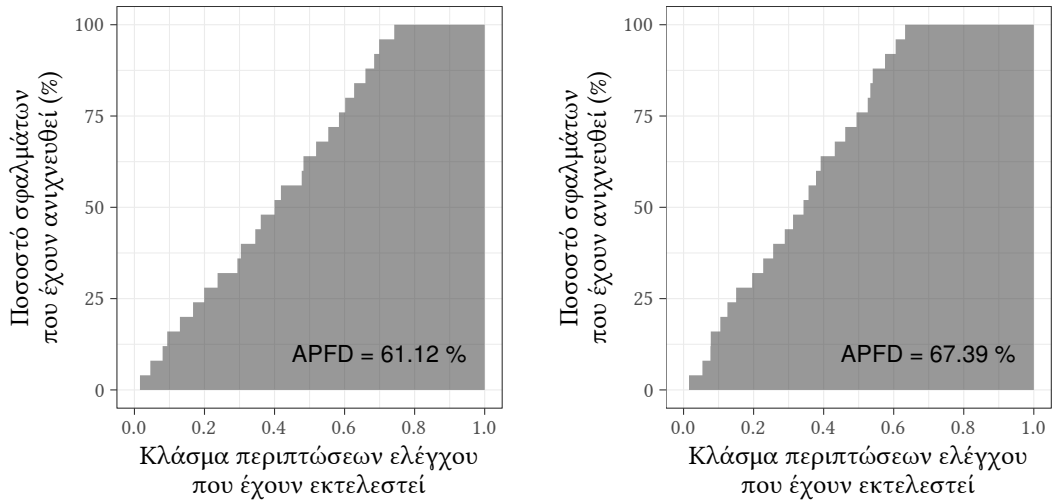
(δ) ΤΠ8: Method-addtl

Σχήμα 21: Υπολογισμοί APFD για το PyPy2 5.8.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

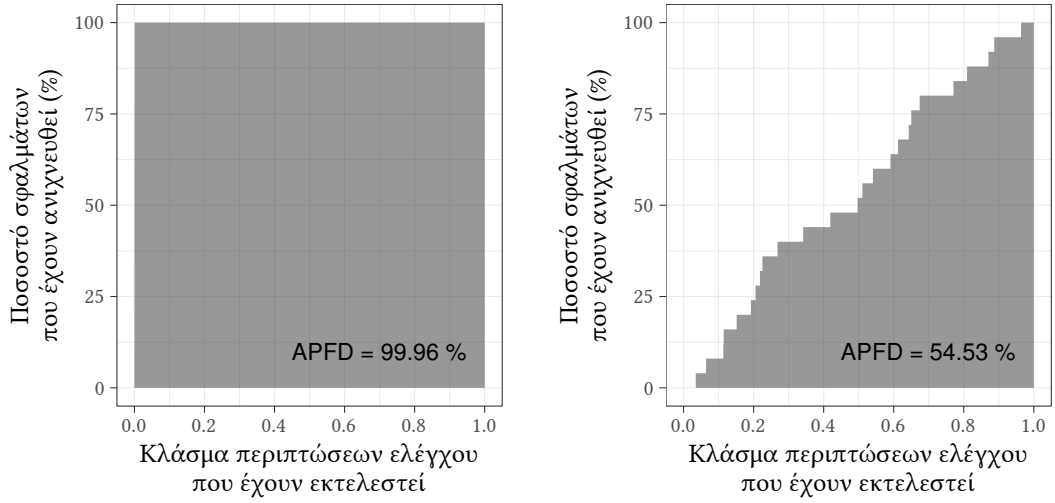
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

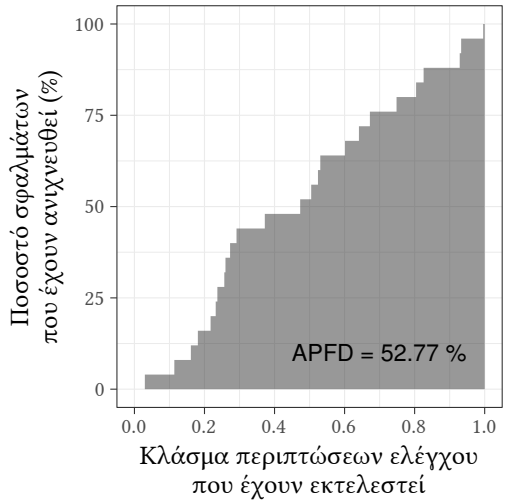
(δ) ΤΠ12: Statement-addtl

Σχήμα 22: Υπολογισμοί APFD για το PyPy2 5.8.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

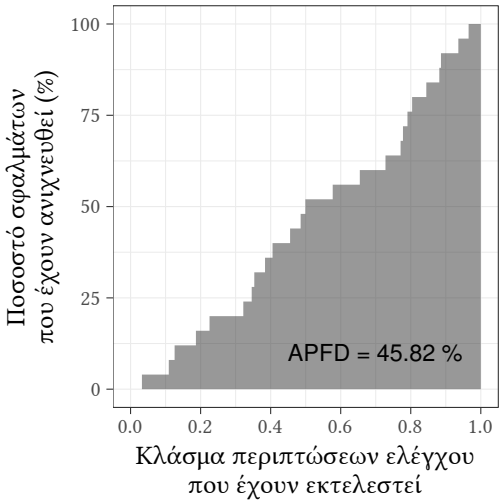


(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

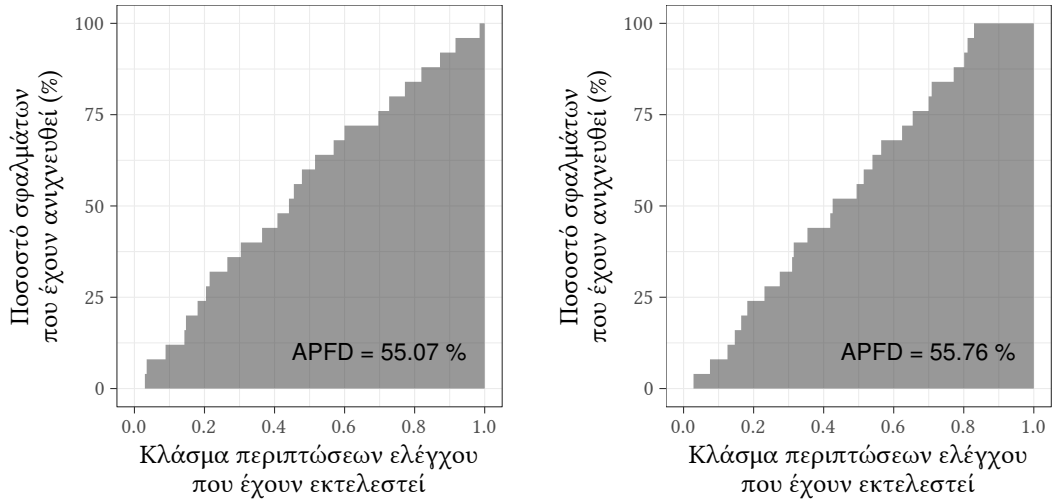


(γ) ΤΠ3: Random



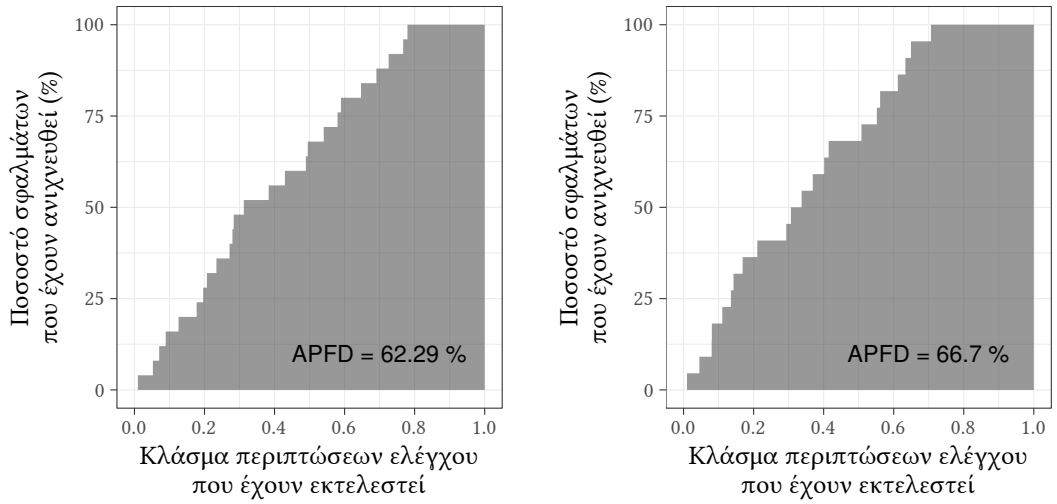
(δ) ΤΠ4: Inverse

Σχήμα 23: Υπολογισμοί APFD για το PyPy2 5.9.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

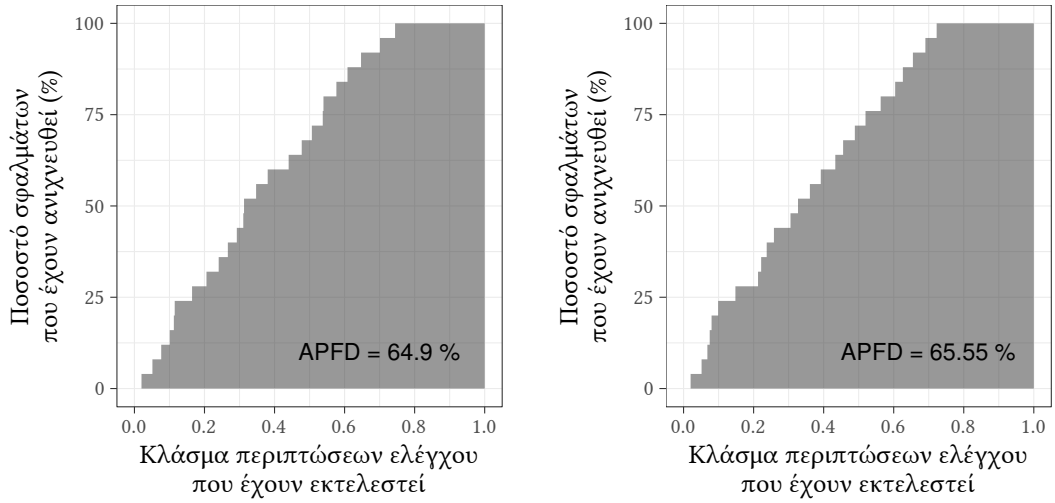
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

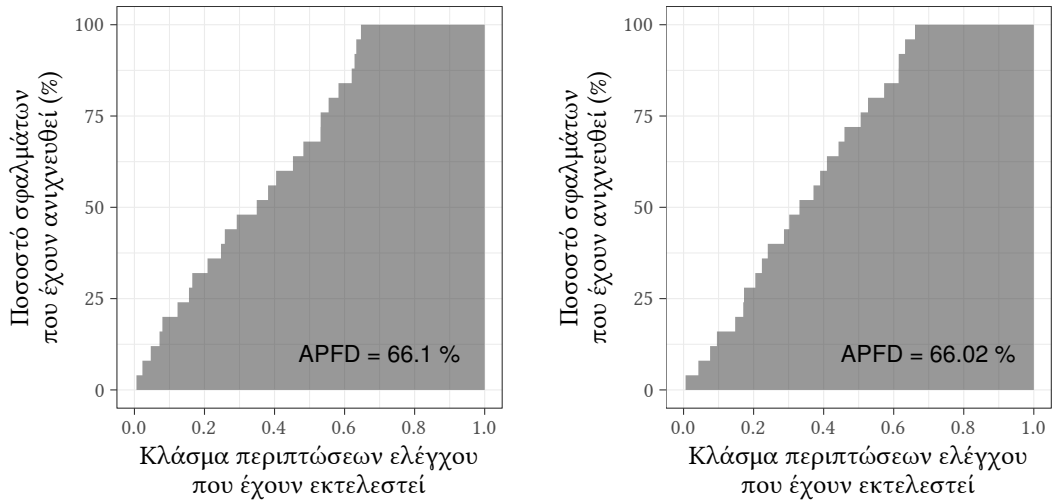
(δ) ΤΠ8: Method-addtl

Σχήμα 24: Υπολογισμοί APFD για το PyPy2 5.9.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

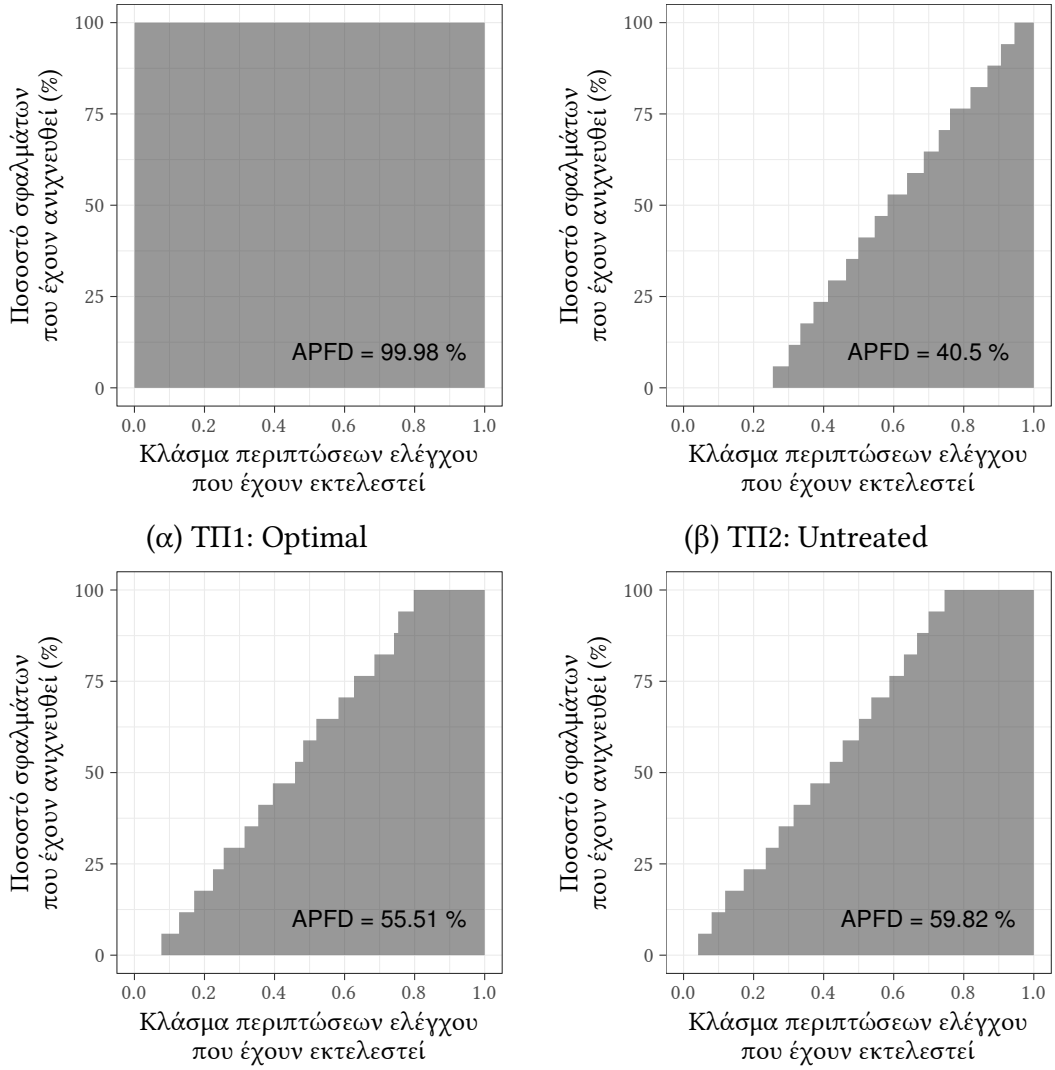
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 25: Υπολογισμοί APFD για το PyPy2 5.9.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



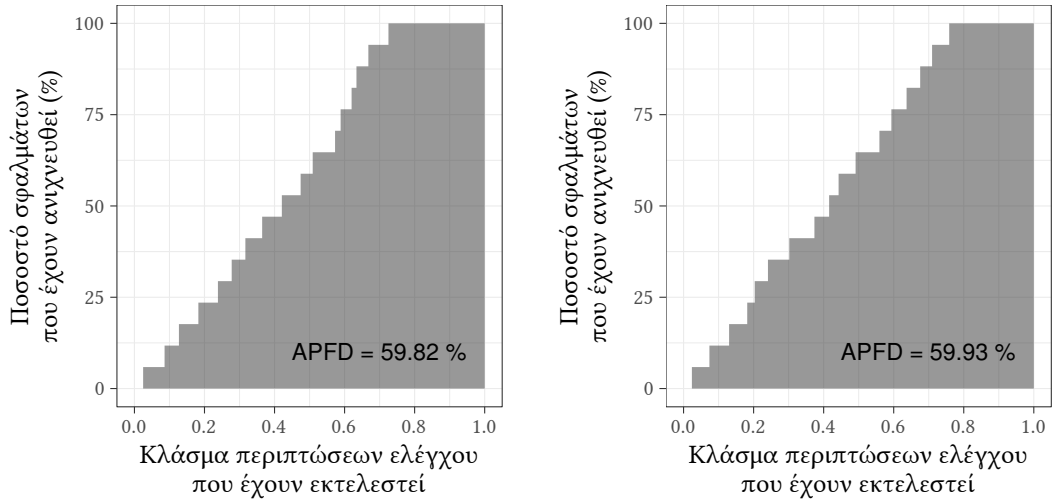
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

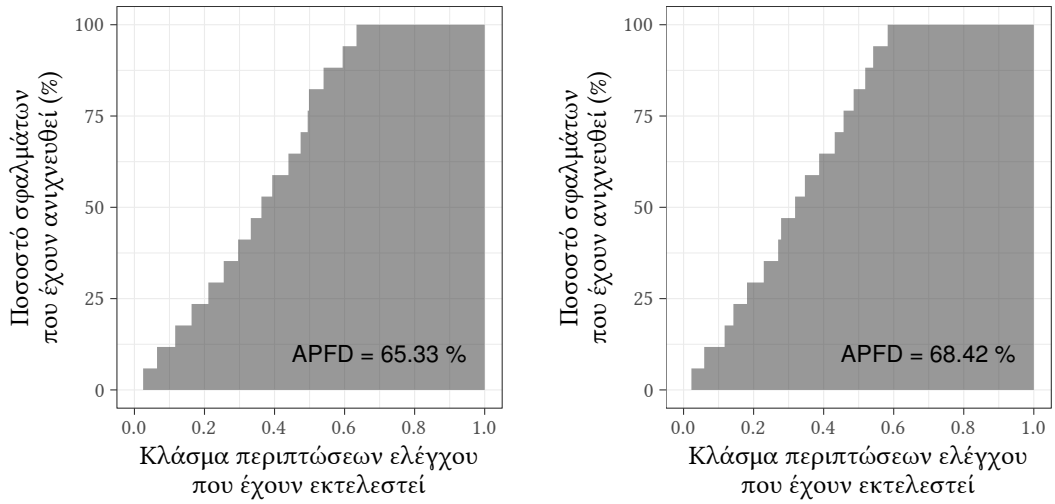
(δ) ΤΠ4: Inverse

Σχήμα 26: Υπολογισμοί APFD για το PyPy3 5.10.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

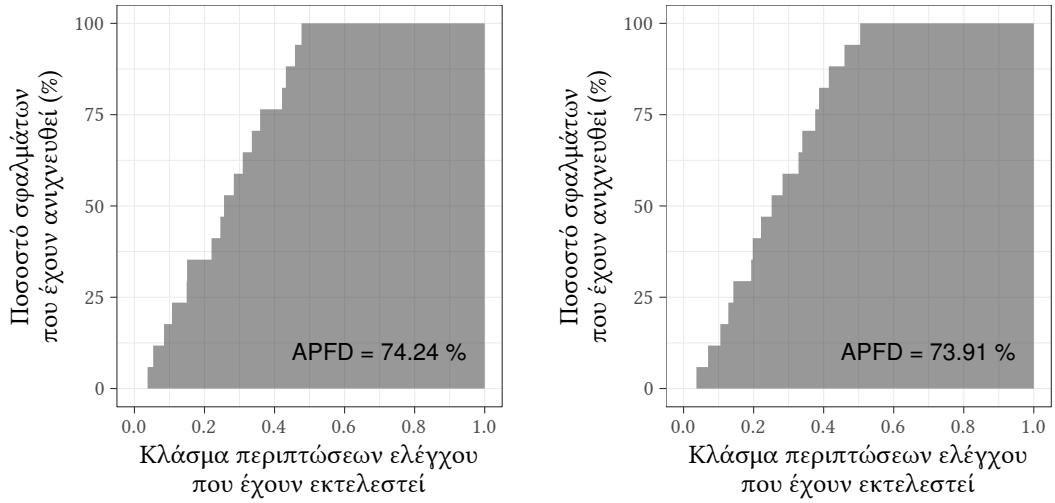
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

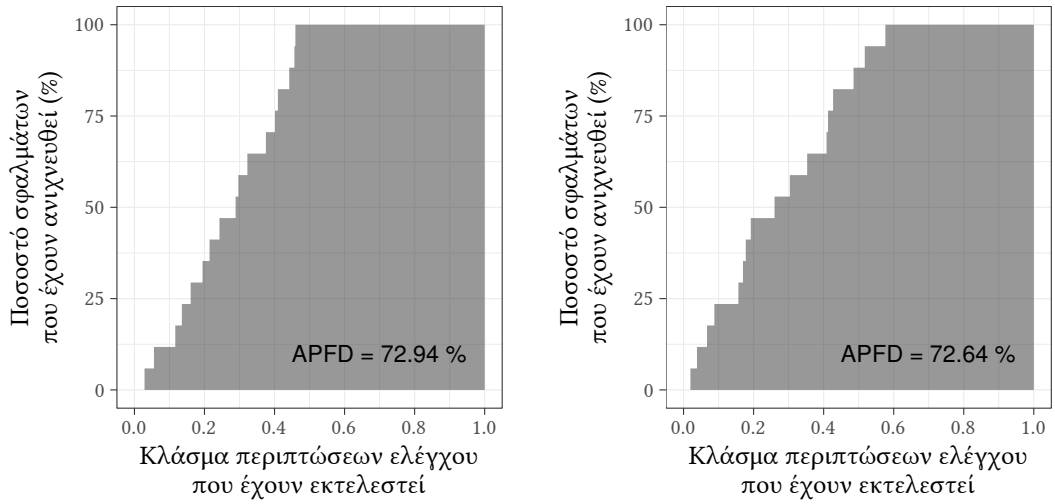
(δ) ΤΠ8: Method-addtl

Σχήμα 27: Υπολογισμοί APFD για το PyPy3 5.10.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

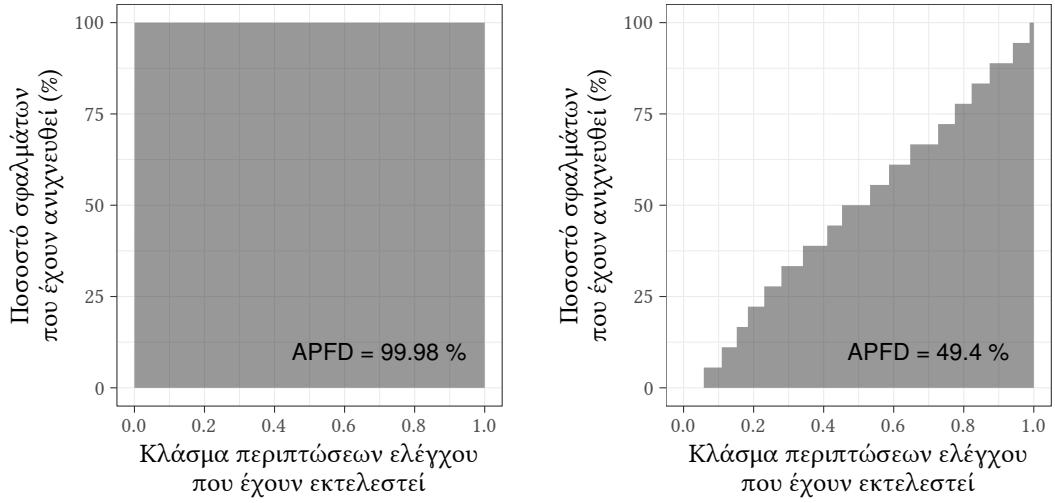
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

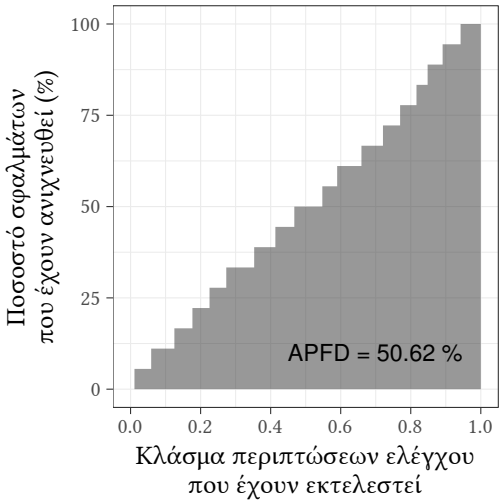
Σχήμα 28: Υπολογισμοί APFD για το PyPy3 5.10.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



(α) ΤΠ1: Optimal

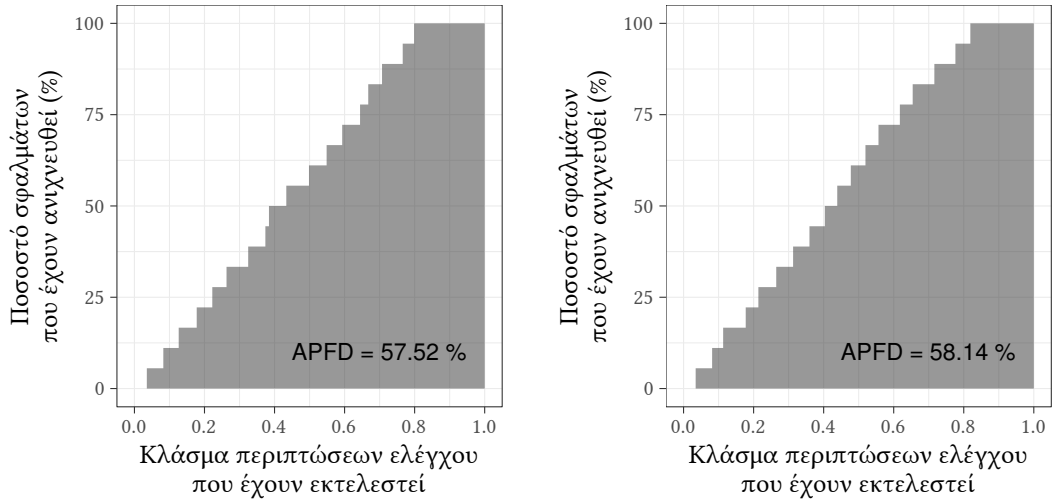
(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random



(δ) ΤΠ4: Inverse

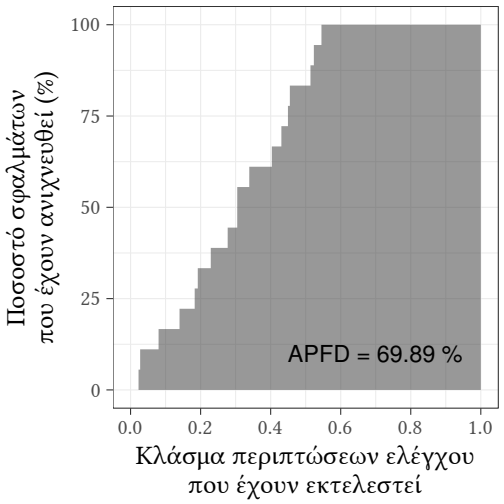
Σχήμα 29: Υπολογισμοί APFD για το PyPy3 5.10.1 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

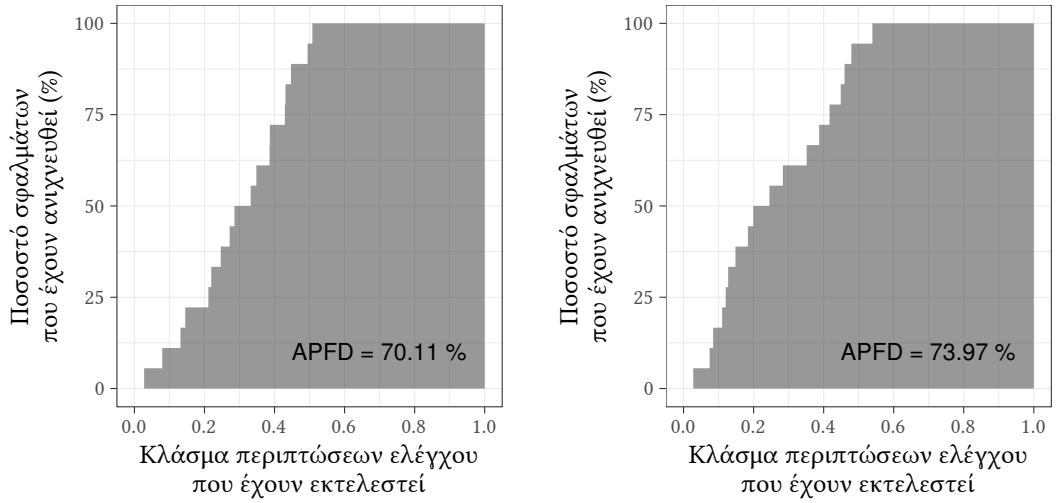
(β) ΤΠ6: File-addtl

(γ) ΤΠ7: Method-total



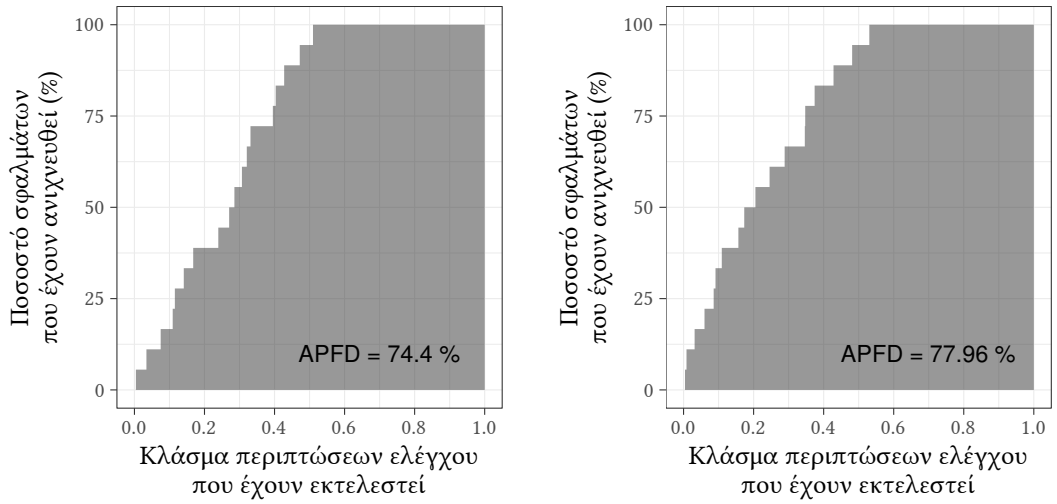
(δ) ΤΠ8: Method-addtl

Σχήμα 30: Υπολογισμοί APFD για το PyPy3 5.10.1 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

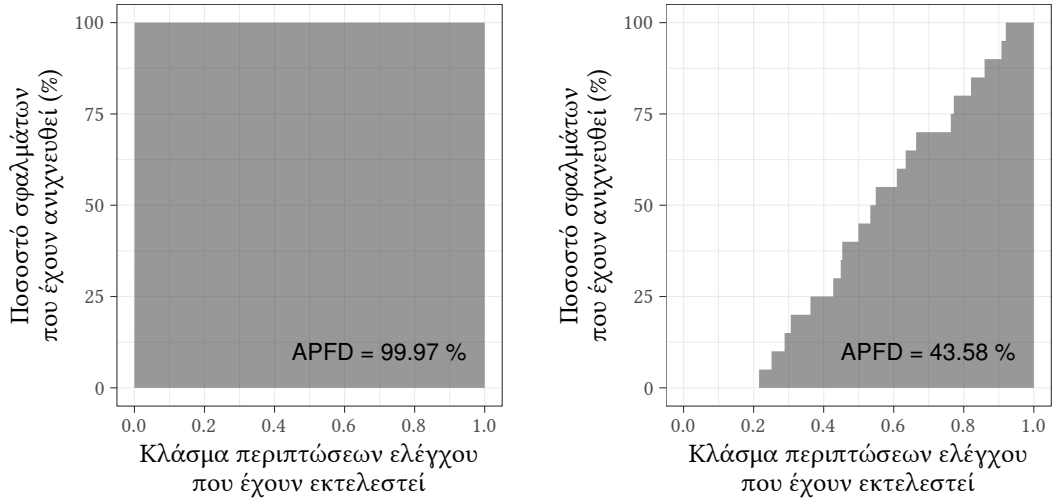
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

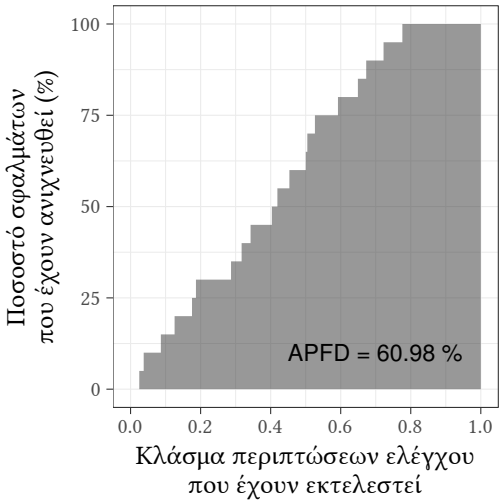
Σχήμα 31: Υπολογισμοί APFD για το PyPy3 5.10.1 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



(α) ΤΠ1: Optimal

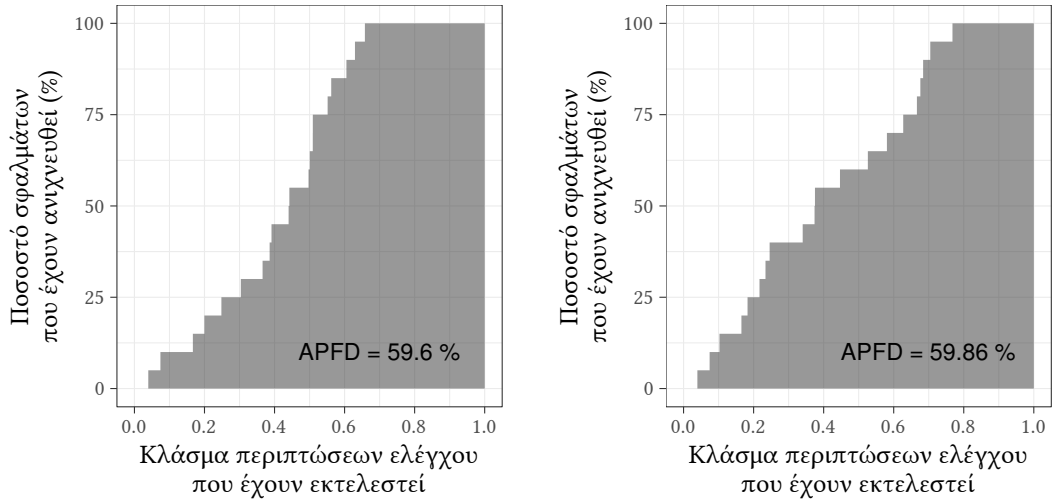
(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random



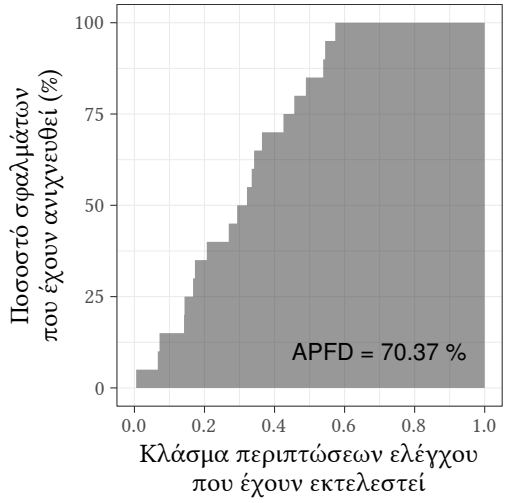
(δ) ΤΠ4: Inverse

Σχήμα 32: Υπολογισμοί APFD για το PyPy3.6 7.1.0 (τεχνικές σύγκρισης)

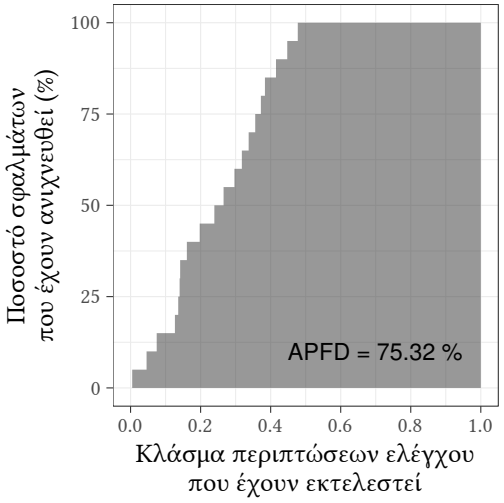


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

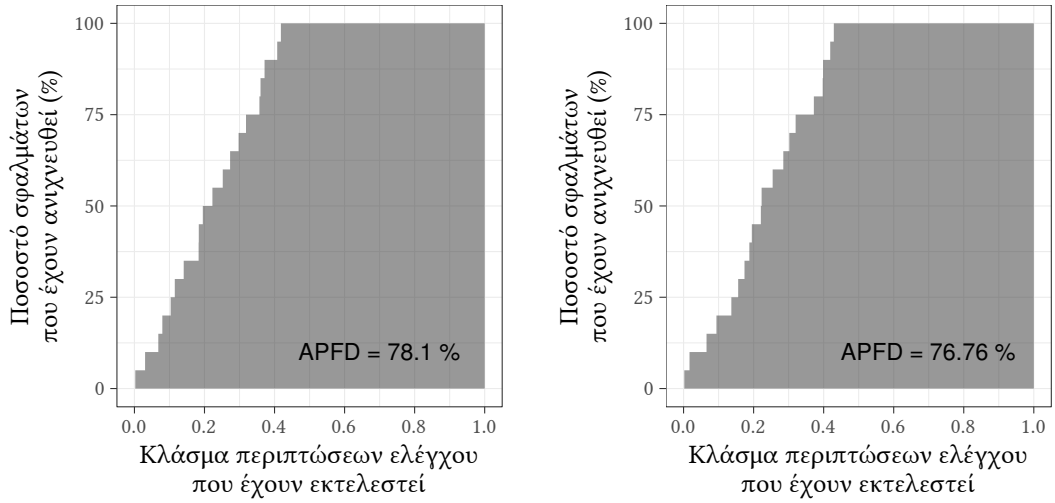


(γ) ΤΠ7: Method-total



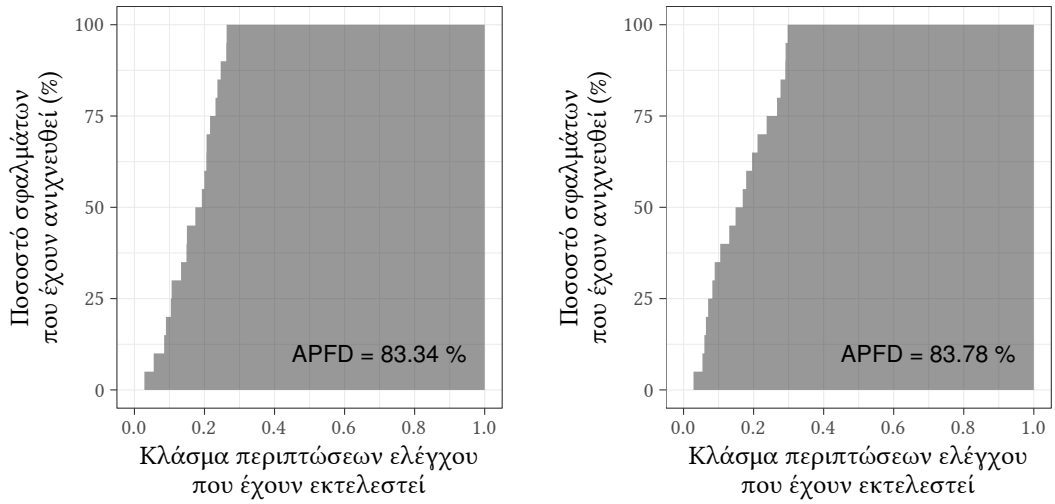
(δ) ΤΠ8: Method-addtl

Σχήμα 33: Υπολογισμοί APFD για το PyPy3.6 7.1.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

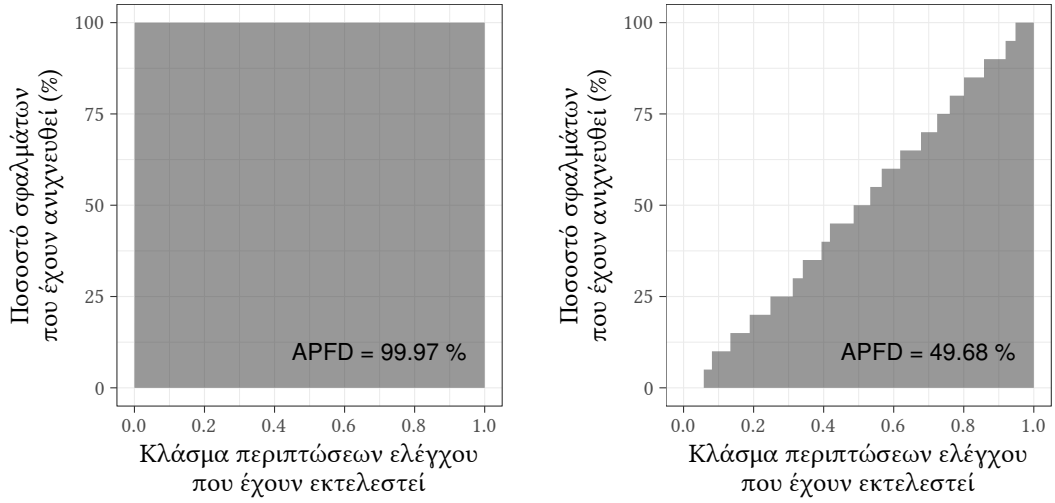
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

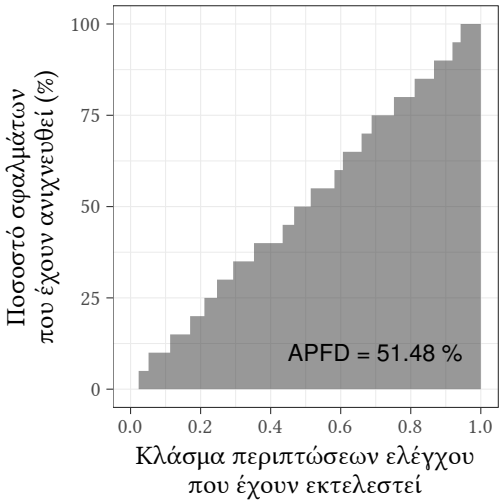
Σχήμα 34: Υπολογισμοί APFD για το PyPy3.6 7.1.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



(α) ΤΠ1: Optimal

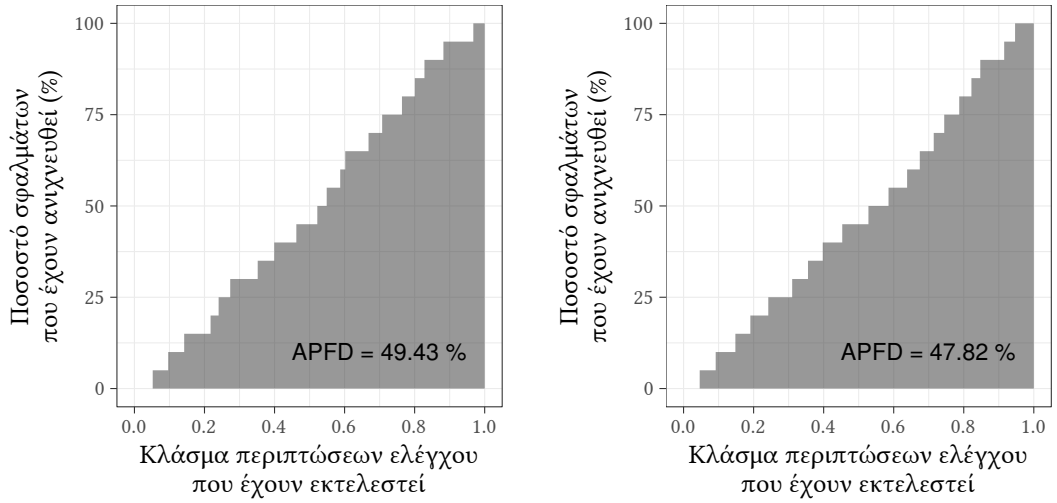
(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random



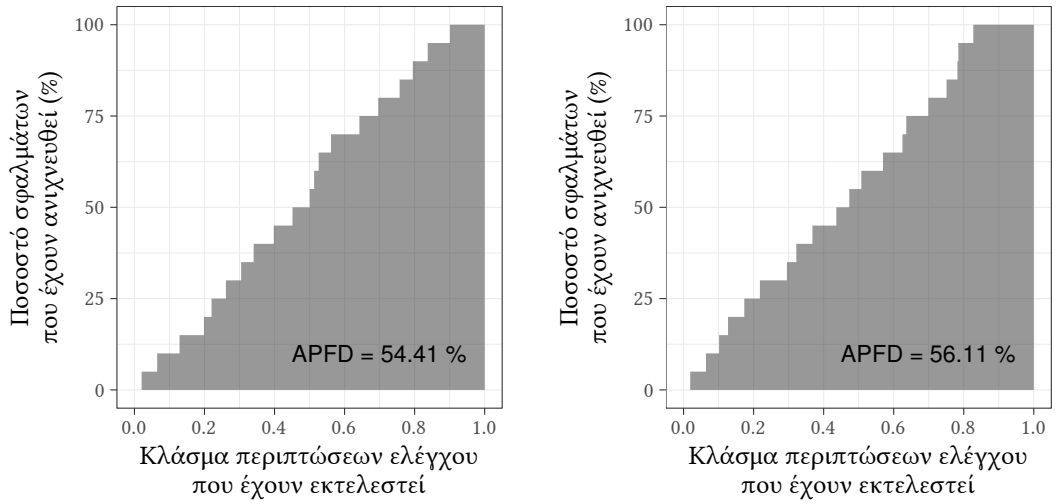
(δ) ΤΠ4: Inverse

Σχήμα 35: Υπολογισμοί APFD για το PyPy3.6 7.1.1 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

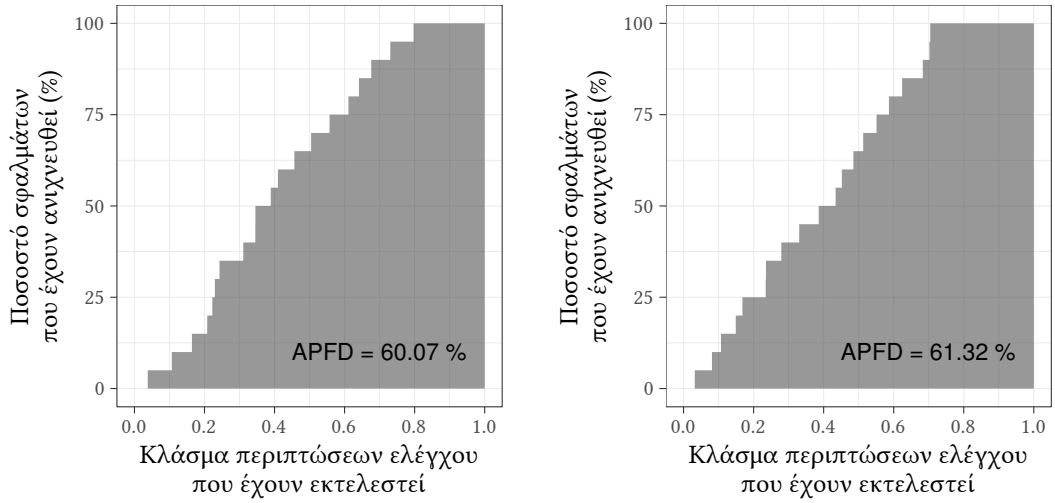
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

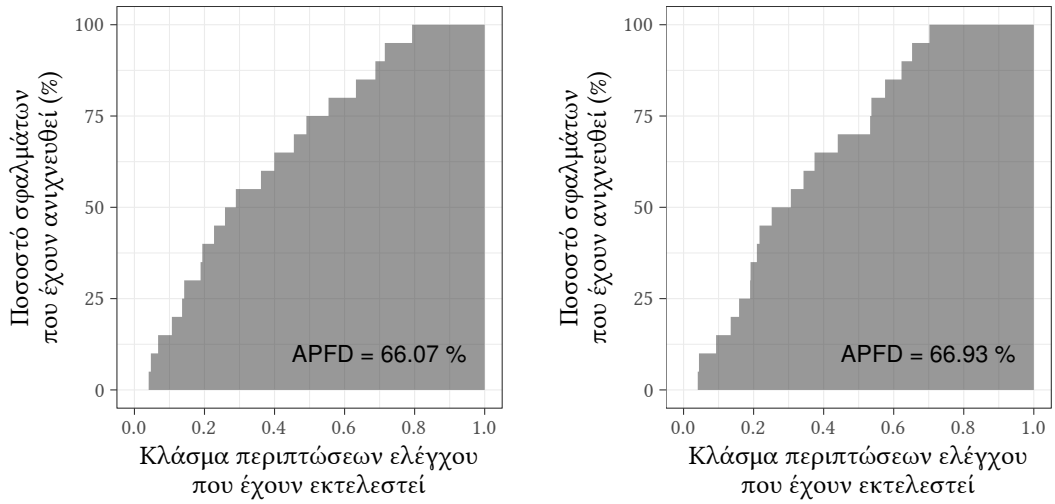
(δ) ΤΠ8: Method-addtl

Σχήμα 36: Υπολογισμοί APFD για το PyPy3.6 7.1.1 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

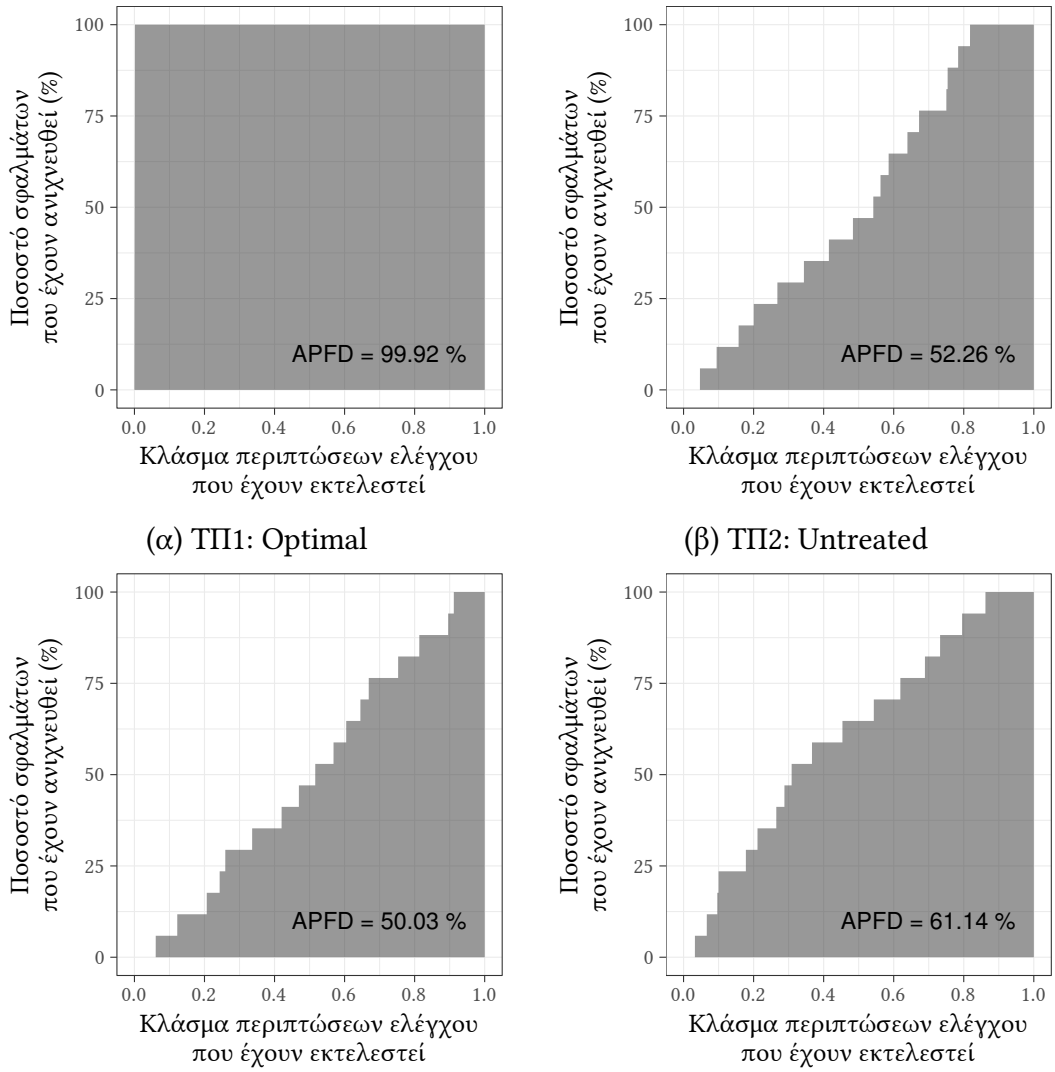


(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 37: Υπολογισμοί APFD για το PyPy3.6 7.1.1 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

Μετρήσεις APFD για το Django



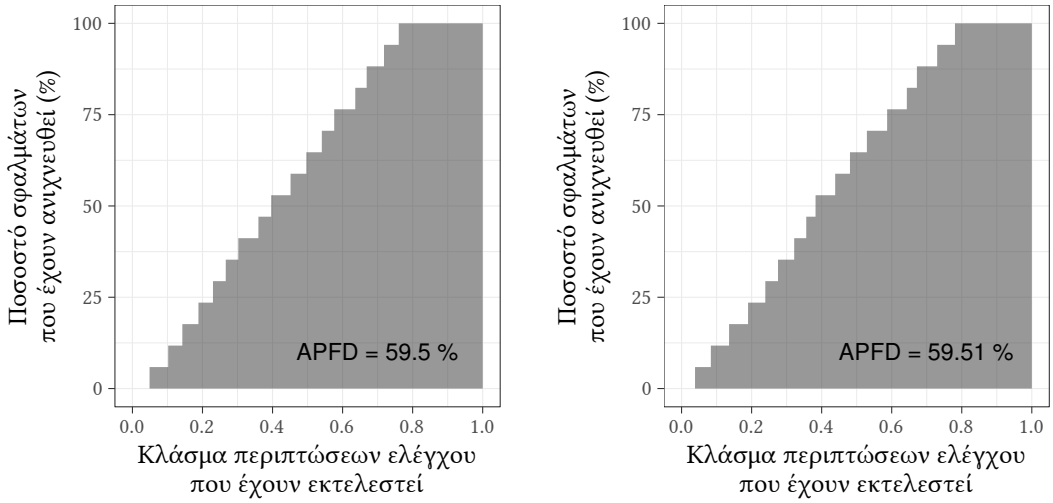
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

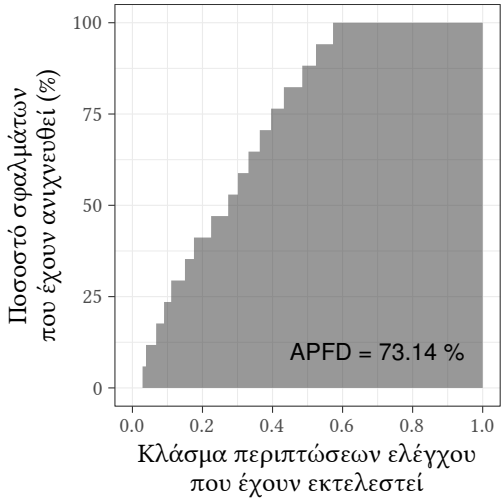
(δ) ΤΠ4: Inverse

Σχήμα 38: Υπολογισμοί APFD για το Django 2.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

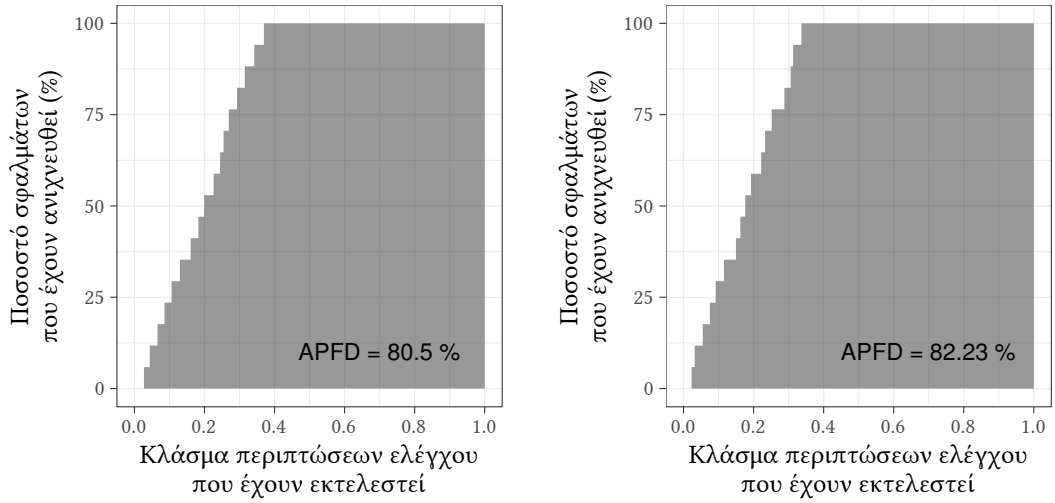
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

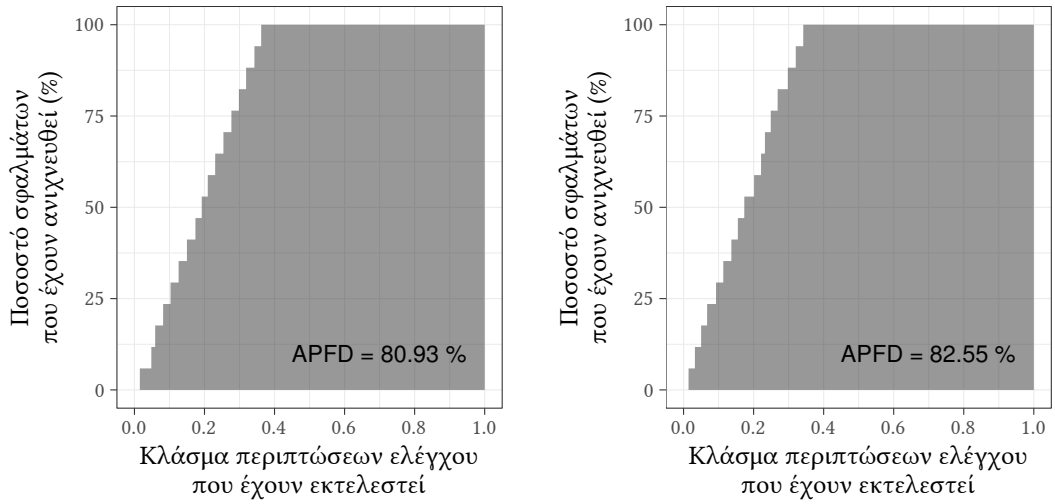
(δ) ΤΠ8: Method-addtl

Σχήμα 39: Υπολογισμοί APFD για το Django 2.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

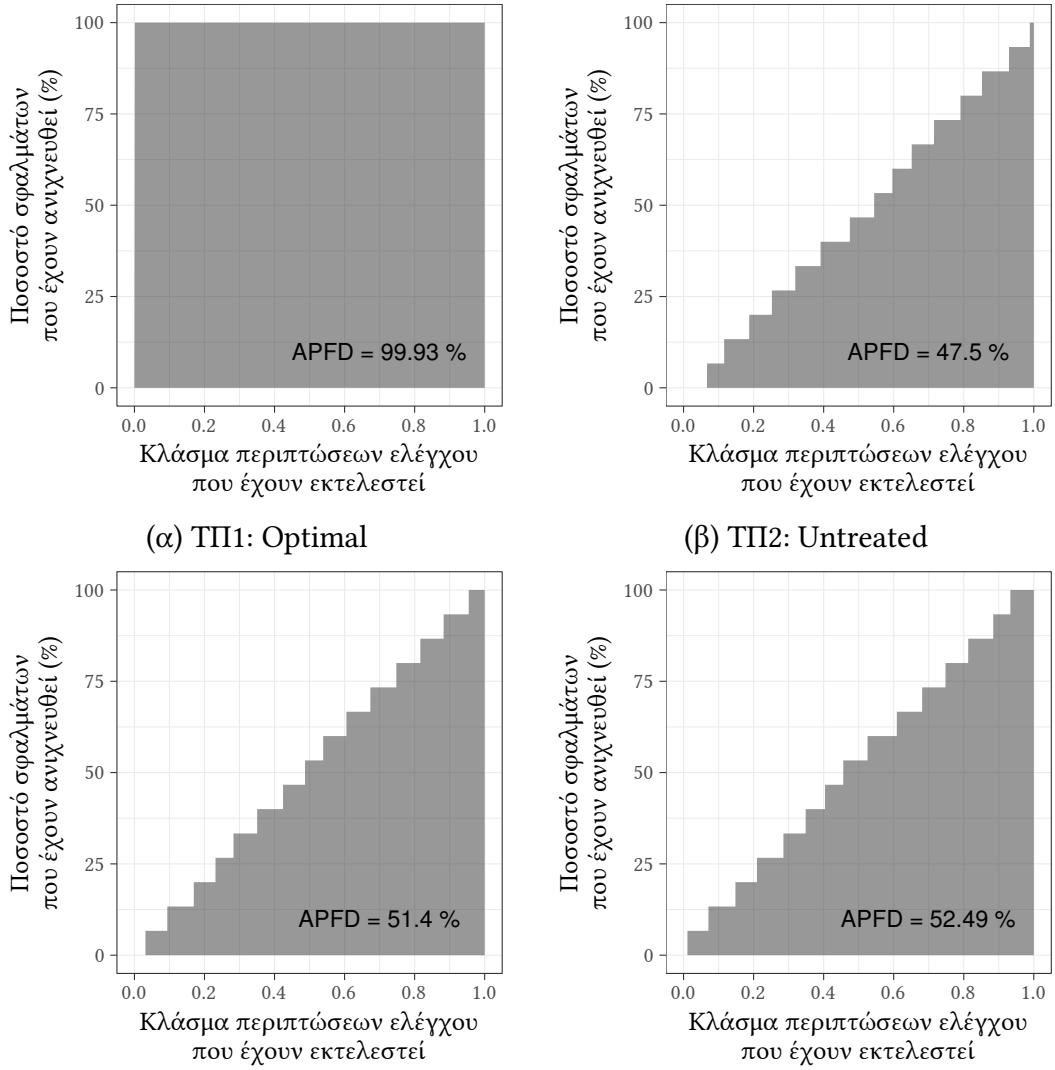
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 40: Υπολογισμοί APFD για το Django 2.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



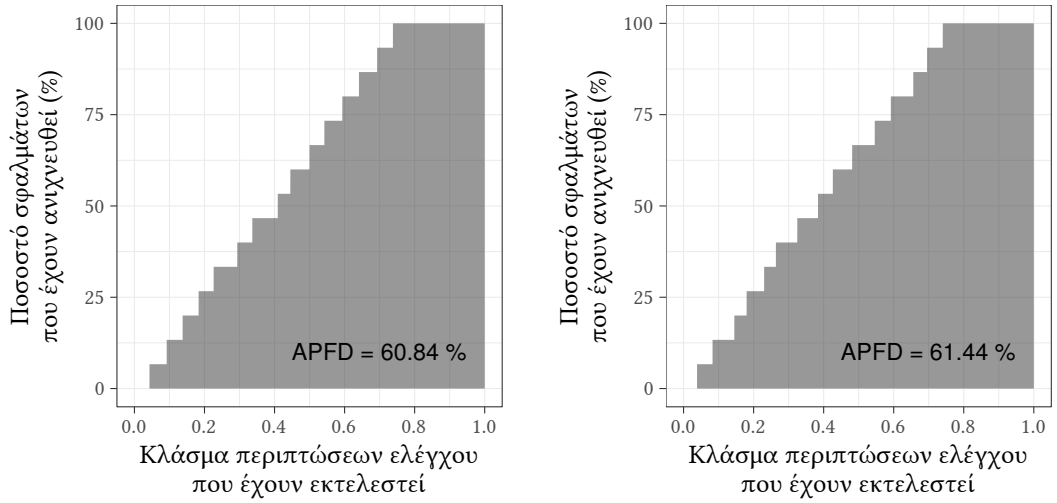
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

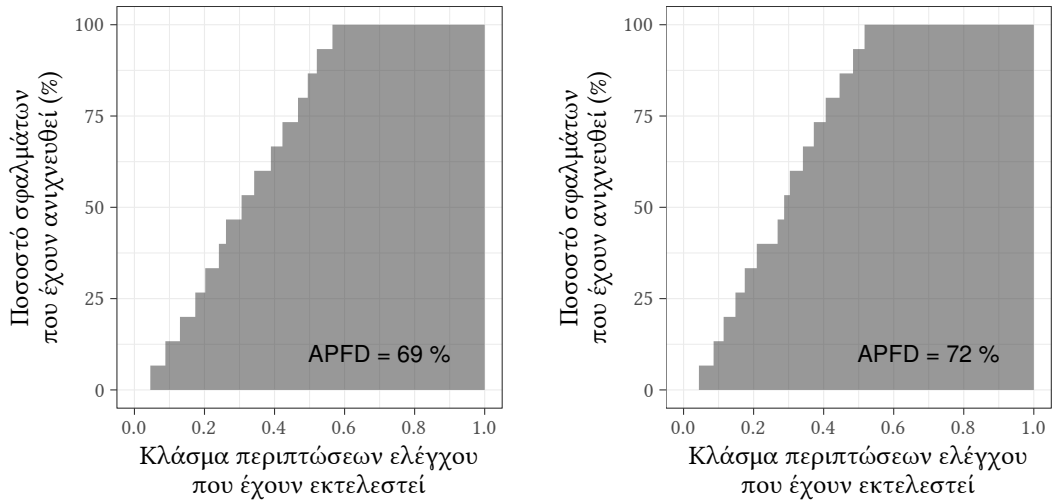
(δ) ΤΠ4: Inverse

Σχήμα 41: Υπολογισμοί APFD για το Django 2.0.1 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

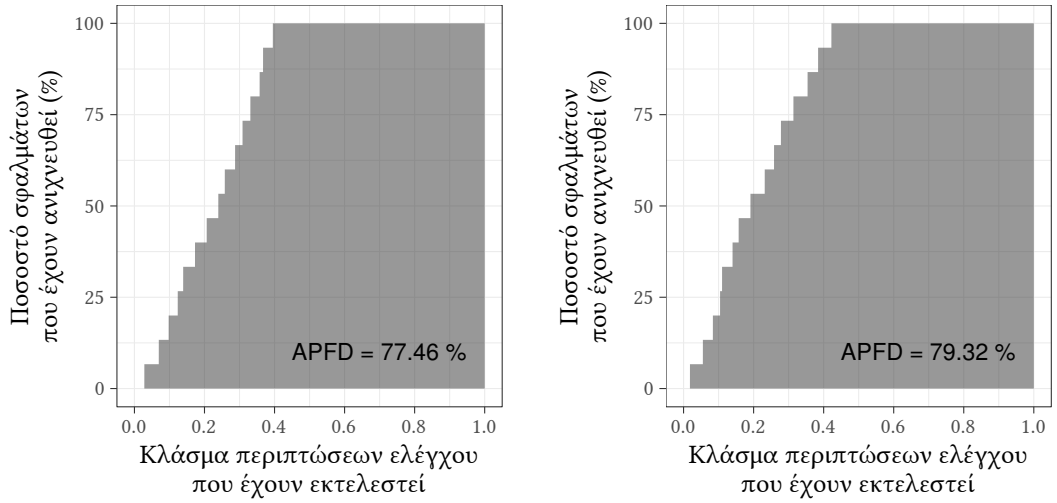
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

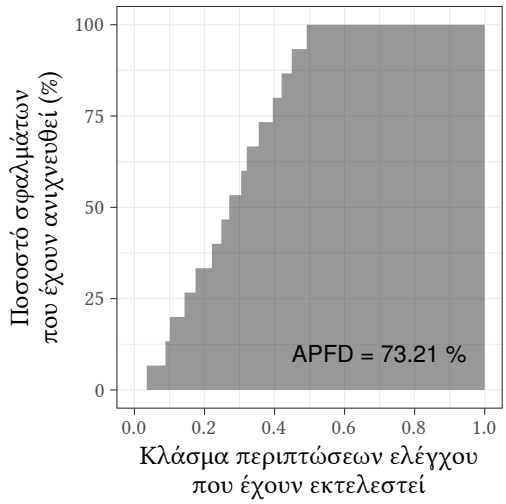
(δ) ΤΠ8: Method-addtl

Σχήμα 42: Υπολογισμοί APFD για το Django 2.0.1 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

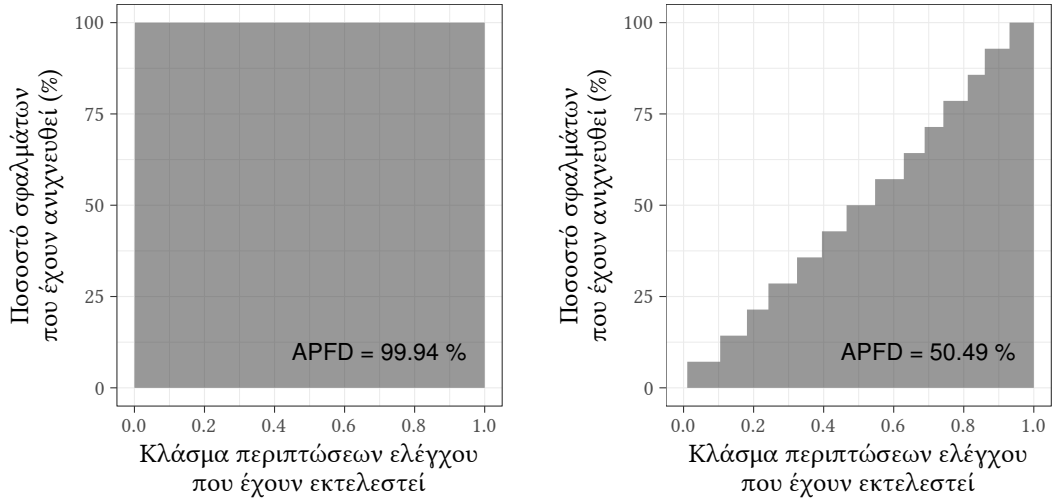
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

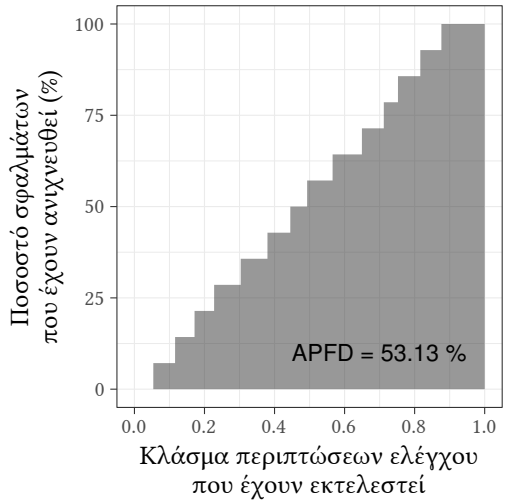
(δ) ΤΠ12: Statement-addtl

Σχήμα 43: Υπολογισμοί APFD για το Django 2.0.1 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

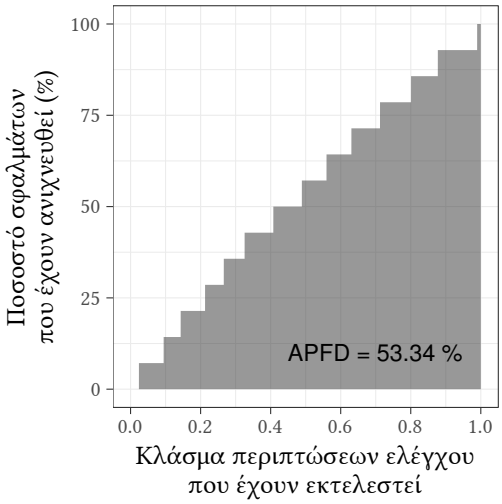


(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

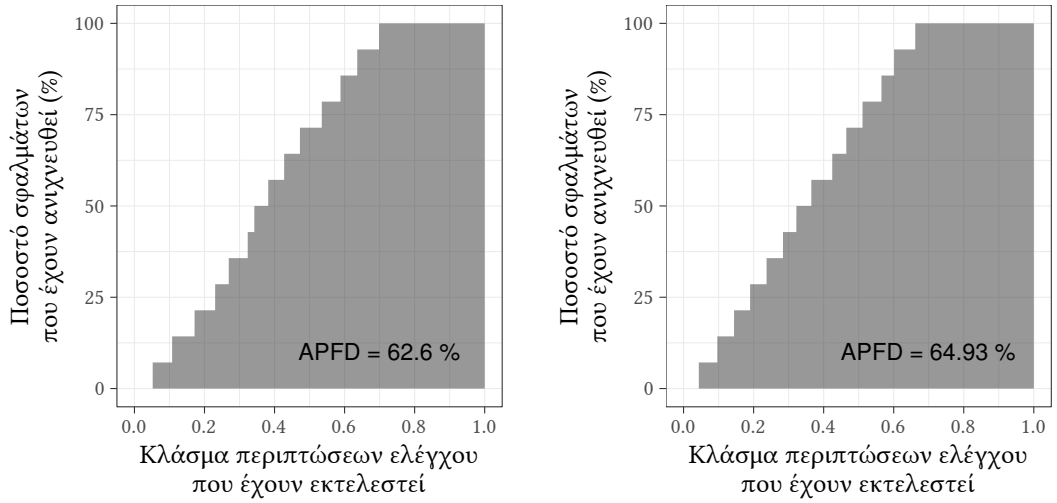


(γ) ΤΠ3: Random



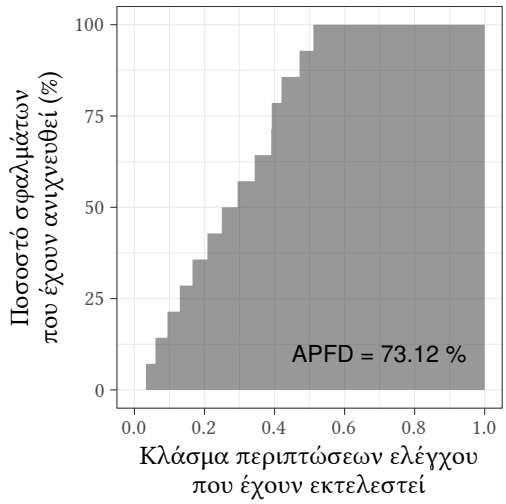
(δ) ΤΠ4: Inverse

Σχήμα 44: Υπολογισμοί APFD για το Django 2.0.2 (τεχνικές σύγκρισης)

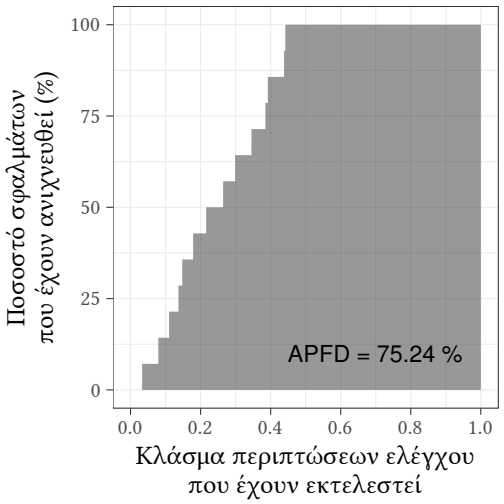


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

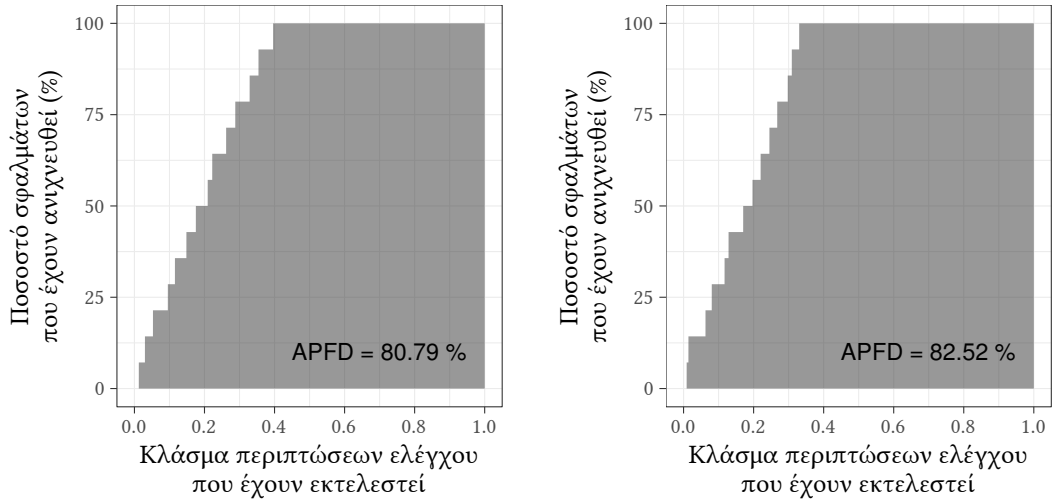


(γ) ΤΠ7: Method-total



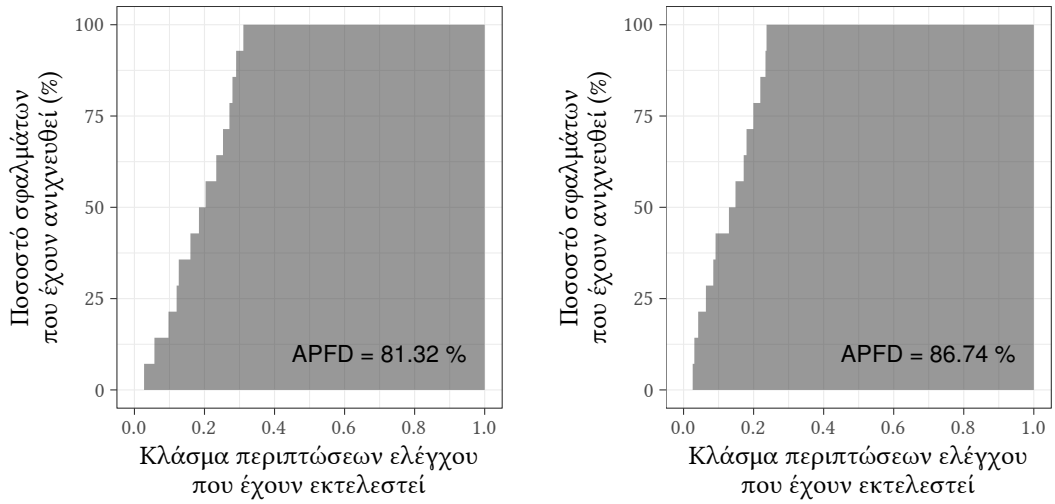
(δ) ΤΠ8: Method-addtl

Σχήμα 45: Υπολογισμοί APFD για το Django 2.0.2 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

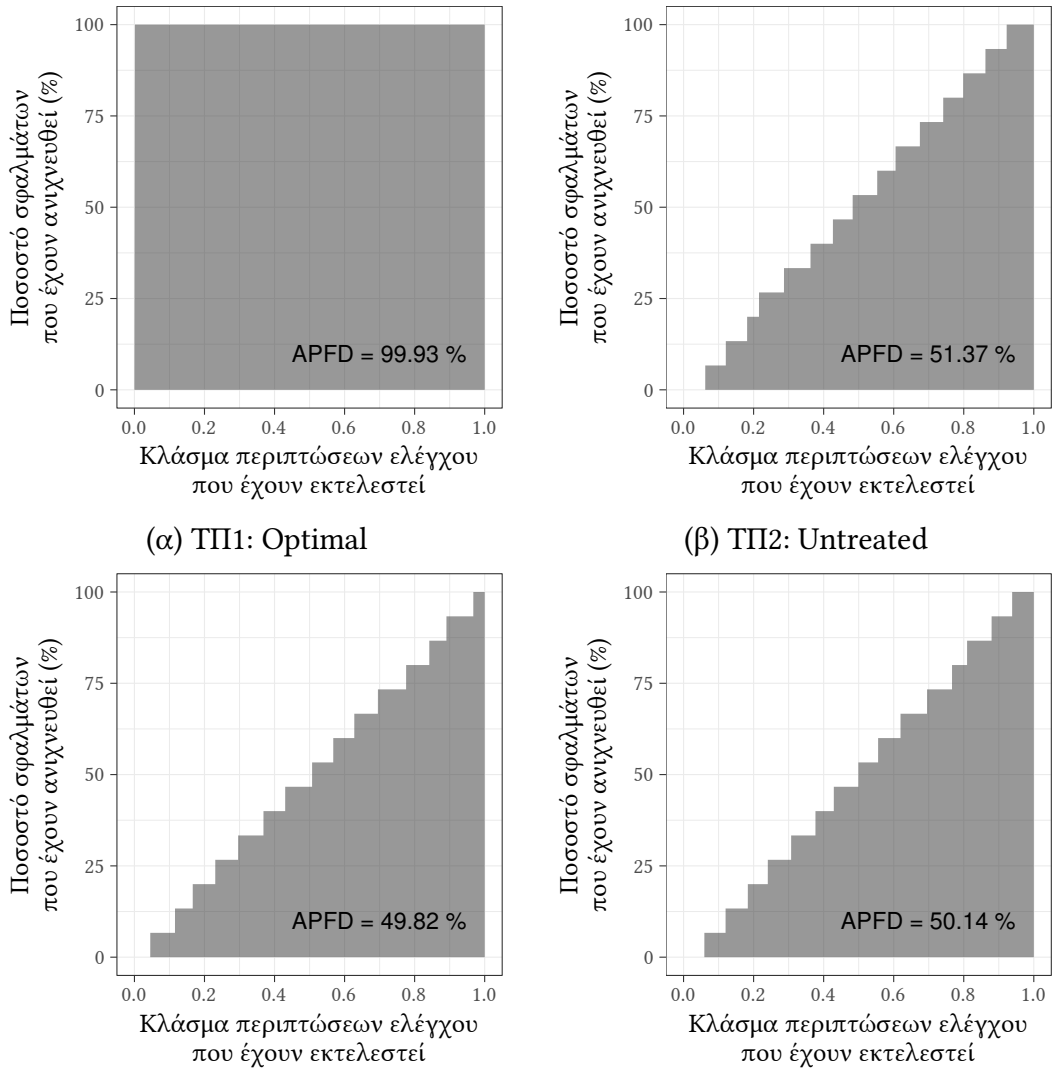
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 46: Υπολογισμοί APFD για το Django 2.0.2 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



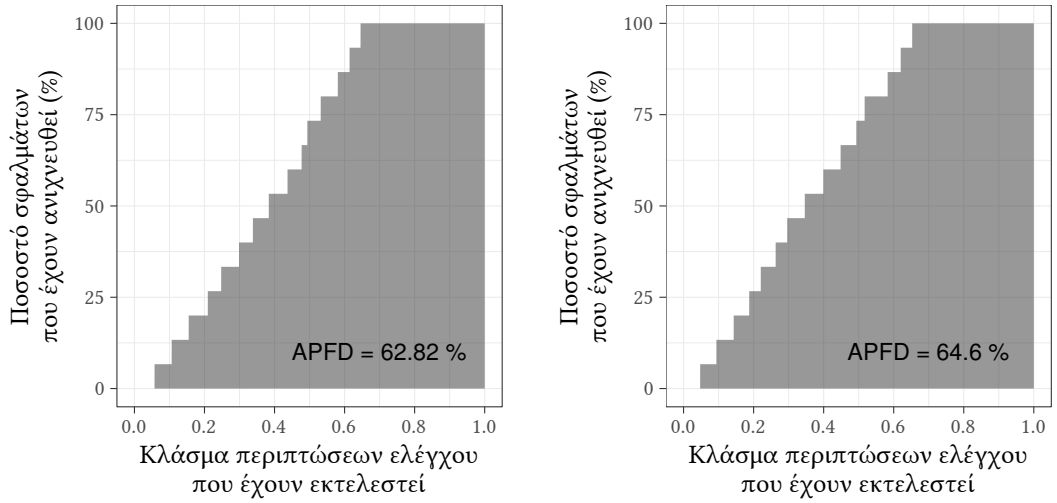
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

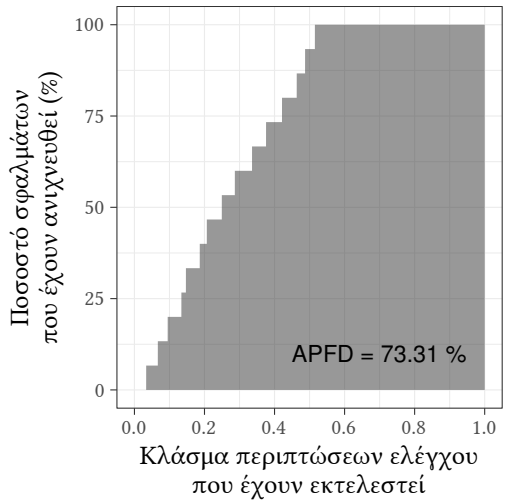
(δ) ΤΠ4: Inverse

Σχήμα 47: Υπολογισμοί APFD για το Django 2.0.3 (τεχνικές σύγκρισης)

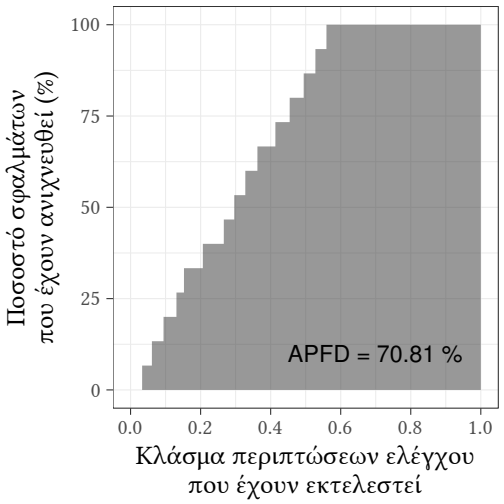


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

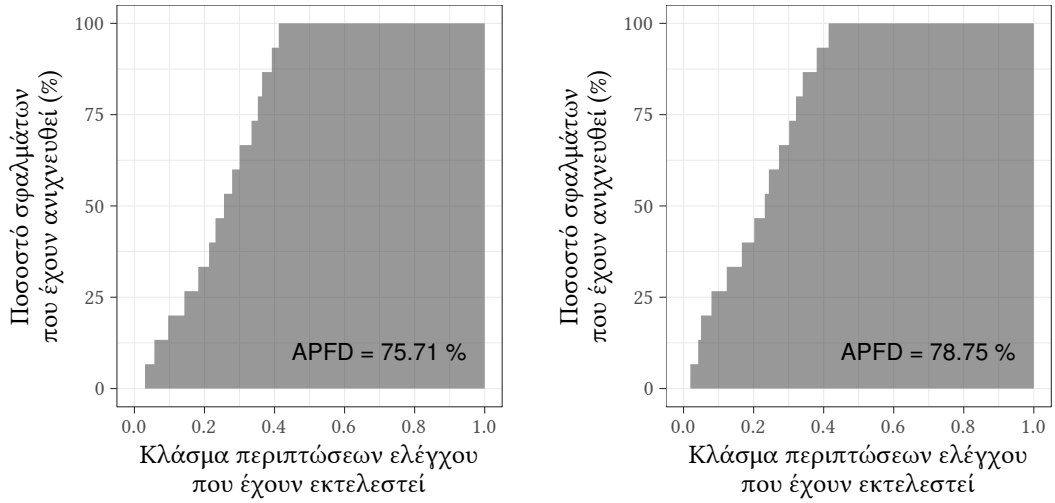


(γ) ΤΠ7: Method-total



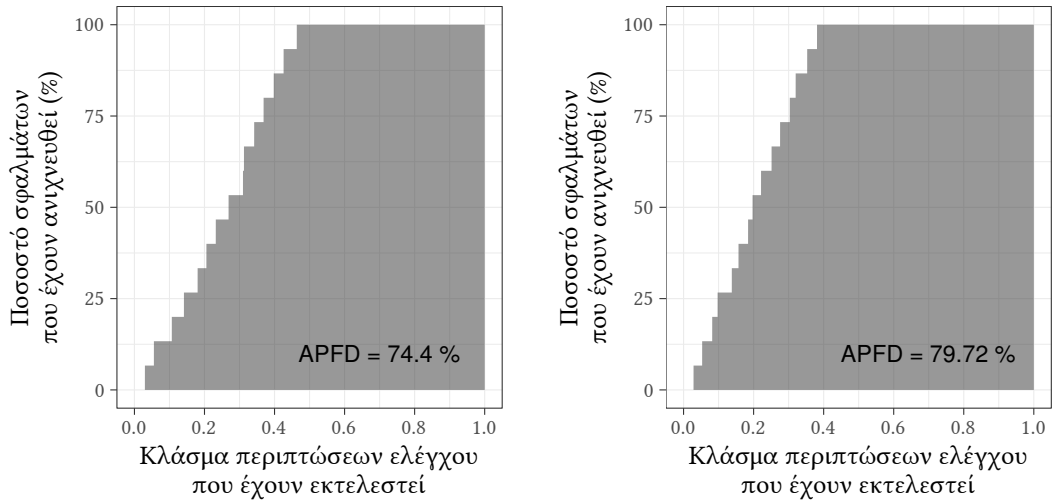
(δ) ΤΠ8: Method-addtl

Σχήμα 48: Υπολογισμοί APFD για το Django 2.0.3 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

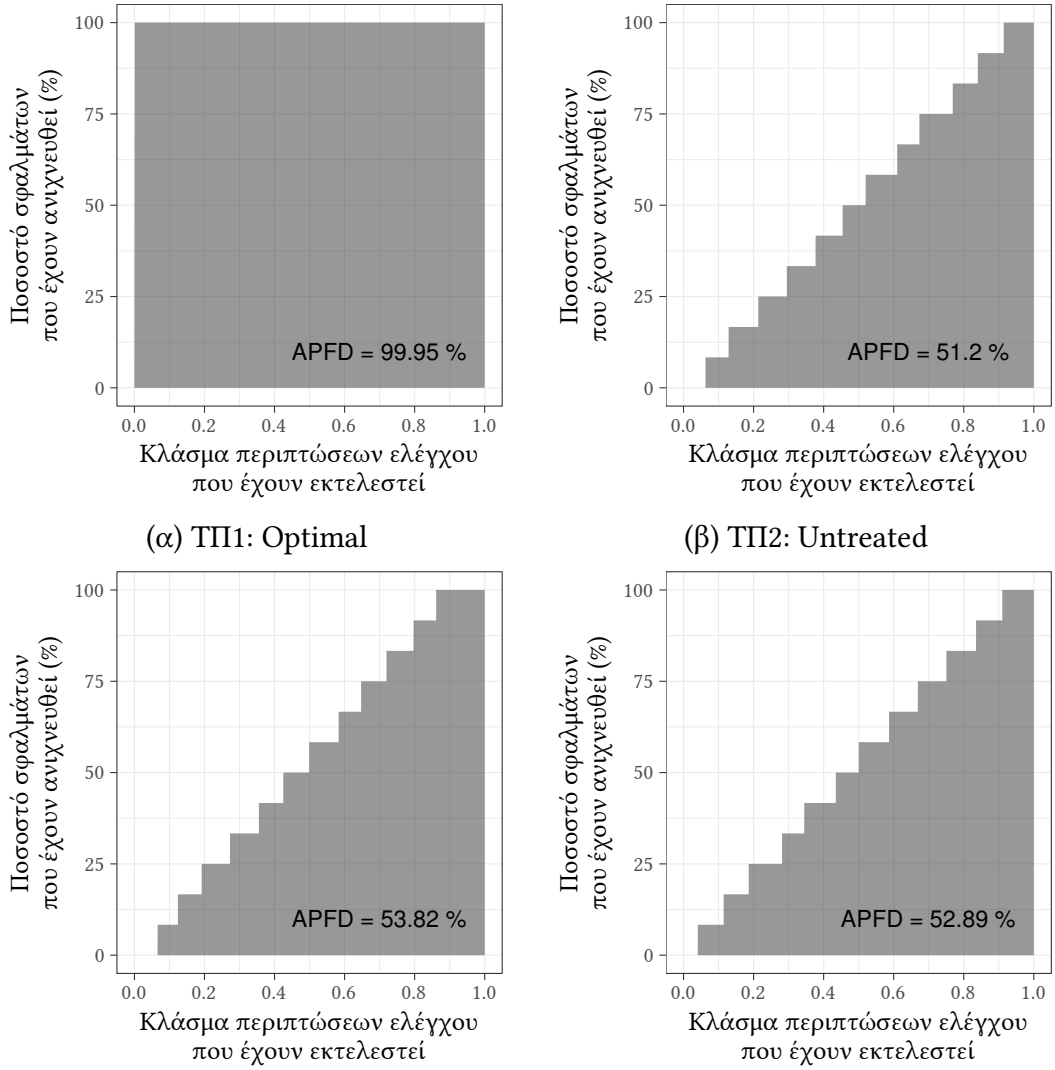
(β) ΤΠ10: Branch-addtl



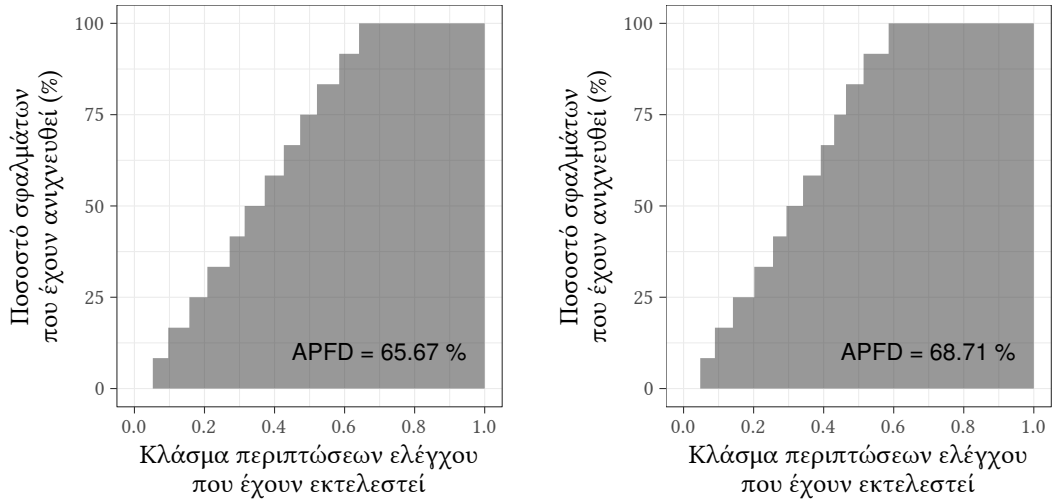
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 49: Υπολογισμοί APFD για το Django 2.0.3 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

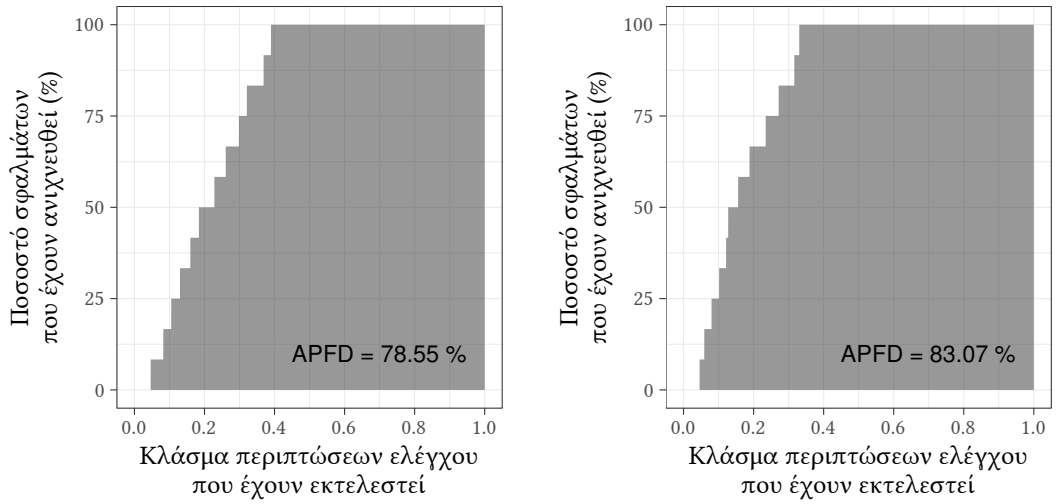


Σχήμα 50: Υπολογισμοί APFD για το Django 2.0.4 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

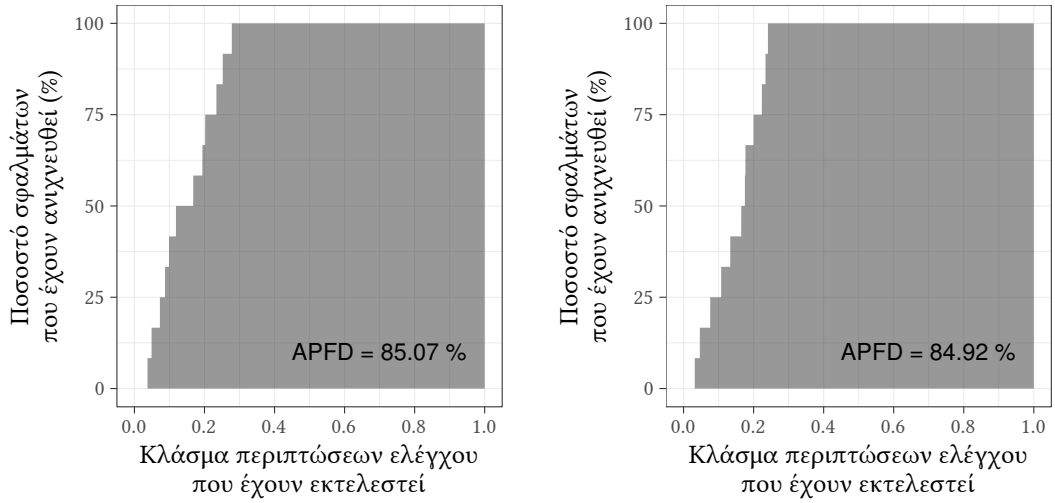
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

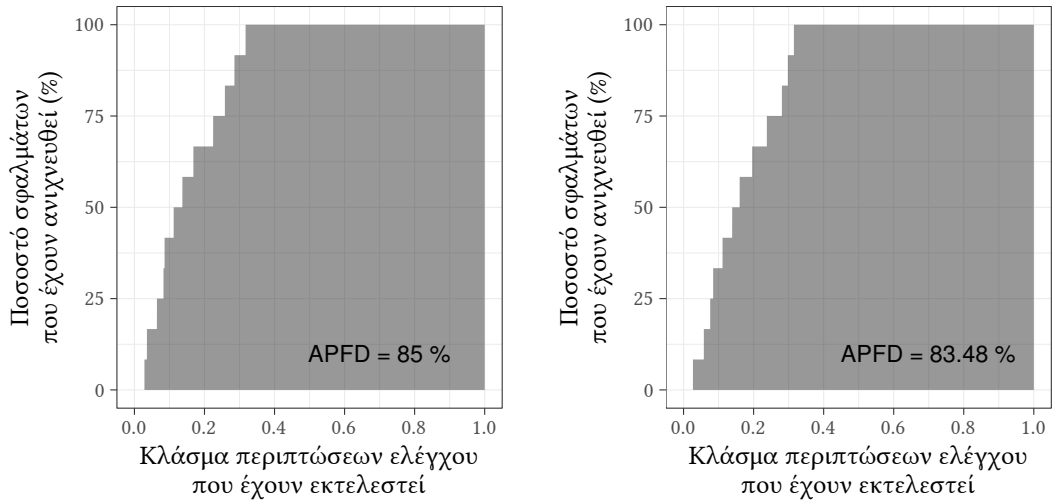
(δ) ΤΠ8: Method-addtl

Σχήμα 51: Υπολογισμοί APFD για το Django 2.0.4 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

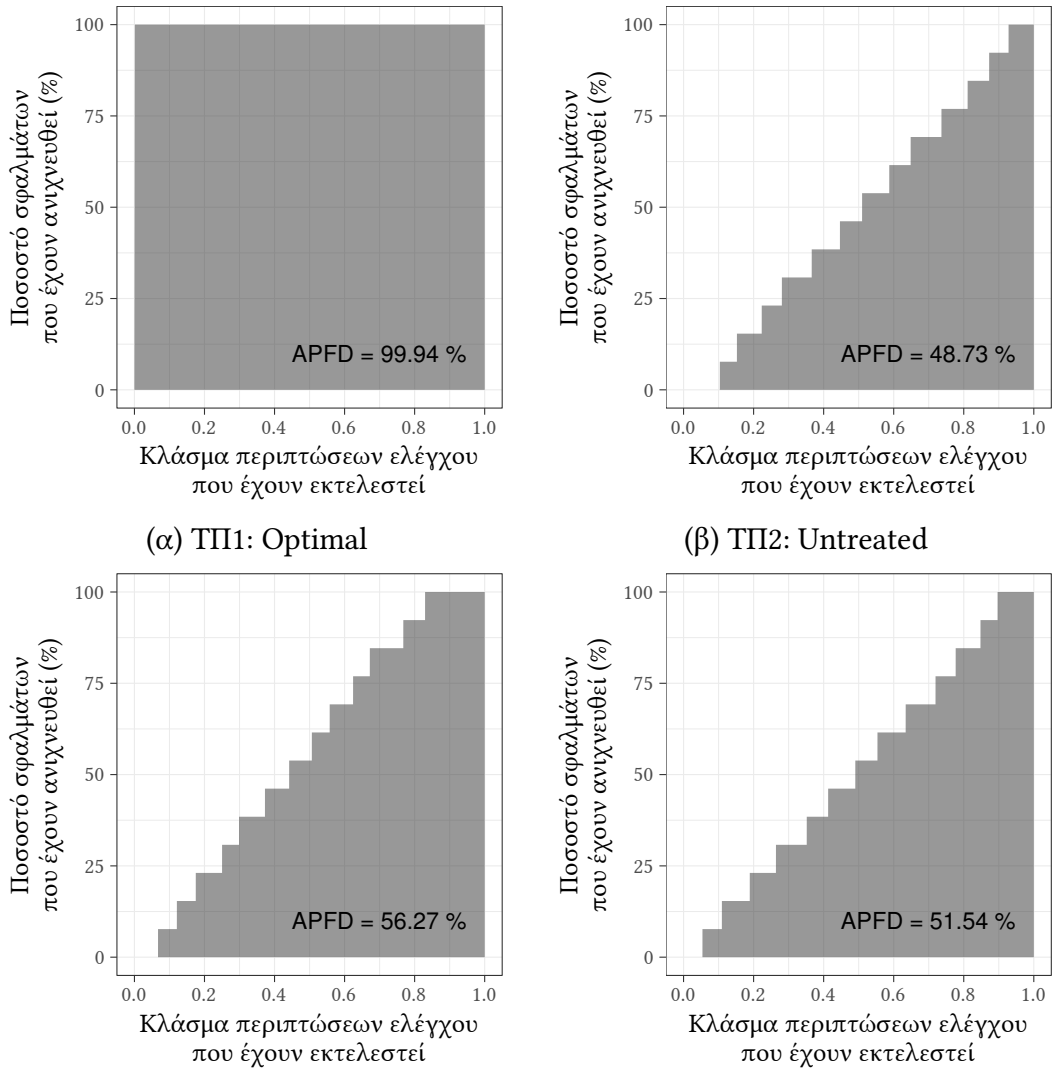
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 52: Υπολογισμοί APFD για το Django 2.0.4 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



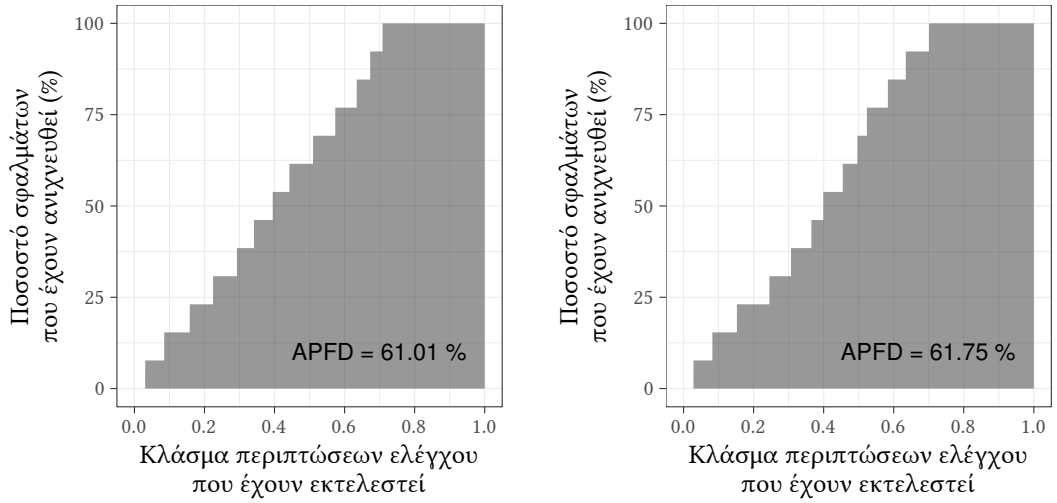
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

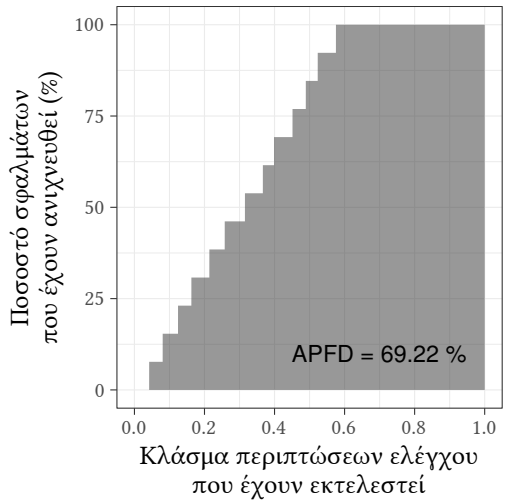
(δ) ΤΠ4: Inverse

Σχήμα 53: Υπολογισμοί APFD για το Django 2.0.5 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

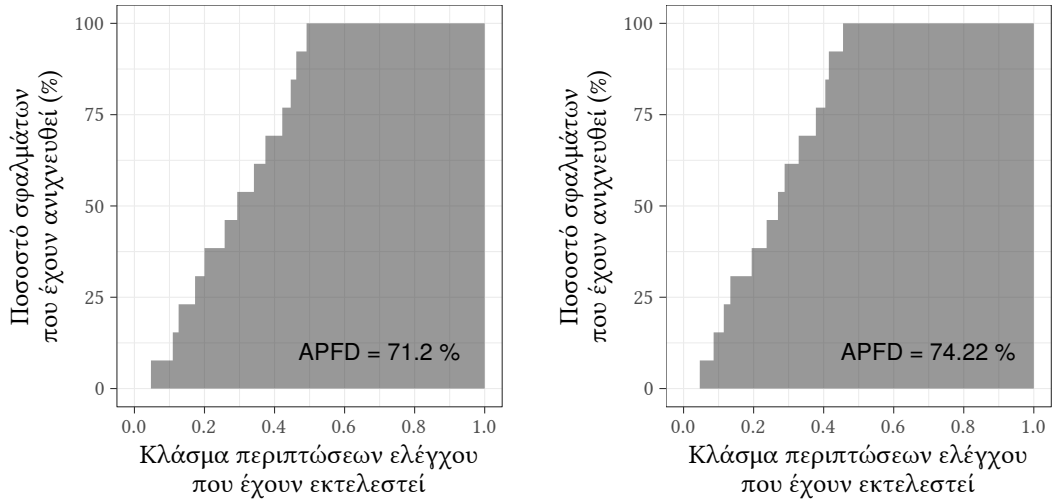
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

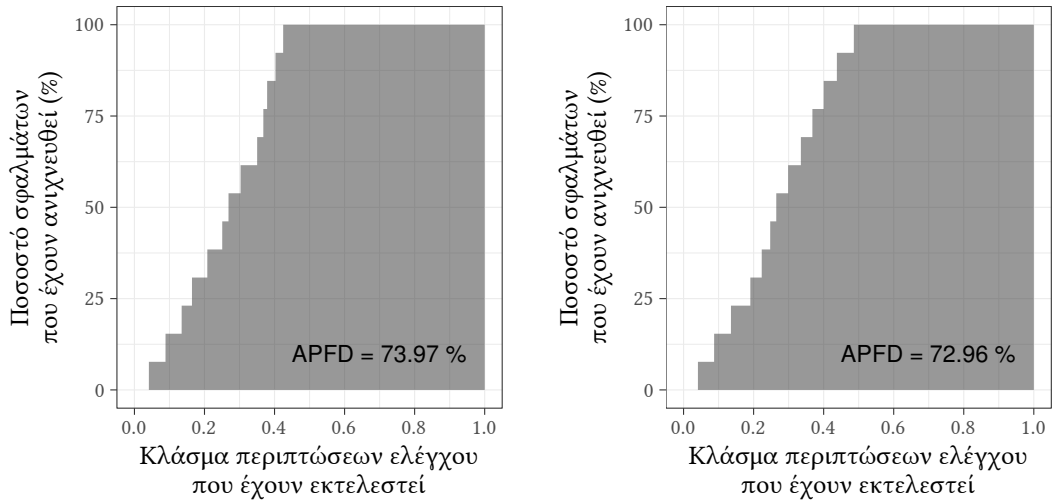
(δ) ΤΠ8: Method-addtl

Σχήμα 54: Υπολογισμοί APFD για το Django 2.0.5 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

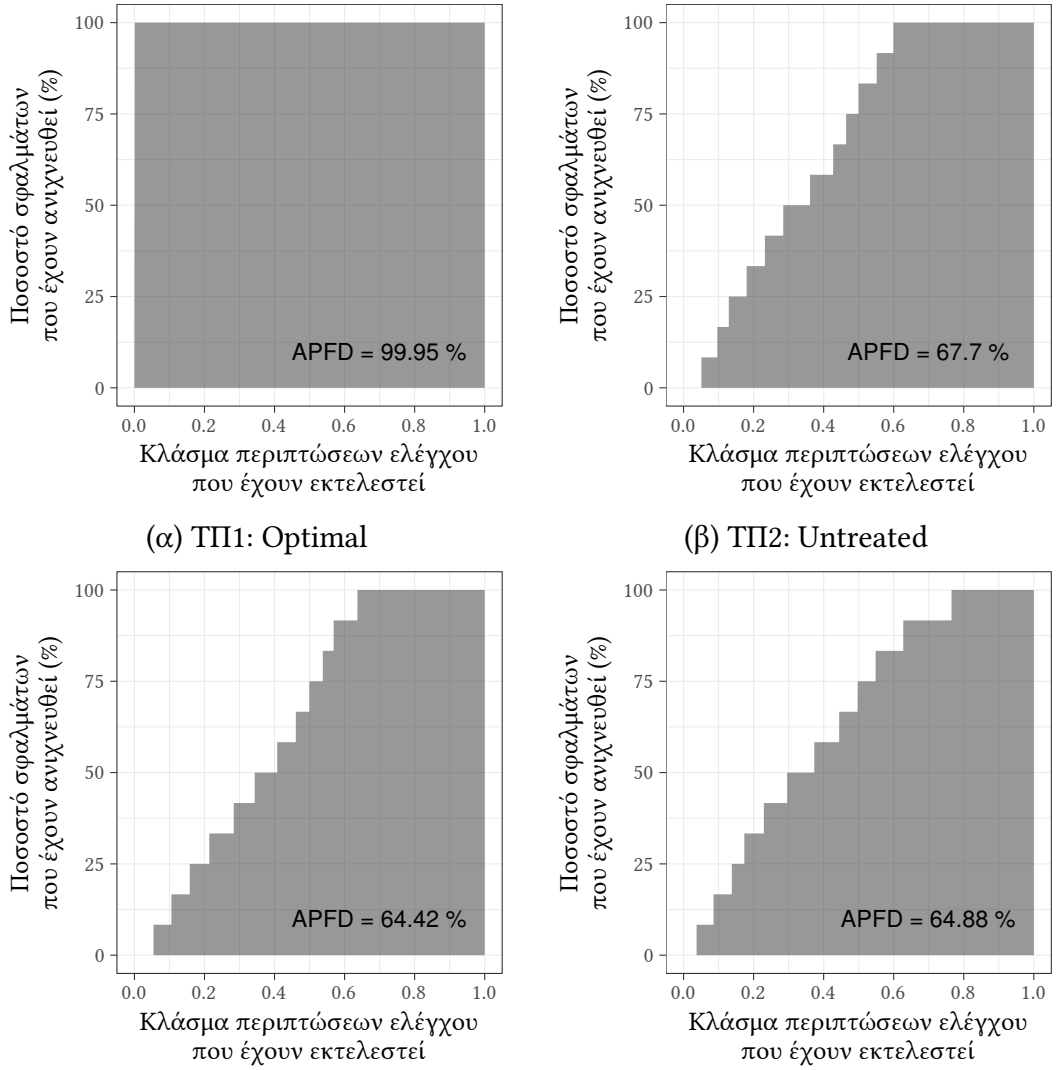
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 55: Υπολογισμοί APFD για το Django 2.0.5 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



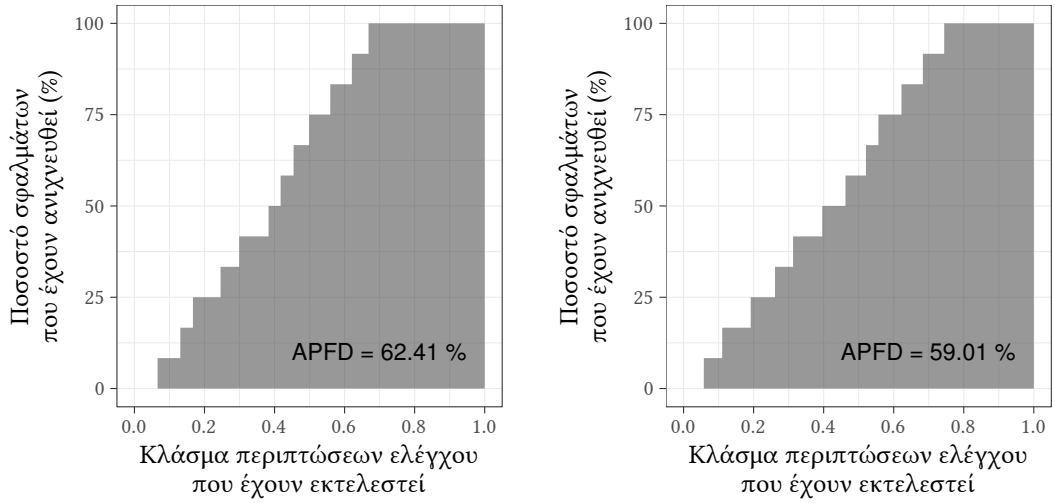
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

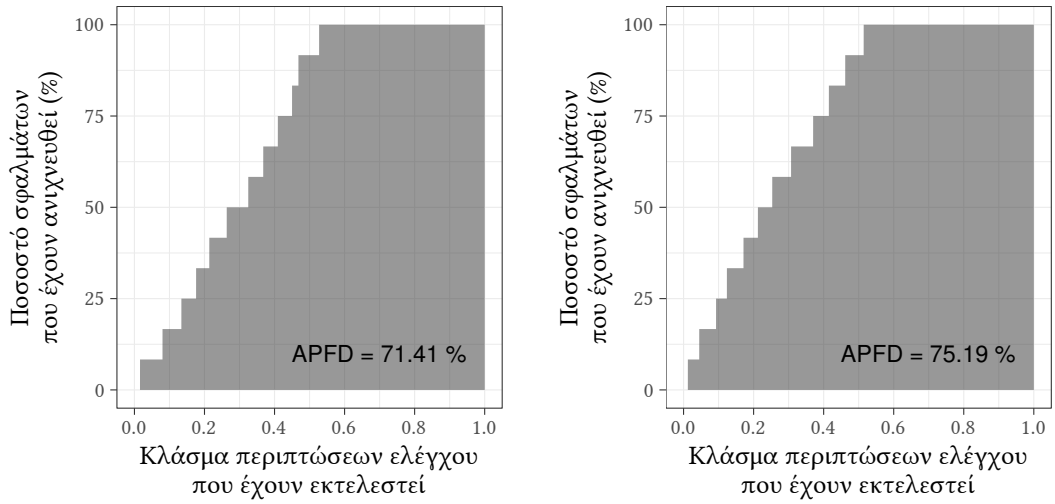
(δ) ΤΠ4: Inverse

Σχήμα 56: Υπολογισμοί APFD για το Django 2.0.6 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

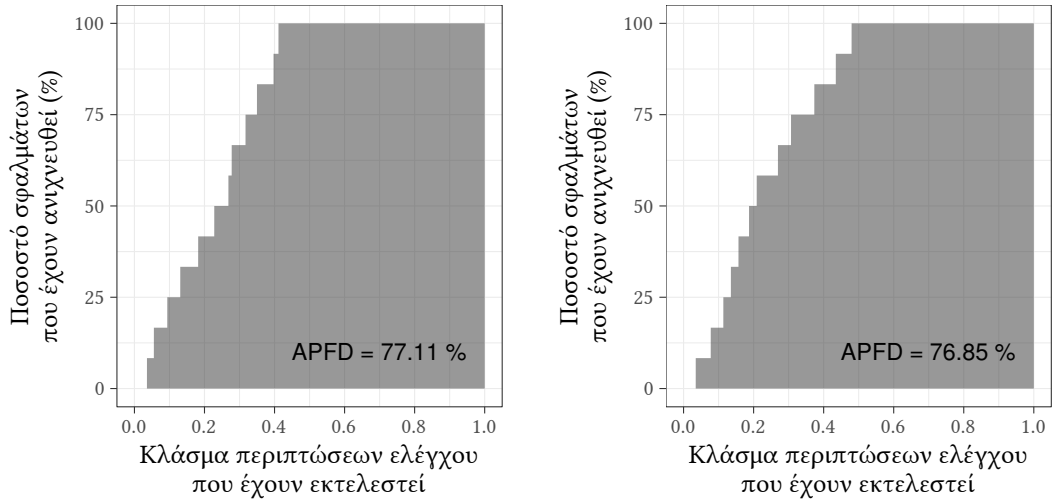
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

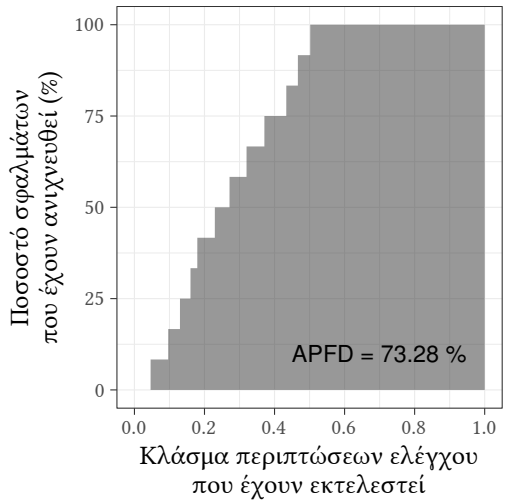
(δ) ΤΠ8: Method-addtl

Σχήμα 57: Υπολογισμοί APFD για το Django 2.0.6 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)

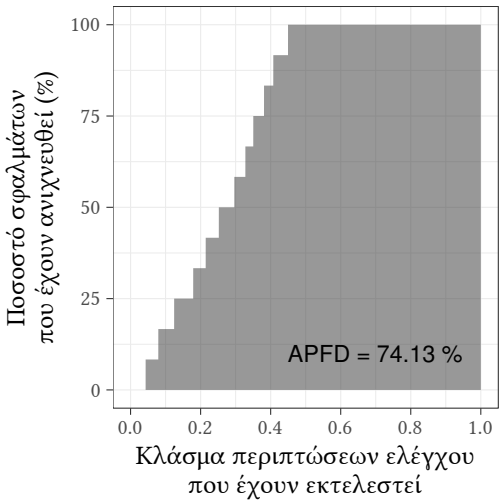


(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

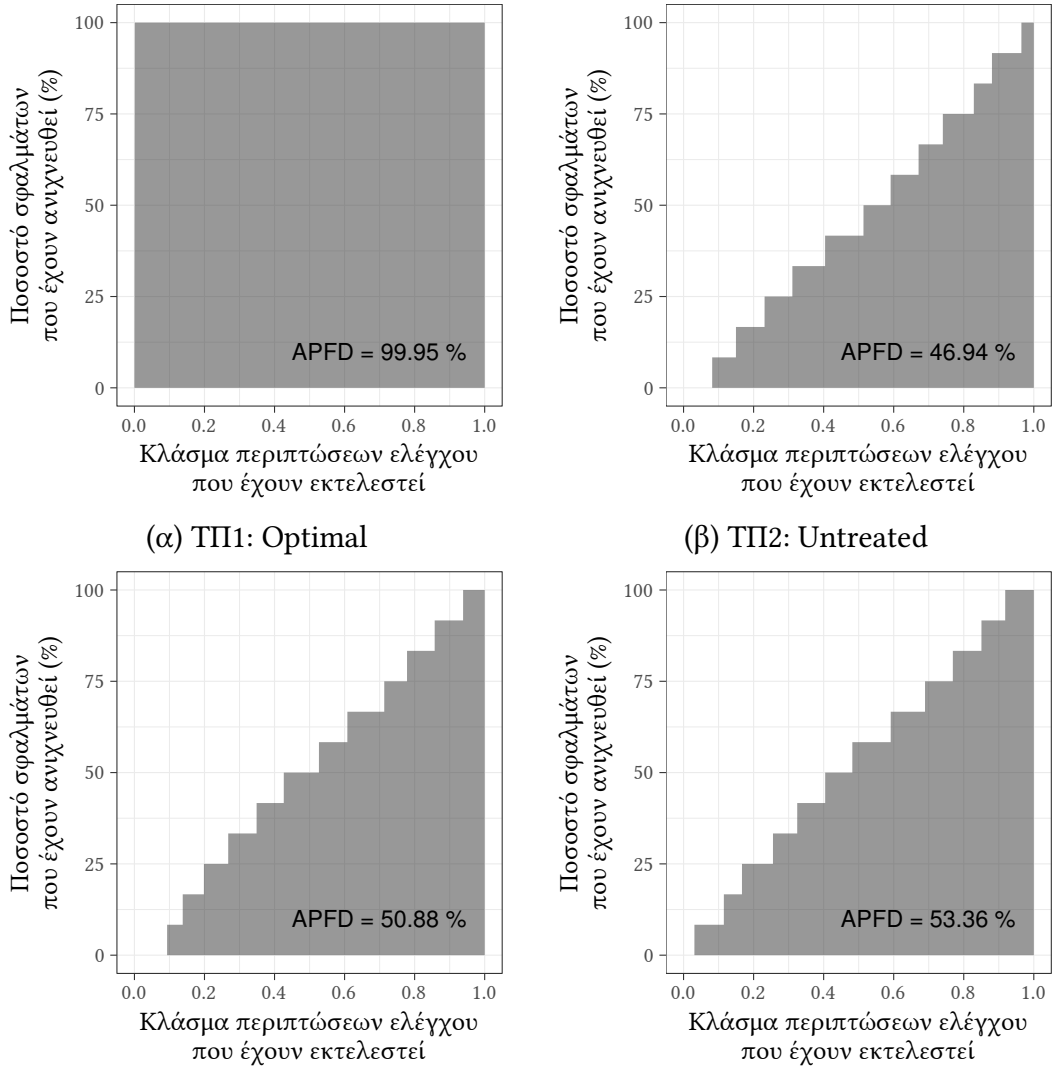


(γ) ΤΠ11: Statement-total



(δ) ΤΠ12: Statement-addtl

Σχήμα 58: Υπολογισμοί APFD για το Django 2.0.6 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



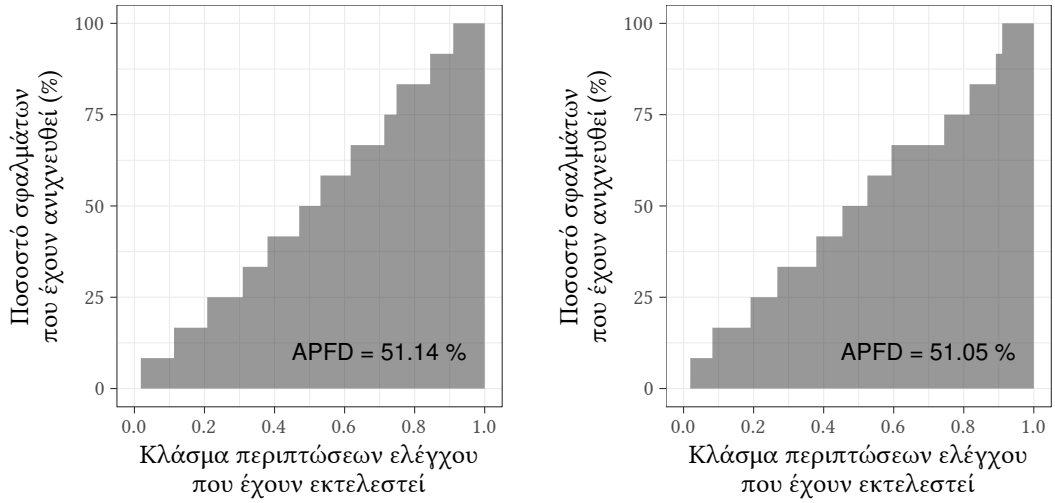
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

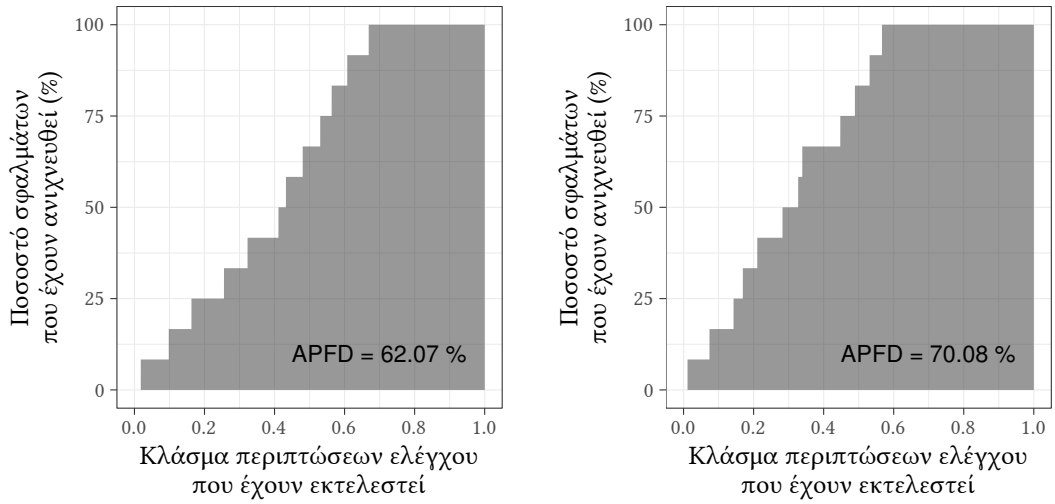
(δ) ΤΠ4: Inverse

Σχήμα 59: Υπολογισμοί APFD για το Django 2.0.7 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

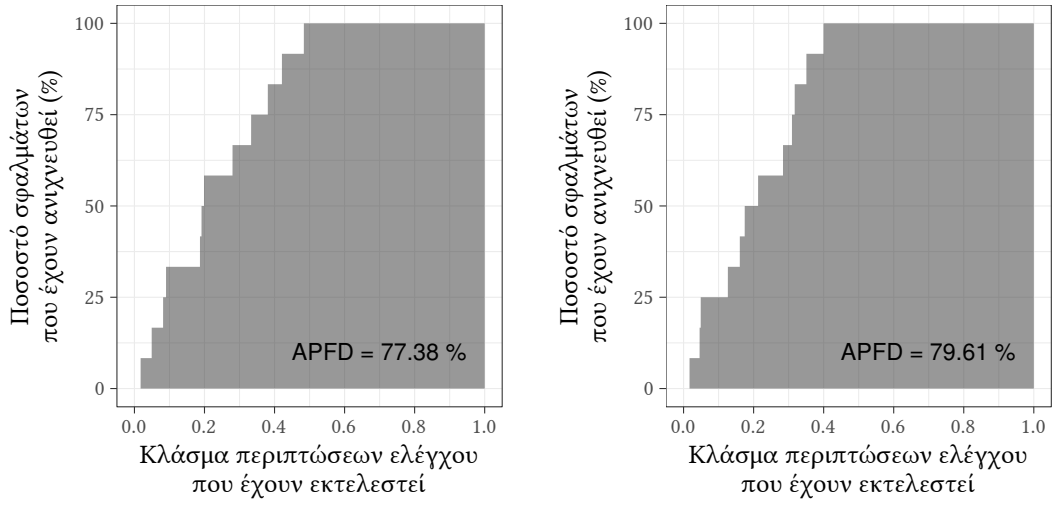
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

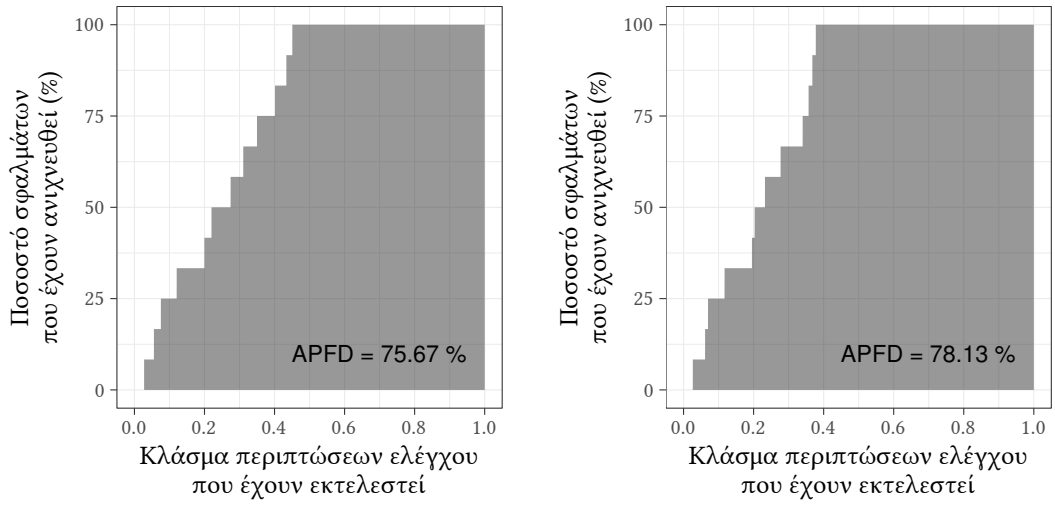
(δ) ΤΠ8: Method-addtl

Σχήμα 60: Υπολογισμοί APFD για το Django 2.0.7 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

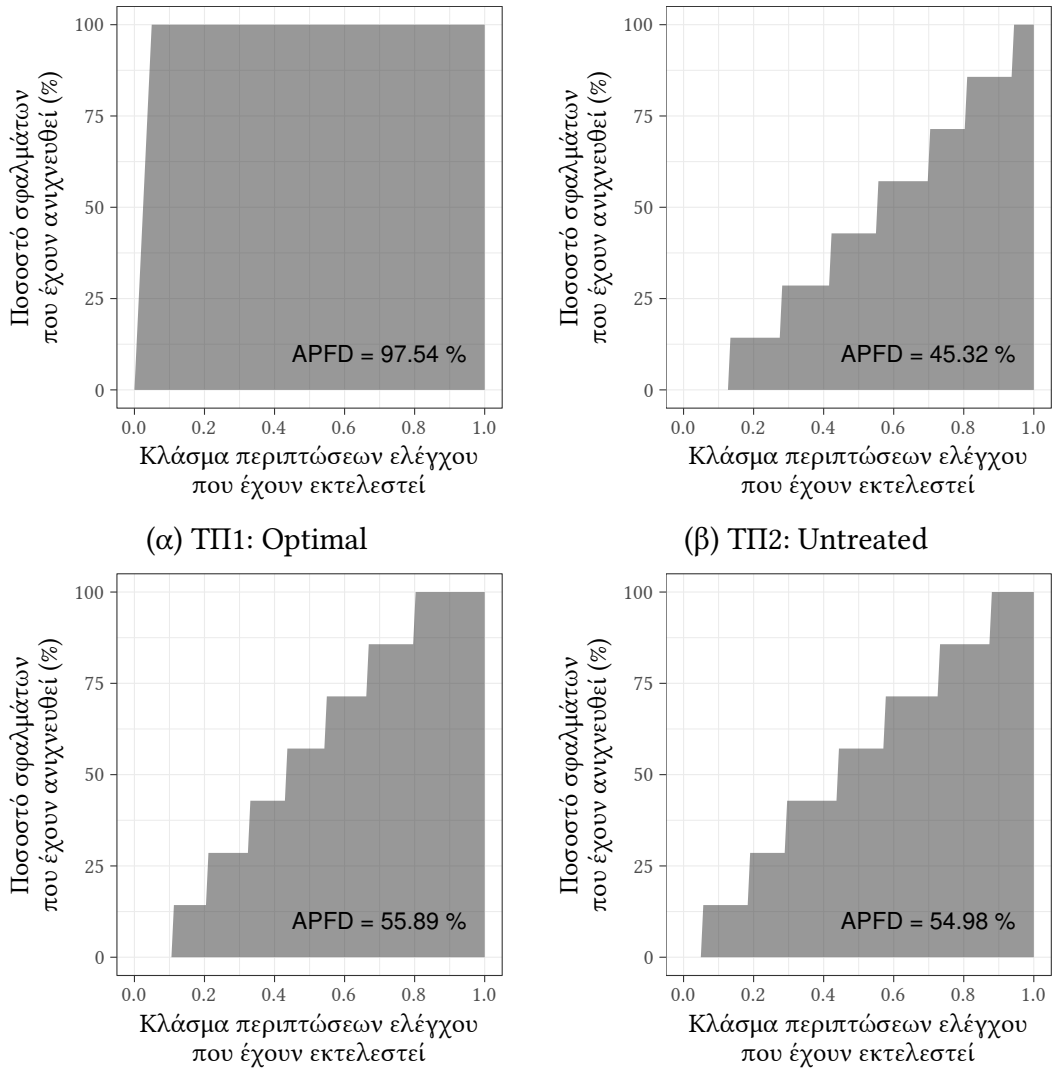


(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 61: Υπολογισμοί APFD για το Django 2.0.7 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

Μετρήσεις APFD για το MoinMoin



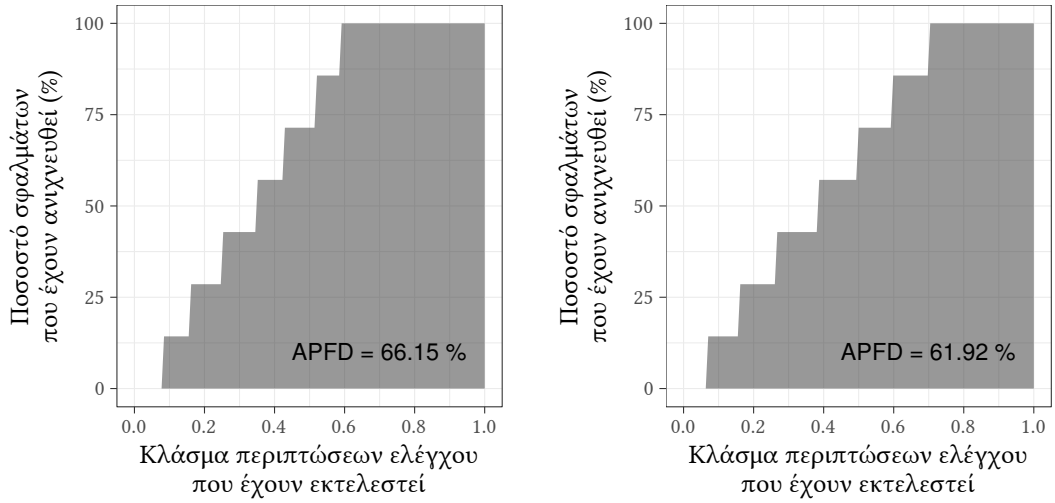
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

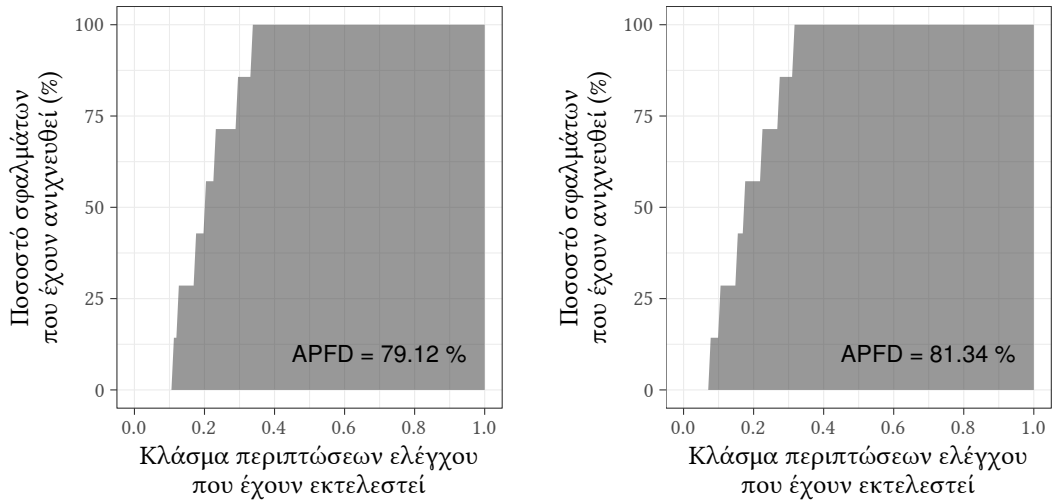
(δ) ΤΠ4: Inverse

Σχήμα 62: Υπολογισμοί APFD για το MoinMoin 1.9.2 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

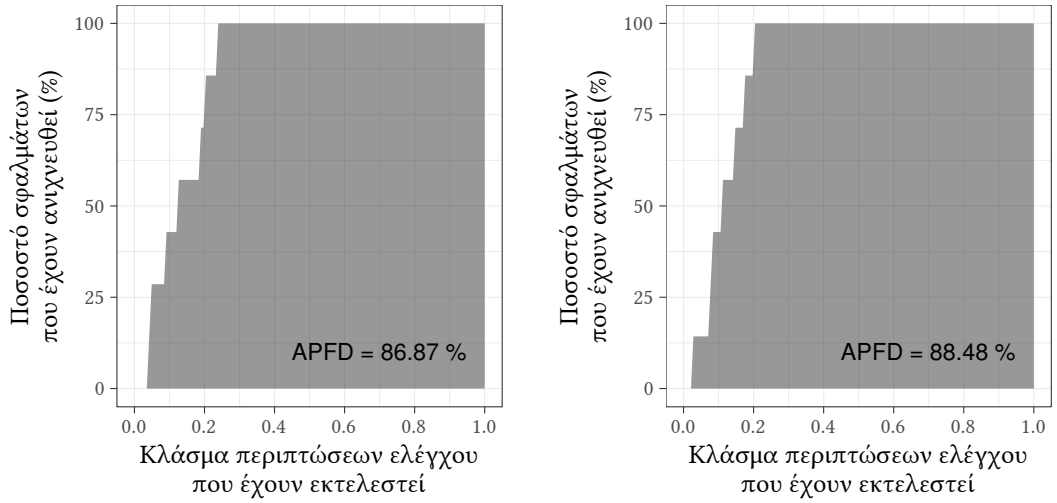
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

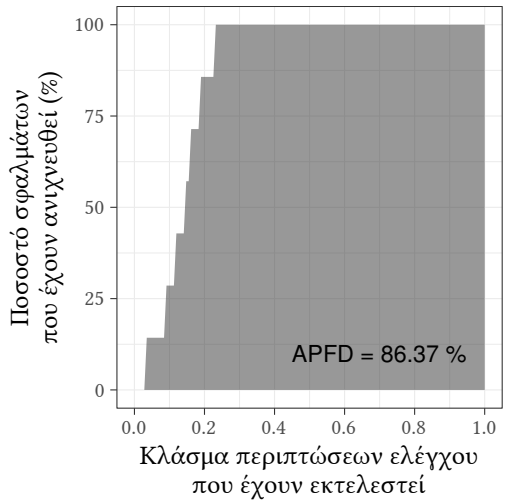
(δ) ΤΠ8: Method-addtl

Σχήμα 63: Υπολογισμοί APFD για το MoinMoin 1.9.2 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)

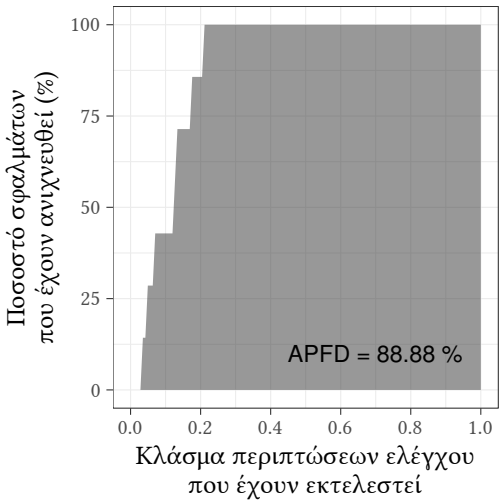


(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

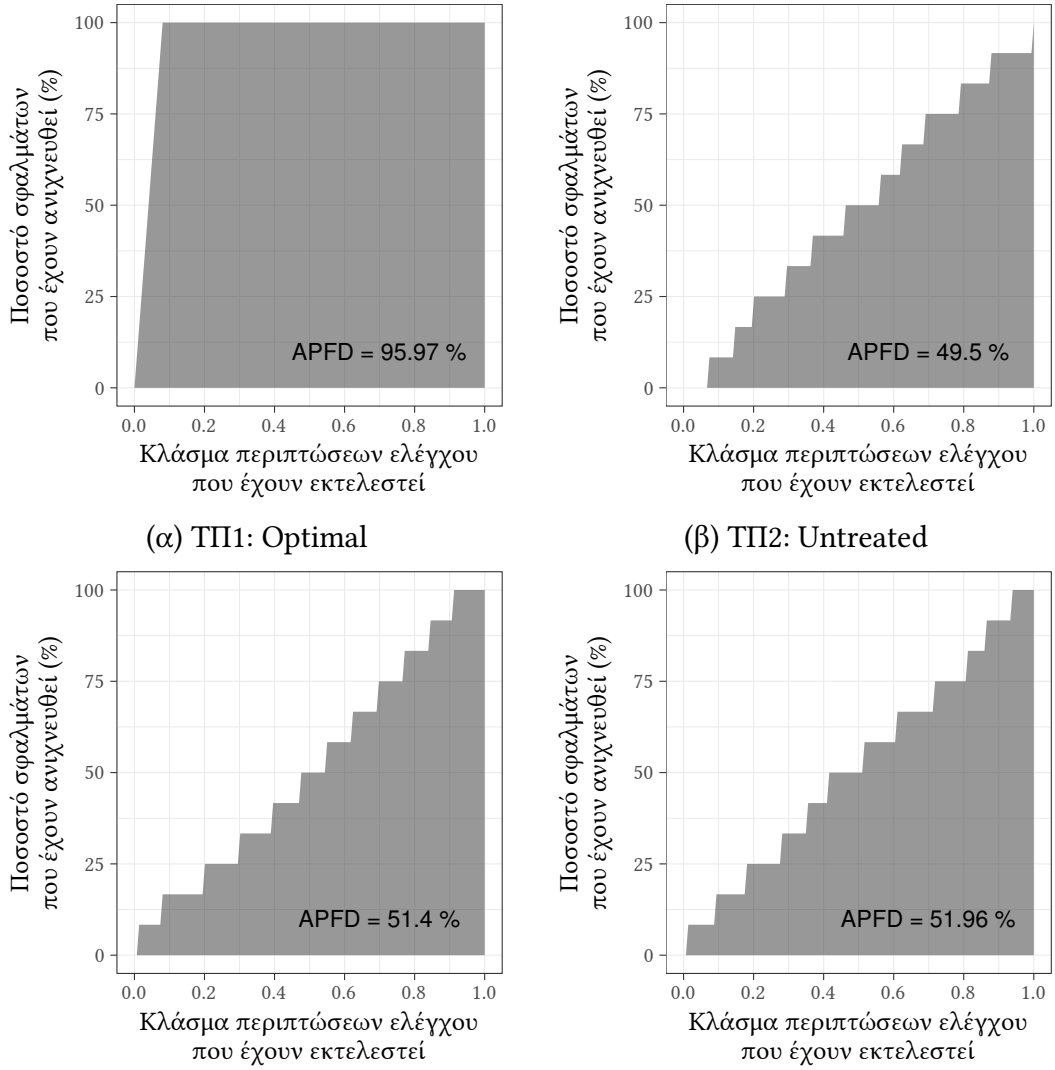


(γ) ΤΠ11: Statement-total

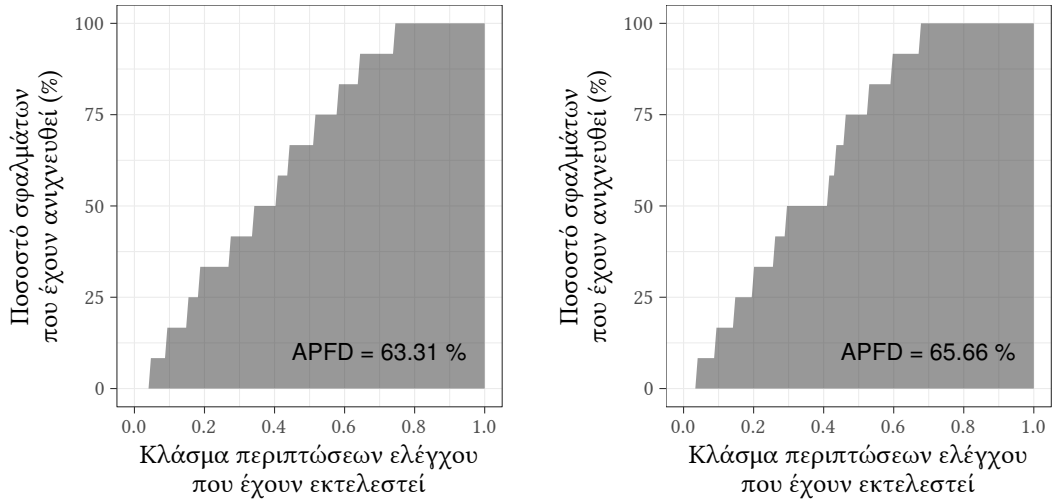


(δ) ΤΠ12: Statement-addtl

Σχήμα 64: Υπολογισμοί APFD για το MoinMoin 1.9.2 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

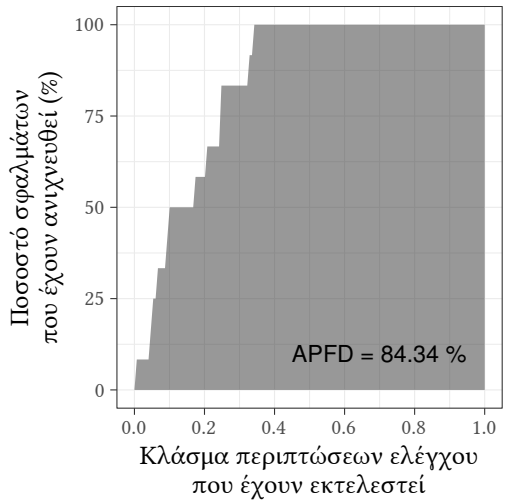


Σχήμα 65: Υπολογισμοί APFD για το MoinMoin 1.9.3 (τεχνικές σύγκρισης)

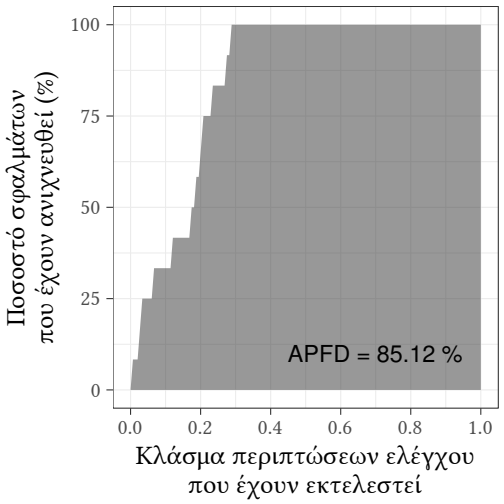


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

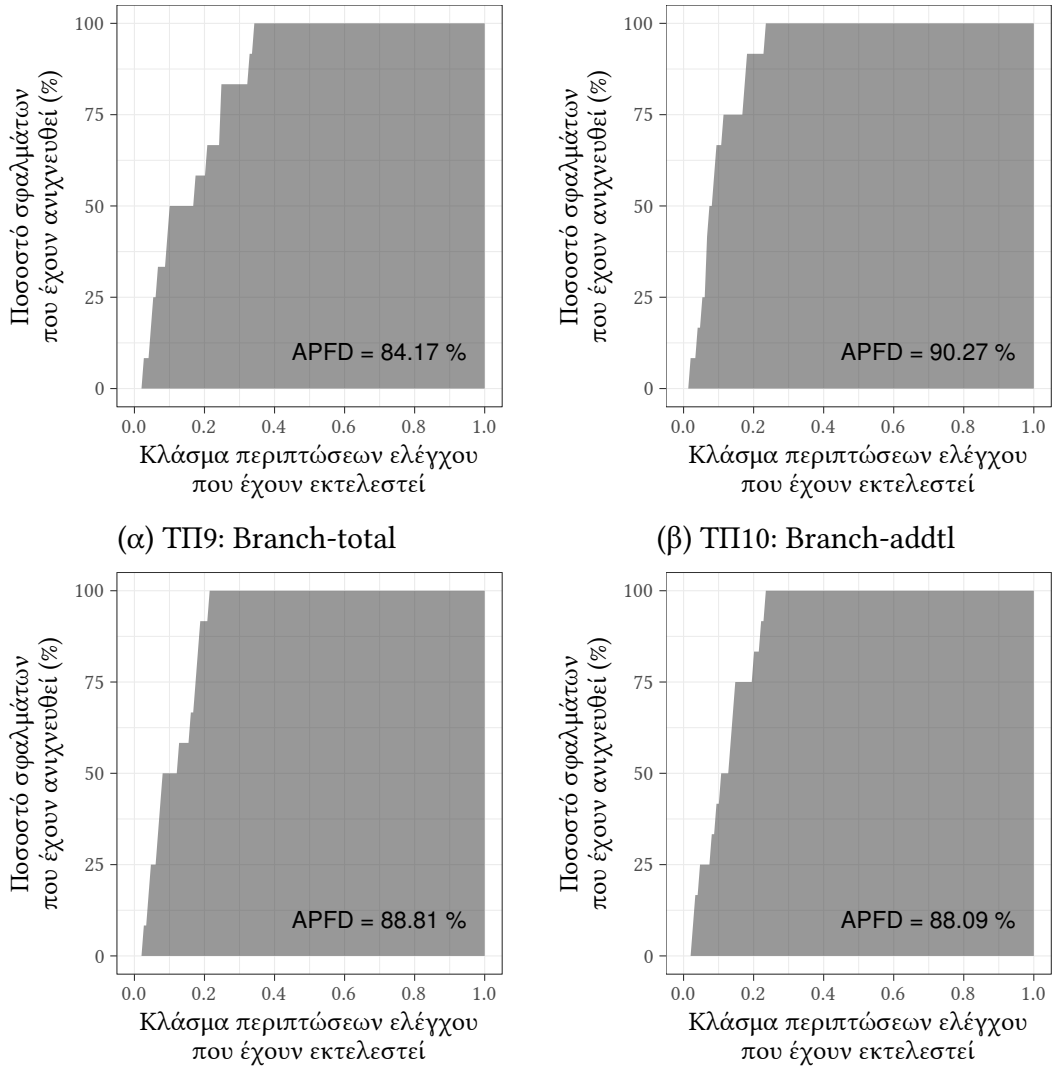


(γ) ΤΠ7: Method-total



(δ) ΤΠ8: Method-addtl

Σχήμα 66: Υπολογισμοί APFD για το MoinMoin 1.9.3 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



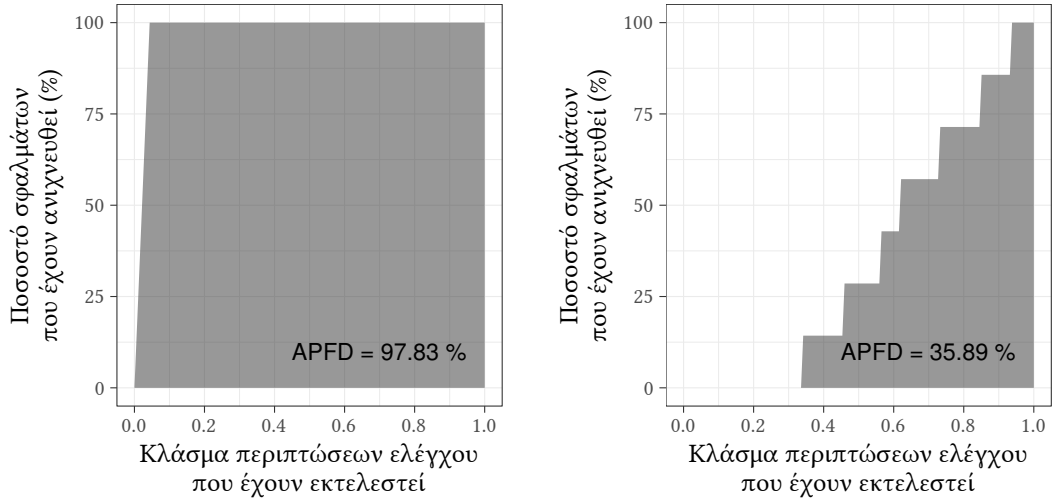
(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

(γ) ΤΠ11: Statement-total

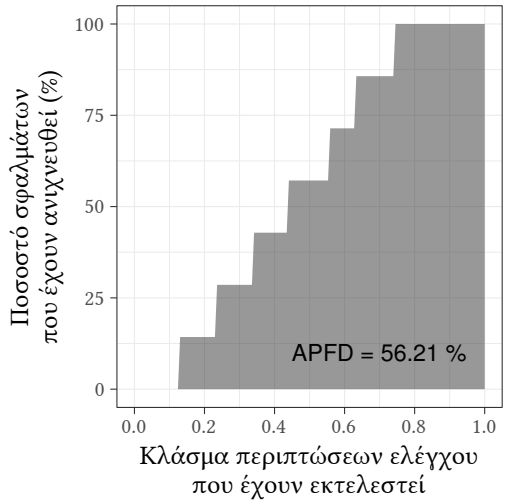
(δ) ΤΠ12: Statement-addtl

Σχήμα 67: Υπολογισμοί APFD για το MoinMoin 1.9.3 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

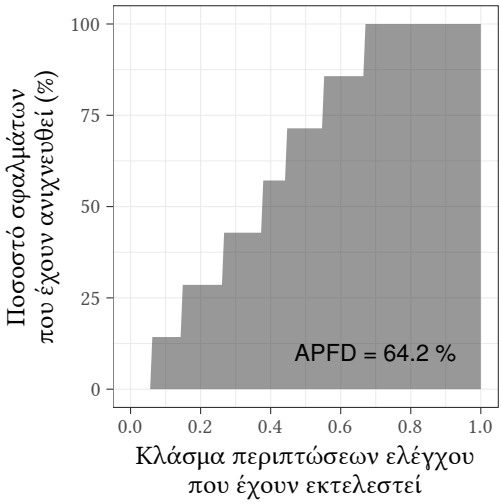


(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

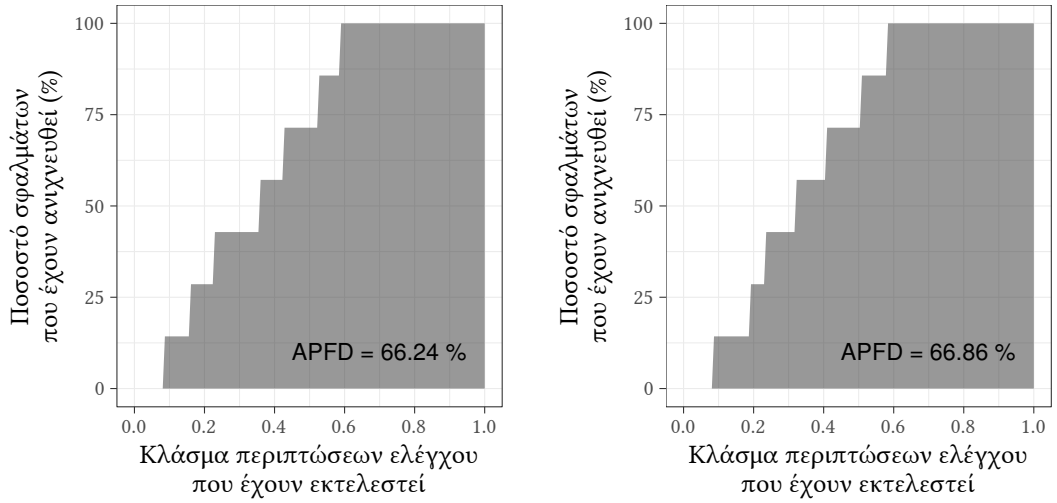


(γ) ΤΠ3: Random



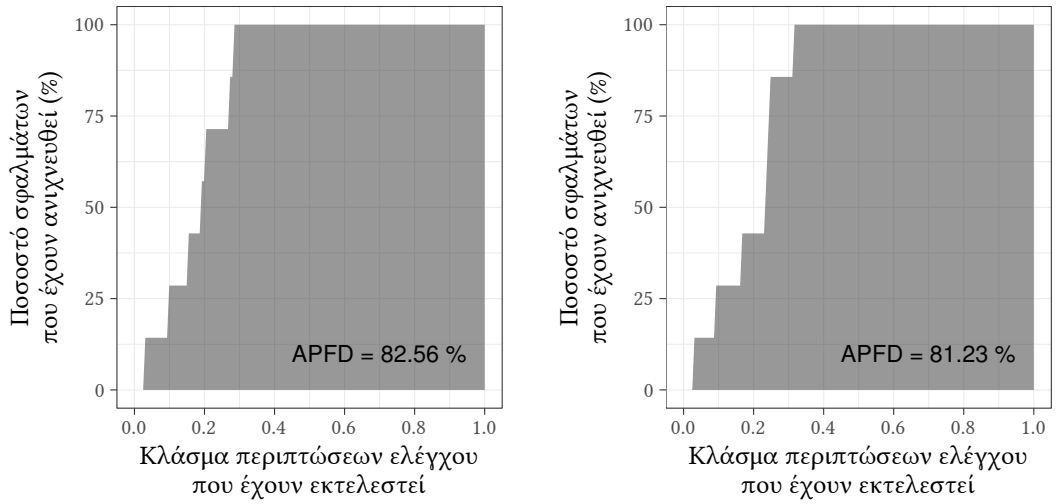
(δ) ΤΠ4: Inverse

Σχήμα 68: Υπολογισμοί APFD για το MoinMoin 1.9.4 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

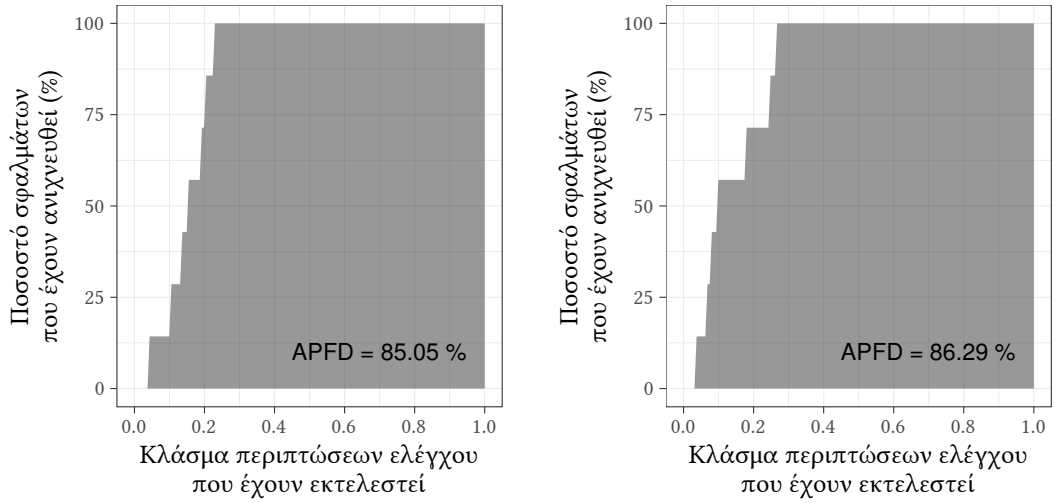
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

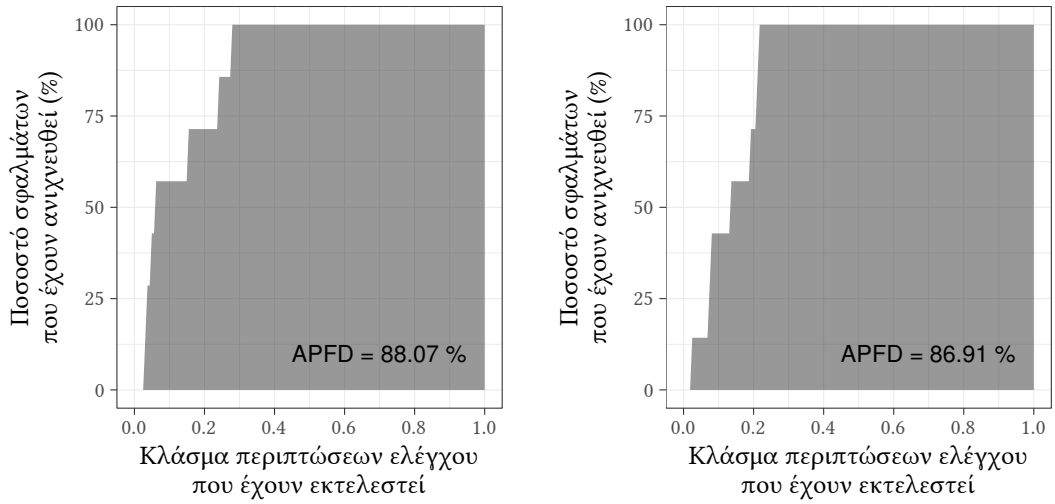
(δ) ΤΠ8: Method-addtl

Σχήμα 69: Υπολογισμοί APFD για το MoinMoin 1.9.4 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

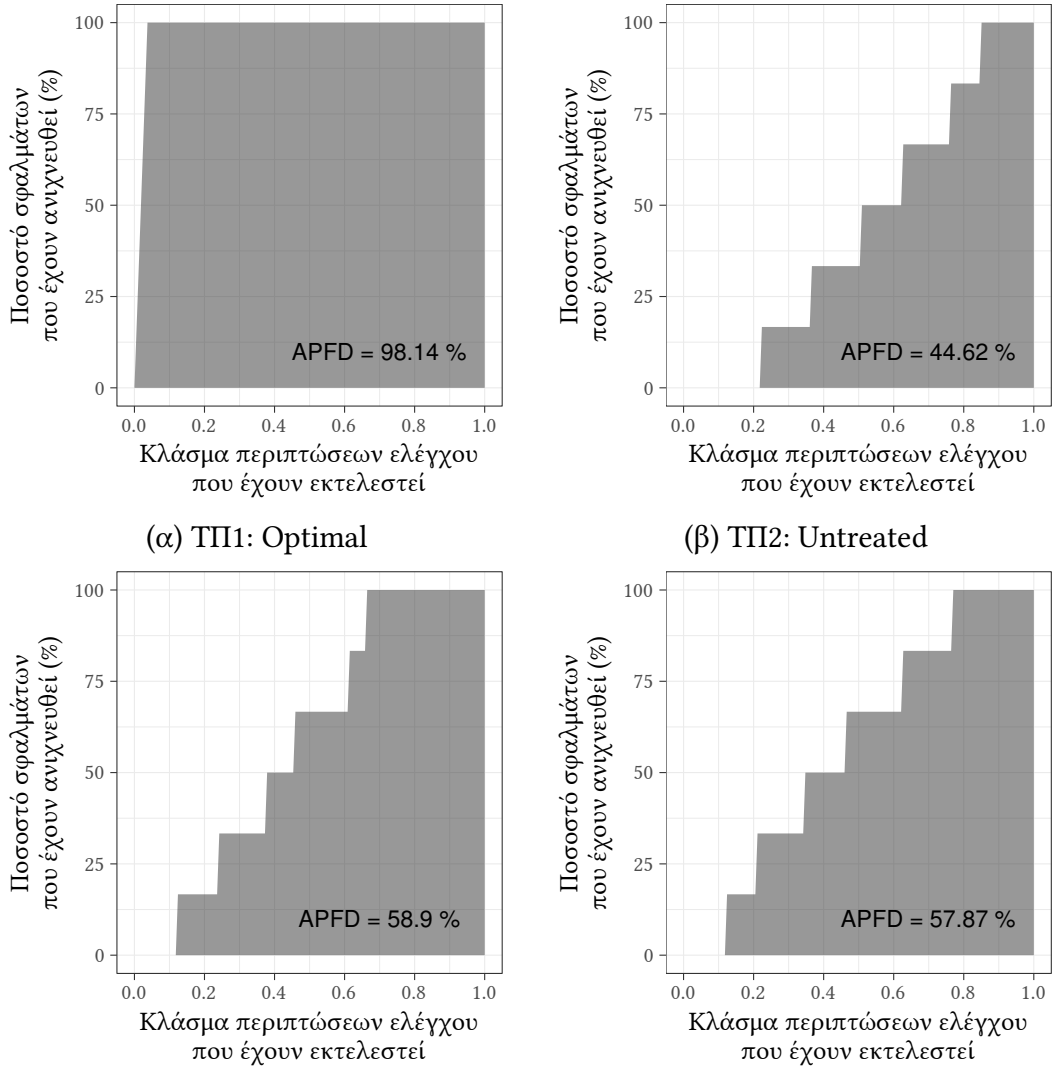
(β) ΤΠ10: Branch-addtl



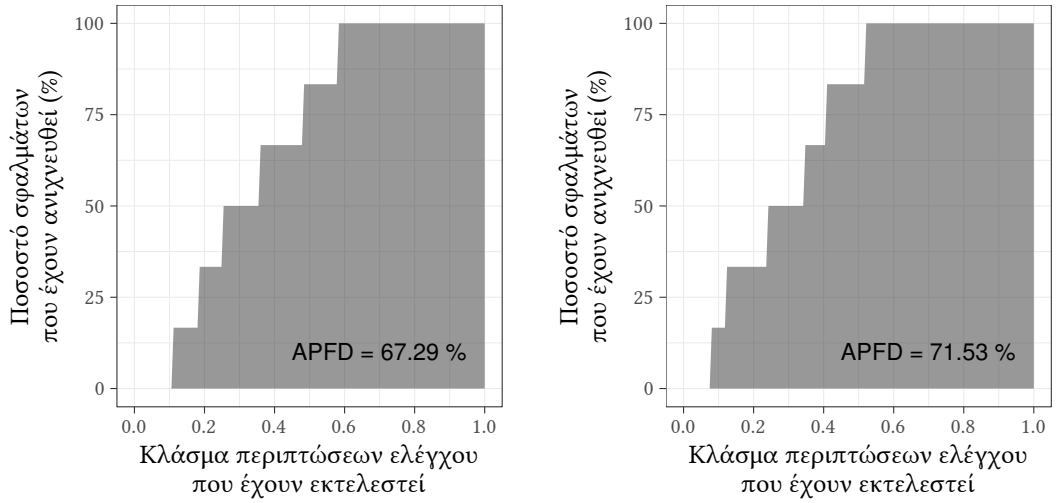
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 70: Υπολογισμοί APFD για το MoinMoin 1.9.4 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

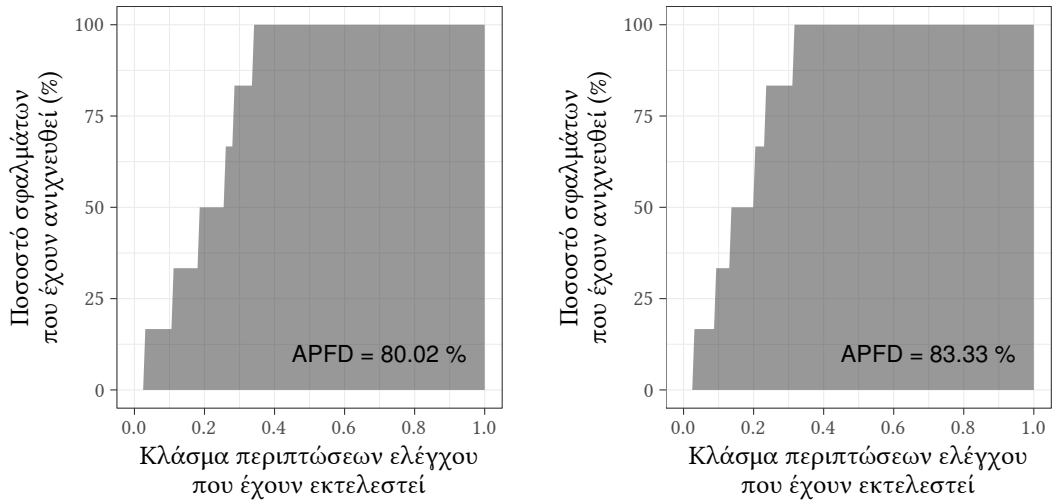


Σχήμα 71: Υπολογισμοί APFD για το MoinMoin 1.9.9 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

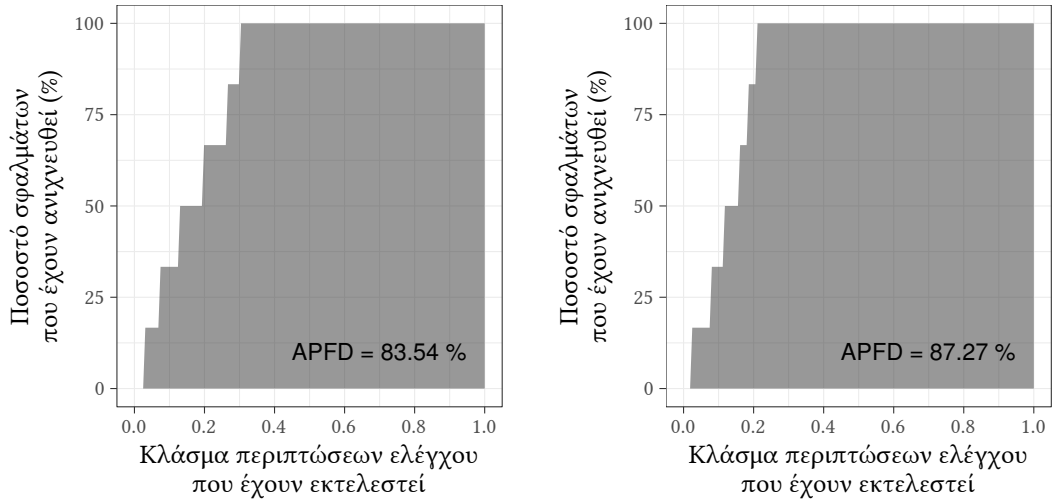
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

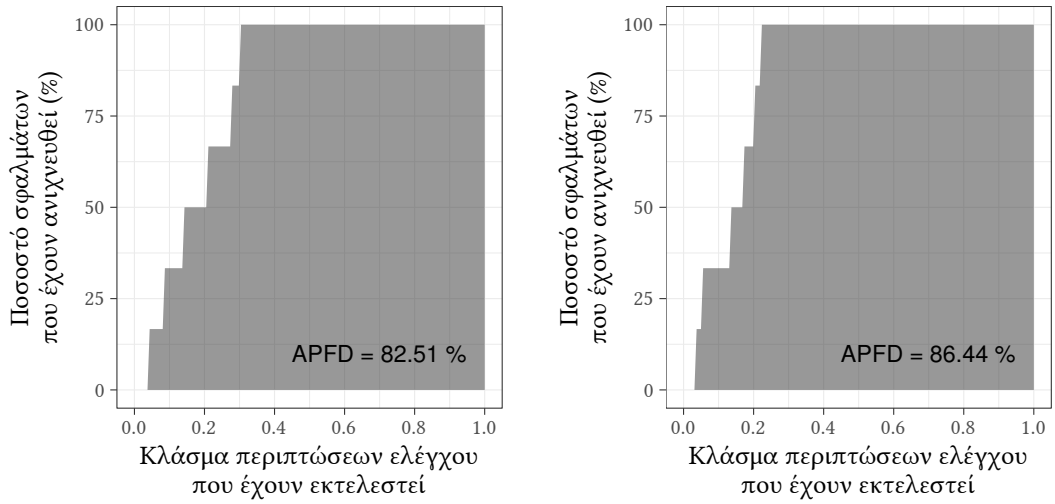
(δ) ΤΠ8: Method-addtl

Σχήμα 72: Υπολογισμοί APFD για το MoinMoin 1.9.9 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

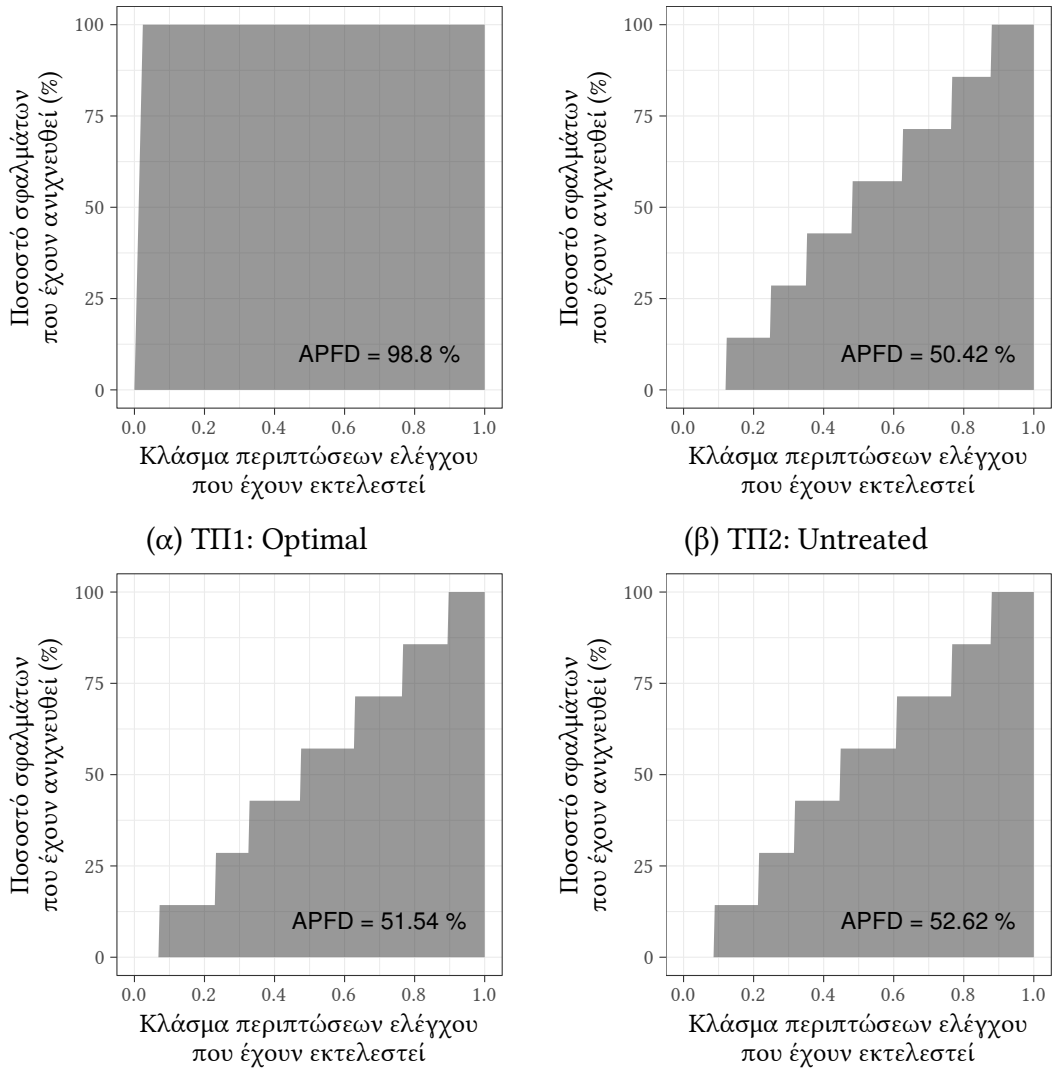


(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 73: Υπολογισμοί APFD για το MoinMoin 1.9.9 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

Μετρήσεις APFD για το Flask



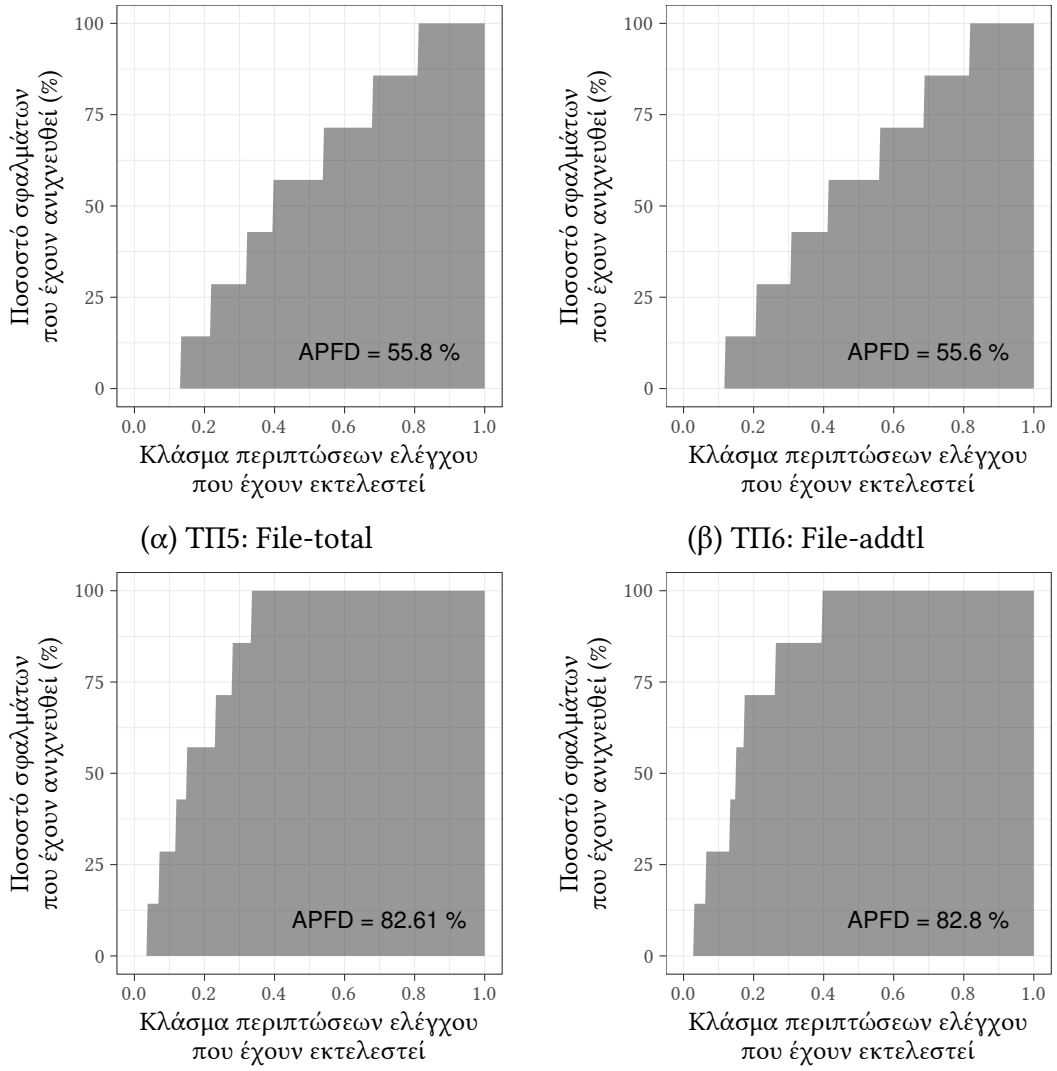
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

(δ) ΤΠ4: Inverse

Σχήμα 74: Υπολογισμοί APFD για το Flask 0.11 (τεχνικές σύγκρισης)



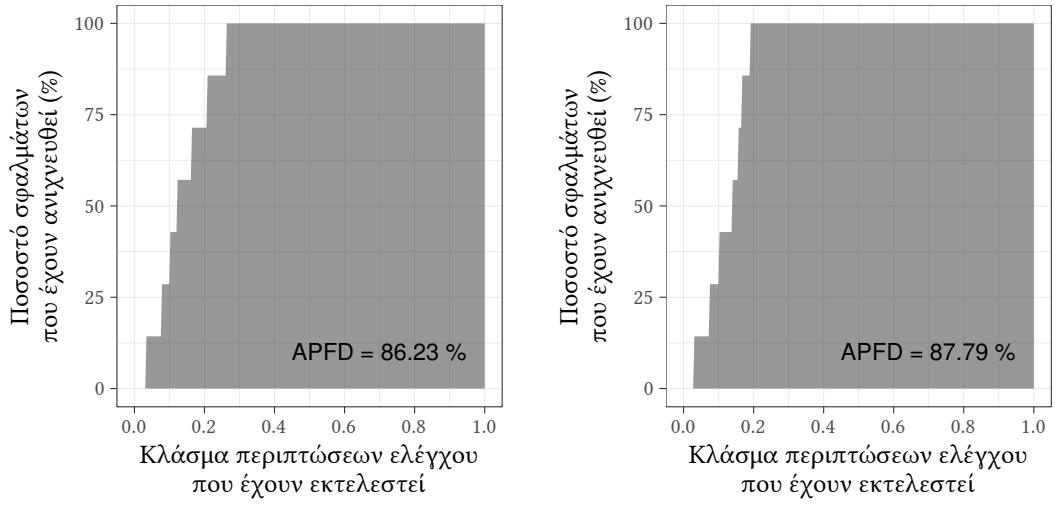
(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

(γ) ΤΠ7: Method-total

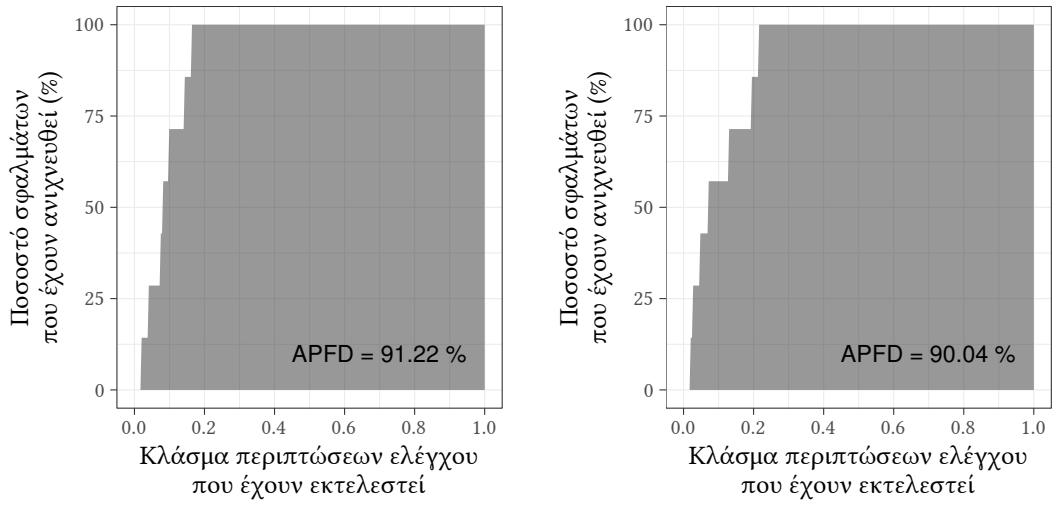
(δ) ΤΠ8: Method-addtl

Σχήμα 75: Υπολογισμοί APFD για το Flask 0.11 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

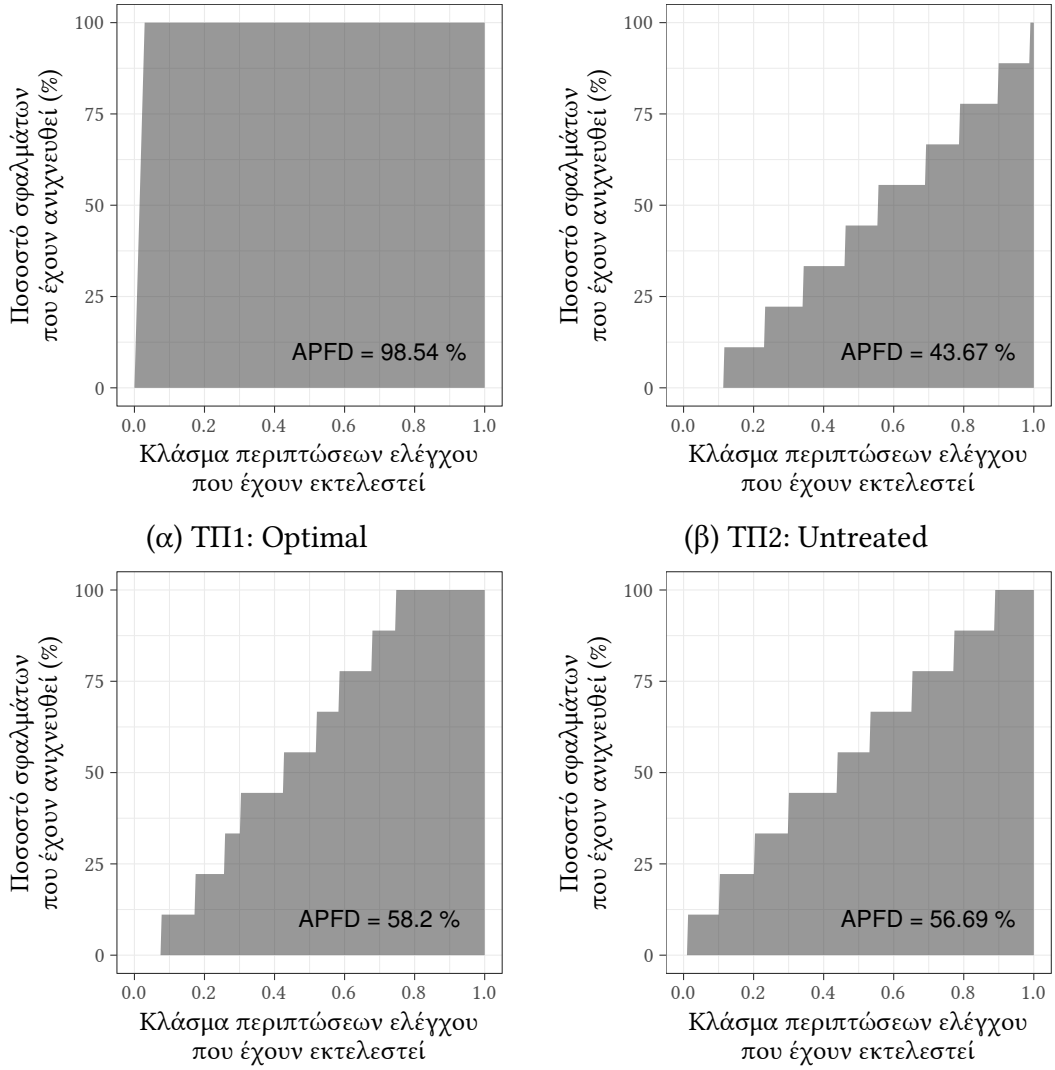
(β) ΤΠ10: Branch-addtl



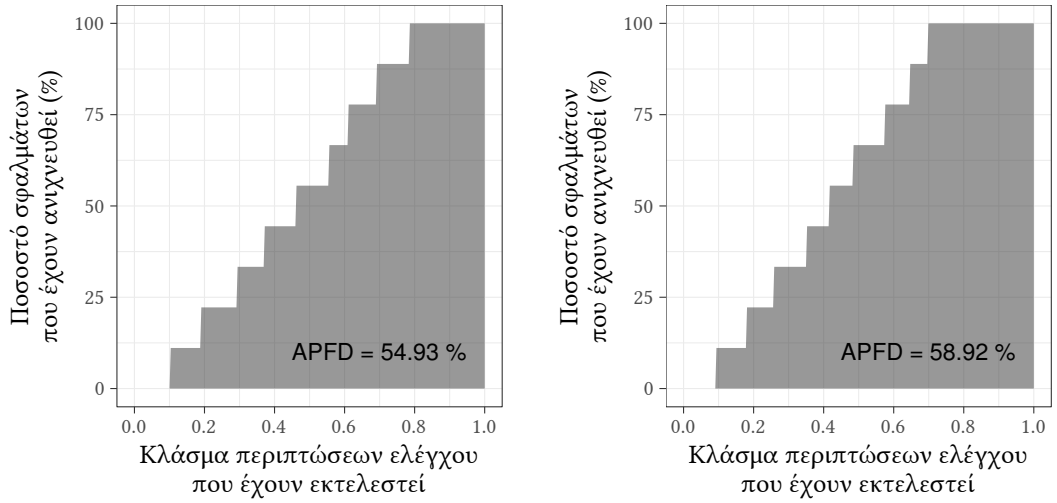
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 76: Υπολογισμοί APFD για το Flask 0.11 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

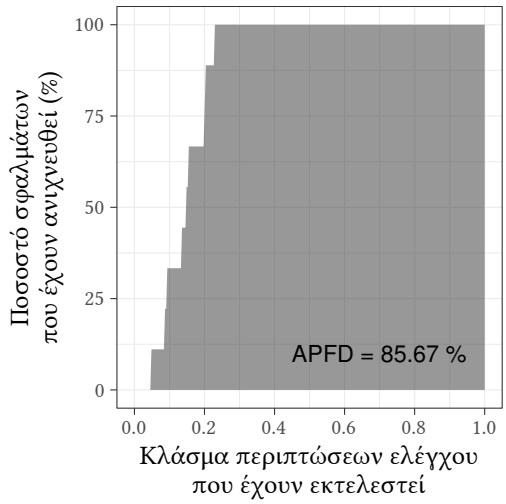


Σχήμα 77: Υπολογισμοί APFD για το Flask 0.12 (τεχνικές σύγκρισης)

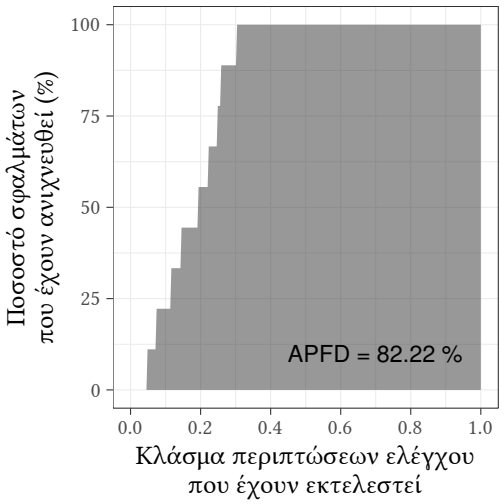


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

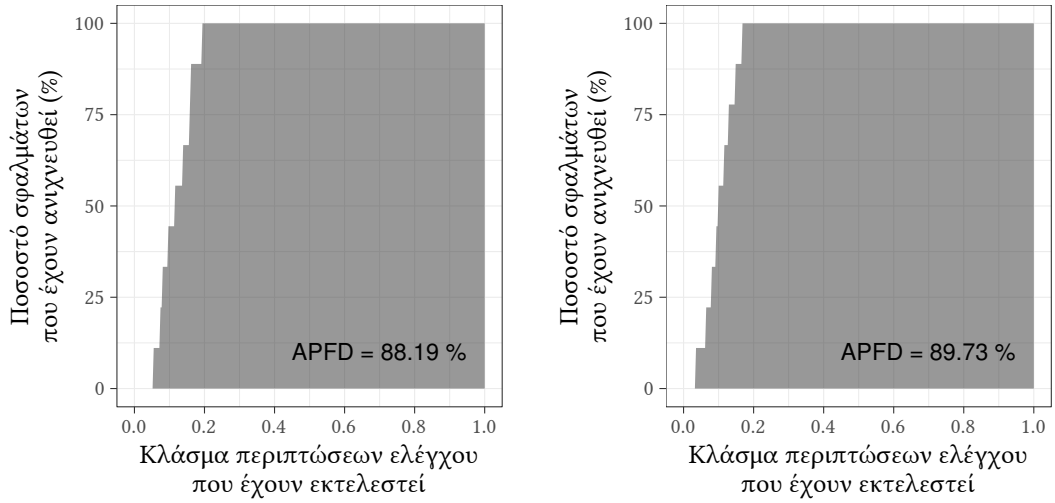


(γ) ΤΠ7: Method-total



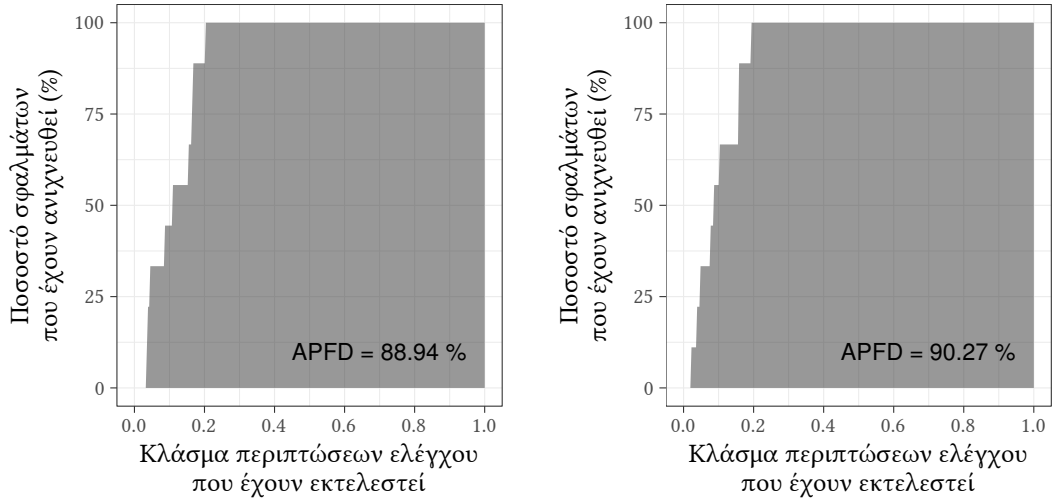
(δ) ΤΠ8: Method-addtl

Σχήμα 78: Υπολογισμοί APFD για το Flask 0.12 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

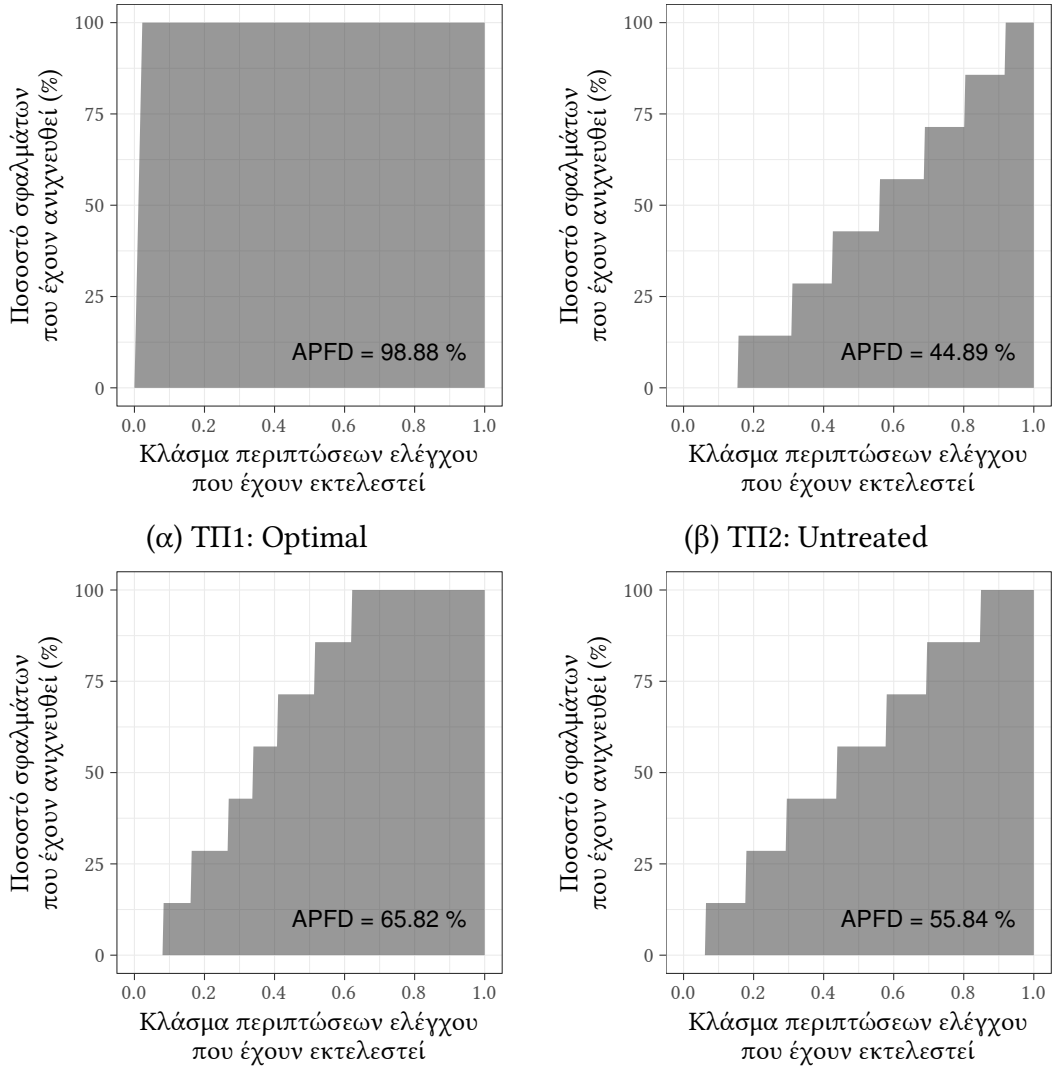
(β) ΤΠ10: Branch-addtl



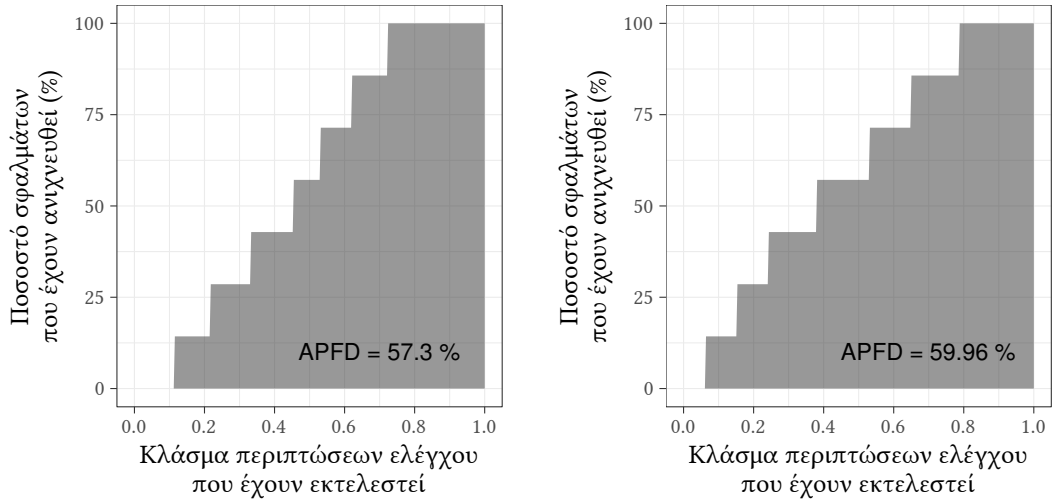
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 79: Υπολογισμοί APFD για το Flask 0.12 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

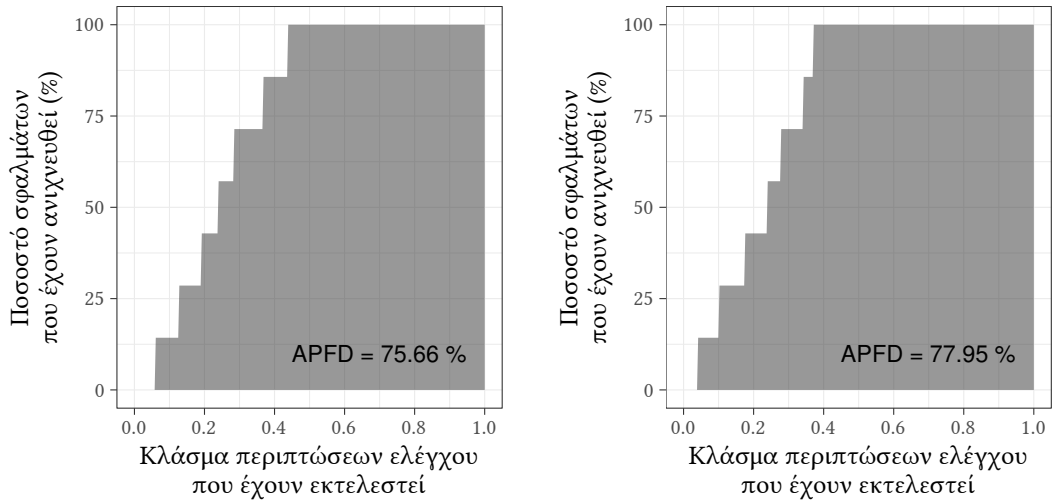


Σχήμα 80: Υπολογισμοί APFD για το Flask 0.12.3 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

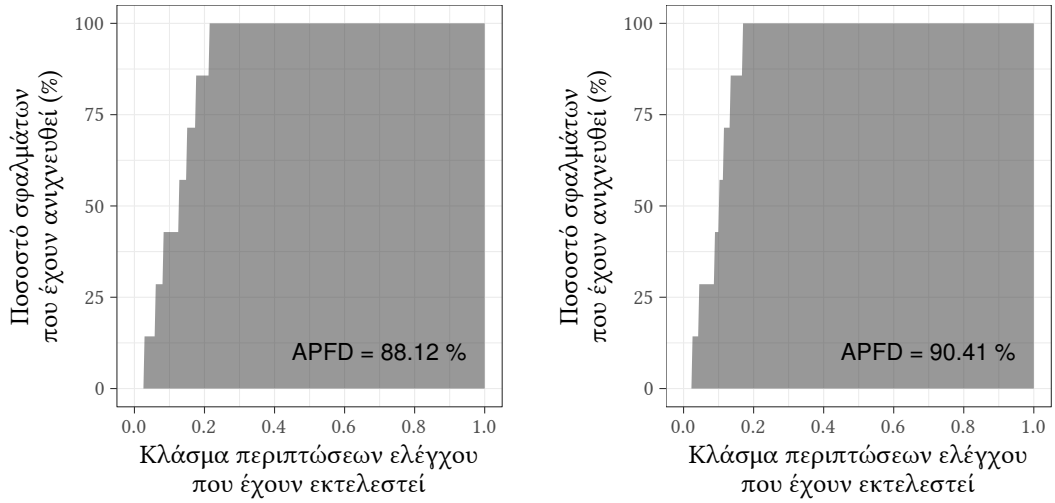
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

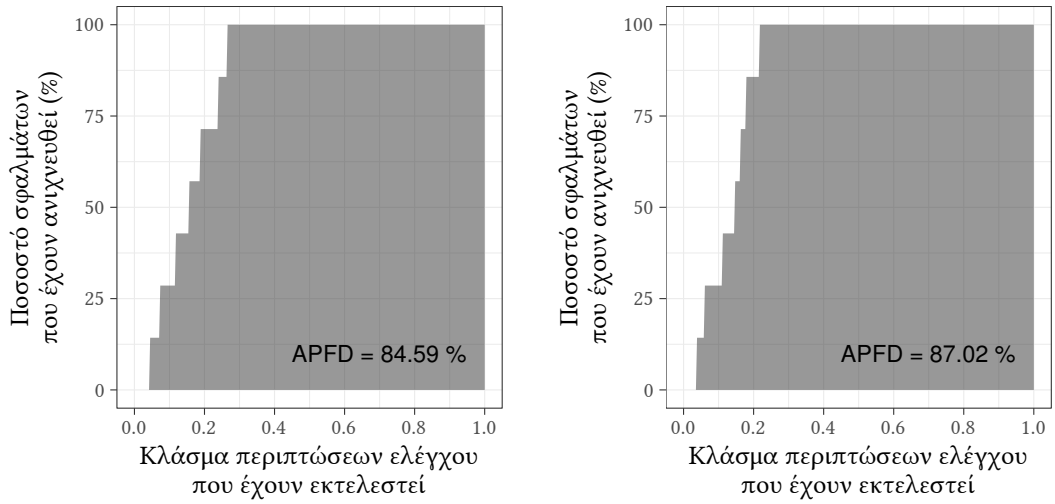
(δ) ΤΠ8: Method-addtl

Σχήμα 81: Υπολογισμοί APFD για το Flask 0.12.3 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

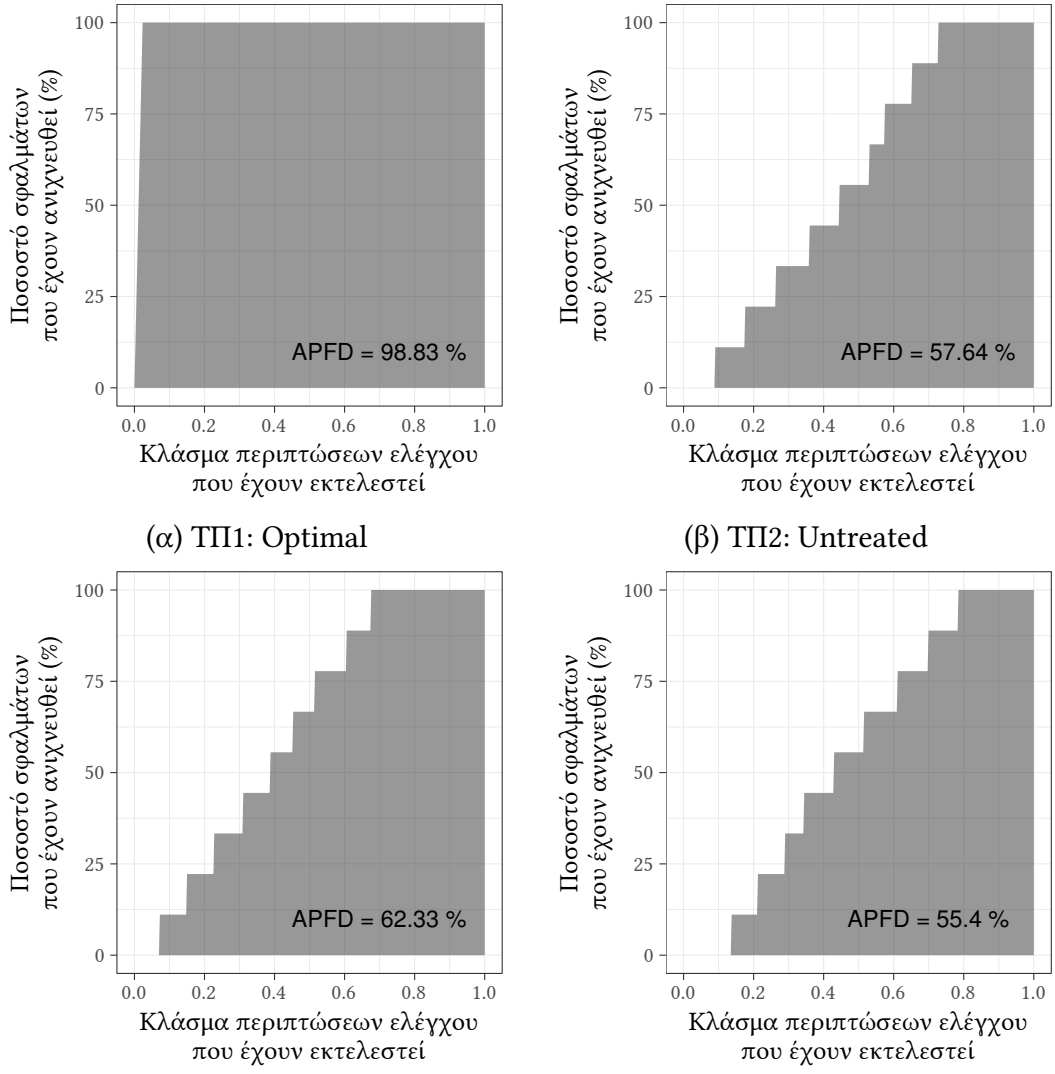
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 82: Υπολογισμοί APFD για το Flask 0.12.3 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)



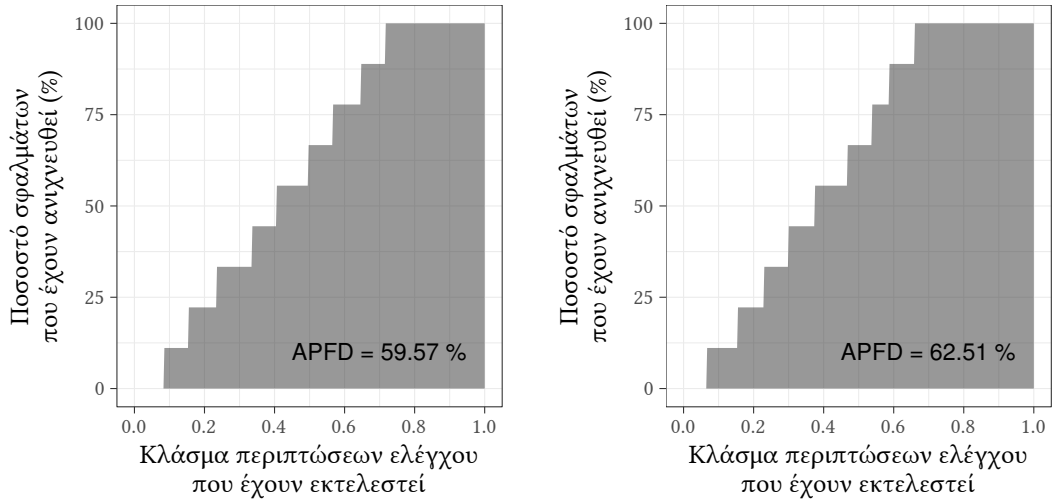
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

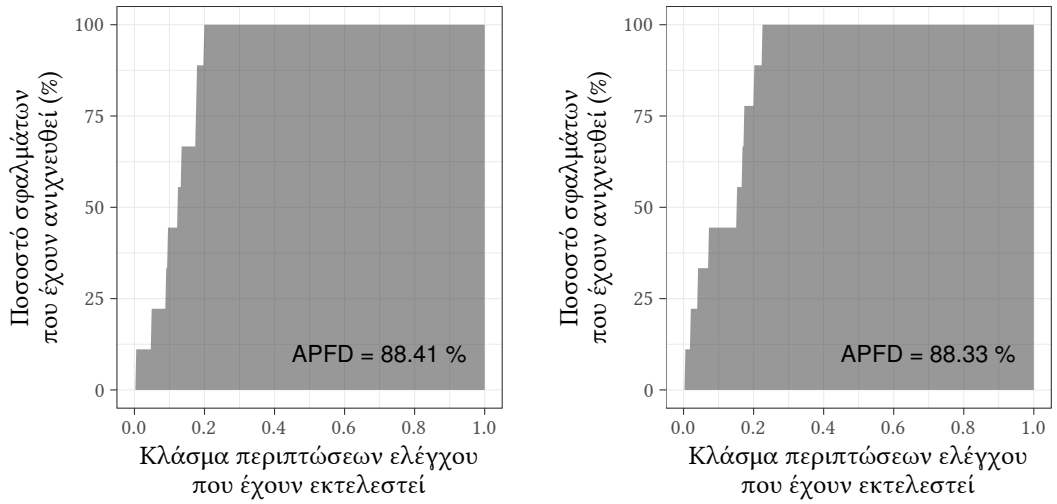
(δ) ΤΠ4: Inverse

Σχήμα 83: Υπολογισμοί APFD για το Flask 1.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

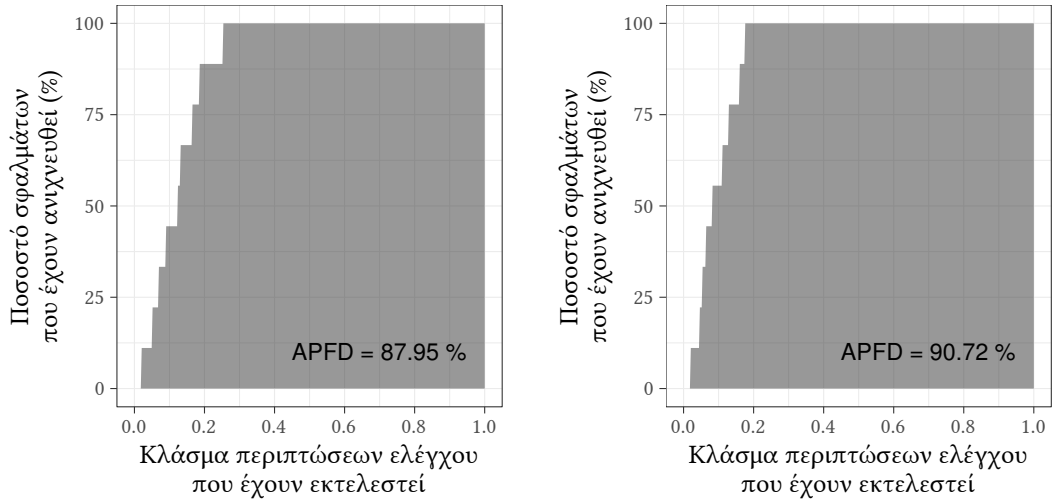
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

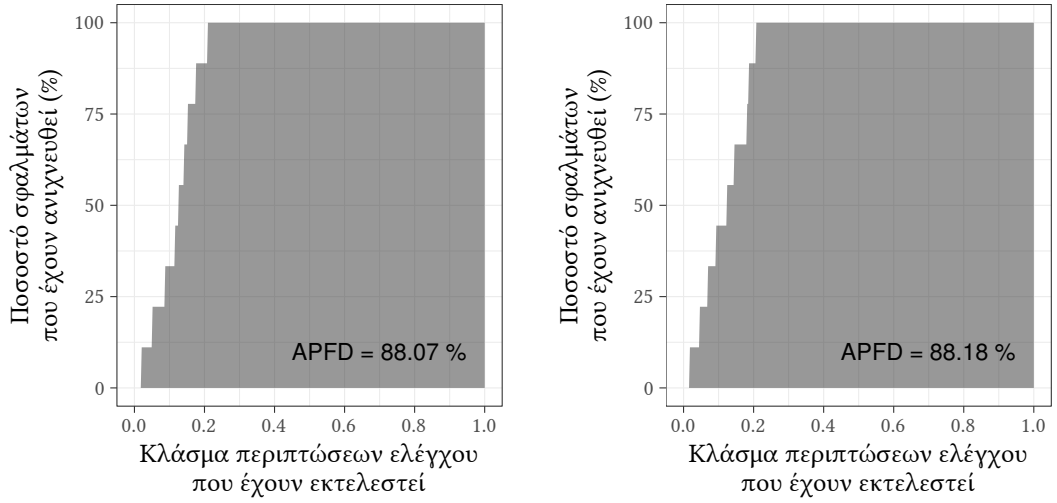
(δ) ΤΠ8: Method-addtl

Σχήμα 84: Υπολογισμοί APFD για το Flask 1.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

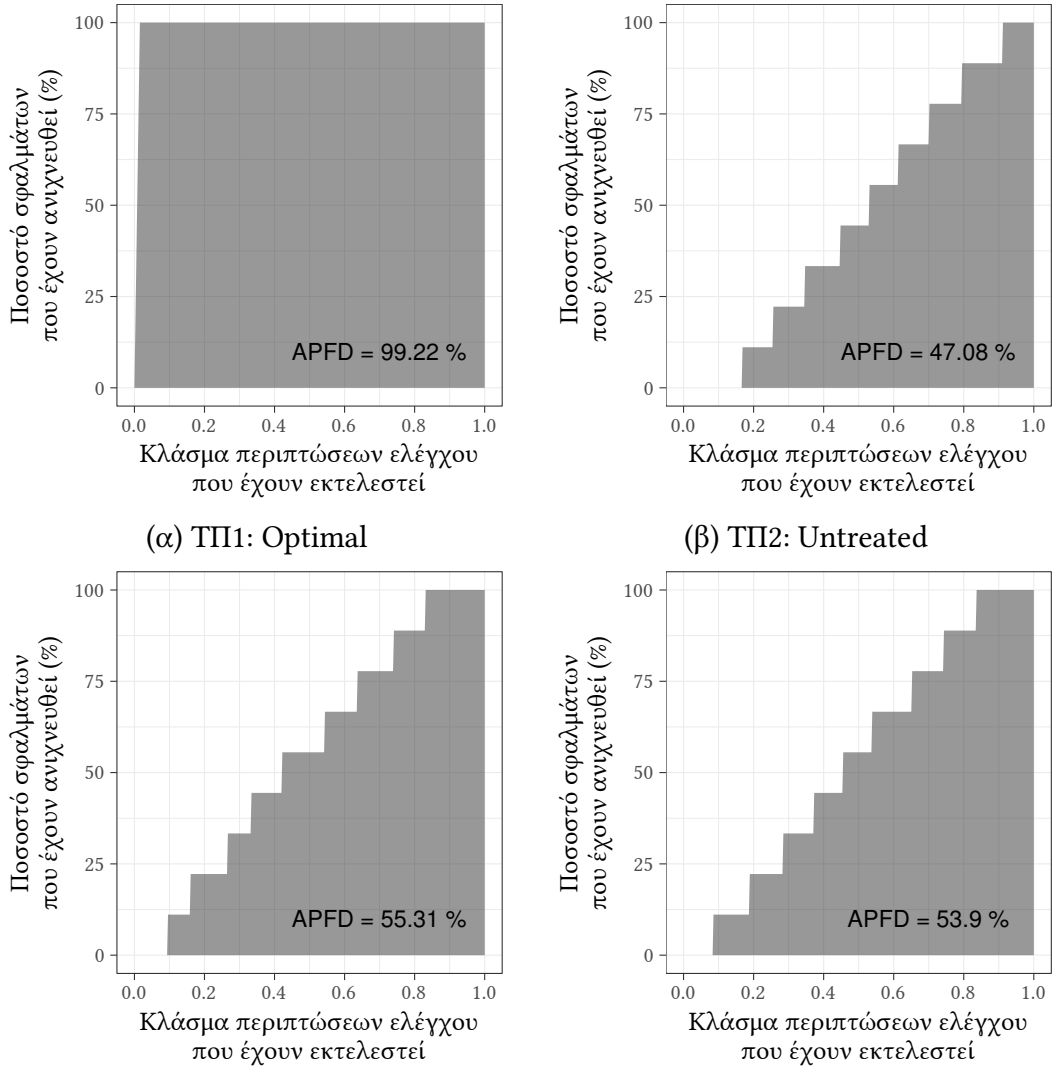
(β) ΤΠ10: Branch-addtl



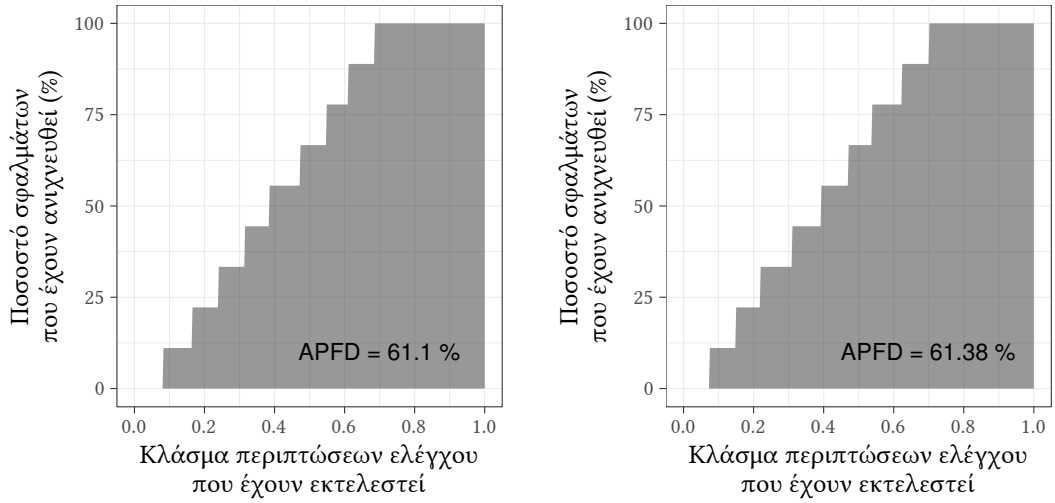
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 85: Υπολογισμοί APFD για το Flask 1.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

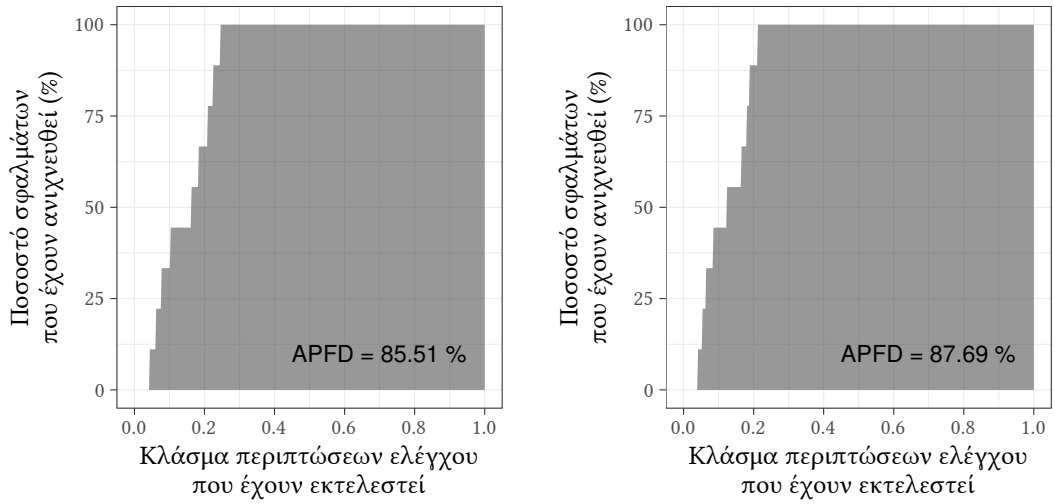


Σχήμα 86: Υπολογισμοί APFD για το Flask 1.0.2 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

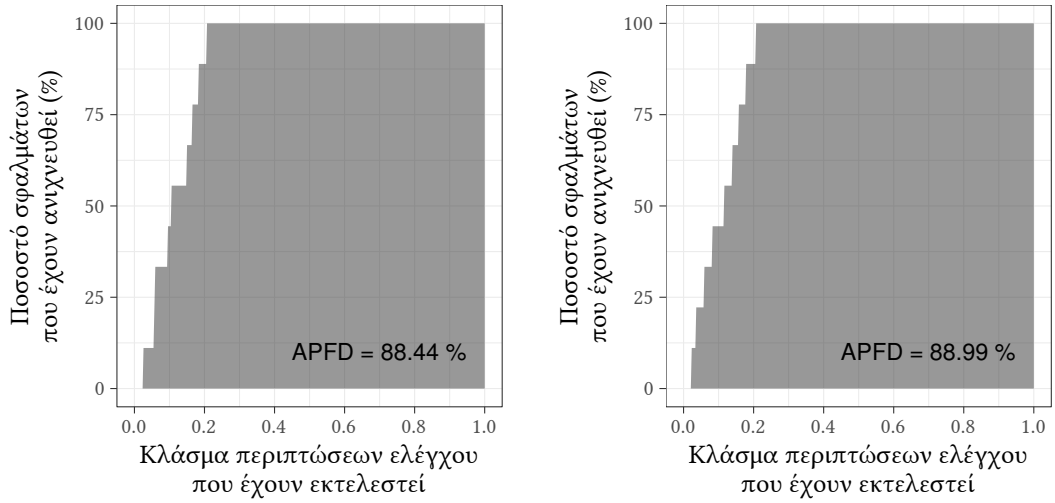
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

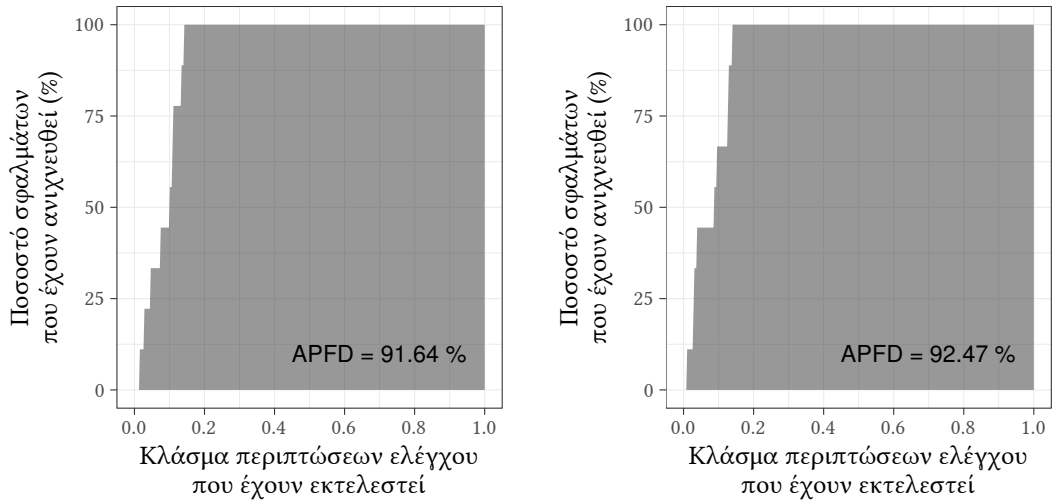
(δ) ΤΠ8: Method-addtl

Σχήμα 87: Υπολογισμοί APFD για το Flask 1.0.2 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

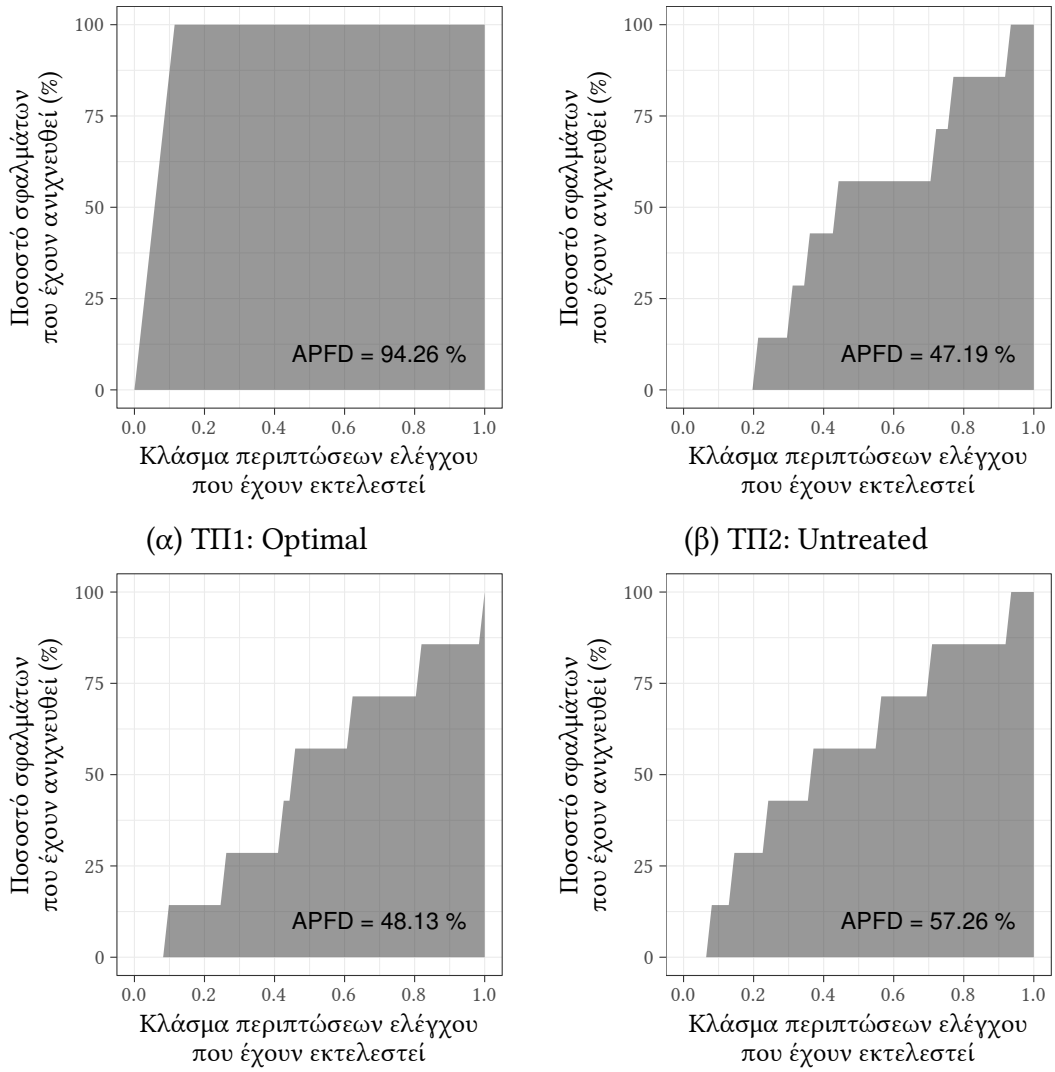


(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 88: Υπολογισμοί APFD για το Flask 1.0.2 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

Μετρήσεις APFD για το Six



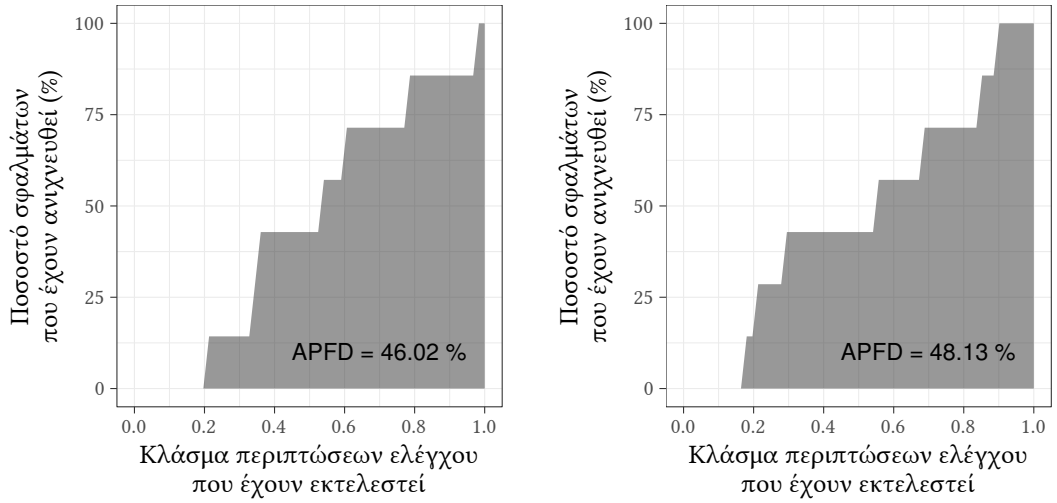
(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

(γ) ΤΠ3: Random

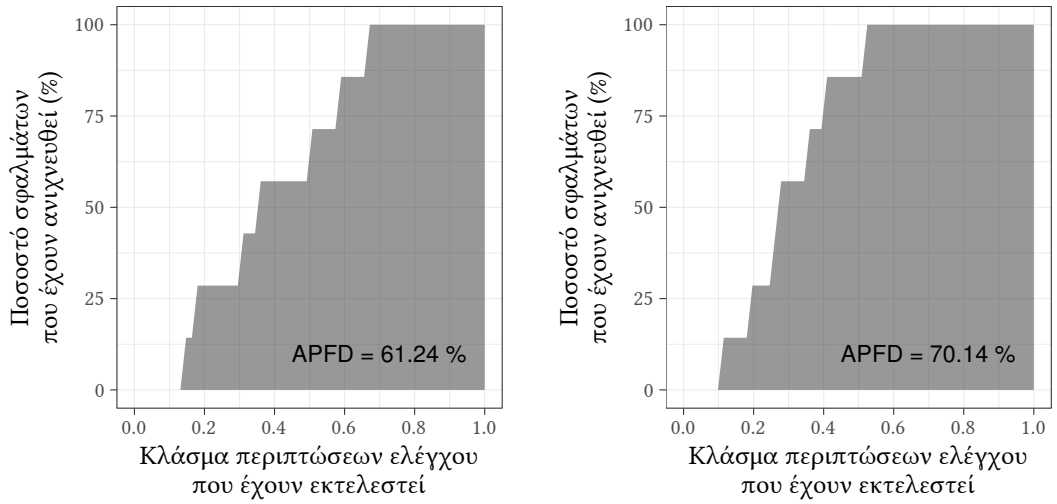
(δ) ΤΠ4: Inverse

Σχήμα 89: Υπολογισμοί APFD για το Six 1.8.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

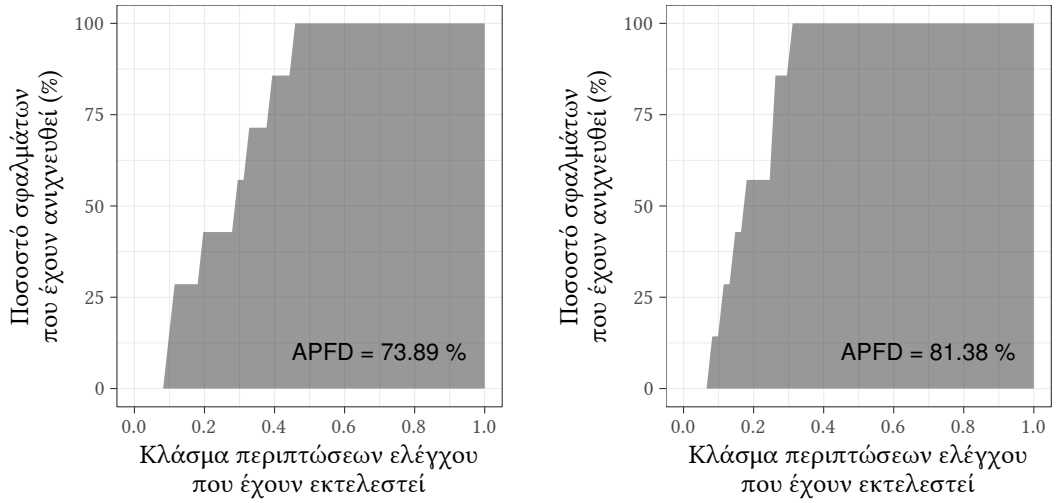
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

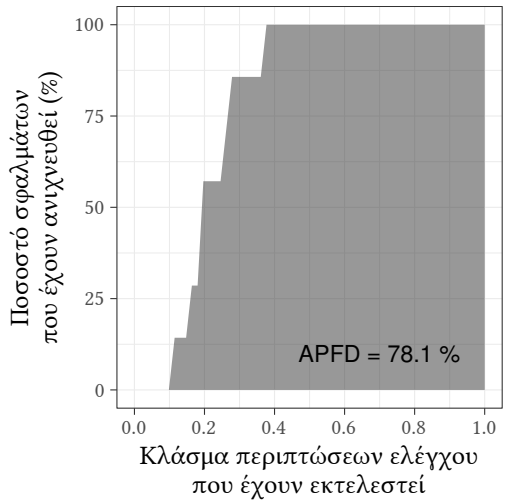
(δ) ΤΠ8: Method-addtl

Σχήμα 90: Υπολογισμοί APFD για το Six 1.8.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)

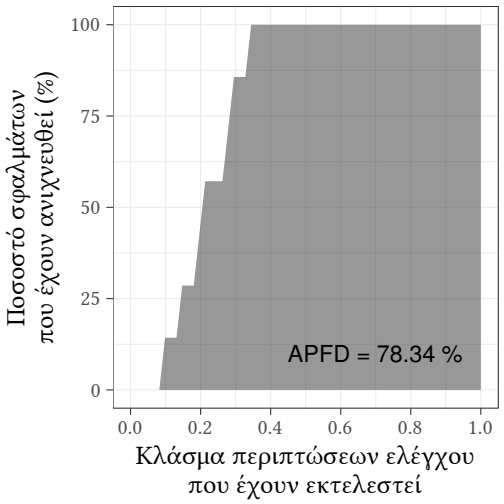


(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl

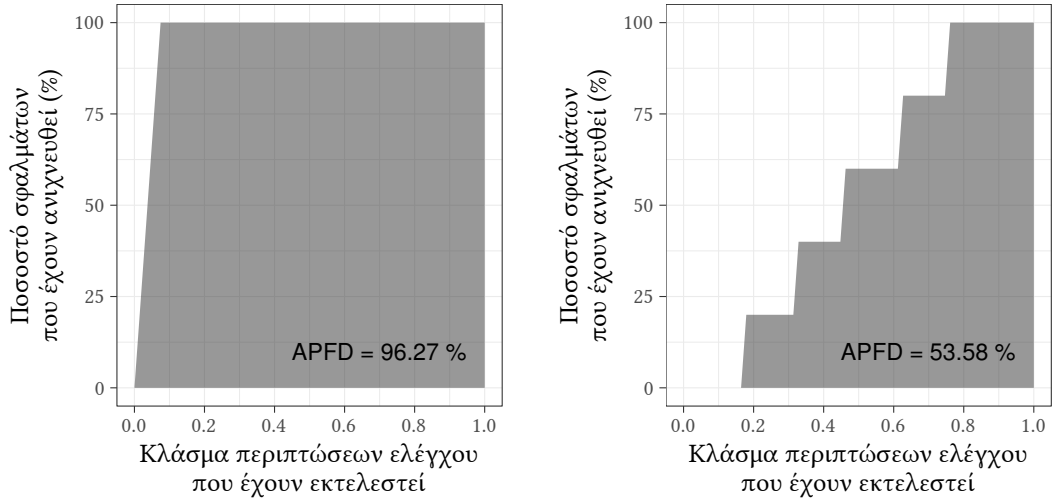


(γ) ΤΠ11: Statement-total



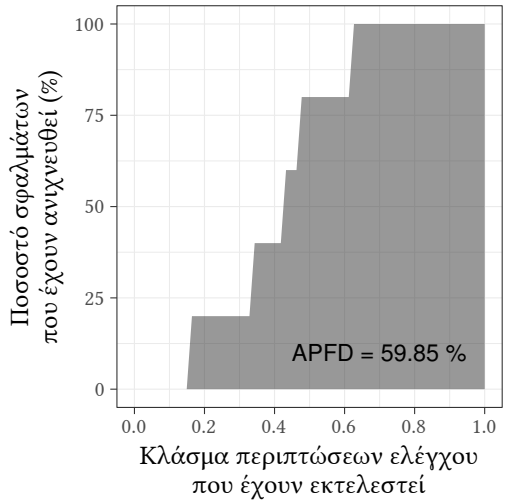
(δ) ΤΠ12: Statement-addtl

Σχήμα 91: Υπολογισμοί APFD για το Six 1.8.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

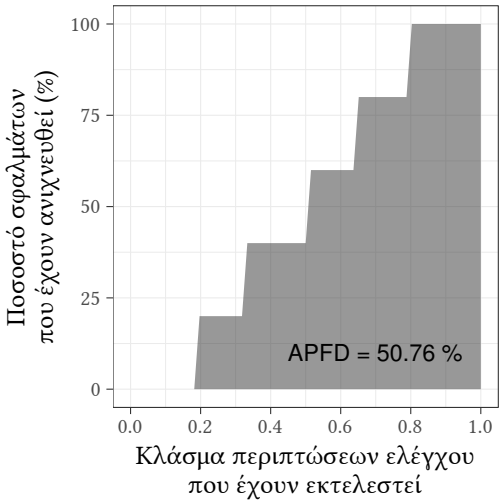


(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

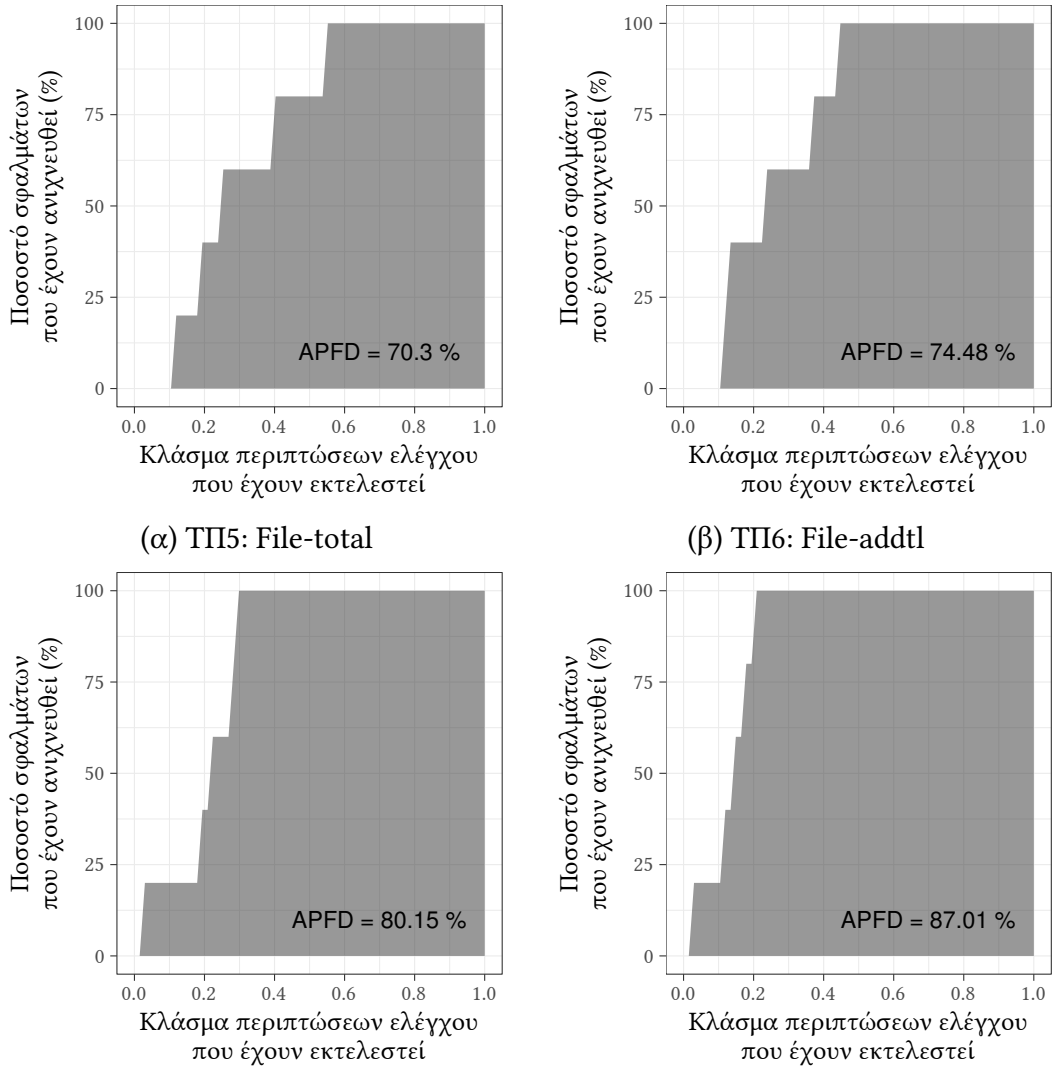


(γ) ΤΠ3: Random



(δ) ΤΠ4: Inverse

Σχήμα 92: Υπολογισμοί APFD για το Six 1.9.0 (τεχνικές σύγκρισης)



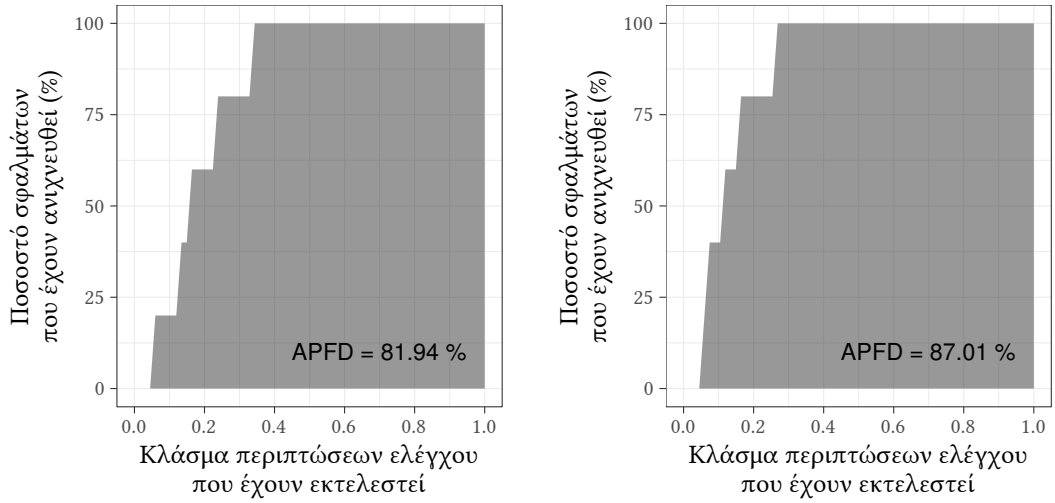
(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

(γ) ΤΠ7: Method-total

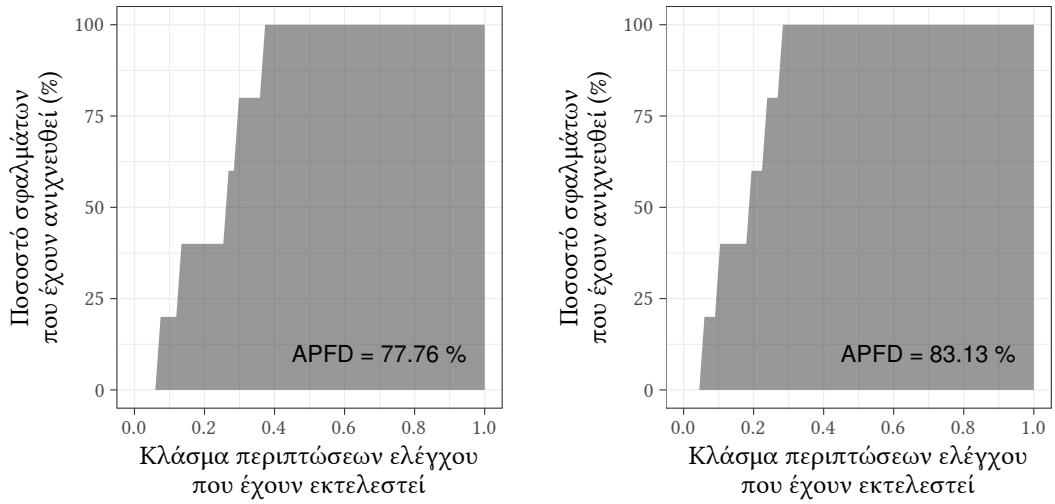
(δ) ΤΠ8: Method-addtl

Σχήμα 93: Υπολογισμοί APFD για το Six 1.9.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

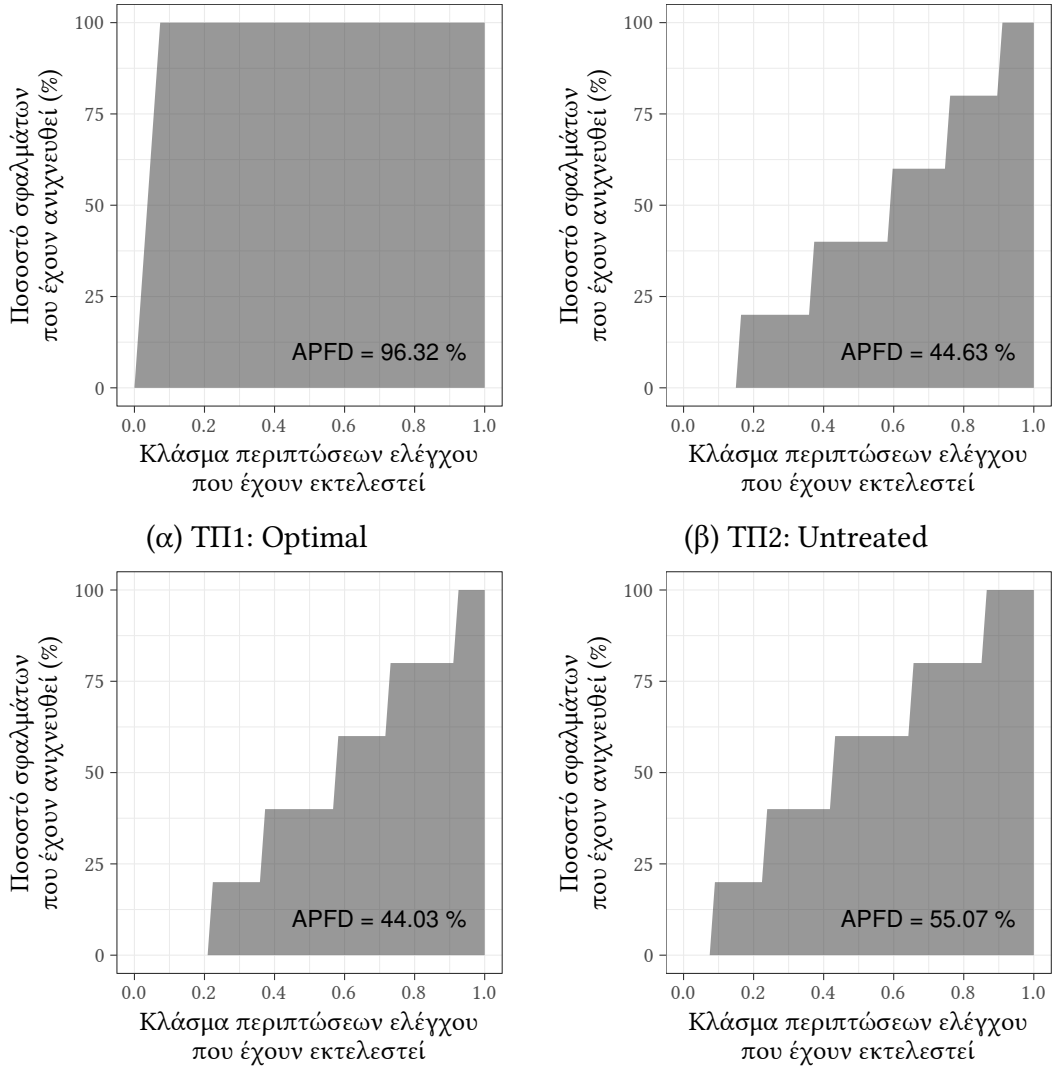
(β) ΤΠ10: Branch-addtl



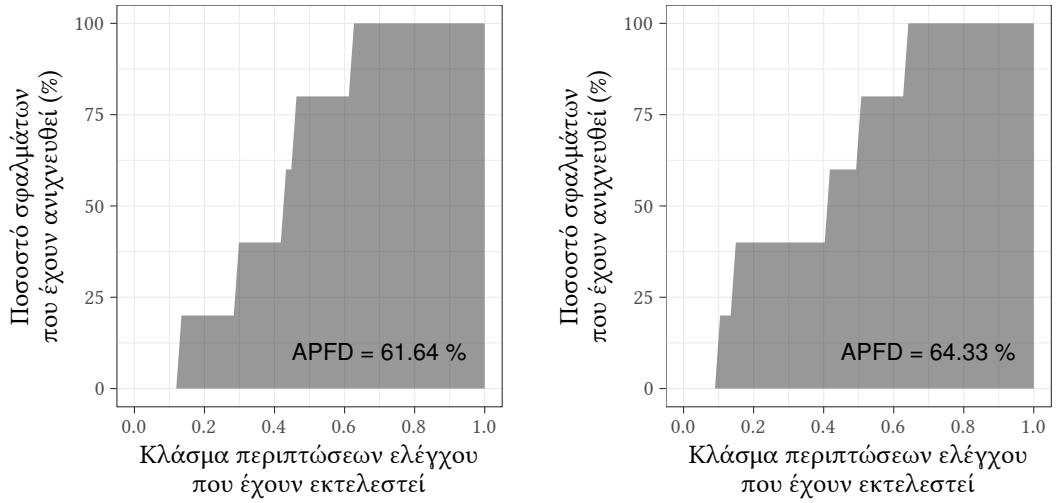
(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 94: Υπολογισμοί APFD για το Six 1.9.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

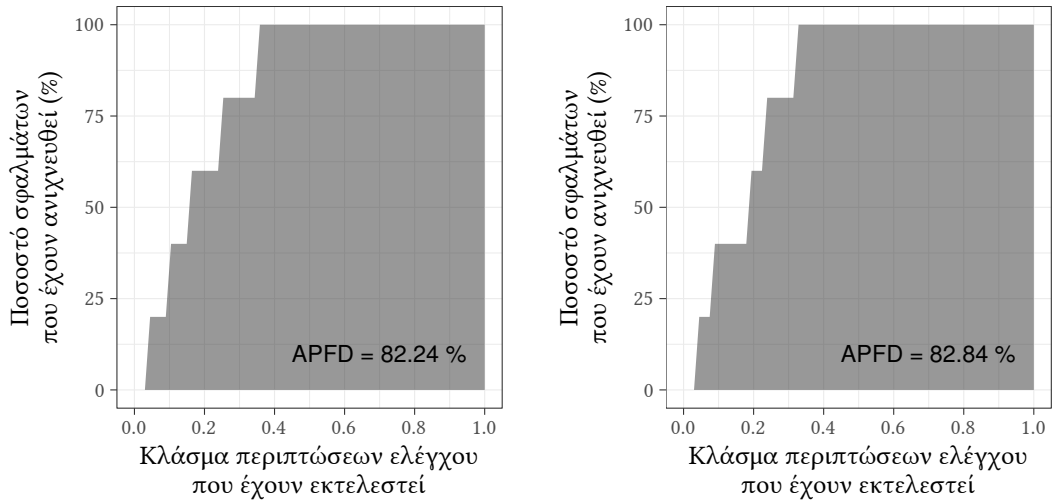


Σχήμα 95: Υπολογισμοί APFD για το Six 1.10.0 (τεχνικές σύγκρισης)



(α) ΤΠ5: File-total

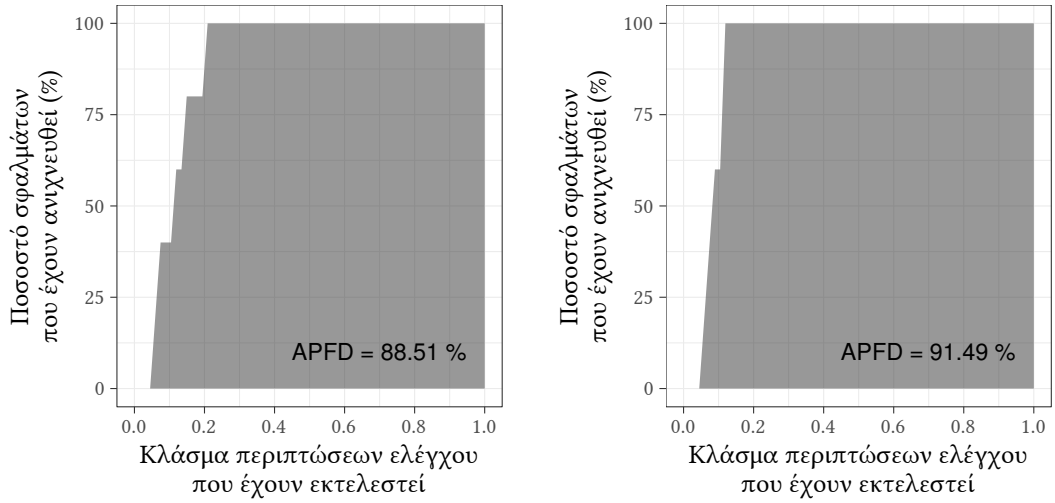
(β) ΤΠ6: File-addtl



(γ) ΤΠ7: Method-total

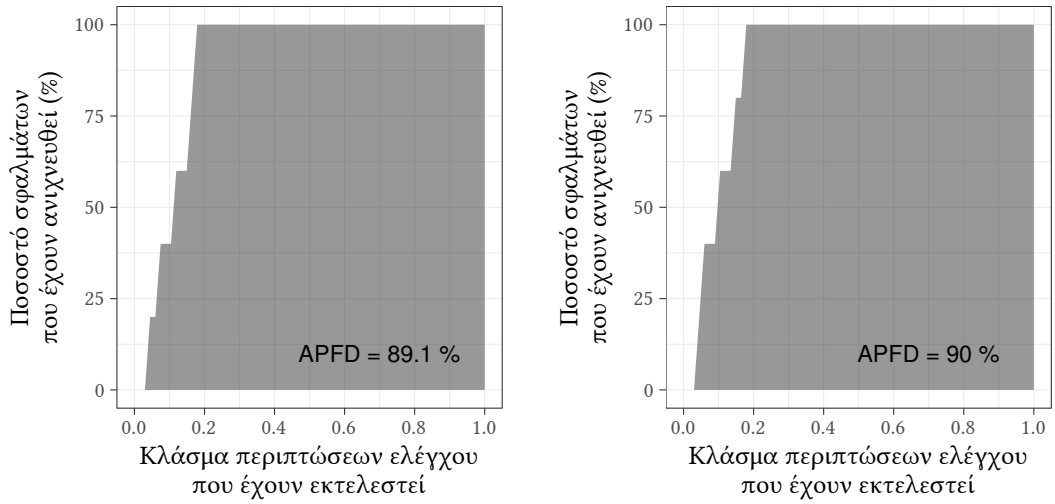
(δ) ΤΠ8: Method-addtl

Σχήμα 96: Υπολογισμοί APFD για το Six 1.10.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

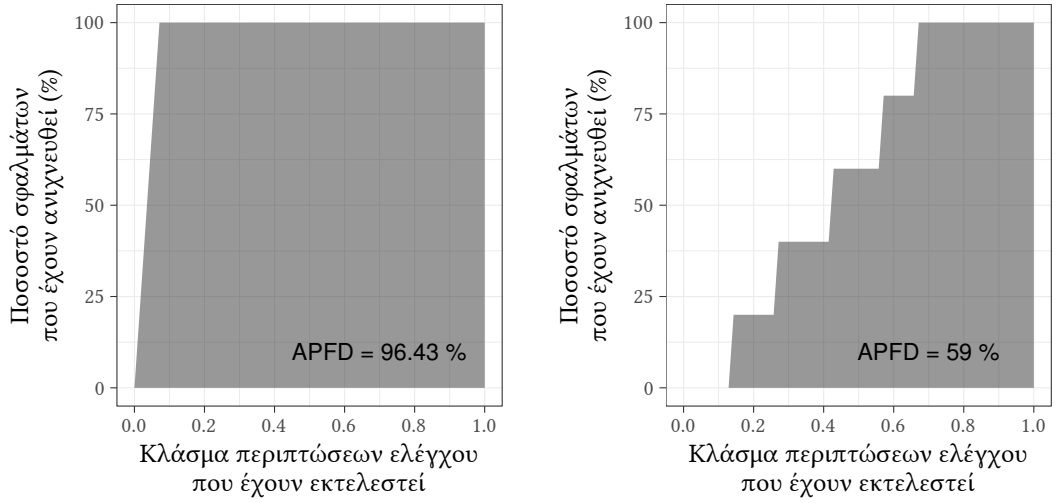
(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

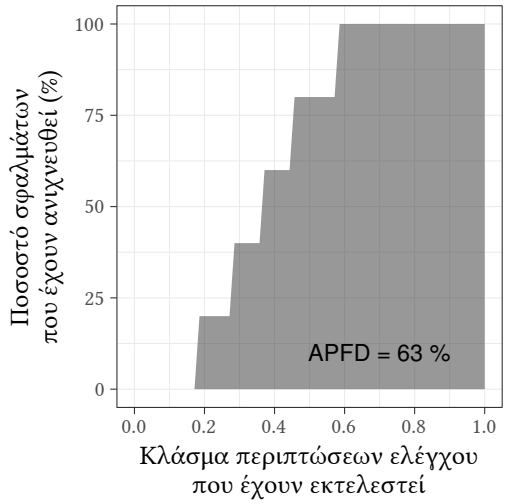
(δ) ΤΠ12: Statement-addtl

Σχήμα 97: Υπολογισμοί APFD για το Six 1.10.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

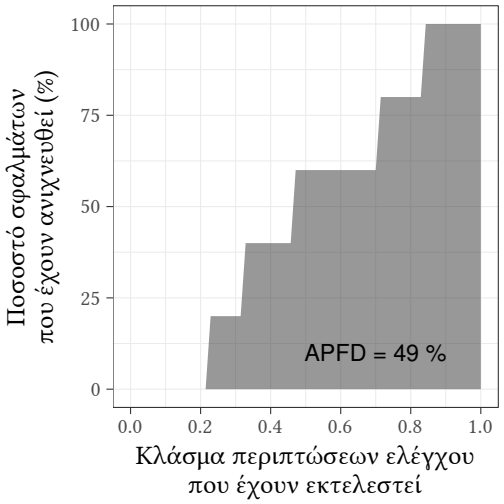


(α) ΤΠ1: Optimal

(β) ΤΠ2: Untreated

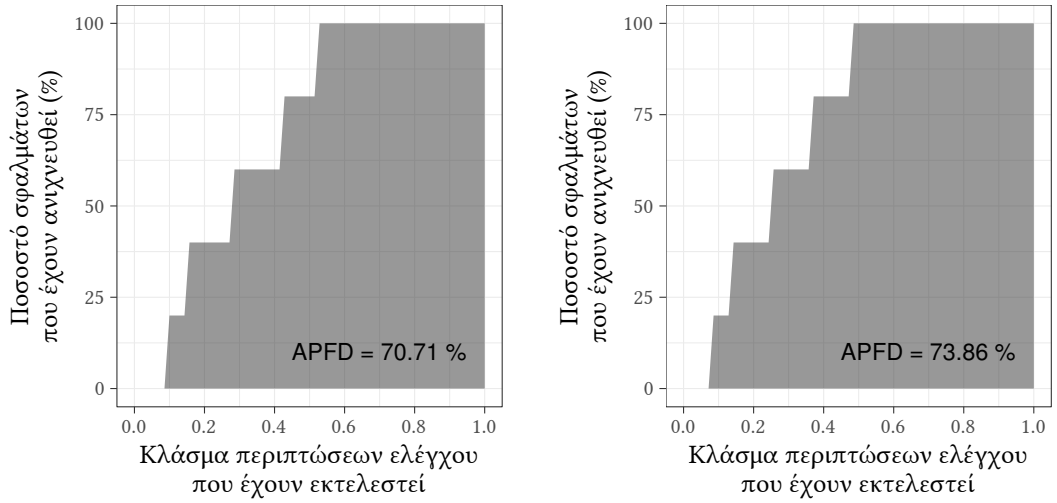


(γ) ΤΠ3: Random



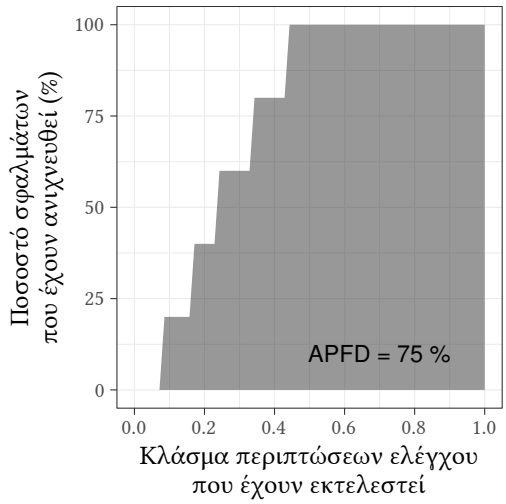
(δ) ΤΠ4: Inverse

Σχήμα 98: Υπολογισμοί APFD για το Six 1.11.0 (τεχνικές σύγκρισης)

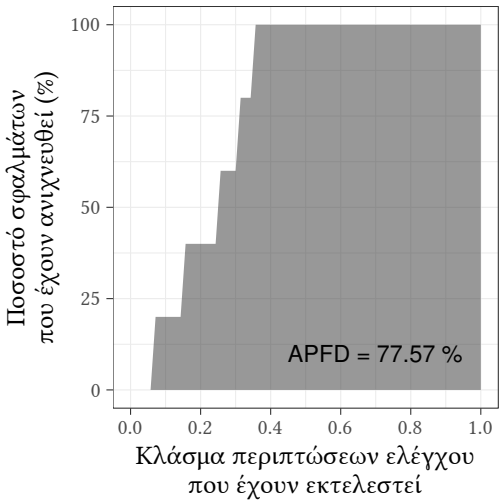


(α) ΤΠ5: File-total

(β) ΤΠ6: File-addtl

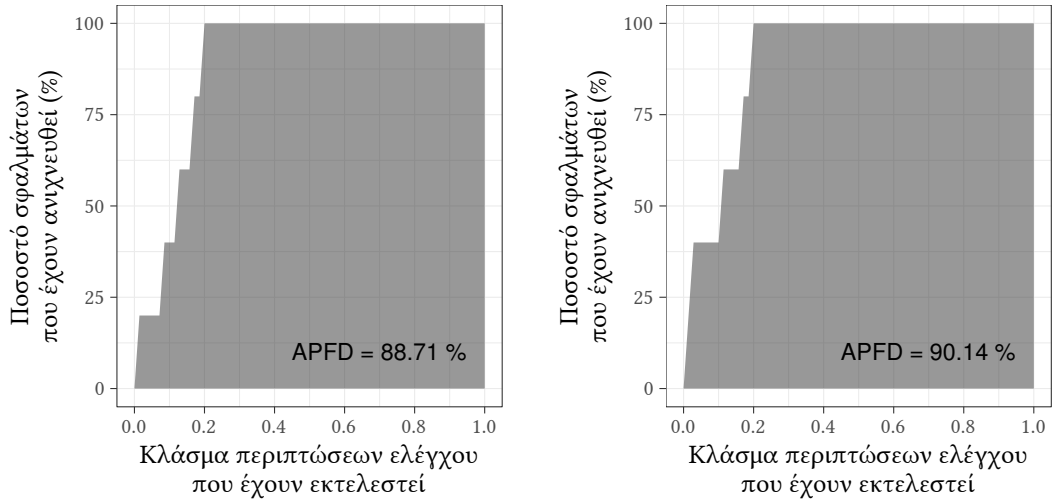


(γ) ΤΠ7: Method-total



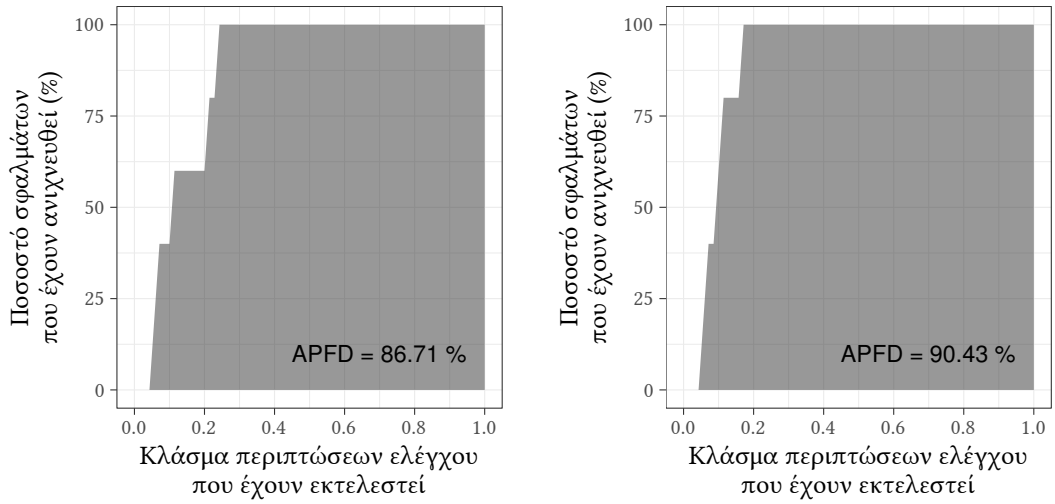
(δ) ΤΠ8: Method-addtl

Σχήμα 99: Υπολογισμοί APFD για το Six 1.11.0 (τεχνικές προτεραιοποίησης σε επίπεδο αρχείο και μεθόδου)



(α) ΤΠ9: Branch-total

(β) ΤΠ10: Branch-addtl



(γ) ΤΠ11: Statement-total

(δ) ΤΠ12: Statement-addtl

Σχήμα 100: Υπολογισμοί APFD για το Six 1.11.0 (τεχνικές προτεραιοποίησης σε επίπεδο διακλάδωσης και εντολής)

Έλεγχος Tukey

Πίνακας 24: Αποτελέσματα ελέγχου κατά ζεύγη συγκρίσεων του Tukey μεταξύ των τεχνικών προτεραιοποίησης

	diff	lwr	upr	p adj
ΤΠ2-ΤΠ1	-49.81	-54.20	-45.43	0.00
ΤΠ3-ΤΠ1	-44.04	-48.42	-39.65	0.00
ΤΠ4-ΤΠ1	-44.35	-48.73	-39.96	0.00
ΤΠ5-ΤΠ1	-38.45	-42.84	-34.06	0.00
ΤΠ6-ΤΠ1	-37.30	-41.69	-32.92	0.00
ΤΠ7-ΤΠ1	-25.34	-29.73	-20.96	0.00
ΤΠ8-ΤΠ1	-22.76	-27.15	-18.38	0.00
ΤΠ9-ΤΠ1	-20.02	-24.41	-15.63	0.00
ΤΠ10-ΤΠ1	-17.61	-22.00	-13.23	0.00
ΤΠ11-ΤΠ1	-19.15	-23.54	-14.77	0.00
ΤΠ12-ΤΠ1	-17.52	-21.91	-13.14	0.00
ΤΠ3-ΤΠ2	5.77	1.39	10.16	0.00
ΤΠ4-ΤΠ2	5.46	1.08	9.85	0.00
ΤΠ5-ΤΠ2	11.36	6.98	15.75	0.00
ΤΠ6-ΤΠ2	12.51	8.12	16.89	0.00
ΤΠ7-ΤΠ2	24.47	20.08	28.86	0.00
ΤΠ8-ΤΠ2	27.05	22.66	31.43	0.00
ΤΠ9-ΤΠ2	29.79	25.41	34.18	0.00
ΤΠ10-ΤΠ2	32.20	27.81	36.58	0.00
ΤΠ11-ΤΠ2	30.66	26.27	35.04	0.00
ΤΠ12-ΤΠ2	32.29	27.90	36.67	0.00
ΤΠ4-ΤΠ3	-0.31	-4.69	4.08	1.00
ΤΠ5-ΤΠ3	5.59	1.20	9.98	0.00
ΤΠ6-ΤΠ3	6.74	2.35	11.12	0.00
ΤΠ7-ΤΠ3	18.70	14.31	23.08	0.00
ΤΠ8-ΤΠ3	21.27	16.89	25.66	0.00

	diff	lwr	upr	p adj
ТП9-ТП3	24.02	19.63	28.40	0.00
ТП10-ТП3	26.43	22.04	30.81	0.00
ТП11-ТП3	24.88	20.50	29.27	0.00
ТП12-ТП3	26.51	22.13	30.90	0.00
ТП5-ТП4	5.90	1.51	10.28	0.00
ТП6-ТП4	7.04	2.66	11.43	0.00
ТП7-ТП4	19.01	14.62	23.39	0.00
ТП8-ТП4	21.58	17.20	25.97	0.00
ТП9-ТП4	24.33	19.94	28.71	0.00
ТП10-ТП4	26.73	22.35	31.12	0.00
ТП11-ТП4	25.19	20.81	29.58	0.00
ТП12-ТП4	26.82	22.44	31.21	0.00
ТП6-ТП5	1.15	-3.24	5.53	1.00
ТП7-ТП5	13.11	8.72	17.49	0.00
ТП8-ТП5	15.69	11.30	20.07	0.00
ТП9-ТП5	18.43	14.04	22.82	0.00
ТП10-ТП5	20.84	16.45	25.22	0.00
ТП11-ТП5	19.29	14.91	23.68	0.00
ТП12-ТП5	20.93	16.54	25.31	0.00
ТП7-ТП6	11.96	7.58	16.35	0.00
ТП8-ТП6	14.54	10.15	18.93	0.00
ТП9-ТП6	17.28	12.90	21.67	0.00
ТП10-ТП6	19.69	15.31	24.08	0.00
ТП11-ТП6	18.15	13.76	22.53	0.00
ТП12-ТП6	19.78	15.39	24.17	0.00
ТП8-ТП7	2.58	-1.81	6.96	0.74
ТП9-ТП7	5.32	0.94	9.71	0.00
ТП10-ТП7	7.73	3.34	12.11	0.00
ТП11-ТП7	6.19	1.80	10.57	0.00

	diff	lwr	upr	p adj
ТП12-ТП7	7.82	3.43	12.20	0.00
ТП9-ТП8	2.74	-1.64	7.13	0.65
ТП10-ТП8	5.15	0.77	9.54	0.01
ТП11-ТП8	3.61	-0.78	8.00	0.23
ТП12-ТП8	5.24	0.85	9.63	0.01
ТП10-ТП9	2.41	-1.98	6.79	0.81
ТП11-ТП9	0.87	-3.52	5.25	1.00
ТП12-ТП9	2.50	-1.89	6.88	0.77
ТП11-ТП10	-1.54	-5.93	2.84	0.99
ТП12-ТП10	0.09	-4.30	4.47	1.00
ТП12-ТП11	1.63	-2.76	6.02	0.99

Πίνακας 25: Αποτελέσματα ελέγχου κατά ζεύγη συγκρίσεων του Tukey μεταξύ των έργων που μελετήθηκαν

	diff	lwr	upr	p adj
Django-PyPy	4.49	2.49	6.50	0.00
MoinMoin-PyPy	9.10	6.64	11.56	0.00
Flask-PyPy	9.45	7.16	11.74	0.00
Six-PyPy	7.16	4.70	9.62	0.00
MoinMoin-Django	4.61	2.15	7.07	0.00
Flask-Django	4.95	2.66	7.24	0.00
Six-Django	2.67	0.21	5.13	0.03
Flask-MoinMoin	0.35	-2.35	3.04	1.00
Six-MoinMoin	-1.94	-4.78	0.90	0.33
Six-Flask	-2.29	-4.98	0.41	0.14