Εργασία Επαλήθευσης, Επικύρωσης και Συντήρησης Λογισμικού
2020 – 2021
Ομάδα :

Χρήστος Αργυρόπουλος 3170010
Διογένης Τσολάκου 3170164

Το παρόν έγγραφο αποτελεί την βιβλιογραφική εργασία του μαθήματος “Επαλήθευση, Επικύρωση και Συντήρηση Λογισμικού” του τμήματος Πληροφορικής του Οικονομικού Πανεπιστημίου Αθηνών (ΟΠΑ), κατά το θερινό εξάμηνο του ακαδημαϊκού έτους 2020 – 2021. Εκπονήθηκε από τους Τσολάκου Διογένη και Αργυρόπουλο Χρήστο.

Το θέμα της εργασίας είναι “Random Testing / Black Box Testing” κι επιλέχθηκε κατόπιν συνεννόησης κι επιβεβαίωσης από τον διδάσκοντα του μαθήματος, Μαλεύρη Νικόλαο.

Λεξικό Όρων

Παρακάτω ακολουθούν κάποιοι βασικοί όροι που θα αναφέρονται συχνά για λόγους σωστής ορολογίας, συντομίας και σωστής περιγραφής των τεχνικών.

Λογισμικό υπό εξέταση : Το λογισμικό πάνω στο οποίο εκτελούμε τον έλεγχο ώστε να διαπιστώσουμε την ορθότητά του, την αξιοπιστία του και την ποιότητά του.

Πεδίο/τιμές εισόδου : Το πεδίο εισόδου περιλαμβάνει όλες τις πιθανές εισόδους ενός λογισμικού, συμπεριλαμβανομένων όλων των παγκόσμιων μεταβλητών, ορίσματα μεθόδων και εξωτερικά εκχωρημένες μεταβλητές.
Test case : Ως test case εννοούμε μία τεχνική που περιγράφει την είσοδο, την ενέργεια και την αναμενόμενη έξοδο που αντιστοιχεί στην είσοδο. Ένα test case δηλώνεται ως επιτυχές εάν η έξοδος που λαμβάνεται μετά την εκτέλεση του συμφωνεί με την αναμενόμενη έξοδο. Εναλλακτικά, το test case δηλώνεται ως αποτυχία εάν η έξοδος που λαμβάνεται μετά την εκτέλεση του δεν συμφωνεί με την αναμενόμενη έξοδο. Μια σουίτα δοκιμών που αποτελείται από μια σειρά από test cases εκτελείται συνήθως για να διαπιστωθεί το επιθυμητό επίπεδο ποιότητας του λογισμικού υπό εξέταση.

Assert/assertion : Assertion είναι ένα κατηγόρημα (μια Boolean συνάρτηση πάνω στο χώρο καταστάσεων, που συνήθως εκφράζεται ως λογική πρόταση χρησιμοποιώντας τις μεταβλητές ενός προγράμματος) που συνδέεται με ένα σημείο του προγράμματος, το οποίο θέλουμε να αποτιμάται ως αληθές σε εκείνο το σημείο της εκτέλεσης του κώδικα.

Unit test(ing) : Το unit testing είναι μια μέθοδος δοκιμής/ελέγχου λογισμικού με την οποία μεμονωμένες μονάδες πηγαίου κώδικα, σύνολα μίας ή περισσότερων ενοτήτων προγράμματος υπολογιστή μαζί με τα σχετικά δεδομένα ελέγχου, τις διαδικασίες χρήσης και τις διαδικασίες λειτουργίας, δοκιμάζονται για να διαπιστωθεί αν είναι κατάλληλες για χρήση και παράδοση στον χρήστη/πελάτη.

Oracle : Ως (test) oracle ορίζεται, "μια πηγή που περιέχει αναμενόμενα αποτελέσματα για σύγκριση με το πραγματικό αποτέλεσμα του λογισμικού υπό εξέταση". Για ένα πρόγραμμα, ένα oracle είναι μια συνάρτηση, η οποία επαληθεύει ότι η έξοδος του προγράμματος είναι η ίδια με την έξοδο μιας σωστής έκδοσης του προγράμματος. Ένα oracle καθορίζει την αποδεκτή συμπεριφορά για την εκτέλεση δοκιμών. Οι τεχνικές δοκιμής/ελέγχου λογισμικού εξαρτώνται από τη διαθεσιμότητα ενός oracle. Ο σχεδιασμός ενός oracle για συνηθισμένο λογισμικό μπορεί να είναι απλό. Ωστόσο, για σχετικά πολύπλοκο λογισμικό, ο σχεδιασμός ενός oracle είναι αρκετά δύσχρηστος και απαιτεί ειδική τεχνογνωσία.

Ορισμένα κοινά ζητήματα που σχετίζονται με το είναι τα ακόλουθα:
1. Θεωρείται ότι τα αποτελέσματα της δοκιμής είναι παρατηρήσιμα και συγκρίσιμα με το oracle.
2. Ιδανικά, ένα oracle θα πρέπει να ικανοποιεί τις επιθυμητές ιδιότητες των προδιαγραφών του προγράμματος.
3. Στην πραγματικότητα γενικευμένα oracles είναι συχνά ανέφικτα.

Οι μεταϋποθέσεις είναι συνήθως χρησιμοποιούμενα oracles στον αυτοματοποιημένο έλεγχο λογισμικού. Οι μεταϋποθέσεις είναι συνθήκες που πρέπει να είναι αληθείς μετά την επιτυχή εκτέλεση μιας μεθόδου. Σε τέτοιες ένα oracle ένα σφάλμα σηματοδοτείται όταν παραβιάζεται μια μεταϋπόθεση.

Μερικές άλλες κοινές τεχνικές που χρησιμοποιούνται ως oracles είναι οι ακόλουθες:
1. Προδιαγραφές και τεκμηρίωση λογισμικού.
2. Προϊόντα παρόμοια με το λογισμικό υπό εξέταση αλλά με διαφορετικούς αλγόριθμους στην υλοποίηση.
3. Ευρετικοί αλγόριθμοι για την παροχή ακριβών αποτελεσμάτων για ένα σύνολο περιπτώσεων δοκιμής.
4. Σύγκριση του αποτελέσματος μιας δοκιμής με μια άλλη για λόγους συνέπειας.
5. Δημιουργία ενός μοντέλου για την επαλήθευση της συμπεριφοράς του λογισμικού υπό εξέταση.
6. Χειροκίνητη ανάλυση από ανθρώπινους εμπειρογνώμονες για την επαλήθευση των αποτελεσμάτων των δοκιμών.

Εισαγωγή Θεωρίας Black Box

Η έννοια black box φέρεται να χρησιμοποιείται στην αγγλική γλώσσα από το 1945, αν και είχε αναφερθεί για πρώτη φορά στην θεωρία κυκλωμάτων από τον Wilhelm Cauer από το 1941. Ωστόσο υπάρχουν μερικές αναφορές ακόμα κι από το 1921. Στην επιστήμη συστημάτων άρχισε η ευρεία χρήση του όρου στην δεκαετία του 1960.

Το black box, περιγράφει καταστάσεις στις οποίες ο παρατηρητής δεν έχει επίγνωση της εσωτερικής δομής του αντικειμένου παρατήρησης. Αντίθετα, γνωρίζει μόνο την εξωτερική μορφή του και εξάγει συμπεράσματα αξιολογώντας τις εισροές και τις εκροές του αντικειμένου. Συναντάται σε διάφορους τομείς από φιλοσοφία έως και εφαρμοσμένες επιστήμες, όπως μαθηματικά και φυσική. Στις ανθρωπιστικές επιστήμες για παράδειγμα, χρησιμοποιείται η θεωρία black box για να περιγράψει και να κατανοήσει ψυχολογικούς παράγοντες σε πεδία όπως το marketing όταν εφαρμόζεται στην ανάλυση συμπεριφοράς καταναλωτών.

Στην συνέχεια αναλύεται πως χρησιμοποιείται στην επιστήμη υπολογιστών και συγκεκριμένα στους ελέγχους λογισμικού, όπως επίσης θα αναλυθεί το Random Testing, μια τεχνική δοκιμής black box.

Black Box Testing

Το Black Box Testing είναι μια μέθοδος δοκιμής λογισμικού που εξετάζει τη λειτουργικότητα μιας εφαρμογής χωρίς να εισχωρεί στις εσωτερικές δομές ή λειτουργίες της. Αυτή η μέθοδος δοκιμής μπορεί να εφαρμοστεί ουσιαστικά σε κάθε επίπεδο ελέγχου λογισμικού: μονάδα (unit), ολοκλήρωση (integration), σύστημα (system) και αποδοχή (acceptance). Μερικές φορές αναφέρεται ως έλεγχος βάσει προδιαγραφών.

Για την εκτέλεση του black box testing δεν απαιτούνται ειδικές γνώσεις του κώδικα της εφαρμογής, της εσωτερικής δομής του και γενικά γνώσεις προγραμματισμού. Ο ελεγκτής γνωρίζει τι πρέπει να κάνει το λογισμικό, αλλά δεν γνωρίζει πώς και γιατί το κάνει. Για παράδειγμα, ο ελεγκτής γνωρίζει ότι μια συγκεκριμένη είσοδος επιστρέφει μια συγκεκριμένη, αμετάβλητη έξοδο, αλλά δεν γνωρίζει πώς το λογισμικό παράγει την έξοδο εξ αρχής.

Δημιουργία Test Cases στο Black Box Testing

Τα test cases κατασκευάζονται γύρω από τις προδιαγραφές και τις απαιτήσεις του λογισμικού, δηλαδή τι υποτίθεται ότι πρέπει να κάνει το λογισμικό. Τα test cases προέρχονται γενικά από εξωτερικές περιγραφές του λογισμικού, συμπεριλαμβανομένων των προδιαγραφών, των απαιτήσεων και των παραμέτρων σχεδιασμού. Παρόλο που οι δοκιμές που χρησιμοποιούνται είναι κυρίως λειτουργικής φύσης, μπορούν επίσης να χρησιμοποιηθούν και μη λειτουργικές δοκιμές. Ο σχεδιαστής δοκιμών επιλέγει έγκυρες και μη έγκυρες εισόδους και καθορίζει τη σωστή έξοδο, συχνά με τη βοήθεια ενός test oracle ή ενός προηγούμενου αποτελέσματος που είναι γνωστό ότι είναι καλό ή σωστό, χωρίς να γνωρίζει την εσωτερική δομή του αντικειμένου δοκιμής.

Τύποι Black Box Testing

Functional Testing

Το black box testing μπορεί να ελέγξει συγκεκριμένες λειτουργίες ή χαρακτηριστικά του λογισμικού υπό εξέταση. Για παράδειγμα, ο έλεγχος ότι είναι δυνατή η είσοδος με τη χρήση σωστών στοιχείων εισόδου χρήστη και ότι δεν είναι δυνατή η είσοδος με λανθασμένα στοιχεία εισόδου.

Το functional testing επικεντρώνεται στις πιο κρίσιμες πτυχές του λογισμικού (δοκιμή καπνού/δοκιμή ανυπαρξίας), στην ολοκλήρωση μεταξύ βασικών στοιχείων (δοκιμή ολοκλήρωσης) ή στο σύστημα ως σύνολο (δοκιμή συστήματος).

Non-functional Testing

Το black box testing μπορεί επίσης να ελέγξει πρόσθετες πτυχές του λογισμικού, πέρα από τα χαρακτηριστικά και τη λειτουργικότητα. Μια μη λειτουργική δοκιμή δεν ελέγχει το "αν" το λογισμικό μπορεί να εκτελέσει μια συγκεκριμένη ενέργεια, αλλά το "πώς" εκτελεί την ενέργεια αυτή.

Οι δοκιμές μαύρου κουτιού μπορούν να αποκαλύψουν αν το λογισμικό είναι:

- Εύχρηστο και εύκολα κατανοητό για τους χρήστες του
- Αποδίδει υπό αναμενόμενα ή μέγιστα φορτία
- Συμβατό με τις σχετικές συσκευές, μεγέθη οθόνης, προγράμματα περιήγησης ή λειτουργικά συστήματα
- Εκτεθειμένο σε ευπάθειες ασφαλείας ή κοινές απειλές ασφαλείας

Regression Testing

Το black box testing μπορεί να χρησιμοποιηθεί για να ελεγχθεί εάν μια νέα έκδοση του λογισμικού παρουσιάζει παλινδρόμηση ή υποβάθμιση των δυνατοτήτων από τη μια έκδοση στην άλλη. Η δοκιμή παλινδρόμησης μπορεί να εφαρμοστεί σε λειτουργικές πτυχές του λογισμικού (για παράδειγμα, ένα συγκεκριμένο χαρακτηριστικό δεν λειτουργεί πλέον όπως αναμενόταν στη νέα έκδοση) ή σε μη λειτουργικές πτυχές (για παράδειγμα, μια λειτουργία που εκτελούσε καλά είναι πολύ αργή στη νέα έκδοση).

Τεχνικές Black Box Testing

Boundary Value Analysis

Η Ανάλυση Οριακών Τιμών (BVA) αποτελεί την πιο συχνή τεχνική ελέγχου black box. Είναι υποκατηγορία του functional testing και εστιάζει στον καθορισμό ορίων στις τιμές που δοκιμάζονται στο σύστημα. Μεταξύ αυτών των οριακών τιμών είναι κάποιες όπως τυπικές τιμές, τιμές σφαλμάτων, το μέγιστο και το ελάχιστο. Η χρησιμότητα της βρίσκεται στην μείωση του πλήθους των test cases και είναι κατάλληλη σε συστήματα που υπάρχει περιορισμένο εύρος εισαγόμενων τιμών.

Κατά την εφαρμογή της ανάλυσης, εξάγονται οι αναμενόμενες τιμές εισροών και εκροών από τις προδιαγραφές του συστήματος. Στη συνέχεια, οι τιμές χωρίζονται σε ομάδες έτσι ώστε οι τιμές που ανήκουν στην ίδια ομάδα, όταν επεξεργαστούν, να παράξουν το ίδιο αποτέλεσμα. Με αυτόν τον τρόπο το πεδίο τιμών χωρίζεται σε περιοχές των οποίων τα όρια καθορίζουν και τη συμπεριφορά του συστήματος, η οποία εμφανίζεται διαφορετική εντός κι εκτός αυτών των ορίων.

Η BVA αποτελεί την βάση της τεχνικής Equivalence Partitioning. Κατά την τεχνική αυτή, τα δεδομένα εισόδου χωρίζονται σε κατηγορίες και κάθε κατηγορία δοκιμάζεται τουλάχιστον μία φορά ώστε να βεβαιωθεί η μέγιστη κάλυψη των δεδομένων εισόδου. Αυτή η τεχνική αποτελεί εξαντλητικό τρόπο ελέγχου.

Decision Table Testing

Decision table ονομάζεται μια μορφή αναπαράστασης συνδυασμού τιμών απέναντι σε ορισμένες δοκιμές ή συνθήκες. Είναι πολύ αποδοτικό στον έλεγχο όλων των δυνατών συνδυασμών.

Το Decision Table Testing είναι μια τεχνική ελέγχου συστημάτων στην οποία συνδυάζονται διάφορες εισαγόμενες τιμές και καταγράφεται το αποτέλεσμα της εκτέλεσης του ελεγχόμενου συστήματος. Τα αποτελέσματα παρουσιάζονται σε

μορφή πίνακα, ο οποίος συχνά αποκαλείται και πίνακας Αιτίου - Αποτελέσματος, λόγω της αναλυτικής καταγραφής εισροών (αίτιο) και εκροών (αποτελέσματος). Έτσι, διευκολύνεται η εξαγωγή έγκυρων συμπερασμάτων για τη συμπεριφορά του συστήματος υπό έλεγχο.

State Transition Testing

Η τεχνική State Transition Testing (STT) χρησιμοποιεί τις εισαγόμενες κι εξαγόμενς τιμές, καθώς και τις ενδιάμεσες καταστάσεις του συστήματος. Με αυτό τον τρόπο, δοκιμάζει το σύστημα απέναντι σε πιθανά σφάλματα κατά την μετάβαση από μια κατάσταση σε μια άλλη.

Είναι πολύ χρήσιμη τεχνική όταν ο στόχος είναι ο έλεγχος του συστήματος για ένα πεπερασμένο σύνολο δεδομένων. Ακόμη, είναι εξαιρετική μέθοδος όταν ο ελεγκτής, θέλει να σιγουρευτεί πως το σύστημα συμπεριφέρεται όπως αναμένεται, μετά από συγκεκριμένη ακολουθία γεγονότων. Για παράδειγμα, μετά την εισαγωγή λανθασμένου κωδικού τρεις φορές, η σελίδα σύνδεσης απαγορεύει στον χρήστη να επιχειρήσει να συνδεθεί ξανά.

Η τεχνική STT, αντίθετα, δεν πρέπει να προτιμάται όταν ο στόχος δεν είναι να ελεγχθεί η συμπεριφορά της εφαρμογής απέναντι σε μη-ακολουθιακές τιμές δεδομένων. Επίσης, δεν πρέπει να προτιμάται, όταν ο ελεγκτής προσπαθεί να δοκιμάσει διαφορετικές λειτουργίες ή συμπεριφορές του υπό έλεγχο συστήματος.

Θετικά και Αρνητικά του Black Box Testing

Όπως είναι προφανές και λογικό, η συγκεκριμένη τεχνική δεν είναι τέλεια και απόλυτη για κάθε λογισμικό υπό εξέταση και για κάθε περίπτωση απαιτήσεων και προδιαγραφών.

Θετικά

- Οι ελεγκτές δεν χρειάζονται ιδιαίτερες τεχνικές γνώσεις, γνώσεις προγραμματισμού ή πληροφορικής
- Οι ελεγκτές δεν χρειάζεται να μάθουν και να γνωρίζουν τις λεπτομέρειες του λογισμικού και πως αυτό λειτουργεί εσωτερικά
- Οι έλεγχοι μπορούν να εκτελεστούν από το κοινό (crowdsourcing) ή από εξωτερικούς ελεγκτές (outsourcing) που πολύ συχνά είναι φθηνότεροι από μία εσωτερική ομάδα ελεγκτών
- Χαμηλή πιθανότητα ψευδώς θετικών αποτελεσμάτων (false positives)
- Οι έλεγχοι έχουν χαμηλότερη πολυπλοκότητα, καθώς απλώς μοντελοποιούν και μιμούνται την κοινή συμπεριφορά του μέσου χρήστη

Αρνητικά

- Δυσκολία αυτοματοποιήσης
- Απαιτεί ιεράρχηση προτεραιοτήτων, καθώς συνήθως είναι ανέφικτη η δοκιμή όλων των μονοπατιών χρήστη
- Δύσκολος ο υπολογισμός του test coverage
- Εάν ένας έλεγχος αποτύχει, μπορεί να είναι δύσκολο να κατανοήσουμε τη βασική αιτία του προβλήματος
- Οι έλεγχοι μπορεί να διεξάγονται σε χαμηλή κλίμακα ή σε περιβάλλον που δεν είναι όμοιο με αυτό της παραγωγής (production)/του πελάτη

Εισαγωγή Random Testing

Το Random Testing αποτελεί μια black box τεχνική ελέγχου λογισμικού όπου το λογισμικό ελέγχεται μέσω της δημιουργίας τυχαίων και ανεξάρτητων εισόδων σύμφωνα με τις απαιτήσεις, τις προδιαγραφές ή άλλα κριτήρια επάρκειας της δοκιμής.

Το λογισμικό που θέλουμε να ελέγξουμε εκτελείται σε σχέση με το σύνολο δεδομένων δοκιμής και τα αποτελέσματα που λαμβάνονται αξιολογούνται για τον προσδιορισμό εάν η παραγόμενη έξοδος ικανοποιεί τα αναμενόμενα αποτελέσματα. Τα αποτελέσματα αυτά συγκρίνονται με τις προδιαγραφές του λογισμικού ώστε να επιβεβαιωθεί ότι το αποτέλεσμα ήταν επιτυχία ή αποτυχία. Σε περιπτώσεις που δεν δίνονται τέτοιες προδιαγραφές χρησιμοποιούνται τα exceptions της γλώσσας προγραμματισμού, δηλαδή αν κατά τη διάρκεια του ελέγχου προκύψει κάποιο exception τότε θεωρούμε ότι υπάρχει κάποιο λάθος στο λογισμικό.

Ιστορική Αναδρομή

Το random testing αναφέρεται στη βιβλιογραφία για πρώτη φορά από τον Hanford το 1970, ο οποίος ανέφερε τη μηχανή σύνταξης (syntax machine) , ένα εργαλείο που παρήγαγε τυχαία δεδομένα για τη δοκιμή μεταγλωττιστών PL/I. Το 1971 χρησιμοποιήθηκε για υλικό (hardware) από τον Melvin Breuer και έγινε προσπάθεια αξιολόγησης της αποτελεσματικότητας της συγκεκριμένης τεχνικής το 1975 από τους Pratima και Vishwani Agraval.

Για λογισμικό εξετάστηκε από τους Duran και Ntafos το 1984, πάνω από μια δεκαετία από την επινόησή του. Η χρήση του ελέγχου μέσω υποθέσεων ως θεωρητική βάση του random testing περιγράφηκε από τον Howden στο βιβλίο του Functional Testing and Analysis (1987).

Τρόπος Λειτουργίας

Ο τρόπος που λειτουργεί το random testing είναι αρκετά απλός, αποδοτικός και εύκολος στην κατανόηση. Τα δεδομένα εισόδου για το εκάστοτε λογισμικό επιλέγονται στη τύχη. Αυτό γίνεται μέσω γεννητριών τυχαίων αριθμών (random number generator – RNG), επειδή όμως η παραγωγή πραγματικά τυχαίων αριθμών είναι αρκετά δαπανηρή, δεν είναι εφικτή προς χρήση στον έλεγχο του λογισμικού. Επομένως, για λόγους ταχύτητας και ευχέρειας χρησιμοποιούνται ψευδοτυχαίοι αριθμοί, οι οποίοι στις περισσότερες περιπτώσεις είναι αρκετοί και επαρκής.

Η παραγωγή των αριθμών αυτών είναι αρκετά εύκολη με λίγες γραμμές κώδικα και με εξίσου απλό τρόπο αυτοματοποίησης ώστε να μην απαιτείται πολλή επίβλεψη από τους ελεγκτές του λογισμικού. Πολλές γλώσσες προγραμματισμού έχουν έτοιμες μεθόδους παραγωγής τυχαίων αριθμών, διευκολύνοντας έτσι την απαιτούμενη δουλειά για το σωστό έλεγχο του υπό εξέταση λογισμικού. Στις παρακάτω εικόνες παρουσιάζονται οι τρόποι παραγωγής αυτών των αριθμών (εισόδων) στις δημοφιλέστερες γλώσσες προγραμματισμού.

```
var rand = Math.random();
```

```
var rand = Math.floor(Math.random * N) + 1;
```

Παραγωγή ενός ψευδοτυχαίου αριθμού στη Javascript, από το 0 ως το 1 και ενός αριθμού από το 1 ως το N αντίστοιχα

```
double rand = Math.random();
```

```
Random rand = new Random();

int r1 = rand.nextInt(N) + 1;
```

Δύο από τους απλούστερους τρόπους παραγωγής ψευδοτυχαίων αριθμών στη Java. Η δεύτερη εικόνα χρησιμοποιεί την κλάση Random από το util πακέτο της Java.

```
int r = rand() % N + 1;
```

Ο απλούστερος τρόπος παραγωγής ενός ψευδοτυχαίου δεκαδικού αριθμού μεταξύ του 1 και του N στην C++

```
#include <iostream>
#include <time.h>

using namespace std;

int main() {
   srand(time(NULL));

   for (int i = 0; i < 5; i++) {
      cout << rand() % 5 << endl;
   }
   return 0;
}
```

Πιο εξειδικευμένος τρόπος παραγωγής τυχαίων αριθμών στην C++. Με τη χρήση της srand χρησιμοποιούμε ένα καινούριο seed κάθε φορά για την παραγωγή των αριθμών σε αντίθεση με την σκέτη rand που χρησιμοποιεί πάντα το ίδιο.

```
import random
print random.randint(0, 5)
```

Όπως στην Java, και στην Python χρειάζεται να συμπεριλάβουμε το πακέτο/βιβλιοθήκη σχετικά με την παραγωγή τυχαίων αριθμών. Η Python έχει αρκετές μεθόδους τυχαίες επιλογής και ανακατέματος αριθμών που είναι αποθηκευμένοι σε δομές δεδομένων.

Όπως είδαμε από τα παραπάνω παραδείγματα η παραγωγή τυχαίων αριθμών μπορεί να γίνει πολύ εύκολα μέσα σε ελάχιστες γραμμές σε αρκετές γλώσσες προγραμματισμού.

Ας δούμε τον παρακάτω κώδικα για μία πιο πρακτική απεικόνιση του τρόπου που λειτουργεί το random testing.

```
int myAbs(int x) {
    if (x > 0) {
        return x;
    }
    else {
        return x; // bug: should be '-x'
    }
}
```

Έστω ο παραπάνω λανθασμένος κώδικας υπολογισμού του απολύτου ενός αριθμού. Μπορούμε να δημιουργήσουμε μία απλή μέθοδο για τον έλεγχο του παραπάνω κώδικα με τυχαία δεδομένα εισόδου.

```
void testAbs(int n) {
    for (int i=0; i<n; i++) {
        int x = getRandomInput();
        int result = myAbs(x);
        assert(result >= 0);
    }
}
```

Όπως παρατηρούμε η συγκεκριμένη μέθοδος καλείται δίνοντας έναν ακέραιο αριθμό ο οποίος χρησιμεύει ως τον αριθμό των επαναλήψεων που θα κληθεί και ελεγχθεί ο κώδικας του αριθμητικού απολύτου. Μέσα στη μέθοδο έχουμε την κλήση μίας μεθόδου που μας δίνει κάποιο τυχαίο δεδομένο εισόδου που το δίνουμε ως παράμετρο στην μέθοδο του απολύτου και μέσω του assert επιβεβαιώνουμε αυτό το οποίο περιμένουμε ως απόλυτα λογικό αποτέλεσμα,

δηλαδή την επιστροφή ενός μη αρνητικού αριθμού ως αποτέλεσμα της μεθόδου του απολύτου.

Παρόλα αυτά, όπως αναφέραμε η μέθοδος του απολύτου είναι λανθασμένη. Για θετικούς τυχαία παραγόμενους αριθμούς η μέθοδος επιστρέφει τα αποτελέσματα που αναμένουμε. Για αρνητικούς αριθμούς όμως η μέθοδος ελέγχου θα δει ότι το assertion δεν εκπληρώνεται, λόγω του λάθος που υπάρχει. Η παρατήρηση αυτή του λογικού λάθους που υπάρχει στη μέθοδο θα ήταν αρκετά εύκολη λόγω της απλότητας της. Όμως πρέπει να αναλογιστούμε τι θα γινόταν για αρκετά πιο περίπλοκο λογισμικό με πολλές περισσότερες γραμμές κώδικα, στο οποίο ο προγραμματιστής ή ο ελεγκτής θα έχει να ασχοληθεί με περισσότερες μεθόδους και κομμάτια κώδικα, δυσκολεύοντας έτσι τη παρατήρηση τέτοιων σχετικά απλών λαθών.

Αντιθέτως, παράγοντας αυτομάτως τυχαίους αριθμούς αυξάνονται σε σημαντικό βαθμό η παρατήρηση λαθών καθώς ο όγκος των δεδομένων εισόδου είναι κατά πολύ μεγαλύτερος και ταχύτερος από αυτούς που θα μπορούσε να σκεφτεί και να δοκιμάσει ένας άνθρωπος.

Εκδοχές του Random Testing

Στην πραγματικότητα χρησιμοποιούνται διάφορες εκδοχές και τροποποιήσεις του Random Testing ανάλογα με το τι απαιτήσεις υπάρχουν στο υπό εξέταση λογισμικό. Για παράδειγμα, σε λιγότερο κρίσιμα λογισμικά, π.χ. σε απλές εφαρμογές που δεν διαχειρίζονται κρίσιμα δεδομένα ή που δεν βασίζονται επάνω τους ανθρώπινες ζωές, δεν θα χρειαστεί να χρησιμοποιήσουμε εξειδικευμένες εκδοχές της τεχνικής, συνήθως οι απλούστερες εκδοχές είναι αρκετές. Αντίθετα, σε κρίσιμα λογισμικά, όπως αεροπλάνα είτε μαχητικά είτε εμπορικά απαιτείται εκτενέστερος έλεγχος του λογισμικού τους, μιας και μιλάμε για πολλές ανθρώπινες ζωές. Επομένως θα χρειαστεί να χρησιμοποιηθούν πιο εξειδικευμένες και εξελιγμένες εκδοχές.

Random+ Testing

Το Random+ Testing είναι μια επέκταση της τυχαίας δοκιμής. Χρησιμοποιεί κάποιες ειδικές προκαθορισμένες τιμές αποθηκευμένες σε λίστα οι οποίες μπορεί να είναι απλές οριακές τιμές ή τιμές που έχουν υψηλή τάση εύρεσης σφαλμάτων στο λογισμικό υπό εξέταση. Οι οριακές αυτές τιμές είναι οι τιμές στην αρχή και το τέλος ενός συγκεκριμένου τύπου δεδομένων. Για παράδειγμα, τέτοιες τιμές για το int θα μπορούσαν να είναι Integer.MAX_VALUE, Integer.MAX_VALUE-1, Integer.MIN_VALUE, Integer.MIN_VALUE+1. Αυτές οι ειδικές τιμές μπορούν να προσθέσουν σημαντική βελτίωση σε μια μέθοδο ελέγχου**.**

Οι περιπτώσεις στις οποίες βοηθάει η συγκεκριμένη εκδοχή είναι περιπτώσεις όπως η δημιουργία δομών δεδομένων μέσω μεταβλητών που μπορεί να έχουν προσαυξηθεί μέσα στον κώδικα και δεν έχουμε άμεσο έλεγχο των τιμών τους κάθε στιγμή. Για παράδειγμα ας υποθέσουμε ότι δημιουργούμε μία λίστα ή έναν πίνακα με μέγεθος τη τιμή μίας μεταβλητής. Η μεταβλητή αυτή προφανώς περιμένουμε ότι θα είναι θετική μιας και δεν έχει λογική η δημιουργία μίας δομής με μηδενικό ή αρνητικό μέγεθος. Έστω επίσης ότι αυτή η μεταβλητή με κάποιο τρόπο έχει πάρει μία αρκετά μεγάλη τιμή, π.χ. την μέγιστη τιμή αναπαράστασης των int στη Java και μέσα στον κώδικα αυξάνεται κατά μία συγκεκριμένη τιμή, τότε θα υπάρξει overflow της μεταβλητής, παίρνοντας έτσι αρνητική τιμή, έχοντας ως αποτέλεσμα την προσπάθεια δημιουργίας της δομής με αρνητικό μέγεθος και πιθανότατα την παύση λειτουργίας του λογισμικού.

Προφανώς αυτές οι ειδικές τιμές δεν είναι καθολικές για κάθε λογισμικό, αλλά συνήθως επιλέγονται από τον κάθε ελεγκτή ανάλογα με τις ιδιαιτερότητες και τις απαιτήσεις του εκάστοτε λογισμικού.

Dirt Spot Sweeping Random Testing

Η στρατηγική Dirt Spot Sweeping Random (DSSR) αποτελεί μια βελτιωμένη εκδοχή του Random+ Testing. Βασίζεται στις ιδέες πως τα όρια τιμών παρουσιάζουν ενδιαφέρουσες τιμές των οποίων η απομονωμένη χρήση είναι χρήσιμη, και πως οι αποτυχίες βρίσκονται σε γειτονικές περιοχές ή λωρίδες.

Στην αρχή των ελέγχων δεν αρχικοποιείται η λίστα με τις ειδικές τιμές αλλά η στρατηγική ενεργοποιείται κατά τον εντοπισμό λαθών. Αν για παράδειγμα, εντοπιστεί λάθος στην τιμή 65, τότε το 65 καθώς και οι αριθμοί που είναι ελάχιστα μεγαλύτεροι ή μικρότεροι του (πχ έστω x οι κοντινοί αριθμοί, με $62 < x < 68$) προστίθενται στη λίστα με τις ειδικές τιμές και δοκιμάζονται με σειρά προτεραιότητας στις επόμενες δοκιμές. Αν η ιδέα πως οι τιμές που προκαλούν λάθη βρίσκονται κοντά μεταξύ τους είναι αληθής, τότε οι νέες τιμές της λίστας θα προκαλέσουν κι αυτές λάθη.

Η ουσιαστική διαφορά με το Random+ Testing έγκειται στο ότι στο R+ η λίστα με ειδικές τιμές είναι σταθερού μεγέθους και αρχικοποιείται πριν την έναρξη των δοκιμών με μερικές προκαθορισμένες τιμές. Αντίθετα, στο DSSR, η λίστα στην αρχή είναι άδεια και όταν εντοπίζεται ένα λάθος, πρώτα ελέγχεται ο τύπος της τιμής που το εντόπισε και στη συνέχεια προστίθεται αυτή η τιμή καθώς και οι γειτονικές της στη λίστα με τις ειδικές τιμές.

Adaptive Random Testing

Το Adaptive Random Testing αποτελεί τροποποιημένη εκδοχή της συνηθισμένης εκδοχής του Random Testing. Αυτό στο οποίο στοχεύει να βελτιώσει από την απλή εκδοχή είναι η δημιουργία δοκιμαστικών εισόδων δεδομένων που μένουν πολύ κοντά ή μακριά από την είσοδο δεδομένων που προκαλεί σφάλματα και την επικείμενη αποτυχία του λογισμικού, αποτυγχάνοντας έτσι την ανακάλυψη αυτών των σφαλμάτων.

Επομένως συγκεκριμένη μέθοδος λειτουργεί ώστε να παραχθούν είσοδοι δεδομένων που στοχεύουν καλύτερα στα πιθανά σφάλματα. Συνήθως χρησιμοποιείται με δύο σύνολα, το σύνολο υποψήφιων και το σύνολο εκτελεσμένων, και τα δύο είναι αρχικά κενά και μόλις αρχίσει ο έλεγχος η τεχνική γεμίζει το σύνολο υποψηφίων με τυχαία τιμές ελέγχου από το πεδίο εισόδου. Η πρώτη τιμή ελέγχου που επιλέγεται τυχαία από το σύνολο υποψηφίων εκτελείται και αποθηκεύεται στο σύνολο εκτελεσμένων. Η δεύτερη τιμή ελέγχου είναι εκείνη που επιλέγεται από το σύνολο υποψηφίων που βρίσκεται μακριά από τη προηγουμένως επιλεγμένη τιμή ελέγχου. Η διαδικασία συνεχίζεται μέχρι την

ολοκλήρωση της δοκιμής και παρέχει μεγαλύτερες πιθανότητες εύρεσης αποτυχιών.

Σε πειράματα με το Adaptive Random Testing χρησιμοποιείται ο αριθμός των τιμών ελέγχου που απαιτούνται για την εύρεση του πρώτου σφάλματος (F-measure) ως μέτρο απόδοσης αντί της παραδοσιακής πιθανότητας ανίχνευσης τουλάχιστον μιας βλάβης (P - measure) και του αναμενόμενου αριθμού βλαβών που ανιχνεύονται (E - measure). Τα αποτελέσματα έδειξαν έως και 50% αύξηση της απόδοσης σε σύγκριση με την τυχαία δοκιμή. Ωστόσο, οι συγγραφείς εξέφρασαν ανησυχίες σχετικά με τη διασπορά των περιπτώσεων δοκιμής σε όλες τις τομέα εισόδου, την αποτελεσματικότητα της επιλογής περιπτώσεων δοκιμής και το υψηλό κόστος των δημιουργίας δοκιμών.

Restricted Random Testing

Το Restricted Random Testing είναι μια εναλλακτική μορφή Adaptive Random Testing. Χρησιμοποιεί ζώνες αποκλεισμού γύρω από εκτελεσμένες περιπτώσεις δοκιμής, οι οποίες όμως δεν οδήγησαν σε αποτυχία. Με αυτόν τον τρόπο, δημιουργεί κυκλικές περιοχές εντός του πεδίου εισαγωγής τιμών από τις οποίες δεν μπορούν να προκύψουν οι επακόλουθες τιμές δοκιμής. Στη συνέχεια, επιλέγονται τιμές οι οποίες πρέπει να βρίσκονται εκτός των ήδη υπάρχοντων περιοχών, ώστε να εξασφαλίζεται πως η κάθε τιμή δοκιμής είναι αρκετά μακριά από κάθε προηγούμενη. Αυτή τη στιγμή, υπάρχουν δύο εκδοχές του, το Ordinary Restricted Random Testing (ORRT) και το Normalized Restricted Random Testing (NRRT).

Το RRT παρουσιάζει, όπως και το Adaptive Random Testing, σαφή βελτίωση σε σχέση με το απλό Random Testing, όπως φανερώνει ο δείκτης F-measure, ο οποίος δείχνει βελτίωση έως και 55%. Επίσης, το RRT αν κι έχει πολύ υψηλές έμμεσες υπολογιστικές δαπάνες, αυτές είναι σαφώς χαμηλότερες από αυτές του Advanced RT, για την ακρίβεια φθάνουν μόλις στο ¼ αυτού. Ωστόσο, πρέπει να σημειωθεί πως απορρίπτει σταθερά όλες τις περιπτώσεις δοκιμής εντός των ζωνών αποκλεισμού, μερικές εκ των οποίων είναι δυνατό να προκαλούν αποτυχίες το συστήματος κι έτσι αυτές δεν λαμβάνονται υπόψιν. Αυτό το

πρόβλημα έχουν προσπαθήσει να λύσουν οι ερευνητές προτείνοντας διάφορες λύσεις όπως το να επιλέγονται περιπτώσεις δοκιμής μέσω ενός καλοσχεδιασμένου προφίλ δοκιμών.

Directed Automated Random Testing

Οι Godefroid κ.ά. πρότειναν το Directed Automated Random Testing (DART). Στη διαδικασία DART, το δεδομένο λογισμικό υπό εξέτασει τροποποιείται για να παρακολουθείται η δυναμική συμπεριφορά του λογισμικού κατά την εκτέλεση. Προσδιορίζει επίσης τις εξωτερικές διεπαφές ενός συγκεκριμένου λογισμικού υπό εξέταση. Αυτές οι διεπαφές περιλαμβάνουν εξωτερικές μεταβλητές, εξωτερικές μεθόδους και την καθορισμένη από τον χρήστη κύρια μέθοδο που είναι υπεύθυνη για την εκτέλεση του προγράμματος.

Στη συνέχεια, δημιουργεί αυτόματα οδηγούς δοκιμών (test driver) για την εκτέλεση των τυχαία παραγόμενων περιπτώσεων δοκιμής (test case).

Τέλος, τα αποτελέσματα που λαμβάνονται αναλύονται σε πραγματικό χρόνο για τη συστηματική καθοδήγηση την εκτέλεση των περιπτώσεων δοκιμής κατά μήκος εναλλακτικών διαδρομών για μέγιστη κάλυψη του κώδικα. Ο αλγόριθμος DART υλοποιείται σε εργαλεία που είναι εντελώς αυτόματα και δέχονται μόνο το λογισμικό υπό εξέταση ως είσοδο. Αφού εξαχθούν οι εξωτερικές διεπαφές χρησιμοποιεί τις προϋποθέσεις και τις μεταϋποθέσεις του λογισμικού υπό εξέταση για την επικύρωση των εισόδων δοκιμής. Για γλώσσες που δεν υποστηρίζουν συμβόλαια (assertions) εντός του κώδικα (όπως η C), χρησιμοποιούνται δημόσιες μέθοδοι ή διεπαφές για να μιμούνται το σενάριο. Το DART προσπαθεί να καλύψει διαφορετικά μονοπάτια του κώδικα του προγράμματος για την ενεργοποίηση σφαλμάτων. Το oracle του αποτελείται από τον έλεγχο για crashes, αποτυχημένα assertions και μη τερματισμό.

Object-oriented Adaptive Random Testing

Η τεχνική Adaptive Random Testing for Object-Oriented (ARTOO) βασίζεται στην απόσταση αντικειμένων. Από έρευνες έχουν οριστεί οι παράμετροι που μπορούν να χρησιμοποιηθούν για τον υπολογισμό της απόσταση μεταξύ των αντικειμένων. Δύο αντικείμενα έχουν μεγαλύτερη απόσταση μεταξύ τους εάν έχουν πιο ανόμοιες ιδιότητες. Οι παράμετροι που καθορίζουν την απόσταση μεταξύ των αντικειμένων είναι: δυναμικοί τύποι, τιμές πρωτόγονων πεδίων (primitive fields) και τιμές πεδίων αναφοράς (reference fields). Οι συμβολοσειρές (string) αντιμετωπίζονται ως άμεσα χρησιμοποιήσιμες τιμές και χρησιμοποιείται ο τύπος Levenshtein ως κριτήριο απόστασης μεταξύ δύο συμβολοσειρών.

Κατά τη δοκιμή ARTOO, λαμβάνονται δύο σύνολα, το σύνολο υποψηφίων που περιέχει τα αντικείμενα που είναι έτοιμα να εκτελεστούν από το σύστημα και το σύνολο που χρησιμοποιείται, το οποίο είναι αρχικά κενό. Το πρώτο αντικείμενο επιλέγεται τυχαία από το σύνολο υποψηφίων που μεταφέρεται στο σύνολο χρήσης μετά την εκτέλεση. Το δεύτερο αντικείμενο που επιλέγεται από το σύνολο υποψηφίων για εκτέλεση είναι αυτό με τη μεγαλύτερη απόσταση από το τελευταίο αντικείμενο που εκτελέστηκε στο σύνολο χρήσης. Η διαδικασία συνεχίζεται μέχρι να βρεθεί το σφάλμα ή μέχρι να τελειώσουν τα αντικείμενα στο σύνολο υποψηφίων. Η τεχνική ARTOO, αξιολογήθηκε σε σύγκριση με τη DART επιλέγοντας κλάσεις από τη βιβλιοθήκη EiffelBase. Τα πειραματικά αποτελέσματα έδειξαν ότι ορισμένα σφάλματα που εντοπίστηκαν από την τεχνική ARTOO δεν εντοπίστηκαν από την DART τεχνική. Επιπλέον, η τεχνική ARTOO βρήκε τα πρώτα σφάλματα με μικρό αριθμό δοκιμών από την τεχνική DART. Ωστόσο, απαιτήθηκε περισσότερος υπολογισμός για την επιλογή ενός test case στην τεχνική ARTOO και η διαδικασία απαιτούσε περισσότερο χρόνο και κόστος για τη δημιουργία περιπτώσεων δοκιμής σε σύγκριση με την DART.

Feedback-directed Random Testing

Το Feedback-directed Random Testing (FDRT) είναι μια τεχνική που δημιουργεί unit tests τυχαία για αντικειμενοστρεφή προγράμματα. Όπως υποδηλώνει το όνομα, η FDRT χρησιμοποιεί την ανατροφοδότηση που λαμβάνεται από την εκτέλεση της πρώτης παρτίδας τυχαία επιλεγμένων unit tests για τη δημιουργία των επόμενων παρτίδων κατευθυνόμενων unit tests. Με αυτόν τον τρόπο, εξαλείφονται οι περιττές και λανθασμένες δοκιμές μονάδας σταδιακά από τη σουίτα δοκιμών με τη βοήθεια του φιλτραρίσματος και της εφαρμογής συμβολαίων. Για παράδειγμα, ένα unit test που παράγει IllegalArgumentException κατά την εκτέλεση απορρίπτεται επειδή τα ορίσματα που χρησιμοποιούνται στο unit test δεν είναι σύμφωνα με τον απαιτούμενο τύπο. Από μελέτες περίπτωσης που έχουν πραγματοποιηθεί στην οποία μια ομάδα ελεγκτών εφάρμοσε το FDRT σε ένα κρίσιμο στοιχείο της αρχιτεκτονικής .NET, τα αποτελέσματα έδειξαν ότι τα σφάλματα που ανακαλύφθηκαν με FDRT σε 15 ώρες χειροκίνητης επεξεργασίας και 150 ώρες επεξεργασίας με CPU είναι περισσότερες από ό,τι ένας μηχανικός δοκιμών βρίσκει σε ένα χρόνο με χειροκίνητες και άλλες αυτοματοποιημένες τεχνικές. Ως αποτέλεσμα, η FDRT είναι στον κατάλογο εργαλείων που χρησιμοποιούνται στη Microsoft για τη βελτίωση του λογισμικού.

Αυτόματα Εργαλεία για Random Testing

JCrasher

Το Java Crasher (JCrasher) είναι ένα αυτόματο εργαλείο ελέγχου ευρωστίας (robustness) που αναπτύχθηκε από τους Csallner και Smaragadakis. Το JCrasher δοκιμάζει το πρόγραμμα Java με τυχαίες εισόδους. Οι εξαιρέσεις που δημιουργούνται κατά τη διαδικασία ελέγχου καταγράφονται και συγκρίνονται με τον κατάλογο των αποδεκτών προτύπων που ορίζονται ως ευρετικά (heuristic) χαρακτηριστικά. Οι απροσδιόριστες εξαιρέσεις κατά την εκτέλεση (runtime exception) θεωρούνται ως αποτυχίες.

Το JCrasher ελέγχει τυχαία μόνο τις δημόσιες μεθόδους του λογισμικού υπό εξέταση με βάση το γεγονός ότι οι χρήστες αλληλεπιδρούν με τα προγράμματα μέσω δημόσιων μεθόδων. Ο μηχανισμός λειτουργίας του JCrasher απεικονίζεται με τη δοκιμή ενός προγράμματος Java. Το πηγαίο αρχείο μεταγλωττίζεται πρώτα για να ληφθεί ο κώδικας byte. Ο λαμβανόμενος κώδικας byte διαβιβάζεται ως είσοδος στο JCrasher, το οποίο χρησιμοποιεί τη βιβλιοθήκη Java reflection για να αναλύσει όλες τις μεθόδους που έχουν δηλωθεί από την κλάση. Το JCrasher χρησιμοποιεί τη μεταβατικούς τύπους παραμέτρων για να δημιουργήσει το καταλληλότερο σύνολο δεδομένων ελέγχου το οποίο εγγράφεται σε ένα αρχείο Java. Το αρχείο μεταγλωττίζεται και εκτελείται από το JUnit. Όλες οι εξαιρέσεις που παράγονται κατά τη διάρκεια της εκτέλεσης των test cases συλλέγονται και συγκρίνονται με την ευρετική αξιοπιστία και οι παραβιάσεις που προκύπτουν αναφέρονται ως σφάλματα.

Το JCrasher είναι ένα πρωτοποριακό εργαλείο με δυνατότητα εκτέλεσης πλήρως αυτόματων δοκιμών, συμπεριλαμβανομένων της δημιουργίας test cases, εκτέλεση, φιλτράρισμα και δημιουργία αναφορών. Το καινοτόμο χαρακτηριστικό του είναι η παραγωγή test cases ως αρχεία JUnit που μπορούν εύκολα να διαβαστούν και να χρησιμοποιηθούν για regression testing. Ένα άλλο σημαντικό χαρακτηριστικό του JCrasher είναι η ανεξάρτητη εκτέλεση κάθε νέας δοκιμής σε μια καθαρή “πλάκα”. Αυτό διασφαλίζει ότι οι αλλαγές που έγιναν από τις προηγούμενες δοκιμές δεν επηρεάζουν τη νέα δοκιμή.

Jartege

Η γεννήτρια τυχαίων δοκιμών Java (Java random test generator - Jartege) είναι ένα εργαλείο ελέγχου που παράγει τυχαία unit tests για κλάσεις Java με συμβόλαια που καθορίζονται σε JML. Τα συμβόλαια περιλαμβάνουν την κλάση αναλλοίωτη και τις προϋποθέσεις και μεταϋποθέσεις των μεθόδων. Αρχικά, το Jartege χρησιμοποιεί τα συμβόλαια για να εξαλείψει τα άσχετα (που παραβιάζουν τις προϋποθέσεις) test cases και αργότερα τα ίδια συμβόλαια χρησιμεύουν ως test oracles (μεταϋποθέσεις). Η Jartege χρησιμοποιεί ομοιόμορφο τυχαίο έλεγχο για να ελέγξει τις κλάσεις και για τη δημιουργία των test cases. Επιπλέον, η δοκιμή ενός συγκεκριμένου τμήματος της κλάσης μπορεί να πάρει προτεραιότητα αλλάζοντας

τις παραμέτρους για να προκύψουν ενδιαφέρουσες ακολουθίες κλήσεων, αν αυτό επιθυμεί ο ελεγκτής. Οι παράμετροι περιλαμβάνουν τα εξής:

- Λειτουργικό προφίλ των κλάσεων, δηλαδή η πιθανή χρήση της υπό εξέτασης κλάσης από άλλες τάξεις.
- Βάρος της κλάσης και της μεθόδου υπό εξέταση. Το υψηλότερο βάρος δίνει προτεραιότητα στην κλάση ή μέθοδο έναντι της χαμηλότερης βαρύτητας κατά τη διάρκεια της διαδικασίας ελέγχου.
- Πιθανότητα δημιουργίας νέων αντικειμένων κατά τη διάρκεια της διαδικασίας ελέγχου. Χαμηλή πιθανότητα σημαίνει δημιουργία λιγότερων αντικειμένων και μεγαλύτερη δυνατότητα επαναχρησιμοποίησης για διαφορετικές λειτουργίες, ενώ υψηλή πιθανότητα σημαίνει πολυάριθμα νέα αντικείμενα με λιγότερη δυνατότητα επαναχρησιμοποίησης.

Η τεχνική Jartege αξιολογεί μια κλάση με βάση τις προϋποθέσεις εισόδου και τις εσωτερικές προϋποθέσεις. Οι προϋποθέσεις εισόδου είναι τα συμβόλαια που πρέπει να πληρούνται από τα παραγόμενα δεδομένα ελέγχου για τη δοκιμή ενώ οι εσωτερικές προϋποθέσεις είναι τα συμβόλαια που βρίσκονται στο εσωτερικό των μεθόδων και οι παραβιάσεις τους θεωρούνται σφάλματα είτε στις μεθόδους είτε στις προδιαγραφές. Το Jartege ελέγχει για σφάλματα στον κώδικα του προγράμματος καθώς και στις προδιαγραφές και τις δοκιμές JUnit
που παράγονται από το Jartege και μπορούν να χρησιμοποιηθούν αργότερα για δοκιμές παλινδρόμησης (regression testing). Ένας από τους περιορισμούς του είναι η προηγούμενη απαίτηση των προδιαγραφών του προγράμματος σε γλώσσα μοντελοποίησης Java (JML).

Eclat

Το Eclat είναι ένα εργαλείο αυτοματοποιημένων δοκιμών το οποίο παράγει και ταξινομεί unit tests για κλάσεις Java. Το εργαλείο λαμβάνει ένα κομμάτι λογισμικού και ένα σύνολο από test cases για τις οποίες το λογισμικό τρέχει σωστά. Με βάση τις σωστές λειτουργίες του λογισμικού δημιουργείται ένα λειτουργικό μοντέλο για να γίνει έλεγχος των επιλεγμένων δεδομένων. Εάν το λειτουργικό πρότυπο των δεδομένων δοκιμής διαφέρει από το μοντέλο, τότε είναι δυνατόν να προκύψουν τα ακόλουθα τρία αποτελέσματα:

α) σφάλμα στο συγκεκριμένο λογισμικό υπό εξέταση
β) παραβίαση του μοντέλου παρά την κανονική λειτουργία
γ) παράνομη είσοδος την οποία το πρόγραμμα δεν είναι σε θέση να χειριστεί.

Η διαδικασία δοκιμής πραγματοποιείται από το Eclat σε τρία στάδια. Στο πρώτο στάδιο, επιλέγεται ένα μικρό υποσύνολο εισόδων δοκιμής, το οποίο ενδέχεται να αποκαλύψει σφάλματα στο δεδομένο λογισμικό υπό εξέταση. Στο δεύτερο στάδιο, χρησιμοποιείται η συνάρτηση μείωσης για την απόρριψη κάθε περιττής εισόδου, αφήνοντας μόνο μία είσοδο ανά λειτουργικό πρότυπο. Στο τρίτο στάδιο, οι αποκτηθέντες δοκιμαστικές είσοδοι μετατρέπονται σε test cases και δημιουργούνται oracles για τον προσδιορισμό της επιτυχίας ή της αποτυχίας της δοκιμής.

Για την αξιολόγηση, συνέκριναν το Eclat με το JCrasher εκτελώντας εννέα προγράμματα και στα δύο εργαλεία. Ανέφεραν ότι το Eclat είχε καλύτερες επιδόσεις από το JCrasher. Κατά μέσο όρο, το Eclat
επέλεξε 5,0 εισόδους ανά εκτέλεση από τις οποίες το 30% αποκάλυψε σφάλματα, ενώ το JCrasher επέλεξε 1,13 εισόδους ανά εκτέλεση από τις οποίες το 0,92% αποκάλυψε σφάλματα. Ο περιορισμός, όμως, του Eclat είναι η εξάρτησή του από την αρχική δεξαμενή σωστών test cases. Οποιοδήποτε σφάλμα στη δεξαμενή μπορεί να οδηγήσει στη δημιουργία
λανθασμένου λειτουργικού μοντέλου, το οποίο θα επηρεάσει αρνητικά τη διαδικασία δοκιμής.

Randoop

Ο Random tester for object-oriented programs (Randoop) είναι ένα εργαλείο που χρησιμοποιείται για την υλοποίηση της FDRT [86]. Το Randoop είναι ένα πλήρως αυτόματο εργαλείο, ικανό να ελέγχει κλάσεις Java και .NET binaries. Λαμβάνει ως είσοδο ένα σύνολο κλάσεων, συμβάσεων, φίλτρων και χρονικού ορίου και δίνει ως έξοδο μια σουίτα προγραμμάτων JUnit για Java και NUnit για .Net. Κάθε unit test σε μία σουίτα δοκιμών είναι μια ακολουθία κλήσεων μεθόδων (εφεξής αναφερόμενη ως ακολουθία).

Η Randoop κατασκευάζει την ακολουθία σταδιακά, επιλέγοντας τυχαία μια δημόσια μέθοδο από την κλάση υπό δοκιμή. Οι παράμετροι για αυτές τις μεθόδους επιλέγονται από ένα προκαθορισμένο σύνολο σε περίπτωση πρωτόγονων τύπων και ως ακολουθία μηδενικών τιμών στην περίπτωση τύπων αναφοράς. Η Randoop διατηρεί δύο σύνολα που ονομάζονται ErrorSeqs και NonErrorSeqs για την καταγραφή της ανατροφοδότησης. Επεκτείνει το σύνολο ErrorSeqs στην περίπτωση παραβίασης σύμβασης ή φίλτρου και το σύνολο NonErrorSeqs όταν δεν καταγράφεται παραβίαση στην ανατροφοδότηση.

Η χρήση αυτής της δυναμικής αξιολόγησης της ανατροφοδότησης κατά την εκτέλεση φέρνει ένα αντικείμενο σε μία ενδιαφέρουσα κατάσταση. Κατά την ολοκλήρωση της δοκιμής, τα ErrorSeqs και NonErrorSeqs παράγονται ως JUnit ή σουίτα δοκιμών NUnit.

Όσον αφορά την κάλυψη και τον αριθμό των σφαλμάτων που ανακαλύφθηκαν, το Randoop χρησιμοποιώντας την τεχνική FDRT συγκρίθηκε με το JCrasher και το JavaPathFinder σε 14 βιβλιοθήκες Java και .Net. προγραμμάτων. Τα αποτελέσματα έδειξαν ότι το Randoop πέτυχε μεγαλύτερη κάλυψη κλάδων και καλύτερη ανίχνευση σφαλμάτων από το JCrasher**.**

Αξιολόγηση Μεθόδου - Θετικά

Το random testing αποτελεί έναν πολύ εύκολο τρόπο να ελεγχθεί λογισμικό και θεωρείται εξίσου αποτελεσματικός με άλλους τρόπους αξιολόγησης, όπως για παράδειγμα οι δοκιμασίες μονάδας.

Η παραγωγή τυχαίων εισόδων μπορεί να γίνει αυτοματοποιημένα, συνεπώς διευκολύνεται η χρήση του. Επιπρόσθετα, το ανθρώπινο δυναμικό που διενεργεί τον έλεγχο μπορεί να είναι ένα ανεξάρτητο σώμα από την ομάδα που αναπτύσσει το λογισμικό, χωρίς να απαιτείται η γνώση προγραμματισμού από τους ελεγκτές. Αυτό σημαίνει πως μπορούν να εντοπισθούν λάθη που υπό άλλες συνθήκες η ομάδα ανάπτυξης δεν θα μπορούσε να εντοπίσει, και το λογισμικό αξιολογείται πιο

αντικειμενικά. Μπορεί επίσης να ανακαλύψει λεπτά σφάλματα σε ένα δεδομένο λογισμικό όταν υποβάλλεται σε μεγάλο αριθμό των περιπτώσεων δοκιμής. Ακόμη, ένα θετικό στοιχείο αυτού του τρόπου είναι πως επιλογή τυχαίων εισόδων καλύπτει μεγάλο εύρος των πιθανών τιμών που μπορούν να εισαχθούν σε ένα πρόγραμμα, ενώ αρκετά συχνά ελέγχονται τιμές που ένας χρήστης δεν θα σκεφτόταν ποτέ να ελέγξει, αυξάνοντας έτσι την ανθεκτικότητα του λογισμικού υπό έλεγχο.

Επισημαίνεται ότι η απλότητα και η αποδοτικότητα κόστους του random testing καθιστά πιο εφικτή την εκτέλεση μεγάλου αριθμού περιπτώσεων δοκιμής σε αντίθεση με άλλες τεχνικές που απαιτούν σημαντικό χρόνο και πόρους για τη δημιουργία περιπτώσεων δοκιμής και εκτέλεσης αυτών.

Αξιολόγηση Μεθόδου - Αρνητικά

Στην αντίπερα όχθη, το random testing δεν μπορεί να καλύψει εξειδικευμένες περιπτώσεις ή συνθήκες. Αυτό την καθιστά επιρρεπή σε κριτική λόγω της παραγωγής πολλών δεδομένων ελέγχου που οδηγούν στην ίδια κατάσταση του λογισμικού.

Συχνά αναφέρεται ότι τα παραγόμενα δεδομένα παραβιάζουν τις προϋποθέσεις που έχουν τεθεί για το λογισμικό. Αυτό δημιουργεί την ανάγκη να χρησιμοποιηθεί σε συνδυασμό με άλλες πιο στοχευμένες μεθόδους αξιολόγησης, ώστε να καλυφθούν περιπτώσεις που δεν μπορούν να αξιολογηθούν με τις τυχαία επιλεγμένες τιμές. Νωρίτερα, αναφέραμε πως ένας πραγματικός χρήστης ενδεχομένως να μην σκεφτόταν ποτέ να εισάγει κάποια από τις τυχαίες εισόδους, ωστόσο είναι πιθανό να συμβεί και το αντίστροφο. Αναλόγως το είδος και τον σκοπό του λογισμικού, μερικές τιμές είναι πιθανότερο από κάποιες άλλες να εισαχθούν, συνεπώς πρέπει να ληφθεί υπόψιν ο ανθρώπινος παράγοντας και να υπάρξει ανθρώπινη παρέμβαση. Σε περίπτωση που αυτό δεν συμβεί, η κατανομή των επιλεγμένων εισόδων δεν θα είναι ρεαλιστική, συνεπώς υπάρχει κίνδυνος να αποτύχει η αξιολόγηση του προγράμματος σε αργότερα στάδια της ανάπτυξης του, όταν θα δοκιμαστεί σε πραγματικές συνθήκες.

Κάποια από τα μειονεκτήματα αυτά μπορούν να λυθούν ή τουλάχιστον να μειωθούν ως κάποιον βαθμό με τις τροποποιήσεις της τεχνικής που αναφέρθηκαν παραπάνω.

Πρακτικές Εφαρμογές

Η μέθοδος Random Testing είναι μια πολύ χρήσιμη και αξιόπιστη μέθοδος ελέγχου σε μεγάλα συστήματα με πολλές συνθήκες και μονοπάτια εκτέλεσης. Είναι εύκολο να παραχθούν πολλοί έλεγχοι γρήγορα, κι αυτοί, όπως αναφέραμε νωρίτερα, μπορούν να καλύψουν ένα σημαντικό ποσοστό του ελέγξιμου κώδικα και στη συνέχεια να ακολουθήσουν πιο στοχευμένες μέθοδοι, όπως για παράδειγμα το White Box Testing.

Μια περίπτωση χρήσης αυτής της μεθόδου είναι η ανάπτυξη compilers. Με την πάροδο του χρόνου, προστίθενται νέες ιδιότητες στην αντίστοιχη γλώσσα προγραμματισμού και οι δοκιμές παλαιότερων εκδόσεων δεν μπορούν πλέον να καλύψουν τις νέες. Συνεπώς, μπορεί το Random Testing να παρέχει γρήγορες δοκιμές σε κάθε νέα έκδοση, ειδικότερα σε γλώσσες που εξελίσσονται με ταχείς ρυθμούς.

Σε ακόμη μία γνωστή περίπτωση, το Random Testing χρησιμοποιήθηκε για την παραγωγή τυχαίων ASCII χαρακτήρων ώστε να ελεγχθεί η συμπεριφορά τερματισμού στα Unix. Στη συνέχεια, με την ίδια τεχνική ανακαλύφθηκαν πολλά σφάλματα σε παλαιά λειτουργικά συστήματα, όπως Windows NT και Mac OS X.

Ερευνητές από την Αυστραλία και την Ινδία διεξήγαγαν έρευνα όπου αξιολογούσαν την χρήση τυχαίων τιμών ελέγχου σε ενσωματωμένα συστήματα. Σύγκριναν το Random Testing με τα Unit Testing, Hand Testing και Black Box Testing. Σύμφωνα με τα ευρήματα τους, όταν κάθε τεχνική χρησιμοποιήθηκε ξεχωριστά, το Random Testing είχε την δεύτερη καλύτερη επίδοση σε branch coverage και την τρίτη σε λόγο εξοντωμένων μεταλλάξεων. Ομοίως, όταν συνδυάστηκαν όλες οι μέθοδοι με Unit Testing, το Random Testing ήρθε ξανά

δεύτερο σε ποσοστό εξόντωσης μεταλλάξεων, αλλά παρουσίασε σαφώς βελτιωμένη εικόνα στην κάλυψη κλάδων.

Μια ειδική κατηγορία ελέγχου με τυχαία δεδομένα αποτελεί το Monkey Testing. Κατά τις δοκιμές, εισάγονται πραγματικά τυχαία δεδομένα, όπως δηλαδή θα τα παρήγαγε μια μαϊμού πατώντας διάφορα κουμπιά. Με αυτόν τον τρόπο, γίνονται υποθέσεις για την ευφυΐα της μαϊμούς και αναλόγως αλλάζει η εγκυρότητα των τιμών, οι οποίες παρόλα αυτά παραμένουν τυχαίες. Σε πραγματικές συνθήκες, αυτή η τεχνική χρησιμοποιείται στον έλεγχο συναλλαγών στις βάσεις δεδομένων. Κατά την εκκίνηση μιας συναλλαγής, εισάγονται τυχαία δεδομένα ή πραγματοποιούνται τυχαίες ενέργειες, και στη συνέχεια γίνεται επαναφορά (roll back) ώστε να ελεγχθεί εάν προκλήθηκε σφάλμα ή φθορά στην βάση.

Τέλος, φέρεται να χρησιμοποιήθηκε η ίδια μέθοδος ελέγχου σε κλήσεις χαμηλών επιπέδων σε συστήματα αρχείων της NASA. Φαίνεται, σύμφωνα με την υπάρχουσα βιβλιογραφία, πως το Random Testing είναι ευρέως διαδεδομένο τόσο στον ακαδημαϊκό, όσο και στον εμπορικό χώρο.

Συμπεράσματα

Το Black Box είναι μια ευρέως διαδεδομένη έννοια σε μεγάλο φάσμα θετικών και θεωρητικών επιστημών, στην κάθε μια με τη δική της αξία και σημασία. Στην επιστήμη υπολογιστών συναντάται κυρίως στο Black Box Testing.

Το Black Box Testing αποτελεί μια πολύ σημαντική και χρήσιμη μέθοδο ελέγχου ποιότητας λογισμικού και συστημάτων. Χρησιμοποιείται ήδη αρκετές δεκαετίες κι έχει συμβάλει σημαντικά σε δοκιμές κρίσιμων λογισμικών ιστορικής σημασίας όπως της NASA και των πρώτων εκδόσεων των Windows και Mac. Με την πάροδο των ετών εφευρίσκονται όλο και πιο βελτιωμένες εκδοχές του, καθιστώντας το από τις πιο αξιόλογες μεθόδους ελέγχου με βάση τις επιδόσεις του. Μια κατηγορία αυτού είναι το Random Testing.

Το Random Testing από όταν ξεκίνησε να χρησιμοποιείται, καθιερώθηκε και πλέον αποτελεί βασικό τρόπο διασφάλισης ποιότητας συστημάτων και λογισμικού. Έχει αναπτυχθεί πληθώρα διαφορετικών τρόπων εφαρμογής του και η έρευνα συνεχίζεται, καθιστώντας το αρκετά εξελίσιμη τεχνική ελέγχου.

Πηγές

https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF4290/v10/undervisningsmateriale/INF4290-RandomTesting.pdf

https://link.springer.com/chapter/10.1007/978-3-540-30502-6_23

https://www.hindawi.com/journals/mpe/2019/9381728/

http://web.cecs.pdx.edu/~hamlet/random.pdf

https://www.researchgate.net/post/What-can-be-the-advantages-and-disadvantages-of-random-testing

https://www.researchgate.net/publication/220344923_Restricted_Random_Testing_Adaptive_Random_Testing_by_Exclusion

https://link.springer.com/chapter/10.1007%2F3-540-47984-8_35

https://ieeexplore.ieee.org/document/5562948

https://www.functionize.com/blog/black-box-testing-techniques-explained-2/

https://reqtest.com/testing-blog/black-box-testing/

https://www.guru99.com/black-box-testing.html

https://www.guru99.com/decision-table-testing.html

https://www.researchgate.net/publication/232616711_Effectiveness_of_Random_Testing_of_Embedded_Systems

https://etheses.whiterose.ac.uk/7981/1/ociamthesismain.pdf