

Unit Testing

Πως μπορούμε να εξασφαλίσουμε την ορθότητα των προγραμμάτων;

Παντελέλης Μιχάλης
email: mpantel@aegean.gr

Ηλεκτρολόγος Μηχανικός & Μηχανικός Η/Υ

Ειδικός Λογαριασμός Ερευνας Π.Αιγαίου
Προϊστάμενος Τμήματος Διαχείρισης Πληροφοριακών Συστημάτων

Μπορούμε να αποδείξουμε ότι ένα πρόγραμμα είναι ορθό;

Για οποιαδήποτε είσοδο;

Αυτοματοποιημένα;

Με το ερώτημα αυτό ασχολήθηκαν από τις αρχές του περασμένου αιώνα...

O David Hilbert (https://en.wikipedia.org/wiki/David_Hilbert):

Θεωρείται πατέρας της "Μαθηματικής Λογικής"

Προσπαθούσε να αποδείξει ότι αν έχουμε ορίσει
ένα σύνολο από κανόνες (αξιώματα και θεωρήματα),
και ένα τρόπο να παράγουμε συμπεράσματα από αυτούς τους κανόνες,

μπορούμε να φτιάξουμε συστήματα πλήρη και συνεπή ...

Αξίωμα: 2 σημεία ορίζουν μία ευθεία

Αξίωμα: $\alpha + \beta = \beta + \alpha$

Πληρότητα: βασισμένοι στα αξιώματα μπορούμε να αποδείξουμε όλα τα θεωρήματα

Συνέπεια: δεν έχουμε αντιφάσεις

Με το ερώτημα αυτό ασχολήθηκαν από τις αρχές του περασμένου αιώνα...

O Kurt Gödel (https://en.wikipedia.org/wiki/Kurt_G%C3%B6del):

Απέδειξε το θεώρημα της ΜΗ πληρότητας (1931)

Οπότε τα σχέδια του Hilbert ναυάγησαν :-(

Με το ερώτημα αυτό ασχολήθηκαν από τις αρχές του περασμένου αιώνα...

O Alan Turing (https://en.wikipedia.org/wiki/Alan_Turing):

Απέδειξε ότι ΔΕΝ υπάρχει αλγόριθμος που να αποφασίζει πάντα και σωστά
για ένα οποιοδήποτε πρόγραμμα και την είσοδό του
αν το πρόγραμμα θα σταματήσει όταν το εκτελέσουμε με την είσοδο αυτή

Γλώσσες Προγραμματισμού – Γενικός Διαχωρισμός

Low Level / High Level

Γλώσσες Προγραμματισμού – Γενικός Διαχωρισμός

Low Level: Machine Code, Assembly

High Level: Java, Javascript, C/C++, ADA, Pascal

Γλώσσες Προγραμματισμού – Γενικός Διαχωρισμός

Low Level: Machine Code, Assembly

High Level: Java, Javascript, C/C++, ADA, Pascal

Compiled / Interpreted

Γλώσσες Προγραμματισμού – Γενικός Διαχωρισμός

Low Level: Machine Code, Assembly

High Level: Java, Javascript, C/C++, ADA, Pascal

Compiled: Java, Javascript, C/C++, ADA, Pascal

Interpreted: Javascript, Python, Ruby . . .

Γλώσσες Προγραμματισμού Python vs Ruby

```
print "Hello, world!"
```

Γλώσσες Προγραμματισμού Python vs Ruby

```
for i in [1,2,3]:  
    print i
```

```
for i in [1,2,3]  
    print i  
end
```

Γλώσσες Προγραμματισμού Python vs Ruby

list

@list

len(list)

@list.length

indentation and colons(:)

begin ... end

Γλώσσες Προγραμματισμού

Έχουν παρόμοιες δομές και συντακτικό

Υλοποιούμε τους ίδιους αλγορίθμους... με άλλα λόγια

Δεν πρέπει να τρομάζουμε βλέποντας μία διαφορετική γλώσσα

Συνήθως η επιλογή γλώσσας είναι θέμα (προσωπικής) προτίμησης

Όσα θα περιγράψουμε στην συνέχεια μπορείτε να τα εφαρμόσετε γενικά

ανά οικογένεια γλωσσών

όλες οι γλώσσες που είδαμε παραπάνω μπορούν να λύσουν τα ίδια προβλήματα
άλλες ευκολότερα, άλλες δυσκολότερα:

https://en.wikipedia.org/wiki/Turing_completeness

Πώς ελέγχουμε τα προγράμματα που φτιάχουμε π.χ. για το εργαστήριο;

Με το χέρι...

Εποπτικά...

Εμπειρικά...

Πώς ελέγχουμε μεγάλα συστήματα λογισμικού;

Regression Suites

End User Testing

Automated Testing

PyUnit <https://wiki.python.org/moin/PyUnit>
Minitest <https://github.com/seattlerb/minitest>
Junit <https://junit.org/junit5/>
Jasmin <https://jasmine.github.io/>

Program Development by Stepwise Refinement

Niklaus Wirth

Communications of the ACM, Vol. 14, No. 4, April 1971

<http://sunnyday.mit.edu/16.355/wirth-refinement.html>

Test-Driven Development

(Martin Fowler: <https://martinfowler.com/bliki/TestDrivenDevelopment.html>)

Είναι τεχνική για τη κατασκευή λογισμικού
που οδηγείται από της συγγραφή tests

Αναπτύχθηκε από τον Kent Beck στα τέλη της δεκαετίας του 1990
ως τμήμα της μεθοδολογίας Extreme Programming.

RED – GREEN – REFACTOR

RED: Γράφουμε tests (που αρχικά δεν ικανοποιούνται)

GREEN: Γράφουμε κώδικα που ικανοποιεί τα tests

REFACTOR: Βελτιώνουμε τη ποιότητα του κώδικα

Refactoring (Martin Fowler, <https://refactoring.com/>)

Ουσιαστικό (refactoring): αλλαγή στην εσωτερική δομή του λογισμικού ώστε:
να γίνει εύκολότερη η κατανόησή του
και οικονομικότερη η εξέλιξή του,
ΧΩΡΙΣ να αλλάξει η εξωτερική του συμπεριφορά.

Ρήμα (refactor): η διαδικασία της αναδιάταξης του κώδικα μέσω
μίας σαφούς αλληλουχίας βημάτων με τρόπο τέτοιο ώστε
να μήν αλλάξει η εξωτερική του συμπεριφορά.

Testing Frameworks

Διαθέσιμα πλέον στις περισσότερες γλώσσες προγραμματισμού

Διαφορετικές Φιλοσοφίες (TDD – testing, BDD – Behaviour)

Διαφορετικά επίπεδα testing (unit, integration, front end...)

Διαφορετική αποδοχή από της κοινότητες προγραμματιστών

TDD: Assertions – Διαβεβαιώσεις (Ruby)

```
assert_equal 2, @size  
  
assert_includes @list, "item"  
  
assert_instance_of Array, @list  
  
assert_kind_of Array, @list  
  
assert_nil
```

Παραδειγματα

```
class MyList

def initialize(a = [])
  @list = a
end

class MyListClassTests

def test_new
  assert_kind_of MyList, MyList.new([1, 2, 3])
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
.
```

```
Finished in 0.000444 seconds.
```

```
1 tests, 1 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
100% passed
```

```
2252.25 tests/s, 2252.25 assertions/s
```

```
class MyList

def contents
  @list
end

class MyListClassTests

def test_contents
  assert_equal [1, 2, 3], MyList.new([1, 2, 3]).contents
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
..
```

```
Finished in 0.000433 seconds.
```

```
2 tests, 2 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
4618.94 tests/s, 4618.94 assertions/s
```

```
class MyList

def append(a)
  @list.append(a)
end

class MyListClassTests

def test_append
  my_list = MyList.new([1, 2, 3])
  my_list.append(4)
  assert_equal [1, 2, 3, 4], my_list.contents
  assert_includes my_list.contents, 4
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
...
```

```
Finished in 0.000676 seconds.
```

```
3 tests, 4 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
4437.87 tests/s, 5917.16 assertions/s
```

```
class MyList

def find_first_position_of(a)
  i = 0
  position = nil
  while i < @list.length
    if @list[i] == a
      position = i
      break
    end
    i += 1
  end
  return position
end

class myListClassTests

def test_find_first_position_of
  my_list = MyList.new([1, 2, 3, 2, 5])
  assert_equal 1, my_list.find_first_position_of(2)
  assert_equal 0, my_list.find_first_position_of(1)
  assert_equal 4, my_list.find_first_position_of(5)
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
....
```

```
Finished in 0.000586 seconds.
```

```
4 tests, 7 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
6825.94 tests/s, 11945.39 assertions/s
```

```
class MyList

def find_positions_of(a)
  positions = []
  i = 0
  while i < @list.length
    if @list[i] == a
      positions.append i
    end
    i += 1
  end
  return positions
end

class MyListClassTests

def test_find_positions_of
  my_list = MyList.new([1, 2, 3, 2, 5])
  assert_equal [1, 3], my_list.find_positions_of(2)
  assert_equal [0], my_list.find_positions_of(1)
  assert_equal [4], my_list.find_positions_of(5)
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
.....
```

```
Finished in 0.000596 seconds.
```

```
5 tests, 10 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
8389.26 tests/s, 16778.52 assertions/s
```

```
class MyList

def maximum
  i = 0
  max = @list[0]
  while i < @list.length
    if @list[i] > max
      max = @list[i]
    end
    i += 1
  end
  return max
end

class myListClassTests

def test_maximum
  my_list = MyList.new([1, 2, 3, 2, 5, 16, 7, 12])
  assert_equal 16, my_list.maximum
  my_list = MyList.new([12, 2, 33, 2, 5, 16, 7, 12])
  assert_equal 33, my_list.maximum
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
.....
```

```
Finished in 0.000768 seconds.
```

```
6 tests, 12 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
7812.50 tests/s, 15625.00 assertions/s
```

```
class MyList

def minimum
  i = 0
  min = @list[0]
  while i < @list.length
    if @list[i] > min
      min = @list[i]
    end
    i += 1
  end
  return min
end

class myListClassTests

def test_minimum
  my_list = MyList.new([1, 2, 3, 2, 5, 16, 7, 12])
  assert_equal 1, my_list.minimum
  my_list = MyList.new([12, 1, 2, 3, 2, 25, 16, 7])
  assert_equal 1, my_list.minimum
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
.....F
```

```
=====
```

```
Failure: test_minimum(MyListClassTests)
```

```
testing.rb:179:in `test_minimum'
```

```
 176:   if in_sequence('T7')
 177:     def test_minimum
 178:       my_list = MyList.new([1, 2, 3, 2, 5, 16, 7, 12])
=> 179:       assert_equal 1, my_list.minimum
 180:       my_list = MyList.new([12, 1, 2, 3, 2, 25, 16, 7])
 181:       assert_equal 1, my_list.minimum
 182:     end
```

```
<1> expected but was
```

```
<16>
```

```
diff:
```

```
? 16
```

```
=====
```

```
.
```

```
Finished in 0.005552 seconds.
```

```
7 tests, 13 assertions, 1 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
85.7143% passed
```

```
1260.81 tests/s, 2341.50 assertions/s
```

```
class MyList

def min_max
  [minimum, maximum]
end

class MyListClassTests

def test_min_max
  my_list = MyList.new([1, 2, 3, 2, 5, 16, 7, 12])
  assert_equal [1, 16], my_list.min_max
  my_list = MyList.new([12, 2, 33, 2, 5, 16, 7, 12])
  assert_equal [2, 33], my_list.min_max
  my_list = MyList.new([12, 1, 2, 3, 2, 25, 16, 7])
  assert_equal [1, 25], my_list.min_max
end
```

Run Tests

```
Loaded suite testing
```

```
Started
```

```
.....
```

```
Finished in 0.000728 seconds.
```

```
8 tests, 17 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
```

```
100% passed
```

```
10989.01 tests/s, 23351.65 assertions/s
```

Συμπεράσματα – Unit Testing

1. Είναι σημαντικό να ελέγχουμε τα προγράμματα που γράφουμε
2. Μπορούμε να επικεντρωθούμε στις συνοριακές περιπτώσεις
3. Είναι σημαντικό ο έλεγχος να γίνεται αυτοματοποιημένα
4. Μπορεί να μας καθοδηγήσει στη βελτίωση του κωδικά που γράφουμε
5. Είναι ίσως το σημαντικότερο εργαλείο που διαθέτουμε ως προγραμματιστές

Η παρουσίαση αυτή έχει υλοποιηθεί στη γλώσσα Ruby...

...εκτελείται ενώ παρουσιάζεται...

είναι διαθέσιμη από το

https://github.com/mpantel/term_note/blob/master/lib/testing.rb