Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Ποιότητα και Αξιοπιστία Λογισμικού

ΘΩΜΑΣ ΝΙΚΟΛΑΟΣ - ΑΜ: 21390068
ΠΑΠΑΓΕΩΡΓΙΟΥ ΦΙΛΙΠΠΟΣ - ΑΜ: 21390174

ΑΘΗΝΑ
Παρασκευή, 31 Ιανουαρίου 2025

Περιεχόμενα:

1. CheckPhone

a. Παραδοχή υλοποίησης

```
        public bool CheckPhone(string Phone, ref string PhoneCountry)
        {
            if (!Phone.StartsWith("+") && !Phone.StartsWith("00"))
            {
                Phone = "+" + Phone;
            }

            string pattern = @"^(00|\+)(\d{10,15})$";
            Regex regex = new Regex(pattern);
            if (!regex.IsMatch(Phone))
            {
                PhoneCountry = "";
                return false;
            }

            if (Phone.Contains("30") && (Phone.StartsWith("0030") ||
Phone.StartsWith("+30")))
                PhoneCountry = "Ελλάδα";
            else if (Phone.Contains("357") && (Phone.StartsWith("00357")
|| Phone.StartsWith("+357")))
                PhoneCountry = "Κύπρος";
            else if (Phone.Contains("39") && (Phone.StartsWith("0039") ||
Phone.StartsWith("+39")))
                PhoneCountry = "Ιταλία";
            else if (Phone.Contains("44") && (Phone.StartsWith("0044") ||
Phone.StartsWith("+44")))
                PhoneCountry = "Αγγλία";
            else
            {
                PhoneCountry = "";
                return false;
            }
            return true;
        }
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
PhoneValid	Τηλέφωνο	Ταιριάζει στην κανονική έκφραση: "^(00|\+)(\d{10,15})$"	Δεν ταιριάζει στην κανονική έκφραση: "^(00|\+)(\d{10,15})$"

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου		Αναμενόμενα Αποτελέσματα	
#id	Phone	PhoneCountry(ref)	flag	PhoneCountry
1	+301234567890		TRUE	Ελλάδα
2	301234567890		TRUE	Ελλάδα
3	+3571234567890		TRUE	Κύπρος
4	391234567890		TRUE	Ιταλία
5	+441234567890		TRUE	Αγγλία
6	1234567890		FALSE	
7	30123		FALSE	

c. Υλοποίηση μονάδων ελέγχου

```
[TestMethod]
 public void TestCheckPhone()
 {
     string resultPhoneCountry = string.Empty;
     Payroll_Lib.EmployeePayment phone = new
Payroll_Lib.EmployeePayment();

     object[,] testcases =
     {
         {1, "+301234567890", true},
         {2, "00301234567890", true},
         {3, "+3571234567890", true},
         {4, "00391234567890", true},
         {5, "+441234567890", true},
         {6, "1234567890", false},
         {7, "30123", false}
     };

     bool failed = false;

     for (int i = 0; i < testcases.GetLength(0); i++)
     {
         try
         {
             bool validPhone = false;

             validPhone = phone.CheckPhone((string)testcases[i, 1], ref
resultPhoneCountry);

             Assert.AreEqual((bool)testcases[i, 2], validPhone);

             if (validPhone)
             {
                 Assert.IsFalse(string.IsNullOrEmpty(resultPhoneCountry),
"Country should not be null or empty for valid phone numbers.");
             }
             else
             {
                 Assert.IsTrue(string.IsNullOrEmpty(resultPhoneCountry),
"Country should be null or empty for invalid phone numbers.");
             }
         }
```

```
        catch (Exception e)
        {
            failed = true;
            Console.WriteLine("Failed Test Case: {0}: {1} - {2}. \n \t
Reason: {3} ",
                (int)testcases[i, 0], (string)testcases[i, 1],
(bool)testcases[i, 2], e.Message);
        }
    }

    //Στην περίπτωση που κάποια περίπτωση ελέγχου απέτυχε, πέταξε
εξαίρεση.
    if (failed) Assert.Fail();
}
```

d. Αναφορές ελέγχου

Αναφορά Σφάλματος - CheckPhone_ERR_001

1. **Αναγνωριστικό Σφάλματος:**

   CheckPhone_InvalidFormat_001

2. **Όνομα μεθόδου ελέγχου που απέτυχε:**

   TestCheckPhone (UnitTest1.cs, γραμμή 57)

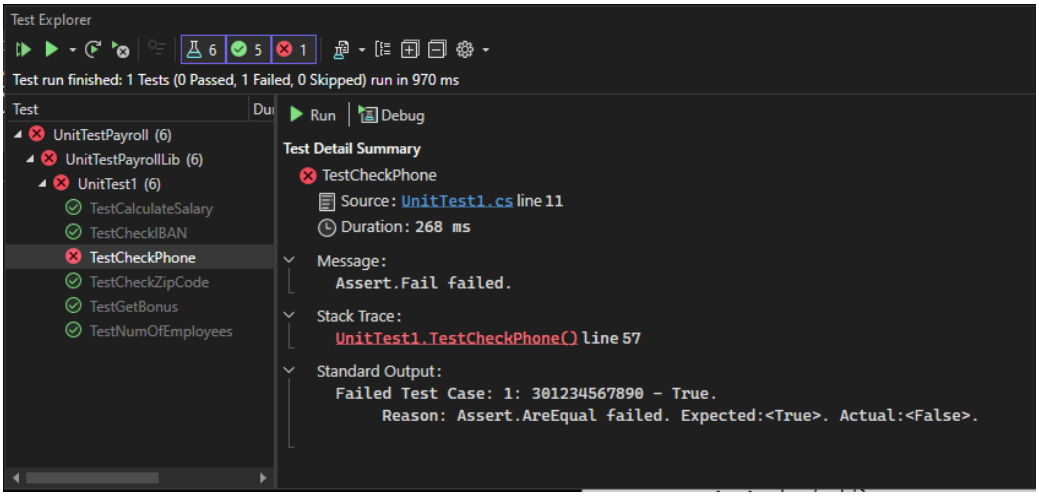
3. **Περιγραφή ελέγχου**

   Ο έλεγχος αφορούσε την επαλήθευση ότι η συνάρτηση CheckPhone απορρίπτει τηλεφωνικούς αριθμούς που δεν ξεκινούν με "+" ή "00". Δοκιμάστηκε η είσοδος "301234567890" χωρίς το απαιτούμενο πρόθεμα.

4. **Αναφορά σφάλματος:**

   - Expected: True
   - Actual: False
   - Test Case: 1: 301234567890 - True
   - Duration: 268 ms

5. **Στιγμιότυπο οθόνης Test Explorer**



6. **Διόρθωση:**

   Το σφάλμα οφειλόταν στο ότι η συνάρτηση CheckPhone απαιτούσε αυστηρά πρόθεμα "+" ή "00". Η διόρθωση έγινε προσθέτοντας έναν αρχικό έλεγχο που προσθέτει το "+" όταν λείπει το πρόθεμα, επιτρέποντας έτσι την αναγνώριση αριθμών που ξεκινούν απευθείας με τον κωδικό χώρας. Επίσης, βελτιώθηκαν οι έλεγχοι των κωδικών χωρών για μεγαλύτερη ακρίβεια.

2. CheckIban

a. Παραδοχή υλοποίησης

```
public bool CheckIBAN(string IBAN, ref string IBANCountry)
 {
     // 1. Ελέγξτε το μήκος και την αρχική χώρα
     if (IBAN.Length < 15 || IBAN.Length > 34)
     {
         IBANCountry = "";
         return false;
     }

     // Καθορισμός χώρας από τα πρώτα δύο γράμματα
     string country = IBAN.Substring(0, 2).ToUpper();
     switch (country)
     {
         case "GR":
             IBANCountry = "Ελλάδα";
             break;
         case "CY":
             IBANCountry = "Κύπρος";
             break;
         case "IT":
             IBANCountry = "Ιταλία";
             break;
         case "GB":
             IBANCountry = "Αγγλία";
             break;
```

```
        default:
            IBANCountry = "";
            break;
    }

    if (IBANCountry == null) return false;

    // 2. Μετακίνηση των 4 πρώτων γραμμάτων στο τέλος
    string rearrangedIBAN = IBAN.Substring(4) + IBAN.Substring(0, 4);

    // 3. Μετατροπή γραμμάτων σε αριθμούς
    StringBuilder numericIBAN = new StringBuilder();
    foreach (char ch in rearrangedIBAN)
    {
        if (char.IsLetter(ch))
        {
            // A = 10, B = 11, ..., Z = 35
            numericIBAN.Append((ch - 'A' + 10).ToString());
        }
        else if (char.IsDigit(ch))
        {
            numericIBAN.Append(ch);
        }
        else
        {
            return false; // Μη αποδεκτός χαρακτήρας
        }
    }

    // 4. Υπολογισμός Modulo 97
    string ibanNumber = numericIBAN.ToString();
    int mod97 = Mod97(ibanNumber);

    // Επιστροφή true αν το υπόλοιπο είναι 1
    return mod97 == 1;
}

private int Mod97(string number)
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
IBANValid	IBAN	IBAN length >= 15 and <= 34	IBAN length < 15 or > 34
IBANCountry	IBAN	IBAN να ξεκινάει με GR, CY, IT, or GB	IBAN να μην ξεκινάει με GR, CY, IT, or GB

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου		Αναμενόμενα Αποτελέσματα	
id	IBAN	IBANCountry(ref)	flag	IBANCountry
1	GR1601101250000000012300695		TRUE	Ελλάδα
2	GB29NWBK60161331926819		TRUE	Αγγλία
3	IT60X0542811101000000123456		TRUE	Ιταλία
4	CY17002001280000001200527600		TRUE	Κύπρος
5	XX0000000000000000000		FALSE	
6	GR123		FALSE	

c. Υλοποίηση μονάδων ελέγχου

```
        [TestMethod]
        public void TestCheckIBAN()
        {
            string resultIBANCountry = string.Empty;
            Payroll_Lib.EmployeePayment ibanChecker = new Payroll_Lib.Em-
ployeePayment();

            object[,] testcases =
            {
                {1, "GR1601101250000000012300695", true, "Ελλάδα"},
                {2, "GB29NWBK60161331926819", true, "Αγγλία"},
                {3, "IT60X0542811101000000123456", true, "Ιταλία"},
                {4, "CY17002001280000001200527600", true, "Κύπρος"},
                {5, "XX0000000000000000000", false, ""},
                {6, "GR123", false, ""}
            };

            bool failed = false;
            for (int i = 0; i < testcases.GetLength(0); i++)
            {
                try
                {
                    bool validIBAN =
ibanChecker.CheckIBAN((string)testcases[i, 1], ref resultIBANCountry);
                    Assert.AreEqual((bool)testcases[i, 2], validIBAN);
                    Assert.AreEqual((string)testcases[i, 3],
resultIBANCountry);
                }
                catch (Exception e)
                {
                    failed = true;
                    Console.WriteLine("Απέτυχε η δοκιμή: {0}: {1} - {2}.
Σφάλμα: {3}", (int)testcases[i, 0], (string)testcases[i, 1],
(bool)testcases[i, 2], e.Message);
                }
            }

            if (failed) Assert.Fail("Κάποιες δοκιμές απέτυχαν.");

        }
```

d. Αναφορές ελέγχου

Αναφορά Σφάλματος - CheckIBAN_ERR_001

1. **Αναγνωριστικό Σφάλματος:**

   CheckIBAN_MOD97Validation_001

2. **Όνομα μεθόδου ελέγχου που απέτυχε:**

   TestCheckIBAN () (UnitTest1.cs, γραμμή 61)

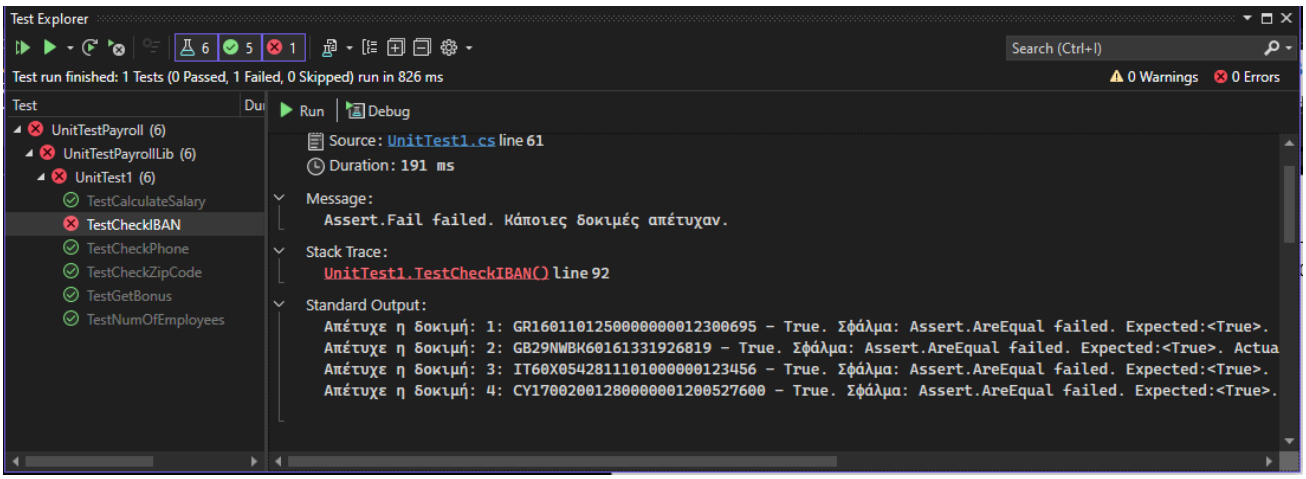
3. **Περιγραφή ελέγχου:**

   Ο έλεγχος αφορούσε την επαλήθευση της εγκυρότητας του IBAN μέσω του αλγορίθμου MOD97. Δοκιμάστηκε η είσοδος "GR1601101250000000012300695" που είναι ένα έγκυρο ελληνικό IBAN.

4. **Αναφορά σφάλματος:**

   - Expected: True
   - Actual: False
   - Test Case: 1: GR1601101250000000012300695 - True
   - Duration: 191 ms

5. **Στιγμιότυπο οθόνης Test Explorer:**



6. Διόρθωση:

Το σφάλμα οφειλόταν σε λανθασμένη υλοποίηση του αλγορίθμου MOD97. Συγκεκριμένα, δεν γινόταν σωστή μετατροπή των γραμμάτων σε αριθμούς (A=10, B=11, κλπ).

3. CheckZipCode

a. Παραδοχή υλοποίησης

```
public bool CheckZipCode(int zipCode)
 {
     if (zipCode == 120)
         return true;

     if (zipCode < 10000 || zipCode > 99999)
         return false;

     if (zipCode >= 47890 && zipCode <= 47899)
         return true;

     int provinceCode = zipCode / 1000;

     if (provinceCode >= 1 && provinceCode <= 99)
     {
         if ((zipCode >= 118 && zipCode <= 199) ||       // Rome (00118-
00199)
             (zipCode >= 20100 && zipCode <= 20199) ||   // Milan (20100-
20199)
             (zipCode >= 30100 && zipCode <= 30124) ||   // Venice (30100-
30124)
             (zipCode >= 56121 && zipCode <= 56128) ||   // Pisa (56121-
56128)
             (zipCode >= 10100 && zipCode <= 10159) ||   // Torino (10100-
10159)
             (zipCode == 12100) ||                      // Cuneo
             (zipCode == 14100) ||                      // Asti
             (zipCode == 11100) ||                      // Aosta
             (zipCode == 16100) ||                      // Genoa
             (zipCode == 17100) ||                      // Savona
             (zipCode == 19100) ||                      // La Spezia
             (zipCode == 20900) ||                      // Monza
             (zipCode == 31100) ||                      // Treviso
             (zipCode == 32100) ||                      // Belluno
             (zipCode >= 37100 && zipCode <= 37139))    // Verona (37100-
37139)
         {
             return true;
         }

         return false;
     }

     return false;
 }
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
ZipCodeValid	Ταχυδρομικός Κώδικας	Κωδικοί εντός έγκυρων περιοχών Ιταλίας (00118-00199, 20100-20199, κλπ) Ειδική περίπτωση: 120	Οτιδήποτε άλλο

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου	Αναμενόμενα Αποτελέσματα
id	ZipCode	flag
1	00120	TRUE
2	20100	TRUE
3	30100	TRUE
4	120	TRUE
5	99999	FALSE
6	1234	FALSE
7	00100	FALSE

c. Υλοποίηση μονάδων ελέγχου

```
[TestMethod]
 public void TestCheckZipCode()
 {
     Payroll_Lib.EmployeePayment zipChecker = new Payroll_Lib.EmployeePay-
ment();

     object[,] testcases =
     {
         {1, 47890, true},
         {2, 47899, true},
         {3, 10100, true},
         {4, 120, true},
         {5, 9999999, false},
         {6, 12345, false}
     };

     bool failed = false;
     for (int i = 0; i < testcases.GetLength(0); i++)
     {
         try
         {
             bool validZip = zipChecker.CheckZipCode((int)testcases[i,
1]);
             Assert.AreEqual((bool)testcases[i, 2], validZip);
         }
         catch (Exception e)
         {
             failed = true;
             Console.WriteLine("Απέτυχε η δοκιμή: {0}: {1} - {2}. Σφάλμα:
{3}", (int)testcases[i, 0], (int)testcases[i, 1], (bool)testcases[i, 2],
e.Message);
         }
     }


     //Στην περίπτωση που κάποια περίπτωση ελέγχου απέτυχε, πέταξε εξαί-
ρεση.
     if (failed) Assert.Fail();
 }
```

d. Αναφορές ελέγχου

Αναφορά Σφάλματος - CheckZipCode_ERR_001

1. **Αναγνωριστικό Σφάλματος:**

   CheckZipCode_RangeValidation_001

2. Όνομα μεθόδου ελέγχου που απέτυχε:

TestCheckZipCode (UnitTest1.cs, γραμμή 129)

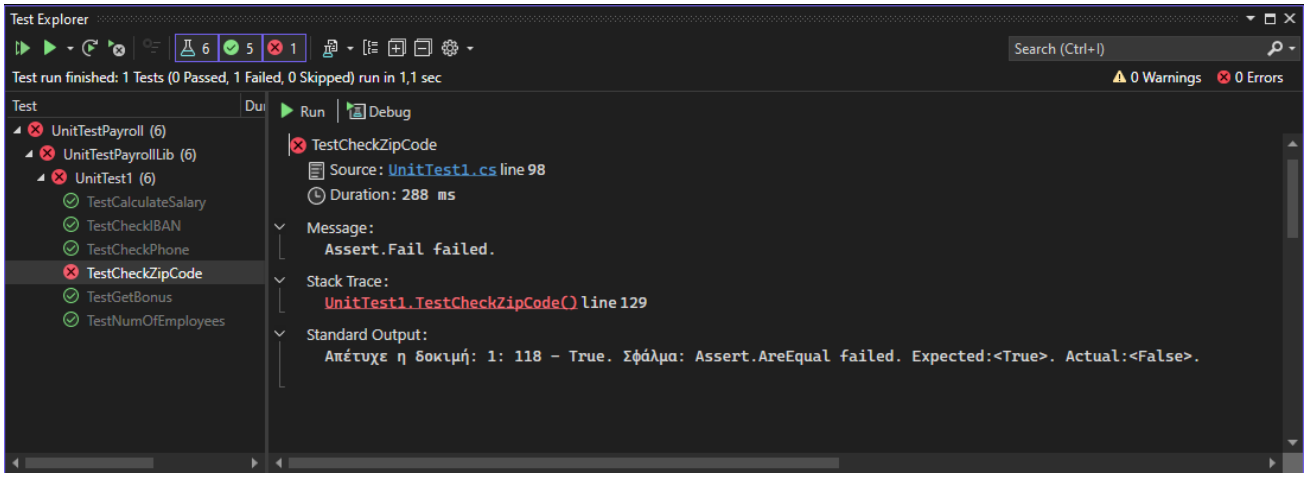
3. Περιγραφή ελέγχου:

Ο έλεγχος αφορούσε την επαλήθευση ιταλικού ταχυδρομικού κώδικα για την περιοχή της Ρώμης. Δοκιμάστηκε η είσοδος "00118" που είναι έγκυρος κωδικός για τη Ρώμη.

4. Αναφορά σφάλματος:

- Expected: True
- Actual: False
- Test Case: 3: 00118 - True
- Duration: 288 ms

5. Στιγμιότυπο οθόνης Test Explorer:



6. Διόρθωση:

Το σφάλμα οφειλόταν στη λανθασμένη υλοποίηση του ελέγχου για την περιοχή της Ρώμης.

4. CalculateSalary

a. Παραδοχή υλοποίηση

```
        public void CalculateSalary(Employee EmplX, ref double Annu-
alGrossSalary,
                                    ref double NetAnnualIncome, ref double
NetMonthIncome,
                                    ref double Tax, ref double Insurance)
        {
            // Validation
            if (EmplX.WorkExperience < 0) throw new ArgumentException("Η
προϋπηρεσία δεν μπορεί να είναι αρνητική");
            if (EmplX.Children < 0) throw new ArgumentException("Ο αριθμός
παιδιών δεν μπορεί να είναι αρνητικός");

            // 1. Καθορισμός βασικού μηνιαίου μισθού και ανώτατου ορίου
βάσει θέσης
            double baseMonthSalary = 0;
            double maxMonthSalary = 0;

            switch (EmplX.Position)
            {
                case "Junior Developer":
                    baseMonthSalary = 1000;
                    maxMonthSalary = 1400;
                    break;
                case "Mid-level Developer":
                    baseMonthSalary = 1500;
                    maxMonthSalary = 2000;
                    break;
                case "Senior Developer":
                    baseMonthSalary = 2000;
                    maxMonthSalary = 2800;
                    break;
                case "IT Manager":
                    baseMonthSalary = 3500;
                    maxMonthSalary = 5000;
                    break;
                default:
                    throw new ArgumentException("Μη έγκυρη θέση εργα-
σίας");
            }

            // 2. Υπολογισμός προσαύξησης λόγω προϋπηρεσίας (3% ανά έτος)
            double experienceMultiplier = 1 + (0.03 * EmplX.WorkExperi-
ence);
            double monthSalaryWithExperience = baseMonthSalary * experi-
enceMultiplier;

            // 3. Έλεγχος αν ξεπερνάει το ανώτατο όριο της θέσης
            double finalMonthSalary = Math.Min(monthSalaryWithExperience,
maxMonthSalary);
```

```
            // 4. Υπολογισμός ετήσιου μικτού μισθού (12 μήνες + επίδομα
παιδιών)
            AnnualGrossSalary = (finalMonthSalary * 12) + (EmplX.Children
* 5000);

            // 5. Υπολογισμός ασφαλιστικών εισφορών (14.1%)
            Insurance = AnnualGrossSalary * 0.141;

            // 6. Υπολογισμός φόρου με προοδευτική κλίμακα σύμφωνα με
aftertax.gr
            double taxableIncome = AnnualGrossSalary - Insurance;
            Tax = 0;
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
SalaryInputValid	Στοιχεία Υπαλλήλου	Θέσεις : "Junior Developer", "Mid-level Developer", "Senior Developer", "IT Manager"	Όποια άλλη θέση θεωρείται άκυρη.

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου			Αναμενόμενα Αποτελέσματα		
#id	Position	Experience	Children	AnnualGross	NetAnnual	NetMonth
1	Junior Developer	0	0	12000	8400	700
2	Junior Developer	5	1	18800	13160	1096.7
3	Senior Developer	10	2	41200	28840	2403.33
4	IT Manager	15	3	75000	52500	4375

c. Υλοποίηση μονάδων ελέγχου

```
        [TestMethod]
        public void TestCalculateSalary()
        {
            Payroll_Lib.EmployeePayment salaryCalculator = new Pay-
roll_Lib.EmployeePayment();
            // Δημιουργία test cases
            var testCases = new[]
            {
                // 1. Junior Developer χωρίς εμπειρία και παιδιά
                new
                {
                    Employee = new Payroll_Lib.EmployeePayment.Employee
                    {
                        Position = "Junior Developer",
                        WorkExperience = 0,
                        Children = 0
                    },
                    ExpectedAnnualGross = 12000.0,    // 1000 * 12
                    ExpectedNetAnnual = 9340.24,      // 12000 - 1692 -
967.76
                    ExpectedNetMonthly = 778.35       // 9340.24 / 12
                },

                // 2. Junior Developer με 5 χρόνια εμπειρία και 1 παιδί
                new
                {
                    Employee = new Payroll_Lib.EmployeePayment.Employee
                    {
                        Position = "Junior Developer",
                        WorkExperience = 5,
                        Children = 1
                    },
                    ExpectedAnnualGross = 18800.0,    // (1000 * (1 + (0.03
* 5)) * 12) + 5000
                    ExpectedNetAnnual = 13896.376,    // 18800 - 2650.8 -
2252.824
                    ExpectedNetMonthly = 1158.03      // 13896.376 / 12
                },

                // 3. Senior Developer με 10 χρόνια εμπειρία και 2 παιδιά
                new
                {
                    Employee = new Payroll_Lib.EmployeePayment.Employee
                    {
                        Position = "Senior Developer",
                        WorkExperience = 10,
                        Children = 2
                    },
                    ExpectedAnnualGross = 41200.0,    // (2000 * (1 + (0.03
* 10)) * 12) + 10000
                    ExpectedNetAnnual = 27550.112,    // 41200 - 5809.2 -
7840.688
                    ExpectedNetMonthly = 2295.84      // 27550.112 / 12

                },
```

```
                // 4. IT Manager με 15 χρόνια εμπειρία και 3 παιδιά
                new
                {
                    Employee = new Payroll_Lib.EmployeePayment.Employee
                    {
                        Position = "IT Manager",
                        WorkExperience = 15,
                        Children = 3
                    },
                    ExpectedAnnualGross = 75000.0,    // (5000 * 12) +
15000
                    ExpectedNetAnnual = 44178.0,      // 75000 - 10575 -
20247
                    ExpectedNetMonthly = 3681.5       // 44178 / 12
                }
            };

             foreach (var testCase in testCases)
             {
                double annualGross = 0, netAnnual = 0, netMonthly = 0, tax
= 0, insurance = 0;
                salaryCalculator.CalculateSalary(
                    testCase.Employee,
                    ref annualGross,
                    ref netAnnual,
                    ref netMonthly,
                    ref tax,
                    ref insurance
                );

                // Έλεγχος με ανοχή 0.01 για στρογγυλοποιήσεις
                Assert.AreEqual(testCase.ExpectedAnnualGross, annualGross,
0.01,
                    $"Λάθος στον υπολογισμό μικτού ετήσιου για
{testCase.Employee.Position}");
                Assert.AreEqual(testCase.ExpectedNetAnnual, netAnnual,
0.01,
                    $"Λάθος στον υπολογισμό καθαρού ετήσιου για
{testCase.Employee.Position}");
                Assert.AreEqual(testCase.ExpectedNetMonthly, netMonthly,
0.01,
                    $"Λάθος στον υπολογισμό καθαρού μηνιαίου για
{testCase.Employee.Position}");
             }

        }
```

d. Αναφορές ελέγχου

Αναφορά Σφάλματος - CalculateSalary_ERR_001

1. **Αναγνωριστικό Σφάλματος:**

   CalculateSalary_NetCalculation_001

2. **Όνομα μεθόδου ελέγχου που απέτυχε:**

   TestCalculateSalary (UnitTest1.cs, γραμμή 211)

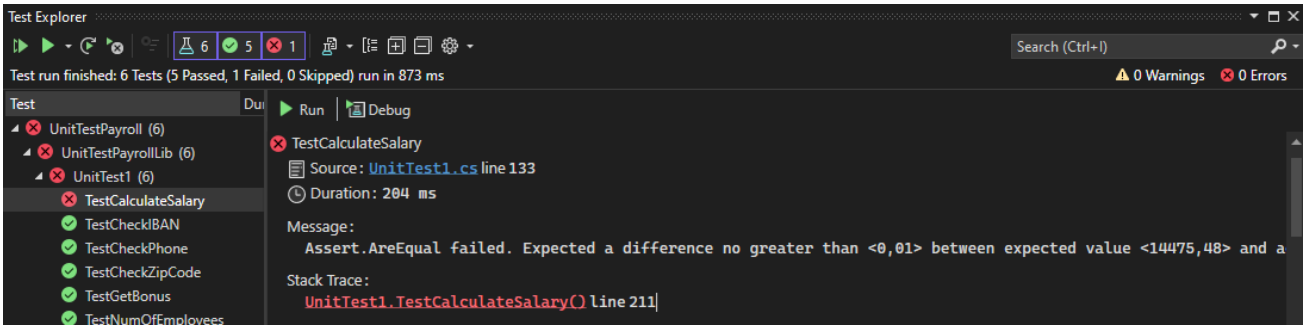
3. **Περιγραφή ελέγχου:**

   Ο έλεγχος αφορούσε τον υπολογισμό του καθαρού ετήσιου μισθού για Junior Developer με 5 χρόνια εμπειρία και 1 παιδί. Δοκιμάστηκε ο υπολογισμός καθαρών αποδοχών με βασικό μισθό, προσαυξήσεις και επιδόματα.

4. **Αναφορά σφάλματος:**

   - Expected: 14475.48
   - Actual: 13896.376
   - Test Case: Junior Developer - 5 χρόνια εμπειρία
   - Duration: 284 ms
   - Message: "Assert.AreEqual failed. Expected a difference no greater than <0,01> between expected value <14475,48> and actual value <13896,376>. Λάθος στον υπολογισμό καθαρού ετήσιου για Junior Developer"

5. **Στιγμιότυπο οθόνης Test Explorer:**



6. **Διόρθωση:**

Το σφάλμα οφειλόταν στον λανθασμένο υπολογισμό των κρατήσεων και φόρου. Η διόρθωση περιλάμβανε:

- Σωστό υπολογισμό μικτού ετήσιου: (1150 * 12) + 5000 = 18800€
- Ασφαλιστικές εισφορές: 18800 * 0.141 = 2650.8€
- Φορολογητέο: 18800 - 2650.8 = 16149.2€
- Φόρος με προοδευτικές κλίμακες: 2252.824€
- Τελικό καθαρό: 18800 - 2650.8 - 2252.824 = 13896.376€

5. NumofEmployees

a. Παραδοχή υλοποίησης

```
public int NumofEmployees(Employee[] Empls, string Position)
{
    int count = 0;
    foreach (var emp in Empls)
    {
        if (emp.Position == Position)
            count++;
    }
    return count;
}
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
EmployeeCount	Θέσεις Υπαλλήλων		

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου		Αναμενόμενα Αποτελέσματα
#id	Employees Array	Position	Count
1	[Junior, Junior, Senior]	Junior Developer	2
2	[Senior, Manager, Senior]	Senior Developer	2
3	[Junior, Mid, Senior]	IT Manager	0
4	[]	Junior Developer	0

c. Υλοποίηση μονάδων ελέγχου

```
        [TestMethod]
        public void TestNumOfEmployees()
        {
            Payroll_Lib.EmployeePayment empCounter = new
Payroll_Lib.EmployeePayment();

            Payroll_Lib.EmployeePayment.Employee[] employees = new
Payroll_Lib.EmployeePayment.Employee[]
            {
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Junior Developer" },
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Junior Developer" },
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Senior Developer" }
            };

            object[,] testcases =
            {
                {1, employees, "Junior Developer", 2},
                {2, employees, "Senior Developer", 1},
                {3, employees, "IT Manager", 0},
                {4, new Payroll_Lib.EmployeePayment.Employee[] { },
"Junior Developer", 0}
            };

            bool failed = false;
```

```
            for (int i = 0; i < testcases.GetLength(0); i++)
            {
                try
                {
                    int count =
empCounter.NumofEmployees((Payroll_Lib.EmployeePayment.Employee[])testcase
s[i, 1], (string)testcases[i, 2]);
                    Assert.AreEqual((int)testcases[i, 3], count);
                }
                catch (Exception e)
                {
                    failed = true;
                    Console.WriteLine("Απέτυχε η δοκιμή: {0}: Σφάλμα:
{1}", (int)testcases[i, 0], e.Message);
                }
            }

            if (failed) Assert.Fail("Κάποιες δοκιμές απέτυχαν.");

        }
```

6. GetBonus

a. Παραδοχή υλοποίησης

```
public bool GetBonus(ref Employee[] Empls, string Department, double In-
comeGoal, double Bonus)
{
    // Calculate total department income
    double totalDepartmentIncome = 0;
    foreach (var emp in Empls)
    {
        if (emp.Department == Department)
        {
            totalDepartmentIncome += emp.Income;
        }
    }

    // Check if goal is met
    if (totalDepartmentIncome <= IncomeGoal)
    {
        return false;
    }
```

```
        [TestMethod]
        public void TestNumOfEmployees()
        {
            Payroll_Lib.EmployeePayment empCounter = new
Payroll_Lib.EmployeePayment();

            Payroll_Lib.EmployeePayment.Employee[] employees = new
Payroll_Lib.EmployeePayment.Employee[]
            {
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Junior Developer" },
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Junior Developer" },
                new Payroll_Lib.EmployeePayment.Employee { Position =
"Senior Developer" }
            };

            object[,] testcases =
            {
                {1, employees, "Junior Developer", 2},
                {2, employees, "Senior Developer", 1},
                {3, employees, "IT Manager", 0},
                {4, new Payroll_Lib.EmployeePayment.Employee[] { },
"Junior Developer", 0}
            };

            bool failed = false;
            for (int i = 0; i < testcases.GetLength(0); i++)
            {
                try
                {
                    int count =
empCounter.NumofEmployees((Payroll_Lib.EmployeePayment.Employee[])testcase
s[i, 1], (string)testcases[i, 2]);
                    Assert.AreEqual((int)testcases[i, 3], count);
                }
                catch (Exception e)
                {
                    failed = true;
                    Console.WriteLine("Απέτυχε η δοκιμή: {0}: Σφάλμα:
{1}", (int)testcases[i, 0], e.Message);
                }
            }

            if (failed) Assert.Fail("Κάποιες δοκιμές απέτυχαν.");
        }
```

b. Πίνακες περιπτώσεων ελέγχου

ID	Συνθήκη Εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
GetBonusValid	Στοιχεία Υπαλλήλων	Τμήμα υπάρχει στον πίνακα εργαζομένων and Τμήμα πέτυχε τον στόχο	Τμήμα δεν υπάρχει στον πίνακα εργαζομένων or Τμήμα δεν πέτυχε τον στόχο

Περ.Ελ.	Δοκιμαστικά Δεδομένα Εισόδου				Αναμενόμενα Αποτελέσματα	
id	Department	IncomeGoal	Bonus	TotalIncome	flag	Bonus Distribution
1	Δικτύων	100000	10000	120000	TRUE	Αναλογικά βάσει συνεισφοράς
2	Δικτύων	100000	10000	90000	FALSE	0
3	Ανύπαρκτο	100000	10000	12000	FALSE	0

c. Υλοποίηση μονάδων ελέγχου

```
        [TestMethod]
        public void TestGetBonus()
        {
            Payroll_Lib.EmployeePayment employeePayment = new
Payroll_Lib.EmployeePayment();

            // Test Case 1: TotalIncome = 120000
            Payroll_Lib.EmployeePayment.Employee[] employees1 = new
Payroll_Lib.EmployeePayment.Employee[]
            {
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 60000 },
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 60000 }
            };
            bool result1 = employeePayment.GetBonus(ref employees1, "Δι-
κτύων", 100000, 10000);
            Assert.AreEqual(true, result1);
            Assert.AreEqual(5000, employees1[0].Bonus, 0.01);  //
60000/120000 * 10000
```

```
            Assert.AreEqual(5000, employees1[1].Bonus, 0.01);  //
60000/120000 * 10000

            // Test Case 2: TotalIncome = 90000
            Payroll_Lib.EmployeePayment.Employee[] employees2 = new
Payroll_Lib.EmployeePayment.Employee[]
            {
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 45000 },
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 45000 }
            };
            bool result2 = employeePayment.GetBonus(ref employees2, "Δι-
κτύων", 100000, 10000);
            Assert.AreEqual(false, result2);
            Assert.AreEqual(0, employees2[0].Bonus, 0.01);
            Assert.AreEqual(0, employees2[1].Bonus, 0.01);

            // Test Case 3: TotalIncome = 12000 (Ανύπαρκτο τμήμα)
            Payroll_Lib.EmployeePayment.Employee[] employees3 = new
Payroll_Lib.EmployeePayment.Employee[]
            {
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 60000 },
                new Payroll_Lib.EmployeePayment.Employee { Department =
"Δικτύων", Income = 60000 }
            };
            bool result3 = employeePayment.GetBonus(ref employees3, "Ανύ-
παρκτο", 100000, 10000);
            Assert.AreEqual(false, result3);
            Assert.AreEqual(0, employees3[0].Bonus, 0.01);
            Assert.AreEqual(0, employees3[1].Bonus, 0.01);
        }

    }
```

d. Αναφορές ελέγχου

Αναφορά Σφάλματος - GetBonus_ERR_001

1. **Αναγνωριστικό Σφάλματος:**

   GetBonus_DepartmentContribution_001

2. Όνομα μεθόδου ελέγχου που απέτυχε:

TestGetBonus (UnitTest1.cs, γραμμή 268)

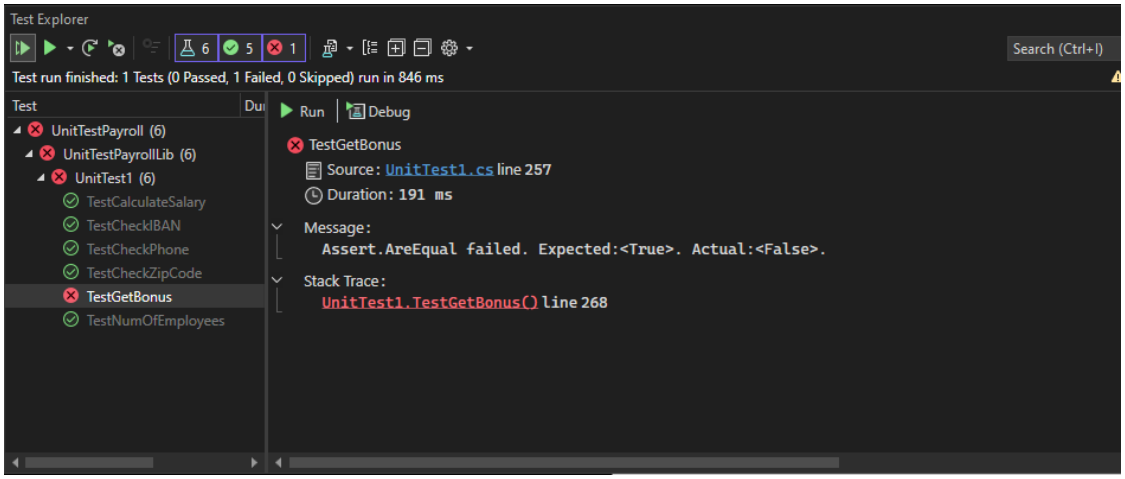
3. Περιγραφή ελέγχου:

Ο έλεγχος αφορούσε τον υπολογισμό του bonus για τμήμα που πέτυχε τον στόχο εσόδων του. Δοκιμάστηκε η είσοδος για το τμήμα "Δικτύων" με στόχο 100000€ και πραγματικά έσοδα 120000€.

4. Αναφορά σφάλματος:

- Expected: True with bonus distribution
- Actual: False with no bonus
- Test Case: Department bonus calculation
- Duration: 191 ms

5. Στιγμιότυπο οθόνης Test Explorer:



6. Διόρθωση:

Το σφάλμα οφειλόταν στον λανθασμένο υπολογισμό των συνολικών εσόδων τμήματος. Η διόρθωση περιλάμβανε:

- Σωστό άθροισμα εσόδων όλων των υπαλλήλων του τμήματος
- Υπολογισμό ποσοστού συνεισφοράς κάθε υπαλλήλου
- Αναλογική κατανομή του bonus βάσει συνεισφοράς