2017


**ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ**

**ΙΔΡΥΜΑ ΗΠΕΙΡΟΥ**

**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.**


# [ΕΛΕΓΧΟΣ ΜΟΝΑΔΩΝ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΠΡΟΚΑΘΟΡΙΣΜΕΝΑ ΔΕΔΟΜΕΝΑ ]

Software Unit Testing with Data Driven Tests


Αντώνης Ψωμάς

Επιβλέπων Αδάμ Σταύρος,


Αθήνα, Σεπτέμβριος, 2017

**ΤΕΙ ΗΠΕΙΡΟΥ**

**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ**

**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# ΕΛΕΓΧΟΣ ΜΟΝΑΔΩΝ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΠΡΟΚΑΘΟΡΙΣΜΕΝΑ ΔΕΔΟΜΕΝΑ

Αντώνης Ψωμάς

Επιβλέπων Αδάμ Σταύρος,

Αθήνα, Σεπτέμβριος, 2017

**SOFTWARE UNIT TESTING WITH DATA DRIVEN TESTS**

# ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Ψωμάς, Αντώνης

……

# ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ θερμά τον καθηγητή Αδάμ Σταύρο για την εμπιστοσύνη του αναθέτοντας μου την παρούσα πτυχιακή εργασία καθώς και για την καθοδήγηση του στην πτυχιακή. Ευχαριστώ επίσης και το τμήμα Μηχανικών Πληροφορικής του ΤΕΙ Ηπείρου για την συμβολή του στην εκπαίδευση όλα αυτά τα χρόνια.

# ΠΕΡΙΛΗΨΗ

Το Unit Test είναι μια συνάρτηση η οποία ελέγχει ένα κομμάτι κώδικα για την ορθότητα του, οι συναρτήσεις αυτές ονομάζονται μέθοδοι ελέγχου. Στόχος του Unit Test είναι να βοηθήσει τον προγραμματιστή να ελέγξει αν το πρόγραμμα του φέρει το επιθυμητό αποτέλεσμα εύκολα και γρήγορα.

Ο κώδικας των προγραμματιστών διαφέρει από άτομο σε άτομο και το μόνο μέτρο σύγκρισης είναι να δουλεύει ο κώδικας και να παράγει το ορθό αποτέλεσμα σε όσο τον δυνατόν λιγότερο χρόνο εκτέλεσης του. Αυτό επιτυγχάνεται δοκιμάζοντας τον κώδικα και ελέγχοντας το αποτέλεσμα που επιστρέφει εάν είναι το επιθυμητό και μετρώντας τον χρόνο εκτέλεσης του. Αυτήν ακριβώς την μέθοδο υλοποιεί το Unit Test, εξοικονομώντας χρόνο για τον προγραμματισμό ενός προγράμματος χωρίς να αναλώνεται πολύτιμος χρόνος στις δοκιμές του κώδικα σε κάθε σημείο. Κατακερματίζοντας ο προγραμματιστής τον κώδικα σε μεθόδους ελέγχου οι οποίες ελέγχουν τα αποτελέσματα για κάθε πιθανή παράμετρο και επιστρέφουν εάν πέρασαν ή όχι το Unit Test. Ο λόγος για τον οποίο προτιμάται ένα Unit Test είναι επειδή σε ένα πρόγραμμα με πολλές γραμμές κώδικα μπορούμε να εισάγουμε ένα Unit Test για να ελέγξει την ορθότητα του για όλα τα πιθανά αποτελέσματα βλέποντας αν τελικά το πρόγραμμα λειτουργεί και αν όχι πού αποτυγχάνει χωρίς να χρειάζεται να τρέξει ολόκληρο το πρόγραμμα ή η εφαρμογή. Συνεπώς μπορούμε να ελέγξουμε το πρόγραμμα πιο τακτικά και πιο αποτελεσματικά ανά πάσα στιγμή.

Η παρούσα πτυχιακή χωρίζεται σε δύο μέρη, το πρώτο αποτελεί το θεωρητικό τμήμα στο οποίο αναλύονται τα Unit Tests, η φιλοσοφία τους και η χρηστικότητα τους και με ποιούς τρόπους ή εργαλεία επιτυγχάνονται και ποιες είναι οι κύριες μέθοδοι τους.

Στο δεύτερο μέρος παρουσιάζεται η ανάπτυξη ενός προγράμματος στο οποίο θα εφαρμόσουμε το Unit Test για να υποδείξει αν το πρόγραμμα επιστρέφει το σωστό αποτέλεσμα για κάθε παράμετρο που δέχεται.

Ένα επί πλέον αποτέλεσμα της πτυχιακής εργασίας είναι να ενημέρωση για τις νέες τεχνικές ελέγχου λογισμικού και τον τρόπο χρήσης τους και λειτουργίας τους καθώς αποτελούν μια νέα και άγνωστη τεχνική στον χώρο του προγραμματισμού.

**Λέξεις-κλειδιά:** έλεγχος μονάδων λογισμικού, μέθοδος ελέγχου, έλεγχος μονάδων λογισμικού με προκαθορισμένα δεδομένα, ανάθεση, δράση, βεβαίωση.

# ABSTRACT

Unit Testing is a function which is used to test if a part of computer source code is correct. The main objective is to help the programmer to test easily if a part of code has the intended result.

Programming differs from person to person and the only way to compare different implementations is to test if they deliver the intended results. This is achieved by testing if the code delivers the expected results. Unit testing helps to write a program without losing valuable time at debugging at every new line of code. The source code is organized in control test methods which test results for every possible parameter and return a boolean result indicating if a test is successful or fails. Unit tests are popular because they can be used to test many input parameters and in case of failures, they can deliver traceable error messages without having to fully run the application. Consequently, a program with unit testing can be tested regularly and fast and more often at any point of the program development.

This thesis consists of two parts. In the first part, the theory and background of unit testing, the usage and the practical tools are presented and analyzed. In the second part, we apply the Unit testing method to the development of a program. The Unit Test indicates if the program delivers the expected result depending on its input parameters. An additional contribution of this thesis is that information is provided for new software testing technics, their usage and their operation.



**Keywords:** Unit Test**,** Software Unit Testing, Test Method, Data Driven Tests, Arrange, Act, Assert.

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

# ΓΛΩΣΣΑΡΙΟ

- Unit Test (UT): Έλεγχος Μονάδων Λογισμικού
- Data Driven Tests (DDT): Έλεγχος με προκαθορισμένα δεδομένα
- Test Driven Development (TDD): Ανάπτυξη Κατευθυνόμενη από τους Ελέγχους
- Arrange: Ανάθεση
- Act: Δράση
- Assert: Βεβαίωση
- Test First Development (TFD): Τεχνική ανάπτυξης κώδικα προγραμματισμού
- Extreme Programming (XP): Ακραίος Προγραμματισμός

# ΕΙΣΑΓΩΓΗ

Στην παρούσα πτυχιακή εργασία εμπεριέχονται όλες οι πληροφορίες για το πώς να επιτύχουμε τον έλεγχο ενός τμήματος λογισμικού και ποίες οι μέθοδοι και τα εργαλεία που χρειάζονται. Συνοπτικά το unit test είναι ο έλεγχος που πραγματοποιείται σε έναν κώδικα προγραμματισμού σε μορφή συνάρτησης η οποία με την εκσφαλμάτωση της μας ενημερώνει αν το αποτέλεσμα που επιστρέφει είναι το επιθυμητό. Επειδή όμως δεν μπορεί ο προγραμματιστής να εκσφαλμάτωση μόνο ένα κομμάτι κώδικα εάν δεν έχει ολοκληρωθεί ο προγραμματισμός ολόκληρου του προγράμματος δημιουργήθηκαν τα unit test για αυτόν τον σκοπό. Πλέον μπορεί κάποιος να εισάγει τέτοιες μεθόδους σε οποιαδήποτε γλώσσα προγραμματισμού και στα περισσότερα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού.

Για την υλοποίηση των Unit Tests της πτυχιακής εργασίας χρησιμοποιήθηκε το Microsoft Visual Studio Professional 2013 το οποίο υπήρξε το πρώτο περιβάλλον που ενσωμάτωσε την δυνατότητα δημιουργίας τους. Ένα από τα αντικείμενα της πτυχιακής ήταν να δείξει πως είναι δυνατόν να δημιουργηθούν προγράμματα τα όποια θα μπορούν να ελέγχουν άλλα προγράμματα για την ορθότητα τους και να παρέχουν την πληροφορία σχετικά με το αν επιτυγχάνουν το επιθυμητό αποτέλεσμα ή όχι. Η επίτευξη αυτού του στόχου της πτυχιακής θα μπορούσε να αποτελέσει τη βάση ώστε να δημιουργηθεί το κατάλληλο πλαίσιο για την αυτόματη διόρθωση εργασιών ή την εξέταση σε μαθήματα προγραμματισμού. Έτσι θα μπορούν να αξιοποιηθούν τέτοια αυτοματοποιημένα προγράμματα και να βρεθεί ένα αξιόλογο πεδίο εφαρμογής στους κατά την διδασκαλία και την εκμάθηση προγραμματισμού εφαρμογών στην εκπαίδευση. Μια άλλη εφαρμογή των αποτελεσμάτων της πτυχιακής θα ήταν η δημιουργία μιας εφαρμογής η οποία θα μπορεί να βοηθήσει ακόμα και τον φοιτητή να αξιολογήσει τις γνώσεις του χωρίς την φυσική παρουσία του καθηγητή σε μια αίθουσα άπλα μεταφορτώνοντας το πρόγραμμα του σε αυτήν την εφαρμογή.

Στην παρούσα πτυχιακή έχει υλοποιηθεί ένα πρότζεκτ μέσω του περιβάλλοντος ανάπτυξης λογισμικού της Microsoft, το Visual Studio 2013, σε γλώσσα προγραμματισμού C#. Το πρότζεκτ αποτελεί μια κλάση βιβλιοθήκης της γλώσσας C# (Class Library) και περιλαμβάνει τέσσερις συναρτήσεις που θεωρούνται θεμελιώδεις στην εκπαίδευση του προγραμματισμού. Οι συναρτήσεις αυτές είναι ο αλγόριθμος εύρεσης μέσου όρου, ο αλγόριθμος ελάχιστου και μέγιστου, ο αλγόριθμος ταξινόμησης φυσαλίδας και ο

αλγόριθμος δυαδικής αναζήτησης. Σε αυτό το πρότζεκτ θα εφαρμόσουμε Unit Test για να ελέγξουμε αν οι συναρτήσεις που δημιουργήσαμε επιστρέφουν το επιθυμητό αποτέλεσμα για κάθε τιμή που θα παίρνουν ως παράμετρο.

# 1. ΘΕΩΡΗΤΙΚΗ ΕΠΙΣΚΟΠΗΣΗ

## 1.1 UNIT TESTS

Πότε όμως προέκυψε η ανάγκη για την δημιουργία του Unit Test στο λογισμικό και πώς επιτεύχθηκε; Η λειτουργία που προσφέρουν δεν διαφέρει καθόλου από τον τρόπο ανάπτυξης ενός προγράμματος από έναν προγραμματιστή λογισμικού η οποία δεν είναι άλλη από την ικανότητα να καθορίσει με την ανθρώπινη όραση και σκέψη πότε ένας κώδικας θα συμπεριφερθεί σωστά. Ο προγραμματιστής είναι αυτός που καθορίζει μια συμπεριφορά ως σωστή ή λανθασμένη και αυτός είναι που θα εξετάσει κάθε σενάριο στο οποίο υπάρχει πιθανότητα αποτυχίας του κώδικα του.

Η ανάγκη υπήρχε από τα πρώτα χρόνια του προγραμματισμού απλώς καλυπτόταν με διαφορετικούς τρόπους. Όλοι οι προγραμματιστές πριν ξεκινήσουν ένα πρόγραμμα σε καινούργιο περιβάλλον ανάπτυξης ή ένα πρόγραμμα σε μια νέα γλώσσα προγραμματισμού που δεν γνωρίζουν συνηθίζουν να δημιουργούν πρώτα το πιο απλό πρόγραμμα ενός αρχάριου, το ‘Hello World!’ για να το δοκιμάσουν. Το ‘Hello World!’ είναι μια συνάρτηση που καλείται από ένα πρόγραμμα και η μόνη εντολή που περιλαμβάνει είναι η εκτύπωση του μηνύματος ‘Hello World!’ στην οθόνη. Η κίνηση που περιγράφηκε και περιλαμβάνει την συνάρτηση ‘Hello World!’ θα μπορούσε να χαρακτηριστεί ως ένα unit test καθώς ο προγραμματιστής βλέποντας το μήνυμα της συνάρτησης εκτυπωμένο στην οθόνη του καταλαβαίνει πως το αποτέλεσμα της είναι το επιθυμητό. Ο προγραμματιστής αφού κατανοήσει τον τρόπο λειτουργείας της νέα γλώσσας προγραμματισμού και πειραματιστεί με την προαναφερόμενη συνάρτηση στην συνέχεια θα την διαγράψει και θα ξεκινήσει τον προγραμματισμό ενός νέου προγράμματος πιο σύνθετου από μια απλή συνάρτηση με μια εντολή. Η διαφορά μιας συνάρτησης του Unit Test όμως από μια απλή συνάρτηση είναι πως περιέχεται σε διαφορετική κλάση από το πρόγραμμα που θέλουμε να ελέγξουμε τα αποτελέσματα του. Συνεπώς δεν χρειάζεται να διαγραφεί αυτή η συνάρτηση καθώς δεν επηρεάζει καθόλου την δομή του προγράμματος της νέας γλώσσας που δοκιμάζουμε. Επίσης πολλοί προγραμματιστές έχουν την τάση να μην χρησιμοποιούν σχολιασμό ανάμεσα στον κώδικα τους για διάφορους λόγος ο καθένας και με την

ολοκλήρωση ενός προγράμματος ή μιας εφαρμογής ο κώδικας τους να είναι δυσανάγνωστος ακόμα και για τον ίδιο τον δημιουργό του προγράμματος με το πέρασμα του χρόνου. Σε περίπτωση που ένας άλλος προγραμματιστής κληθεί να επιλύσει ένα τυχόν σφάλμα της εφαρμογής μετά από διάστημα θα αντιμετωπίσει δυσκολίες στην εύρεση της αιτίας του σφάλματος και ακόμα και αν το διορθώσει δεν θα γνωρίζει με σιγουριά πως δεν θα επαναληφθεί ακόμα και σε άλλο σημείο του κώδικα. Με την ενσωμάτωση Unit Test στα προγράμματα μας μπορούμε να ελέγξουμε περισσότερες φορές τον κώδικα μας και με διαφορετικές παραμέτρους κάθε φορά μειώνοντας την πιθανότητα μετά την ολοκλήρωση του προγράμματος ή της εφαρμογής να εμφανιστεί κάποιο σφάλμα.

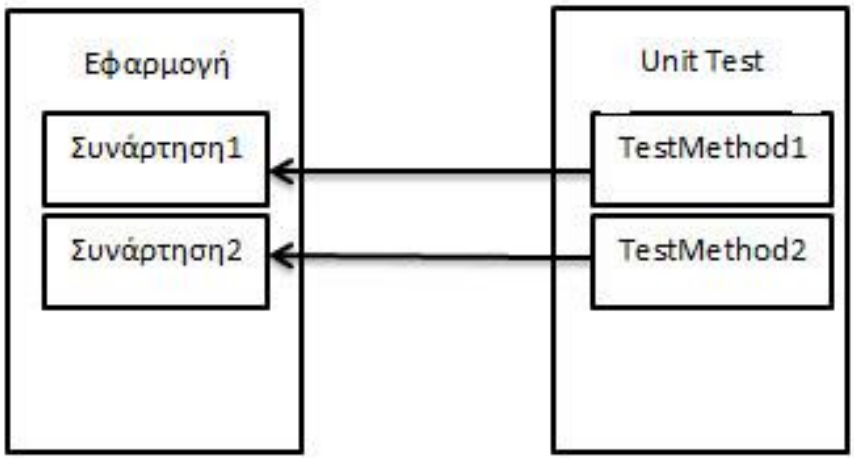
Όταν χρησιμοποιούμε UT για να ελέγξουμε το αποτέλεσμα του προγράμματος μας είναι πιο πιθανό να οδηγηθούμε σε ένα υψηλής ποιότητας πρόγραμμα το όποιο θα περιλαμβάνει συναρτήσεις όπου θα έχουν ελεγχθεί διεξοδικά για κάθε πιθανό αποτέλεσμα, εξαλείφοντας την πιθανότητα σφάλματος κατά την ολοκλήρωση του προγράμματος. Επίσης το Unit Test το οποίο αποτελείται από μεθόδους ελέγχου σε μια κλάση αποτελεί ένα πιο ευανάγνωστο κώδικα μεθοδικά μοιρασμένο. Αυτή η δομή κάνει το Unit Test εύκολο στην ανάγνωση του και στην περεταίρω ανάπτυξη του από διαφορετικούς προγραμματιστές.

Τα Unit Tests προέκυψαν μέσα από τον Extreme Programming (XP) και στηρίζονται σε μεθόδους ελέγχου. Ο προγραμματιστής Kent Beck είναι ο άνθρωπος που ανέπτυξε την τεχνική του Extreme Programming ενώ ταυτόχρονα ήταν επικεφαλής έργου στην Chrysler Comprehensive Compensation για ένα μακροπρόθεσμο πρόγραμμα για τον προγραμματισμό εκ νέου της εφαρμογής της μισθοδοσίας της Chrysler. Στην συνέχεια διατύπωσε την μεθοδολογία ανάπτυξης σε ένα βιβλίο με τίτλο Extreme Programming Explained: Embrace Change. (Copeland, 2001)

Οι πρακτικές που συναντώνται στον XP είναι η πρώιμη συγγραφή μεθόδων ελέγχου, η συχνή ανακατασκευή της εφαρμογής και η συνεχής δοκιμή σε κάθε ενσωμάτωση νέου κώδικα, η βαθμιαία ανάπτυξη και ο επαναληπτικός προγραμματισμός. (Beck, 1999)

Συνοψίζοντας ως Unit Tests ορίζονται τα εργαλεία λογισμικού τα οποία επιτρέπουν τον έλεγχο ενός προγράμματος ή τμήματος κώδικα και αναφέρουν το αποτέλεσμα του. Το αποτέλεσμα αυτό είναι τύπου Boolean και επιστρέφει αληθές σε περίπτωση όπου επιτύχει το test και ψευδές σε διαφορετική περίπτωση. Αποτελεί μια νέα εισαγωγή στα περιβάλλοντα προγραμματισμού και υποστηρίζεται σε όλες τις γλώσσες προγραμματισμού με προσθήκη των κατάλληλων βιβλιοθηκών UT. Δεν μπορεί όμως να χαρακτηριστεί κάτι

εντελώς καινούργιο στον τομέα της πληροφορικής καθώς είναι μια πρακτική που είναι όμοια με τους ελέγχους που πραγματοποιούνται καθώς ολοκληρώνεται ο προγραμματισμός μιας εφαρμογής ή ακόμα και η συγγραφή δύο, τριών γραμμών κώδικα απλώς και μόνο για να δοκιμαστεί μια νέα γλώσσα προγραμματισμού. Τα Unit Test δεν αποτελούν τμήμα της εφαρμογής που δοκιμάζουν, δηλαδή δεν ενσωματώνονται μέσα στον κώδικα, προγραμματίζονται παράλληλα όμως με αυτό και συνδέονται μεταξύ τους στην συνέχεια (Hamill, 2005), όπως φαίνεται και στην παρακάτω εικόνα. (εικόνα 1)




Εικόνα 1

Ένα Unit Test αποτελείται από τουλάχιστον μια κλάση ελέγχου (Test Class) και μια μέθοδο ελέγχου (Test Method), κάθε μέθοδος διακρίνεται από τρία βασικά χαρακτηριστικά την ανάθεση (Arrange), την δράση (Act) και την βεβαίωση (Assert) γνωστά και ως τα τρία Άλφα (AAA) από την αγγλική γλώσσα. Στην ανάθεση ορίζουμε τις μεταβλητές οι οποίες θα χρησιμοποιηθούν ως παράμετροι στη συνάρτηση του προγράμματος που θέλουμε να ελέγξουμε. Ενώ στη δράση ορίζουμε τις μεταβλητές που θα χρησιμοποιηθούν ως παράμετροι στην βεβαίωση και είναι πάντα δύο μεταβλητές. Η μια μεταβλητή περιέχει το επιθυμητό αποτέλεσμα και η άλλη το πραγματικό αποτέλεσμα. Συνηθίζεται η μεταβλητή που θα περιέχει το επιθυμητό αποτέλεσμα να ονομάζεται ‘αναμενόμενη’ (expected) και η μεταβλητή που θα περιέχει το πραγματικό αποτέλεσμα της συνάρτησης που ελέγχτηκε από το Unit Test ‘πραγματική’ (actual). Τέλος στη βεβαίωση οι μεταβλητές που ορίστηκαν στην δράση συγκρίνονται ως προς την τιμή τους

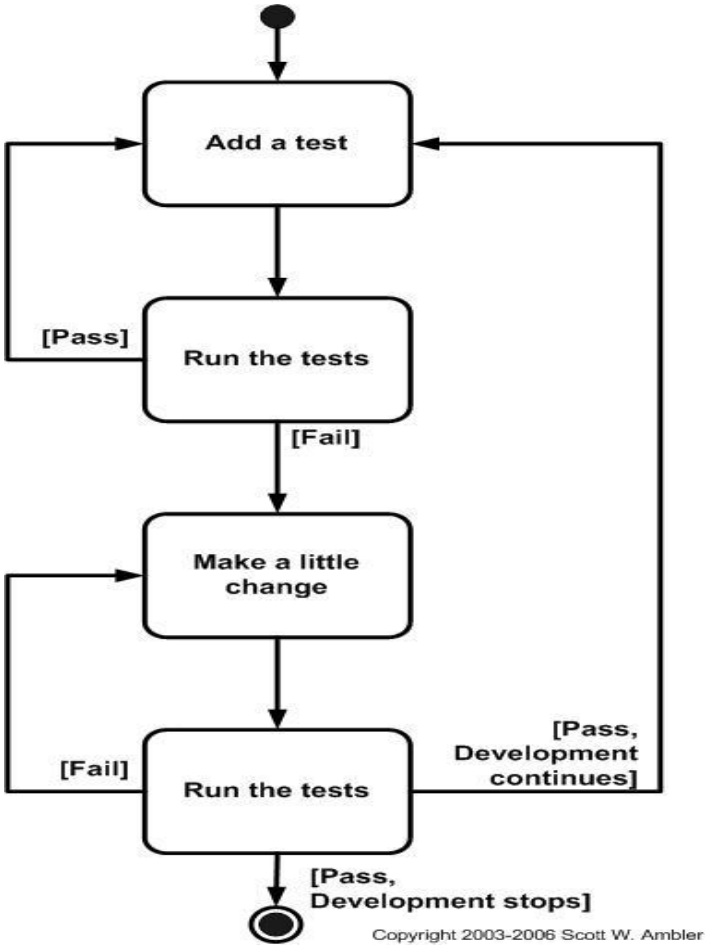
αν είναι η ίδια. Συνεπώς αν οι τιμές είναι ίσες τότε η συνάρτηση του προγράμματος επιστρέφει τα αναμενόμενα αποτελέσματα, ενώ σε διαφορετική περίπτωση η συνάρτηση αποτυγχάνει στον προγραμματισμό της και επιστρέφει λανθασμένες τιμές. Η βέλτιστη πρακτική που συνιστάται είναι κάθε μέθοδος ελέγχου να δοκιμάζεται και με ψευδώς θετική ή ψευδώς αρνητική τεχνική. Η τεχνική ψευδώς αρνητική είναι όταν ο προγραμματιστής σκόπιμα οδηγεί σε σφάλμα τον κώδικα του και αντιθέτως ψευδώς θετική όταν ο προγραμματιστής εν γνώσει του προσπερνάει ένα σφάλμα στον κώδικα του. Η τεχνική ψευδώς θετική συναντάται συχνά στον προγραμματισμό σε γλώσσα PHP όπου για να αποκρύψει διάφορα μηνύματα προειδοποίησης σε μια ιστοσελίδα ο προγραμματιστής χρησιμοποιεί τον χαρακτήρα '@' μπροστά από μεταβλητές ή συναρτήσεις. Παράδειγμα ψευδώς αρνητικής τεχνικής είναι η σύγκριση δύο μεταβλητών που περιέχουν τους ακέραιους αριθμούς 2 και 5 αντίστοιχα πως είναι ίσοι. Με αυτόν τον τρόπο ο προγραμματιστής μπορεί να ελέγξει πέρα από την συνάρτηση του προγράμματος και την ίδια την μέθοδο ελέγχου πως μεταχειρίζεται σωστά τις μεταβλητές που δημιουργεί.

Χρησιμοποιώντας το Unit Test έχουμε ως στόχο όσον δυνατόν μεγαλύτερο ποσοστό του προγράμματος μας να ελέγχεται και από μια μέθοδο ελέγχου καθώς είτε στο τέλος του προγραμματισμού είτε κατά την διάρκεια του μπορούμε να εκτελέσουμε το Unit Test και να μας ενημερώσει ποιες μέθοδοι ελέγχου αποτύχαν. Αναλόγως το μέγεθος του κώδικα που ελέγχει μια μέθοδος ελέγχου μπορεί να χρειαστεί από μερικά millisecond έως λεπτά ή ώρες. Αυτό έχει ως αποτέλεσμα εκτός από τον εύκολο εντοπισμό των σφαλμάτων σε μεγάλα προγράμματα και την εξοικονόμηση πολύτιμου χρόνου στον προγραμματιστή. Μέσο του Unit Test έχει την δυνατότητα ο προγραμματιστής να προσομοιώσει αρκετές συνθήκες χρήσης μιας εφαρμογής από τους μετέπειτα πελάτες ή χρήστες της και να ελέγξει αν τα δεδομένα εισόδου και εξόδου είναι τα επιθυμητά. Ο έλεγχος μπορεί να πραγματοποιείται τακτικά καθ' όλη την πορεία του προγραμματισμού ακόμα και αν ο κώδικας έχει δεχτεί τροποποιήσεις χωρίς να χρειάζεται να αλλαχτούν οι μέθοδοι ελέγχου. (Wikipedia: Unit Testing, 2017)

## 1.2 TEST DRIVEN DEVELOPMENT ΚΑΙ UNIT TESTS

Το Test Driven Development αποτελεί μια χαρακτηριστική πρακτική ευρέως γνωστή στον Extreme Programming. Πάνω σε αυτήν την τεχνική είναι βασισμένα αρκετά και τα Unit Tests και αποτελούν αναπόσπαστο κομμάτι τους (Wikipedia: Test Driven Development, 2017). Το TDD είναι μια επαναστατική μέθοδος προγραμματισμού η οποία περιέχει την τεχνική Test First Development. Κατά την οποία πρώτα δημιουργούμε ένα μικρό τμήμα

κώδικα για να ελέγξουμε αν είναι ορθός και μετέπειτα αν δεν χρειάζεται διορθώσεις συνεχίζουμε στην δημιουργία του επόμενου τμήματος κώδικα που θα δεχτεί τους επόμενους ελέγχους. Η όλη διαδικασία επαναλαμβάνεται μέχρι την επιτυχή ολοκλήρωση του προγραμματισμού (εικόνα 2). Όπως αρέσκεται να λέει και ο Ron Jeffries, ένα από τους τρείς ιδρυτές του Extreme Programming «στόχος του TDD είναι να γράφεις καθαρό κώδικα που πραγματικά δουλεύει». (Ambler, 2010)




Εικόνα 2

## 1.3 DATA DRIVEN TESTS

Το αρχείο αυτό μπορεί να είναι ένα excel, csv, μια βάση δεδομένων ή ακόμα και ένα απλό αρχείο κειμένου txt.

Το επιθυμητό αποτέλεσμα στα Unit Tests μπορεί να ορισθεί είτε σαν παράμετρο σε μία μέθοδο ελέγχου είτε να παρέχεται στην μέθοδο ελέγχου από ένα εξωτερικό αρχείο τύπου xml, txt, xlsx και csv. Τα αρχεία τύπου xml είναι εκείνα τα οποία περιέχουν δεδομένα που περικλείονται από ετικέτες (αγκύλες). Τα αρχεία τύπου xlsx είναι το έγγραφο εξόδου από

το πρόγραμμα λογιστικών φύλλων της Microsoft, το Microsoft Excel που χρησιμοποιεί ένα πλέγμα κελίων διατεταγμένων σε αριθμημένες σειρές και στήλες, όπως επίσης και τα αρχεία τύπου csv με την μόνη διαφορά πως τα κελία αυτά είναι διαχωρισμένα με τα ελληνικά ερωτηματικά (;). Τα αρχεία τύπου txt είναι το παραγόμενο έγγραφο από το σημειωματάριο των Windows. Στην περίπτωση όπου χρησιμοποιείται κάποιο από αυτά τα αρχεία για τον καθορισμό του επιθυμητού αποτελέσματος τότε λέμε πως έχουμε Data Driven Test στο Unit Test μας. Είναι μια τεχνική σύμφωνα με την οποία κάθε μέθοδος ελέγχου δοκιμάζεται παραπάνω από μια φορά αυτόματα για διαφορετικά δεδομένα την φορά τα όποια βρίσκονται αποθηκευμένα στα προαναφερθέντα αρχεία και τα έχει εισάγει ο προγραμματιστής. Με τα Data Driven Tests μπορούμε να ελέγξουμε διεξοδικά ένα πρόγραμμα χωρίς να χρειάζεται να εισάγουμε κάθε φορά καινούργιες παραμέτρους για να αποφασίσουμε αν έχουμε το σωστό αποτέλεσμα. Διότι στο τέλος της χρήσης των DDT από την μέθοδο ελέγχου θα έχουμε μια αναλυτική εικόνα με όλα τα αποτελέσματα που έφερε για κάθε παράμετρο που εισήχθη καθώς και το ακριβή σε χρόνο millisecond που χρειάστηκε το Unit Test.

# 2. ΑΝΑΠΤΥΞΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Για την κατανόηση της λειτουργίας του Unit Test όπως αναφέρθηκε και στην εισαγωγή δημιουργήθηκε στο περιβάλλον ανάπτυξης της Microsoft Visual Studio ένα πρότζεκ σε γλώσσα προγραμματισμού C#. Η πρόσφατα αναπτυγμένη γλώσσα προγραμματισμού C# (sharp) επιλέχθηκε λόγο των δυνατοτήτων της στον αντικειμενοστραφή προγραμματισμό και την όλο και αυξανόμενη προτίμηση της σε εφαρμογές διαδικτύου και βάσεων δεδομένων. Ο συνολικός κώδικας της εφαρμογής δίνεται στο τέλος στο ανάλογο κεφάλαιο.

Το πρότζεκτ έχει υλοποιηθεί με τέτοιον τρόπο ώστε να προσομοιώνει μια εξέταση μαθήματος προγραμματισμού σε ένα ακαδημαϊκό ίδρυμα. Το ζητούμενο είναι η ανάπτυξη τεσσάρων αλγορίθμων σε μορφή συνάρτησης μέσα σε μια κλάση με την ονομασία Exam και με το όνομα του πρότζεκτ να είναι το MyExam. Ο κάθε αλγόριθμος αντιπροσωπεύει και σε ένα θέμα της εξέτασης. Το πρώτο θέμα περιλαμβάνει τον αλγόριθμο μέσου όρου, το δεύτερο θέμα τον αλγόριθμο ελάχιστου και μέγιστου, το τρίτο θέμα τον αλγόριθμο ταξινόμησης φυσαλίδας και το τέταρτο και τελευταίο θέμα τον αλγόριθμο δυαδικής αναζήτησης. Στο τέλος της ανάπτυξης και των τεσσάρων συναρτήσεων εφαρμόζεται το Unit Test για να ελέγξει αν οι συναρτήσεις που δημιουργήθηκαν επιστρέφουν το επιθυμητό αποτέλεσμα για κάθε τιμή που εισάγεται ως παράμετρο.

Επιπλέον στο πρότζεκτ εισάγουμε με την εντολή using τις βιβλιοθήκες System, System.Collections.Generic, System.Threading.Tasks, System.Text και System.Linq οι οποίες είναι απαραίτητες για την σωστή λειτουργεία μιας κλάσης βιβλιοθήκης αλλά και για την χρήση μεταβλητών συμβολοσειράς.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace MyExam
{
   public class Exam
   {
   }
}
```

## 2.1 ΑΛΓΟΡΙΘΜΟΣ ΕΥΡΕΣΗΣ ΜΕΣΟΥ ΟΡΟΥ

Στο πρώτο θέμα περιλαμβάνεται η ανάπτυξη του αλγορίθμου εύρεσης του μέσου όρου δύο πραγματικών αριθμών σε μορφή συνάρτησης με την ονομασία Average που επιστρέφει ένα δεκαδικό αριθμό. Αυτός ο δεκαδικός αριθμός είναι το μέσο όρο των δύο δεκαδικών αριθμών που δέχεται ως παράμετρο η συνάρτηση.

```
public float Average (float first, float second)
{
   float sum = first * second;
   float avg = sum / 2;
   return avg;
}
```

## 2.2 ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΑΙ ΜΕΓΙΣΤΟΥ ΑΡΙΘΜΟΥ

Το δεύτερο θέμα αποτελεί τον αλγόριθμο ελάχιστου και μέγιστου αριθμού ενός ακέραιου αριθμού μεγαλύτερου του 100, σε μορφή συνάρτησης με το όνομα MinMax που επιστρέφει τον μέγιστο ακέραιο.

Στην αρχή της συνάρτησης του θέματος η ακέραια μεταβλητή number που εισάγεται ως παράμετρο μετατρέπεται με την συνάρτηση ToString() της κλάσης System σε συμβολοσειρά και αποθηκεύεται στο πίνακα ακεραίων numberToArray αφού πρώτα κάθε χαρακτήρας της συμβολοσειράς επεξεργάστηκε ως μια λίστα Select(t=>int.Parse(t.ToString())).ToArray(). Ο σκοπός που μετατράπηκε ο ακέραιος σε συμβολοσειρά και αποθηκεύτηκε στην συνέχεια σαν πίνακας ακεραίων είναι διότι με αυτόν τον τρόπο καταφέραμε και διασπάσαμε έναν ακέραιο. Επιπλέον έτσι μπορούμε να επεξεργαστούμε κάθε αριθμό ξεχωριστά και να τους συγκρίνουμε μεταξύ τους για να βρούμε τον μέγιστο αριθμό. Στην συνέχεια αρχικοποιούμε τις μεταβλητές τύπου ακεραίου min και max με το πρώτο στοιχείο του πίνακα numberToArray[] και με το μηδέν αντίστοιχα. Στην επανάληψη όπου συγκρίνουμε το κάθε στοιχείο της numberToArray[] με τα min και max τα αντικαθιστούμε μόνο στην περίπτωση όπου αληθεύει η ισότητα. Στο

τέλος της επανάληψης η μεταβλητές min και max περιέχουν τον μικρότερο και μεγαλύτερο αριθμό αντίστοιχα που βρέθηκε στον πίνακα numberToArray[] και επιστρέφουμε τον μέγιστο max.

```
public int MinMax(int number)
{
 var numberToArray=
number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
int min = numberToArray[0];
 int max = 0;
 for (int x = 0; x < numberToArray.Length; x++)
  {
            if (numberToArray[x] < max)
            {
               max = numberToArray[x];
            }
            else if (numberToArray[x] < min)
            {
               min = numberToArray[x];
            }
  }
        return max;
}
```

## 2.3 ΑΛΓΟΡΙΘΜΟΣ ΤΑΞΙΝΟΜΗΣΗΣ ΦΥΣΑΛΙΔΑΣ

Στο τρίτο θέμα συναντάται ο αλγόριθμος ταξινόμησης φυσαλίδας σε μορφή συνάρτησης με το όνομα BubbleSort, με μοναδική παράμερο έναν ακέραιο. Η συνάρτηση ταξινόμησης φυσαλίδας επιστρέφει τον ακέραιο αριθμό που δέχεται ως παράμετρο σε μορφή συμβολοσειράς, ταξινομημένο από τον μικρότερο στον μεγαλύτερο αριθμό, αυτό γίνεται διότι έναν μόνο ακέραιο αριθμό δεν μπορούμε να τον ορίσουμε ως πίνακα ακεραίων.

Επομένως όπως και στην αρχή του αλγόριθμου του θέματος δύο, μετατρέπουμε τον ακέραιο αριθμό number σε συμβολοσειρά με την εντολή ToString() και ύστερα κάθε χαρακτήρας που περιέχει αποθηκεύεται στον πίνακα numberToArray. Στην συνέχεια συγκρίνουμε το κάθε στοιχείο του πίνακα numberToArray[] με το επόμενο στοιχείο του και αν είναι μεγαλύτερο τους αλλάζουμε τις θέσεις, μεταφέροντας στην επόμενη θέση τον

μεγαλύτερο αριθμό. Στο τέλος της συνάρτησης ενώνουμε το κάθε στοιχείο του πίνακα numberToArray με την χρήση της συνάρτησης String.Join(), που περιλαμβάνεται στην κλάση System ως μια ενιαία συμβολοσειρά και το αποθηκεύουμε στην μεταβλητή τύπου συμβολοσειράς (string) την result όπου και επιστρέφεται από την συνάρτηση.

```
public String BubbleSort (int number)
{
var numberToArray = number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
for (int i = 0; i < numberToArray.Length; i++)
 {

   for (int x = 0; x < numberToArray.Length - 1; x++)
     {
        if (numberToArray[x]  >  numberToArray[x + 1])
        {
          int temp = numberToArray[x + 1];
          numberToArray[x + 1] = numberToArray[x];
                numberToArray[x] = temp;
            }
          }
         }
 string result = String.Join("", numberToArray);
 return result;
}
```

## 2.4 ΑΛΓΟΡΙΘΜΟΣ ΔΥΑΔΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Το τέταρτο και τελευταίο θέμα περιλαμβάνει τον αλγόριθμο δυαδικής αναζήτησης σε μορφή συνάρτησης με το όνομα BinarySearch και με παραμέτρους δύο ακεραίους αριθμούς και επιστρέφει μια συμβολοσειρά. Η πρώτη παράμετρος number και η δεύτερη παράμετρος k περιλαμβάνουν έναν ακέραιο αριθμό. Η συμβολοσειρά που επιστρέφεται περιέχει την λέξη αληθής ‘true’ ή ψευδής ‘false’ ανάλογα με το αν η συνάρτηση βρήκε πως η μεταβλητή k περιέχεται στην μεταβλητή number. Η μεταβλητή number που δέχεται ως παράμετρο πρέπει να περιέχει ένα ακέραιο από αριθμούς ταξινομημένους σε αύξουσα σειρά για να επιτύχει η συνάρτηση.

Μια αναζήτηση ονομάζεται δυαδική όταν για να βρούμε το στοιχείο που ψάχνουμε σε ένα ταξινομημένο μονοδιάστατο πίνακα στοιχείων αντί να συγκρίνουμε ένα προς ένα τα στοιχεία με το ζητούμενο διαιρούμε το μέγεθος του πίνακα δια δύο για να πάρουμε την μεσαία θέση του πίνακα. Ύστερα αν το στοιχείο που βρέθηκε στην μεσαία θέση δεν είναι το ζητούμενο κινούμαστε προς την αρχή ή το τέλος του πίνακα και επαναλαμβάνουμε την διαδικασία ως ότου το ζητούμενο στοιχείο βρεθεί.

Για να επεξεργαστεί ο αριθμός number ορίζεται ξανά ως πίνακας ακεραίων στον πίνακα numberToArray[] ακολουθώντας την ίδια λογική με τις προηγούμενες δύο συναρτήσεις των θεμάτων. Έπειτα δηλώνεται και αρχικοποιείται η συμβολοσειρά found με την λέξη ψευδής ‘false’ και τρείς ακόμα ακέραιες μεταβλητές, η mid, η left και η right. Οι μεταβλητές left και right χρησιμοποιούνται για να αποθηκεύουν τα όρια ενός πίνακα, ώστε να μην πραγματοποιείται αναζήτηση πέρα από αυτά, καθώς θα οδηγήσει σε σφάλματα την συνάρτηση. Η μεταβλητή left που υποδεικνύει την αρχή έχει ως αρχική τιμή το μηδέν (0), ενώ η right το μήκος του πίνακα numberToArray[] μείον ένα, καθώς υποδεικνύει το τέλος του πίνακα. Η μεταβλητή mid χρησιμοποιείται για να αποθηκεύει την μεσαία θέση του στοιχείου σε ένα πίνακα με την βοήθεια των προηγούμενων μεταβλητών και αρχικοποιείται με μια αρνητική τιμή. Στην συνέχεια με την βοήθεια μια επανάληψης η οποία εξασφαλίζει πως δεν θα αναζητήσουμε έξω από τα όρια του πίνακα ορίζουμε την μεσαία θέση προσθέτοντας τα δύο όρια και διαιρώντας τα δια δύο. Ελέγχουμε αν σε αυτή την μεσαία θέση υπάρχει το ζητούμενο στοιχείο που περιέχεται στην παράμετρο k, αν υπάρχει στην μεταβλητή found αποθηκεύουμε την λέξη ‘true’ και επιστρέφουμε το αποτέλεσμα. Αν δεν υπάρχει στην μεσαία θέση και το στοιχείο που βρέθηκε σε αυτήν είναι μεγαλύτερο του k, μεταθέτουμε το τέλος του πίνακα κατά μια θέση πριν την μεσαία, δηλαδή right = mid – 1. Αντίθετα αν δεν υπάρχει στην μεσαία θέση και το στοιχείο που βρέθηκε σε αυτήν είναι μικρότερο του k, μεταθέτουμε την αρχή του πίνακα κατά μια θέση μετά την μεσαία δηλαδή left = mid + 1. Η όλη διαδικασία επαναλαμβάνεται μέχρι να είτε βρεθεί το ζητούμενο στοιχείο είτε να μην είναι δυνατό να χωρισθεί περαιτέρω ο πίνακας και να επιστρέψει η συνάρτηση την αρχικοποιημένη λέξη της found.

```
public String  BinarySearch (int number, int k)
 {
var numberToArray = number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
string found = "false";
int mid = -1;
```

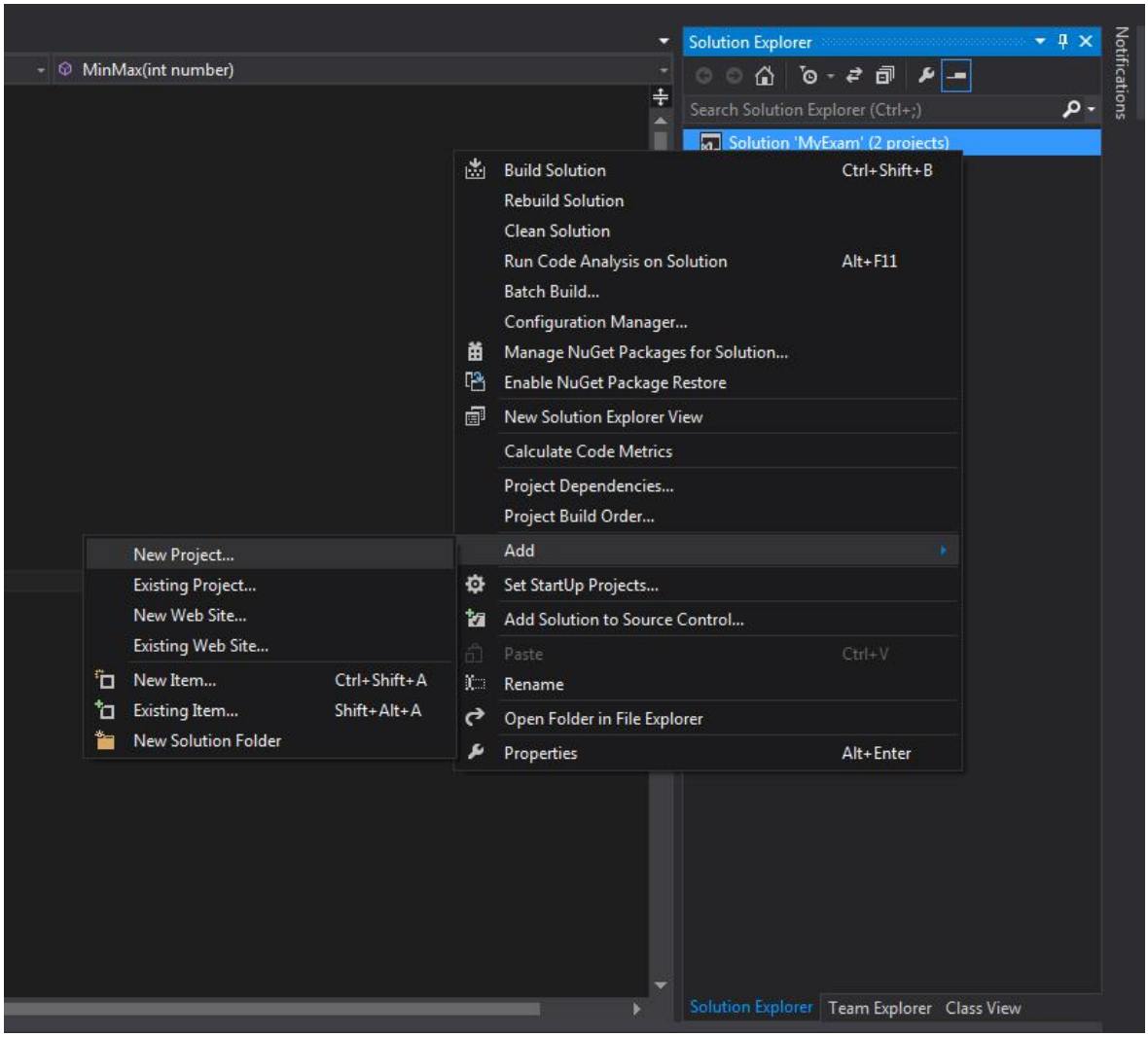
```
int left = 0;
int right = numberToArray.Length - 1;
 while (left <= right)
  {
    mid = (left + right) / 2;
    if (numberToArray[mid] == k)
    {
      found = "true";
       return found;
        }
        else if (numberToArray[mid] > k)
          {
              right = mid - 1;
          }
          else
          {
              left = mid + 1;
          }
        }
        return found;
      }
```

# 3. ΕΦΑΡΜΟΓΗ ΤΟΥ UNIT TEST

## 3.1 ΤΡΟΠΟΣ ΕΙΣΑΓΩΓΗΣ UNIT TEST ΣΕ ΕΝΑ ΠΡΟΓΡΑΜΜΑ

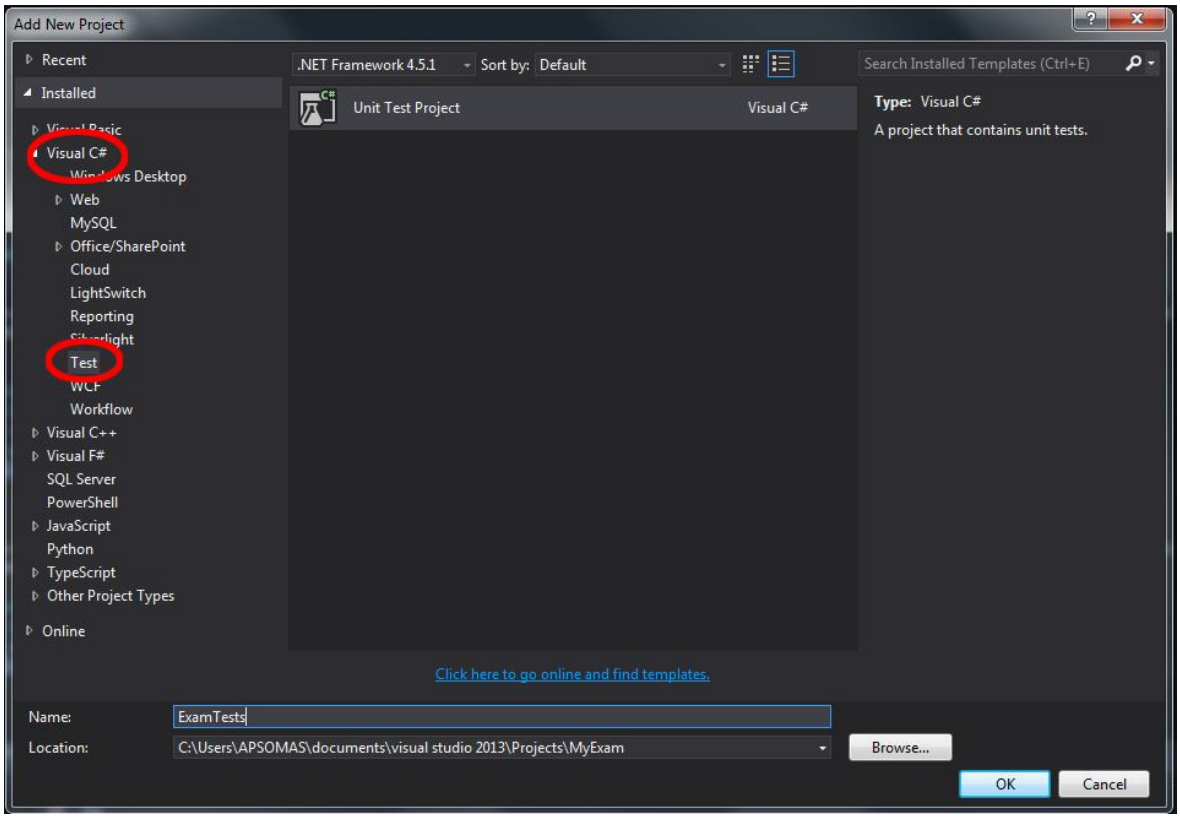
Για να διαπιστωθεί αν οι συναρτήσεις της κλάσης Exam επιστρέφουν το σωστό αποτέλεσμα ο εξεταστής - καθηγητής θα πρέπει είτε να χρησιμοποιήσει τις γνώσεις του για να τις ελέγξει είτε να δημιουργήσει μια εφαρμογή, όπου με μια διεπαφή χρήστη θα εισάγει τις τιμές των παραμέτρων των τεσσάρων συναρτήσεων μια προς μια για κάθε συνάρτηση και να ελέγξει τα αποτελέσματα. Συνεπώς θα πρέπει να δημιουργήσει μια ολοκληρωμένη εφαρμογή που να κληρονομεί την κλάση Exam, να δέχεται τιμές από το πληκτρολόγιο για κάθε συνάρτηση και να εμφανίζει στην οθόνη τα αποτελέσματα που επιστρέφει η κάθε μια συνάρτηση.

Για να αυτοματοποιήσουμε αυτήν την χρονοβόρα διαδικασία, στο αρχικό πρότζεκτ MyExam θα εισάγουμε ένα Unit Tet. Αυτό επιτυγχάνεται με δεξί κλικ επάνω στο πρότζεκτ, ‘Solution’ όπως το ονομάζει η Visual Studio, και επιλέγοντας την προσθήκη ενός νέου πρότζεκτ (εικόνα 3).



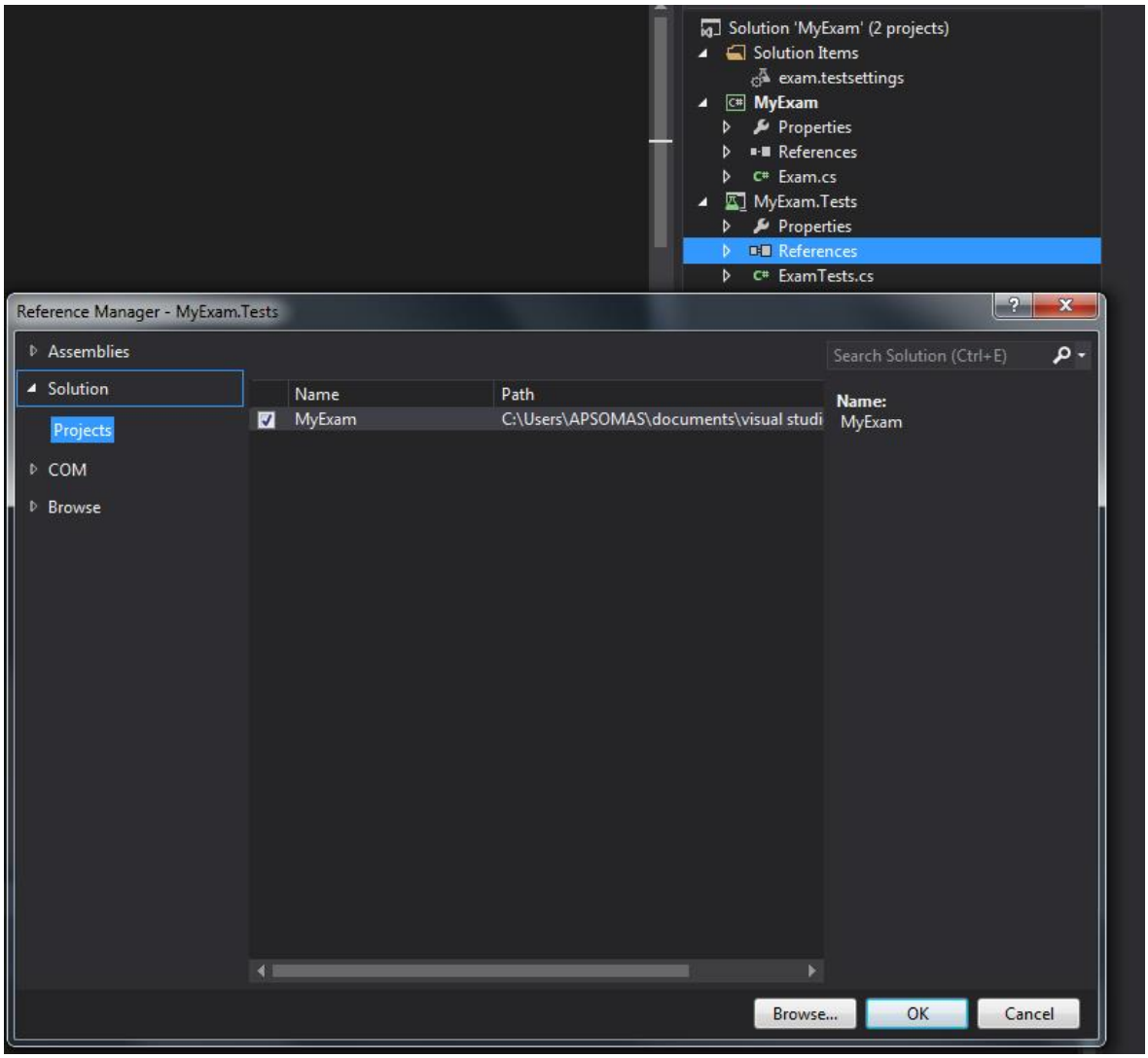

Εικόνα 3

Στο παράθυρο διαλόγου που εμφανίζεται επιλέγουμε την γλώσσα προγραμματισμού την οποία χρησιμοποιούμε ήδη, έπειτα τον τομέα Test και τέλος την επιλογή Unit Test Project. Συνηθίζεται το όνομα του UT να περιλαμβάνει το όνομα του πρότζεκτ στο οποίο προστίθεται με την λέξη Tests να ακολουθεί (εικόνα 4).




Εικόνα 4

Ύστερα η Visual Studio δημιουργεί αυτόματα μια κλάση η οποία κληρονομεί την κλάση Microsoft.VisualStudio.TestTools.UnitTesting όπου περιλαμβάνεται το Unit Test Framework. Το framework περιλαμβάνει κάποια χαρακτηριστικά (attributes) με τα οποία ξεχωρίζει τις μεθόδους ελέγχου από μια κλάση και ποια μέθοδο ελέγχου να χρησιμοποιήσει, τα χαρακτηριστικά αυτά περικλείονται μέσα σε άγκιστρα []. Επιπλέον για να χρησιμοποιήσουμε τις συναρτήσεις της κλάσης Exam πρέπει να προσθέσουμε μια αναφορά στην κλάση MyExam (εικόνα 5) και να προσθέσουμε στην αρχή του κώδικα την εντολή using MyExam.




Εικόνα 5

Καθώς θέλουμε να ελέγξουμε την ορθότητα των τεσσάρων συναρτήσεων της κλάσης Exam δημιουργούμε αντίστοιχα τέσσερις μεθόδους ελέγχου μια για κάθε θέμα της εξέτασης.

```
using MyExam;
namespace MyExam.Tests
{
   [TestClass]
   public class ExamTests
   {
   }
 }
```

## 3.2 ΔΗΜΙΟΥΡΓΙΑ ΤΩΝ DATA DRIVEN TEST

Η χρήση των Data Driven Test αυτοματοποιεί την διαδικασία κατά την οποία ο προγραμματιστής πρέπει να ορίσει για κάθε παράμετρο μιας μεθόδου ελέγχου την τιμή. Τα DDT των μεθόδων ελέγχου του Unit Test δημιουργήθηκαν με την χρήση του Microsoft Excel και αποθηκεύτηκαν ως αρχεία τύπου csv. Για να αναγνωριστούν DDT σε ένα Unit Test πρέπει να προσθέσουμε μια ακόμα αναφορά στην κλάση System.Data και να δημιουργήσουμε μια δημόσια μεταβλητή με το όνομα TestContext τύπου TestContext η οποία χρησιμοποιείται για να αποθηκεύσει τις τιμές που χρειάζεται το Unit Test. Στην αρχή της κλάσης ExamTests προθέτουμε το κώδικα public TestContext TestContext { get; set; }.

Για την πρώτη μέθοδο ελέγχου που ελέγχει τον αλγόριθμο μέσου όρου δύο πραγματικών αριθμών ορίζουμε το όνομα του αρχείου DDT σε Thema_01. Στο Thema_01 προσθέτουμε στο πρώτο κελί τις λέξεις first, second, avg διαχωρισμένες με κόμματα για να προσδιορίσουμε την ονομασία της κάθε στήλης. Η πρώτες δύο στήλες αντιπροσωπεύουν τις δύο παραμέτρους του αλγορίθμου μέσου όρου και η τρίτη το σωστό αποτέλεσμα για τις τιμές που βρίσκονται στην ίδια γραμμή. Στην συνέχεια εισάγουμε διαδοχικά πραγματικούς αριθμούς διαχωρισμένους με κόμματα σε κάθε γραμμή. Όσες γραμμές έχει το αρχείο τόσες φορές θα εκτελεστεί και θα ελεγχθεί η κάθε συνάρτηση του πρότζεκτ MyExam. Σημαντική παρατήρηση είναι η προσθήκη διπλών εισαγωγικών στους πραγματικούς αριθμούς που περιέχουν και δεκαδικό μέρος, καθώς σε διαφορετική περίπτωση δεν θα αναγνωριστούν από την μέθοδο ελέγχου και θα επιστρέψει σφάλμα που θα αφορά την σύνδεση με το DDT.

| first, second, avg |
|---|
| 0,4,2 |
| 6,9,"7,5" |
| 154,63,"108,5" |
| 1564,896,1230 |
| "19,5","44,3","31,9" |

Για τη δεύτερη μέθοδο ελέγχου που δοκιμάζεται ο αλγόριθμος μέγιστου και ελάχιστου ενός ακέραιου αριθμού, δημιουργούμε το αρχείο DDT με όνομα Thema_02 και εισάγουμε τις τιμές γραμμή, γραμμή στο πρώτο κελί. Αυτή η μέθοδο ελέγχου έχει μια παράμετρο ως όρισμα για αυτό και οι στήλες είναι μόνο δύο.

| numbers,max |
|---|
| 130,3 |
| 128,8 |
| 34596,9 |
| 165571,7 |
| 321544,5 |

Ο έλεγχος για τον αλγόριθμο ταξινόμησης φυσαλίδας περιέχεται στην τρίτη μέθοδο ελέγχου η οποία χρησιμοποιεί για DDT το αρχείο Thema_03, όπως και το προηγούμενο αρχείο έτσι και αυτό περιέχει μόνο δύο στήλες.

| numbers,sorted |
|---|
| 254,245 |
| 3695,3569 |
| 78965,56789 |
| 4596321,1234569 |
| 21238,12238 |

Στην τελευταία μέθοδο ελέγχου χρησιμοποιείται το αρχείο Thema_04 για το DDT ώστε να ελεγχθεί ο αλγόριθμος δυαδική αναζήτησης.

| numbers,search,found |
|---|
| 12345678,4,true |
| 23459,3,true |
| 34789,6,false |
| 236789,5,false |

Στο τελευταίο Data Driven Test της μεθόδου ελέγχου παρατηρείτε πως εισήχθησαν ψευδώς θετικά (False Positive) δεδομένα στην τρίτη και τέταρτη γραμμή του πίνακα. Με αυτόν τον τρόπο μπορούμε να διαπιστώσουμε πως ακόμα και το Unit Test λειτούργησε σωστά πέρα από την συνάρτηση που ελέγχτηκε.

## 3.3 ΜΕΘΟΔΟΙ ΕΛΕΓΧΟΥ ΤΟΥ UNIT TEST

Η μέθοδος ελέγχου περιλαμβάνει τις ενέργειες που χρειάζεται για να ελέγξει το UT μια συνάρτηση ενός προγράμματος ως προς το αποτέλεσμα του. Μετά την δήλωση της κλάσης ελέγχου δημιουργούμε μια συνάρτηση αρχικοποίησης (TestInitialize) για την ιδιωτική μεταβλητή με το όνομα exam τύπου Exam η οποία χρησιμοποιείται κάθε φορά που καλείται μια συνάρτηση της κλάσης Exam.

```
private Exam exam;
 [TestInitialize]
  public void TestInitialize()
 {
        exam = new Exam();
 }
```

Αν μια μέθοδος ελέγχου περιλαμβάνει Data Driven Test πρέπει να δηλώνεται η τοποθεσία του αρχείου πριν την δημιουργία των μεθόδων ελέγχου. Αυτό πραγματοποιείται με την συνάρτηση DataSource(). Η συνάρτηση αυτή του Unit Test Framework περιλαμβάνει τέσσερις παραμέτρους, στην πρώτη ορίζεται ο τύπος του αρχείου, στην δεύτερη η τοποθεσία που βρίσκεται το αρχείο, στην τρίτη το όνομα του πίνακα ή της καρτέλας στην οποία βρίσκονται τα στοιχεία και στην τέταρτη ο τρόπος προσπέλασης τους.

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\thema01.csv", "thema01#csv", DataAccessMethod.Sequential),
DeploymentItem("thema01.csv"), TestMethod]
```

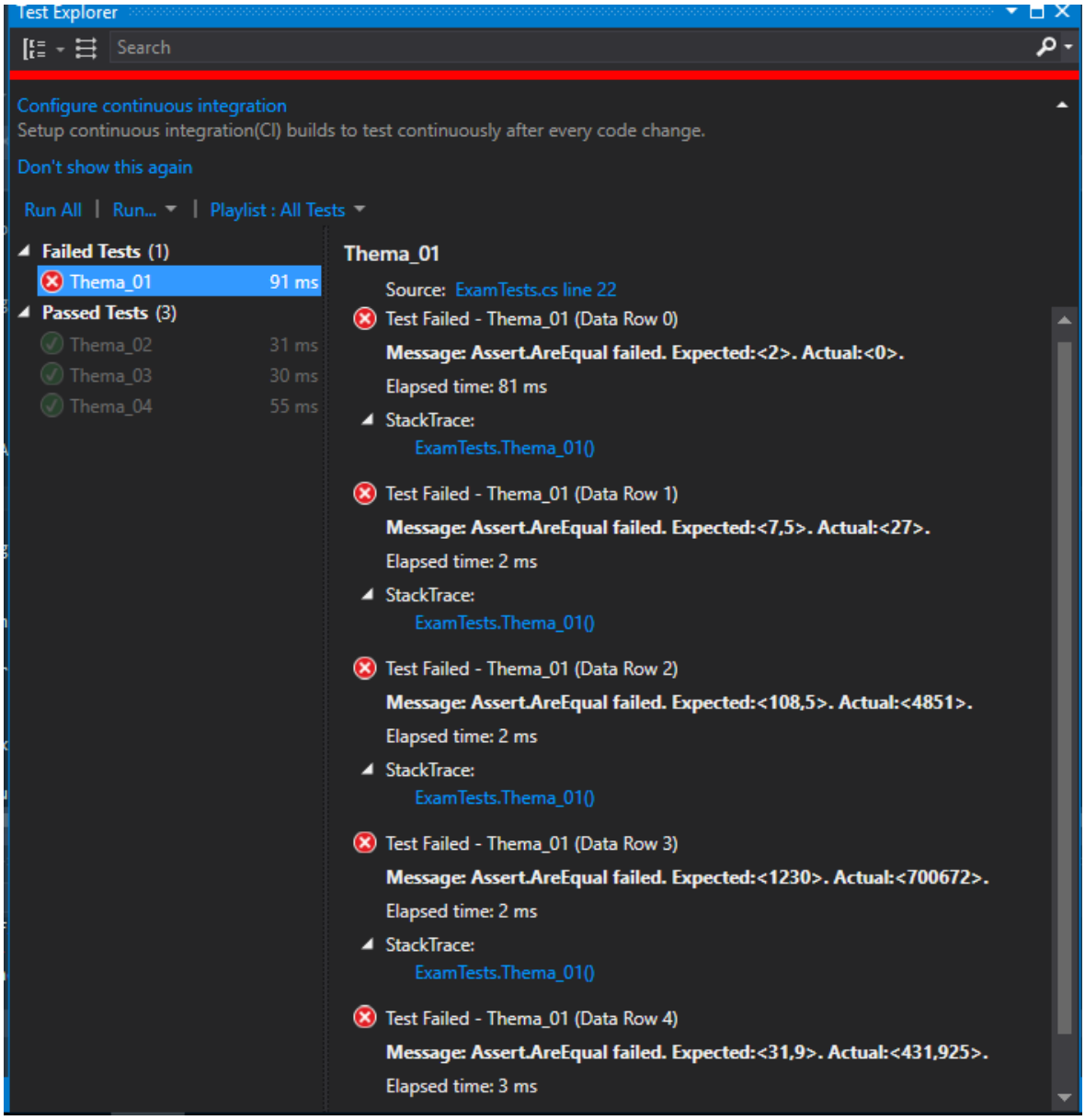
### 3.3.1 ΜΕΘΟΔΟΣ ΕΛΕΓΧΟΥ ΜΕΣΟΥ ΟΡΟΥ

Η μέθοδος ελέγχου που ελέγχει την συνάρτηση Average(float,float) παίρνει το όνομα Thema_01 και περιλαμβάνει τέσσερις μεταβλητές τύπου πραγματικού αριθμού την first, την second, την expected και την actual. Η μεταβλητή first περιλαμβάνει όλες τις τιμές που περιέχονται στην πρώτη στήλη του αρχείου thema01.csv. Η μεταβλητή second όλες τις τιμές της δεύτερης στήλης του ίδιου αρχείου και η μεταβλητή expected περιέχει το επιθυμητό μέσο όρο των τιμών για κάθε τιμή της μεταβλητής first και της μεταβλητής second. Η μεταβλητή actual περιέχει το αποτέλεσμα που επιστρέφει η συνάρτηση Average για τις μεταβλητές first και second. Παρατηρείτε πως κάθε τιμή που βρίσκεται στο αρχείο

thema01.csv πριν μεταφερθεί στην μεταβλητή της μεθόδου ελέγχου μετατρέπεται σε πραγματικός αριθμός με την βοήθεια της συνάρτησης ToSingle() της κλάσης Convert. Στην συνέχεια συναντάται η συνάρτηση AreEqual() της κλάσης Assert που περιλαμβάνεται στο Unit Test framework όπου υποθέτουμε πως οι μεταβλητές actual και expected είναι ίσες.

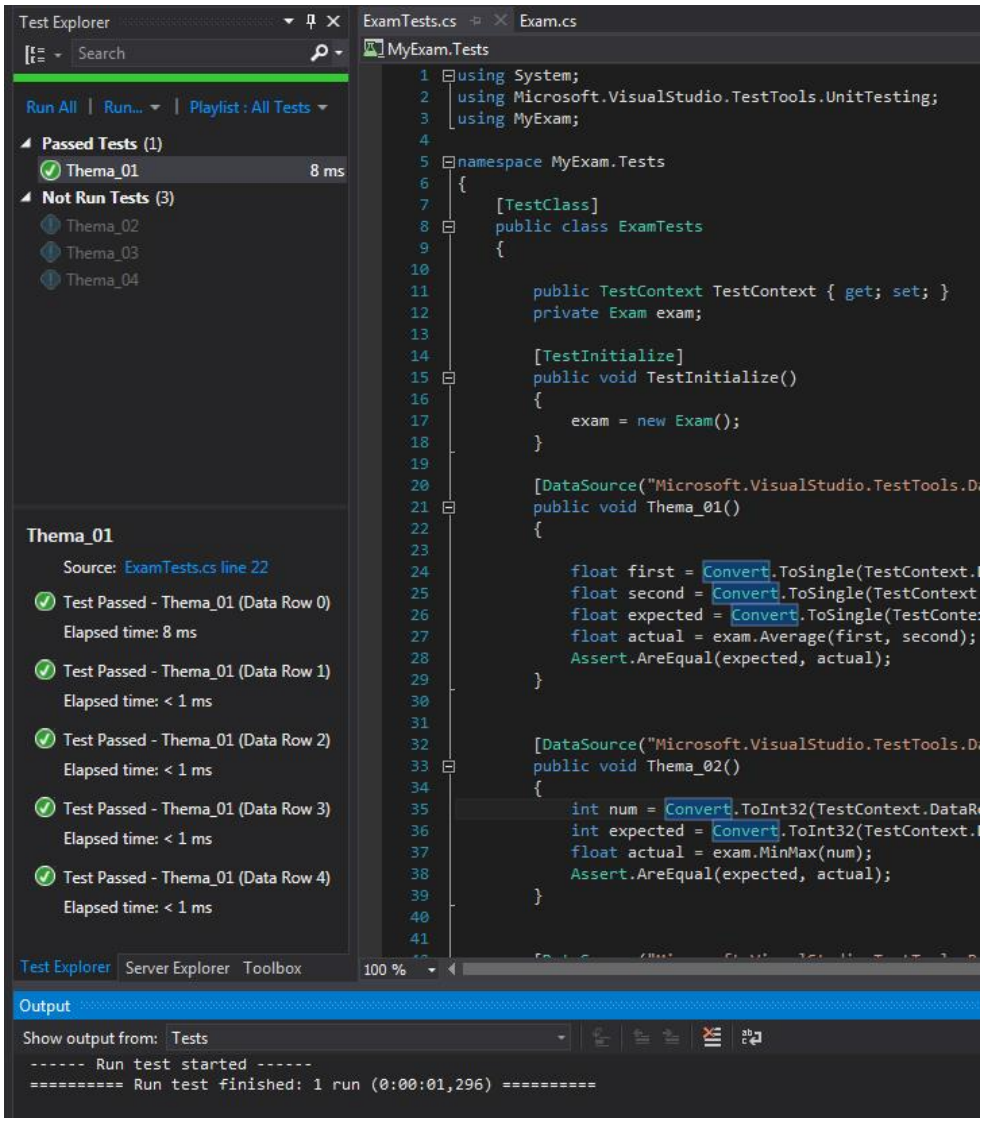
```
public void Thema_01()
{
   float first = Convert.ToSingle(TestContext.DataRow["first"]);
   float second = Convert.ToSingle(TestContext.DataRow["second"]);
   float expected = Convert.ToSingle(TestContext.DataRow["avg"]);
   float actual = exam.Average (first, second);
   Assert.AreEqual(expected, actual);
}
```

Παρατηρούμε όμως πως το UT αποτυγχάνει για κάθε δεδομένο του αρχείου thema01.csv, διότι οι μεταβλητές expected διαφέρουν από τις μεταβλητές actual (εικόνα 6). Σύμφωνα με το μήνυμα σφάλματος της πρώτης γραμμής (Data Row 0) του Unit Test η επιθυμητή τιμή είναι δύο (Expected<2>) αλλά η τιμή που επιστρέφει η συνάρτηση είναι μηδέν (Actual<0>). Το αρχείο thema01.csv στην πρώτη γραμμή περιέχει για τις δύο παραμέτρους της συνάρτησης AreEqual(expected, actual) τις τιμές <0> και <4> αντίστοιχα.




Εικόνα 6

Επανεξετάζοντας τον κώδικα μας παρατηρούμε πως στην συνάρτηση Average(float,float) για να βρούμε το άθροισμα των δύο παραμέτρων τύπου float τους πολλαπλασιάζουμε αντί να τους προσθέσουμε (float sum = first * second;). Διορθώνοντας την γραμμή κώδικα με την εντολή float sum = first + second; το Unit Test ύστερα επιτυγχάνει και ο κώδικας της συνάρτησης Average(float,float) είναι ορθός επιστρέφοντας το σωστό μέσο όρο για όλα τα δεδομένα του αρχείου thema01.csv που δέχεται ως παράμετρο (εικόνα 7).

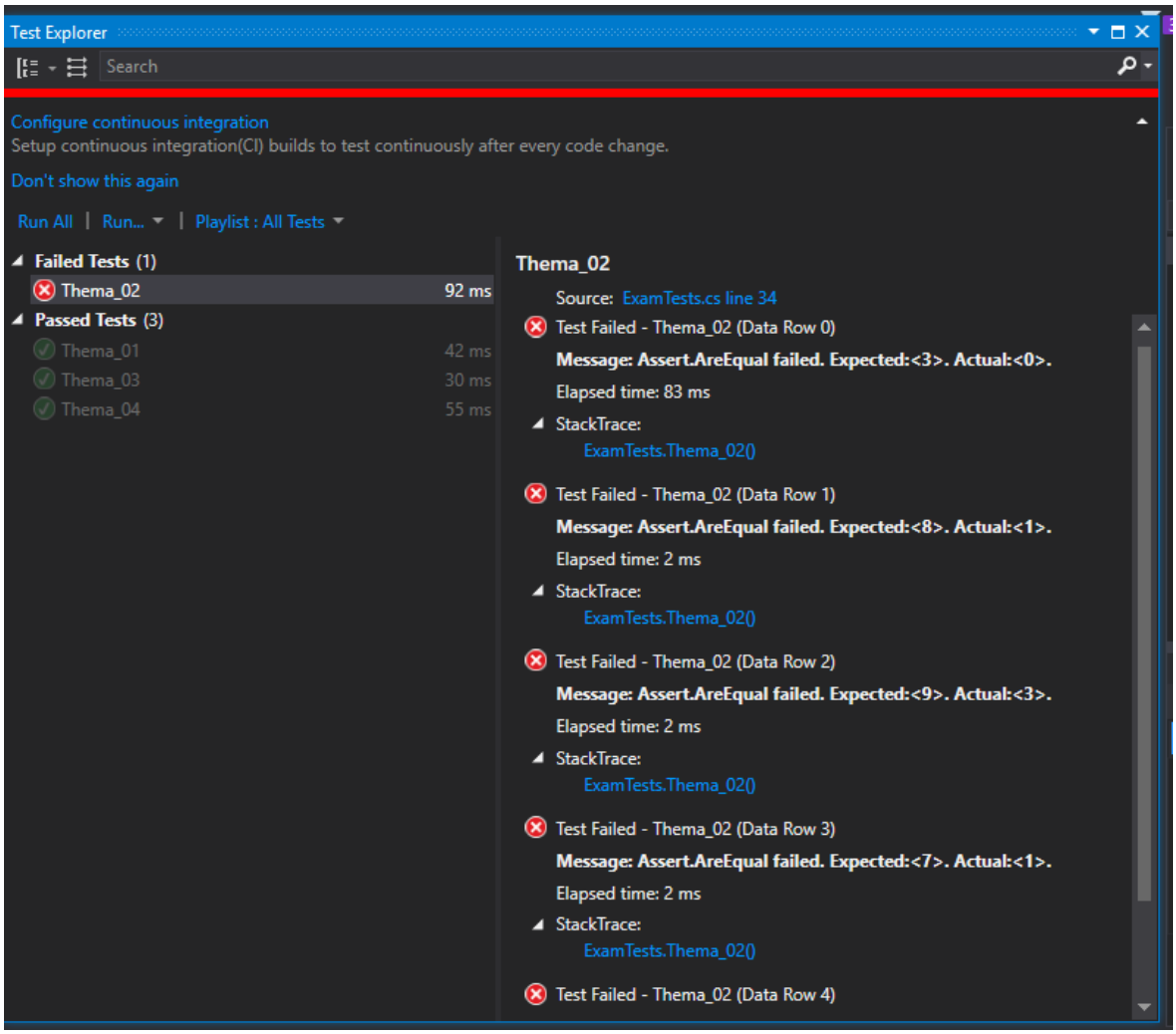



Εικόνα 7

### 3.3.2 ΜΕΘΟΔΟΣ ΕΛΕΓΧΟΥ ΜΕΓΙΣΤΟΥ ΚΑΙ ΕΛΑΧΙΣΤΟΥ

Η συνάρτηση MinMax(int) ελέγχεται με την μέθοδο ελέγχου Thema_02 όπου υπάρχουν τρείς μεταβλητές τύπου ακέραιου αριθμού η num, η expected και η actual. Η μεταβλητή num περιέχει τους αριθμούς στους οποίους η συνάρτηση MinMax(int) χρησιμοποιεί για να επιστρέψει τον μέγιστο αριθμό. Πριν αποθηκευτούν στην μεταβλητή num μετατρέπονται με την συνάρτηση ToInt32() σε ακέραιους αριθμούς. Η μεταβλητή expected περιλαμβάνει τον επιθυμητό μέγιστο αριθμό για κάθε τιμή της num. Στην ακέραια μεταβλητή actual αποθηκεύεται η τιμή που επιστρέφει η κλήση της συνάρτησης MinMax(int) με παράμετρο τις τιμές που περιέχονται στην μεταβλητή num. Τέλος χρησιμοποιείται η συνάρτηση AreEqual() για τις μεταβλητές expected και actual και το αποτέλεσμα τους εμφανίζεται στη οθόνη (εικόνα 8).

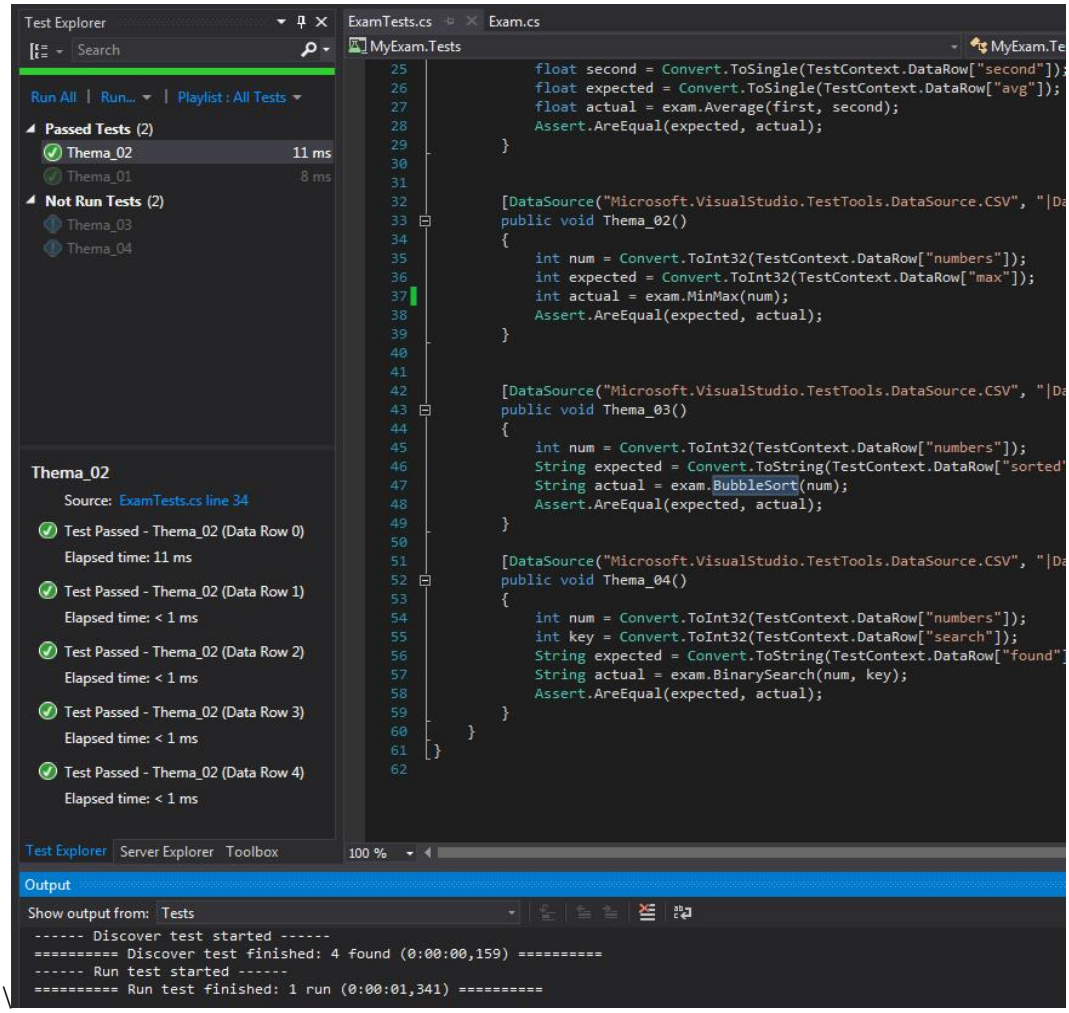
```
public void Thema_02()
{
   int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
   int expected = Convert.ToInt32(TestContext.DataRow["max"]);
   int actual = exam.MinMax (num);
   Assert.AreEqual(expected, actual);
}
```




Εικόνα 8

Η μέθοδος ελέγχου Thema_02() όμως αποτυγχάνει καθώς οι μεταβλητές expected και actual διαφέρουν για όλα τα δεδομένα που δέχεται ως παράμετρο από το αρχείο thema02.csv. Παρατηρώντας τον κώδικα της συνάρτησης MinMax(int) πιο προσεχτικά βλέπουμε πως στην συνθήκη όπου ελέγχουμε αν το περιεχόμενο του πίνακα στην τρέχουσα θέση της επανάληψης είναι μεγαλύτερο από την τιμή της μεταβλητής max έχουμε εισάγει λάθος πρόσημο. Διορθώνοντας το υπάρχον πρόσιμο < με το πρόσημο > if (numberToArray[x] > max) η συνάρτηση AreEqual(expected, actual) επιτυγχάνει για

κάθε δεδομένο που βρίσκεται σε κάθε γραμμή του αρχείου thema02.csv όπως βλέπουμε στην εικόνα 9 παρακάτω.



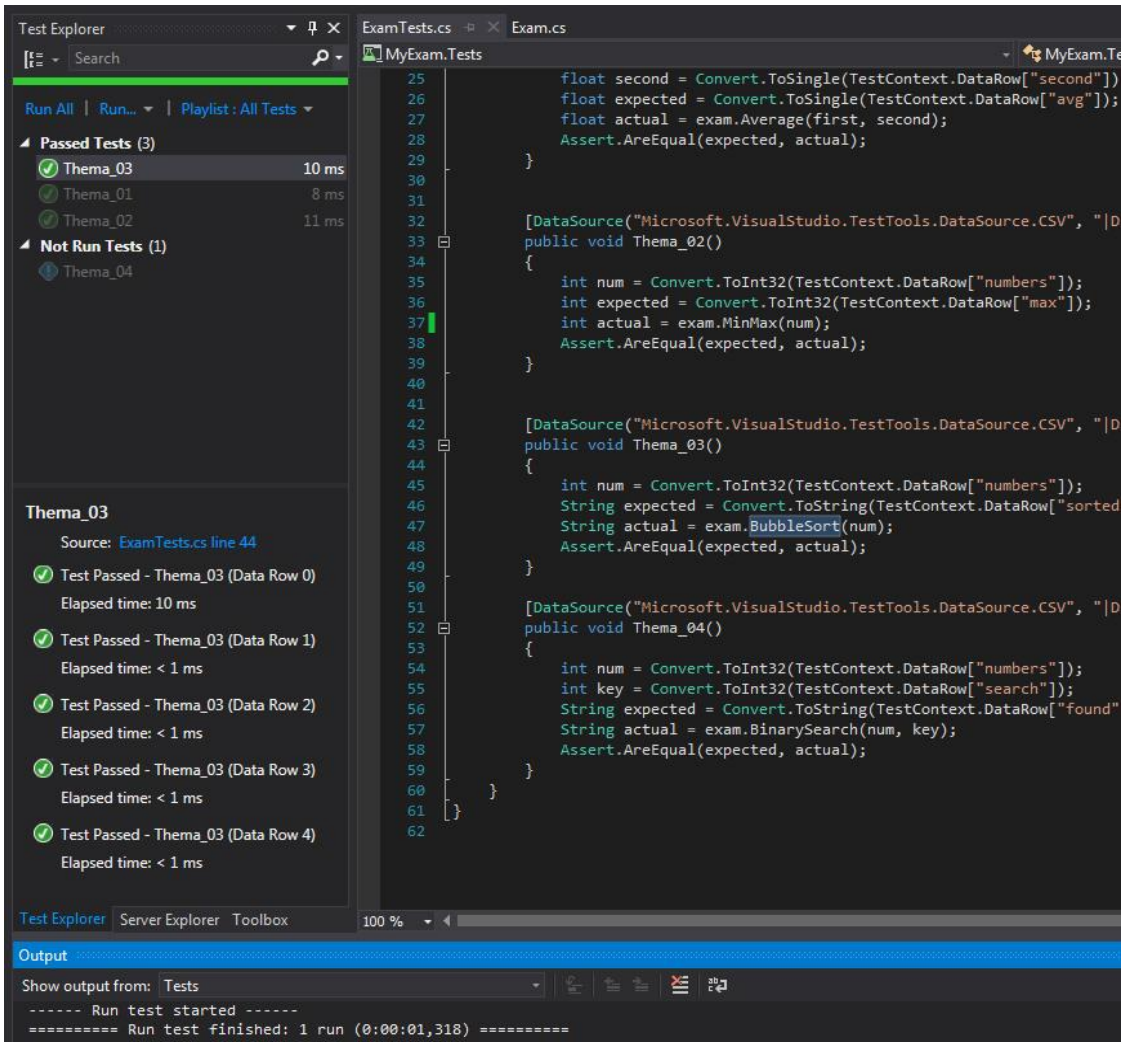

Εικόνα 9

### 3.3.3 ΜΕΘΟΔΟΣ ΕΛΕΓΧΟΥ ΤΑΞΙΝΟΜΗΣΗΣ ΦΥΣΑΛΙΔΑ

Η τρίτη μέθοδος ελέγχου ελέγχει την συνάρτηση BubbleSort(int) και ονομάζεται Thema_03, ενώ περιέχει μια μεταβλητή τύπου ακέραιου αριθμού num και δύο μεταβλητές τύπου συμβολοσειράς expected και actual. Οι τιμές που περιέχονται στην στήλη numbers πριν αποθηκευτούν στην μεταβλητή num μετατρέπονται σε ακέραιους αριθμούς με την συνάρτηση ToInt32() ενώ τα δεδομένα στην στήλη max σε συμβολοσειρά πριν αποθηκευτούν στην μεταβλητή expected. Η μεταβλητή expected περιέχει τον ακέραιο αριθμό της num ταξινομημένο κατά αύξουσα σειρά σε μορφή συμβολοσειράς ενώ η μεταβλητή actual την συμβολοσειρά που επέστρεψε η συνάρτηση BubbleSort(int) για κάθε

num. Τέλος ελέγχεται αν η μεταβλητή expected και actual είναι ίσες μέσο της AreEqual() και εμφανίζουν το αποτέλεσμα στην οθόνη (εικόνα 10).

```
public void Thema_03()
{
   int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
   String expected = Convert.ToString(TestContext.DataRow["sorted"]);
   String actual = exam.BubbleSort (num);
   Assert.AreEqual(expected, actual);
}
```



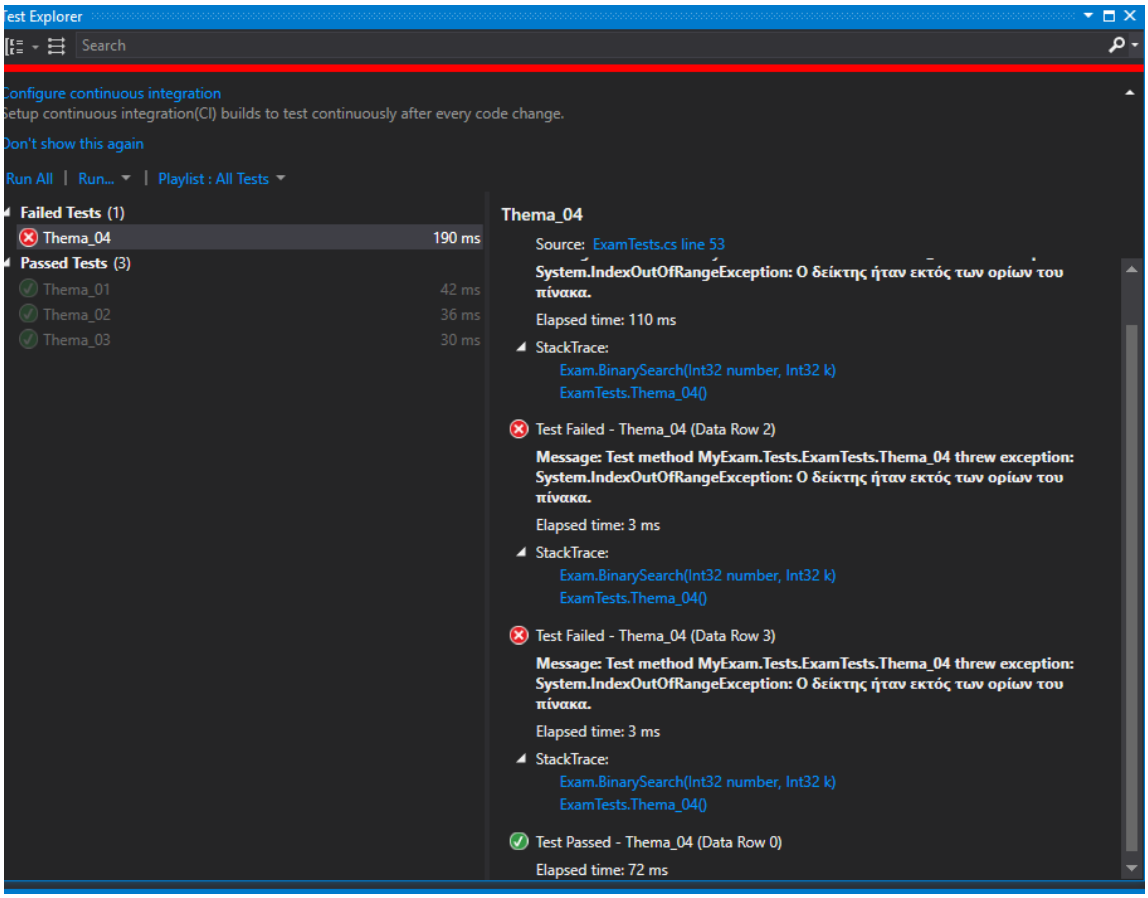

Εικόνα 10

### 3.3.4 ΜΕΘΟΔΟΣ ΕΛΕΓΧΟΥ ΔΥΑΔΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Η τελευταία μέθοδος ελέγχει το αποτέλεσμα της συνάρτησης BinarySearch(int,int) και έχει την ονομασία Thema_04. Περιλαμβάνει δύο ακέραιες μεταβλητές num και key, η μεταβλητή num περιέχει την τιμή στην οποία γίνεται η αναζήτηση για το αν

περιλαμβάνεται η τιμή key μέσα σε αυτήν. Επιπλέον περιλαμβάνει τις μεταβλητές τύπου συμβολοσειράς expected και actual, η μεταβλητή expected περιέχει το επιθυμητό αποτέλεσμα για κάθε τιμή της num και η actual το πραγματικό αποτέλεσμα της συνάρτησης BinarySearch(int,int). Τέλος η AreEqual() συγκρίνει τις μεταβλητές expected και actual και εμφανίζει τα αποτελέσματα στην οθόνη (εικόνα 11).

```
public void Thema_04()
{
   int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
   int key = Convert.ToInt32(TestContext.DataRow["search"]);
   String expected = Convert.ToString(TestContext.DataRow["found"]);
   String actual = exam.BinarySearch(num, key);
   Assert.AreEqual(expected, actual);
}
```

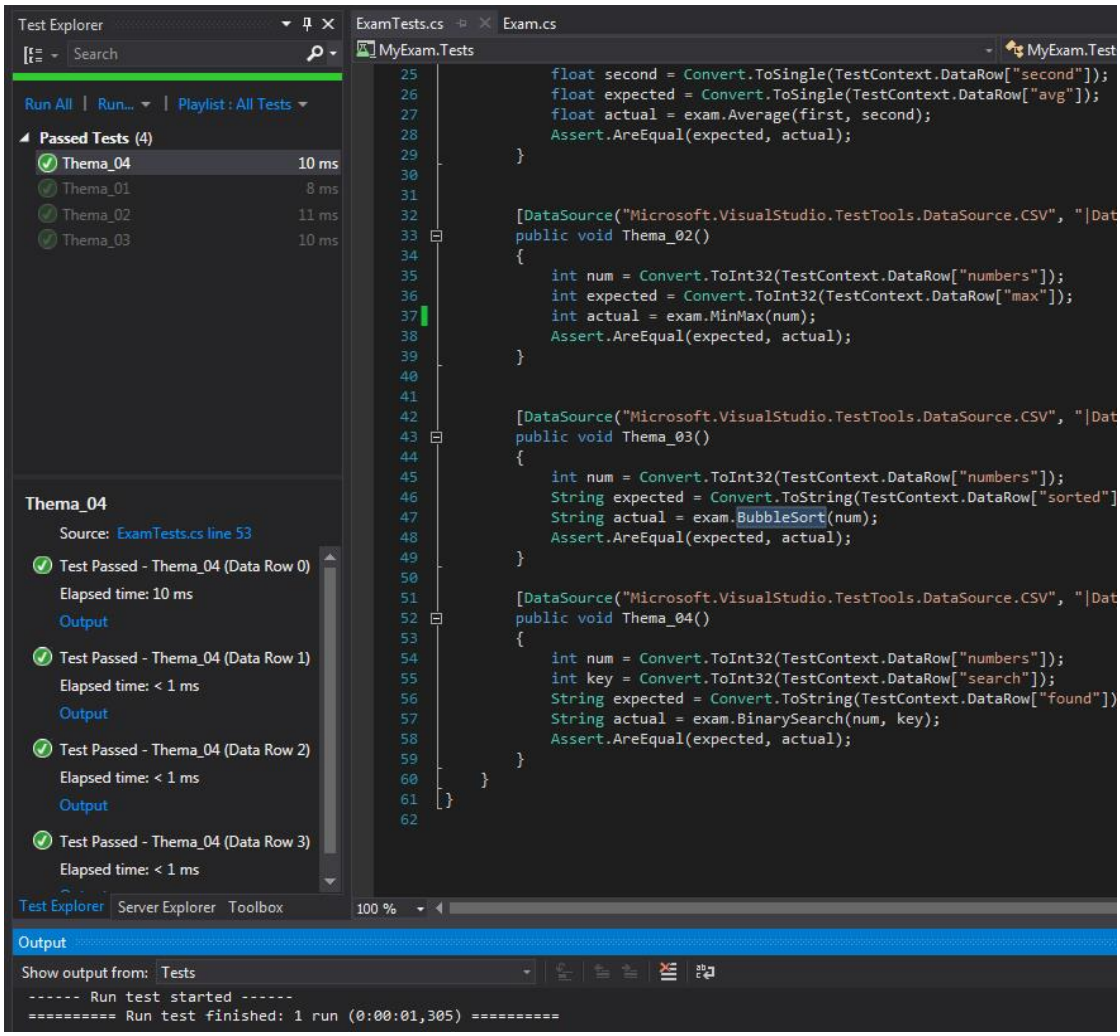



Εικόνα 11

Παρατηρούμε όμως πως η μέθοδο ελέγχου Thema_04() αποτυγχάνει για τα δεδομένα που δέχεται ως παράμετρο η συνάρτηση AreEqual(expected, actual) στις γραμμές 2,3 και 4 του αρχείου thema04.csv, ενώ επιτυγχάνει μόνο για την πρώτη γραμμή (Data Row 0). Σύμφωνα με το μήνυμα η αποτυχία συμβαίνει επειδή ο δείκτης ήταν εκτός των ορίων του πίνακα.

Εξετάζοντας τον κώδικα της συνάρτησης BinarySearch(int,int) βλέπουμε πως όταν το περιεχόμενο της μεσαίας θέσης είναι μεγαλύτερο του αριθμού που αναζητούμε, ως όριο του τέλους του πίνακα numberToArray[] αυξάνουμε κατά ένα την τιμή της μεταβλητής right. Αυτό έχει ως αποτέλεσμα όταν η συνθήκη if (numberToArray[mid] > k) είναι αληθής ο δείκτης του πίνακα (right) να βγαίνει έκτος ορίων καθώς παίρνει τιμή μεγαλύτερη από το μέγεθος του από την εντολή right = right + 1. Για τα δεδομένα που βρίσκονται στην πρώτη γραμμή του αρχείου thema04.csv η μεταβλητή num παίρνει την τιμή 12345678 και η μεταβλητή key την τιμή 4, όπου είναι ο αριθμός που αναζητούμε στην num. Η μεθόδου ελέγχου επιτυγχάνει σε αυτήν την περίπτωση διότι η πρώτη συνθήκη if (numberToArray[mid] == k) της συνάρτησης BinarySearch(int,int) είναι αληθής και δεν χρειάζεται να προχωρήσει η συνάρτηση και στην επόμενη συνθήκη. Η πρώτη επανάληψη έχει ως μεσαία θέση στην μεταβλητή mid την τέταρτη που περιέχει την τιμή τέσσερα (4) που είναι και αυτή που αναζητούσαμε με τα δεδομένα της πρώτης γραμμής του αρχείου thema04.csv. Διορθώνοντας την εντολή right = right + 1 σε right = mid – 1 καταχωρούμε ως όριο του τέλους του πίνακα numberToArray[] στην μεταβλητή right την προηγούμενη θέση της μεσαίας θέσης που περιέχει η μεταβλητή mid.

Εκτελώντας ξανά το Unit Test για την μέθοδο ελέγχου Thema_04() βλέπουμε πως επιτυγχάνει αυτήν την φορά για όλα τα δεδομένα που δέχεται ως παραμέτρους η συνάρτηση AreEqual() (εικόνα 12).


Test Explorer
Search
Run All | Run... | Playlist : All Tests
Passed Tests (4)
Thema_04 10 ms
Thema_01 8 ms
Thema_02 11 ms
Thema_03 10 ms
Thema_04
Source: ExamTests.cs line 53
Test Passed - Thema_04 (Data Row 0)
Elapsed time: 10 ms
Output
Test Passed - Thema_04 (Data Row 1)
Elapsed time: < 1 ms
Output
Test Passed - Thema_04 (Data Row 2)
Elapsed time: < 1 ms
Output
Test Passed - Thema_04 (Data Row 3)
Elapsed time: < 1 ms
Test Explorer Server Explorer Toolbox
ExamTests.cs Exam.cs
MyExam.Tests
MyExam.Test
float second = Convert.ToSingle(TestContext.DataRow["second"]);
float expected = Convert.ToSingle(TestContext.DataRow["avg"]);
float actual = exam.Average(first, second);
Assert.AreEqual(expected, actual);
}
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", "|Dat
public void Thema_02()
{
int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
int expected = Convert.ToInt32(TestContext.DataRow["max"]);
int actual = exam.MinMax(num);
Assert.AreEqual(expected, actual);
}
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", "|Dat
public void Thema_03()
{
int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
String expected = Convert.ToString(TestContext.DataRow["sorted"]
String actual = exam.BubbleSort(num);
Assert.AreEqual(expected, actual);
}
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", "|Dat
public void Thema_04()
{
int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
int key = Convert.ToInt32(TestContext.DataRow["search"]);
String expected = Convert.ToString(TestContext.DataRow["found"])
String actual = exam.BinarySearch(num, key);
Assert.AreEqual(expected, actual);
}
}
}
100 %
Output
Show output from: Tests
------ Run test started ------
========== Run test finished: 1 run (0:00:01,305) ==========

Εικόνα 12

# 4. ΣΥΜΠΕΡΑΣΜΑΤΑ

Ολοκληρώνοντας την συγκεκριμένη εφαρμογή με την χρήση των μεθόδων ελέγχου παρατηρούμε πως ο έλεγχος της κλάσης Exam για κάθε μια από τις συναρτήσεις της πραγματοποιήθηκε ταχύτατα και για αρκετές διαφορετικές παραμέτρους μέσα από μια αυτοματοποιημένη διαδικασία που μας προσφέρουν τα Unit Test Frameworks. Με την εφαρμογή του UT στο αρχικό πρότζεκτ παρατηρήσαμε πως μερικές από τις συναρτήσεις περιείχαν σφάλματα τα οποία δεν ήταν εύκολα αντιληπτά χωρίς την εκσφαλμάτωση του. Με το Unit Test όμως η εύρεση τέτοιων σφαλμάτων και η διόρθωση τους είναι εύκολη και γρήγορη. Τα Unit Tests συνοψίζοντας είναι η δυνατότητα να κατακερματίσεις ένα μεγάλο πρόγραμμα σε μικρότερα τμήματα και να ελεγχθούν αυτά τα τμήματα για τυχόν σφάλματα επανειλημμένα. Πλέον σε κάθε νέα τροποποίηση του κώδικα της συνάρτησης μου επιδέχεται UT ο προγραμματιστής έχει την δυνατότητα να ανατρέξει το UT για να διαπιστώσει αν επηρεάστηκαν τα αποτελέσματα της.

Η χρησιμότητα τους και οι δυνατότητες τους μπορούν να επεκταθούν εκτός από την εκπαίδευση του προγραμματιστή και στο εργασιακό περιβάλλον αλλά ακόμα και στα ακαδημαϊκά ιδρύματα για την βελτίωση του επιπέδου σύνθεσης και δημιουργίας αξιόπιστων εφαρμογών. Η υιοθέτηση αυτών των τεχνικών σε μια εταιρεία θα μπορούσαν να επιφέρουν βελτιωμένη ποιότητα προγραμμάτων καθώς και καλύτερη επικοινωνία με έναν πιθανό πελάτη. Ο τμηματικός έλεγχος του προγράμματος που προσφέρουν τα Unit Test δίνει την δυνατότητα να μπορεί ο πελάτης να έρχεται σε επαφή με το προϊόν συχνότερα και να ελέγχει τα αποτελέσματα των Unit Tests εάν καλύπτουν τις επιθυμίες του σε κάθε στάδιο ανάπτυξης του προγράμματος. Μια ομάδα προγραμματισμού ή ένας προγραμματιστής γνωρίζει έτσι πως τα αποτελέσματα σε κάθε φάση του προγράμματος καλύπτουν τις ανάγκες του πελάτη κερδίζοντας χρόνο στην τελική υλοποίηση του. Έτσι με την ολοκλήρωση του οι απαιτήσεις του πελάτη καλύπτονται πλήρως και αποφεύγονται αλλαγές στο τελικό πρόγραμμα. Μια εταιρεία με αυτόν τον τρόπο κερδίζει χρόνο απέναντι στον πελάτη και εξοικονομεί χρήματα που τυχόν να χρειαζόντουσαν αν παρατείνονταν ο χρόνος λόγο της τροποποίησης του προγράμματος. Επίσης δημιουργείται μια διαφορετική σχέση μεταξύ της εταιρείας και των πελατών της, καθώς ο πελάτης θα αισθάνεται πως συμμετέχει ενεργά στην υλοποίηση του προγράμματος και πως η εταιρεία ενδιαφέρεται να καλύψει πλήρως τις απαιτήσεις του.

Το πρότζεκτ MyExam το όποιο δημιουργήθηκε θα μπορούσε να προσφέρει πολλά οφέλη σε ένα ακαδημαϊκό ίδρυμα όχι μόνο προς τους καθηγητές αλλά και προς τους φοιτητές. Οι φοιτητές υιοθετώντας τα Unit Test ως εργαλεία τα οποία αξιολογούν τα προγράμματα τους σύμφωνα μόνο με το αποτέλεσμα τους θα τους έδινε το κίνητρο να ασχοληθούν περεταίρω με τον προγραμματισμό σε περιβάλλοντα ανάπτυξης λογισμικού. Αυτό θα τους έδινε την ικανότητα του να μπορούν να προβλέψουν ένα σφάλμα στο πρόγραμμα τους πιο άμεσα και να γράφουν πιο σωστό κώδικα. Οι καθηγητές με την ενσωμάτωση UT για την εξέταση των μαθητών σε μαθήματα προγραμματισμού θα μείωναν κατά πολύ τον χρόνο που χρειάζονται για να διορθώσουν όλα τα γραπτά σε κάθε εξεταστική περίοδο. Επιπλέον επειδή με την προσθήκη DDT η όλη διαδικασία αυτοματοποιείται. Ο καθηγητής δεν χρειάζεται να είναι παρόν κατά την διαδικασία ελέγχου των γραπτών των φοιτητών καθώς με την ολοκλήρωση της διαδικασίας εμφανίζονται αναλυτικά αποτελέσματα ποιοι έλεγχοι επέτυχαν και ποιοι όχι. Με αυτόν τρόπο οι φοιτητές γνωρίζουν πολύ πιο γρήγορα τα αποτελέσματα των εξετάσεων τους και συνεπώς έχουν περισσότερο χρόνο στην διάθεση τους για να προετοιμαστούν μέχρι την επόμενη εξέταση.

Μια μελλοντική εργασία η οποία θα μπορούσε να βρει σημαντική εφαρμογή σε ένα ακαδημαϊκό ίδρυμα και γενικότερα στην εκπαίδευση και εκμάθηση του προγραμματισμού θα ήταν η υλοποίηση Unit Tests μέσω μιας πλατφόρμας στο διαδίκτυο η οποία θα εξυπηρετούσε στην εξ αποστάσεως εκπαίδευση. Οι μαθητές να έχουν την δυνατότητα να μεταφορτώνουν τα προγράμματα τους σε μια διαδικτυακή πλατφόρμα και άμεσα μέσω πρωτόκολλων ελέγχου να επιστρέφεται το αποτέλεσμα για το πρόγραμμα τους αν επέτυχε τους ελέγχους ή όχι.

Τέλος ολοκληρώνοντας την παρούσα πτυχιακή εργασία και κατανοώντας την χρησιμότητα των Unit Tests στα προγράμματα έχω υιοθετήσει πλήρως τις αρχές των Unit Tests και του XP καθότι προσθέτουν ευκρίνεια και διαύγεια στον κώδικα και μπορούν πιο εύκολα να προληφθούν σφάλματα και να υλοποιηθούν γρήγορα μεγάλα και αξιόπιστα προγράμματα τα οποία έχουν εξεταστεί λεπτομερώς για τα αποτελέσματα τους με κάθε πιθανή παράμετρο.

## ΣΥΝΟΛΙΚΟΣ ΚΩΔΙΚΑΣ

Παρατηθείτε ο κώδικας για την δημιουργία της κλάσης Exam του προγράμματος MyExam σε γλώσσα C# (Sharp).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyExam
{
   public class Exam
   {
      public float Average (float first, float second)
      {
         float sum = first + second;
         float avg = sum / 2;
         return avg;
      }

      public int MinMax (int number)
      {
         var numberToArray = number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
         int min = numberToArray[0];
         int max = 0;
         for (int x = 0; x < numberToArray.Length; x++)
```

```
                    {
        if (numberToArray[x] > max)
        {
            max = numberToArray[x];
        }
        else if (numberToArray[x] < min)
        {
            min = numberToArray[x];
        }
                    }
    return max;
}

public String BubbleSort(int number)
{
    var numberToArray =
    number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
    for (int i = 0; i < numberToArray.Length; i++)
    {

        for (int x = 0; x < numberToArray.Length - 1; x++)
        {
            if (numberToArray[x] > numberToArray[x + 1])
            {

                int temp = numberToArray[x + 1];
                numberToArray[x + 1] = numberToArray[x];
                numberToArray[x] = temp;
            }
        }
    }
    string result = String.Join("", numberToArray);
    return result;
}
```

```
        public String BinarySearch(int number, int k)
        {
            var numberToArray = number.ToString().Select(t=>int.Parse(t.ToString())).ToArray();
            string found = "false";
            int mid = -1;
            int left = 0;
            int right = numberToArray.Length - 1;
            while (left <= right)
            {
                mid = (left + right) / 2;
                if (numberToArray[mid] == k)
                {
                    found = "true";
                    return found;
                }
                else if (numberToArray[mid] > k)
                {
                    right = mid - 1;
                }
                else
                {
                    left = mid + 1;
                }
            }
            return found;
        }
    }
}
```

Και επιπλέον ο κώδικας για την δημιουργία Unit Test μεθόδους πάνω στο ίδιο πρόγραμμα με την ονομασία ExamTests.cs.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MyExam;
```

```
namespace MyExam.Tests
{
   [TestClass]
   public class ExamTests
   {
      private Exam exam;

      public TestContext TestContext { get; set; }

      [TestInitialize]
      public void TestInitialize()
      {
         exam = new Exam();
      }

 [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\thema01.csv",  "thema01#csv",  DataAccessMethod.Sequential),
DeploymentItem("thema01.csv"), TestMethod]
      public void Thema_01()
      {
         float first = Convert.ToSingle(TestContext.DataRow["first"]);
         float second = Convert.ToSingle(TestContext.DataRow["second"]);
         float expected = Convert.ToSingle(TestContext.DataRow["avg"]);
         float actual = exam.Average(first, second);
         Assert.AreEqual(expected, actual);
      }

 [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\thema02.csv",  "thema02#csv",  DataAccessMethod.Sequential),
DeploymentItem("thema02.csv"), TestMethod]
      public void Thema_02()
      {
         int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
         int expected = Convert.ToInt32(TestContext.DataRow["max"]);
         int actual = exam.MinMax(num);
         Assert.AreEqual(expected, actual);
```

```
        }
 [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\thema03.csv",  "thema03#csv",  DataAccessMethod.Sequential),
DeploymentItem("thema03.csv"), TestMethod]
        public void Thema_03()
        {
            int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
            String expected = Convert.ToString(TestContext.DataRow["sorted"]);
            String actual = exam.BubbleSort(num);
            Assert.AreEqual(expected, actual);
        }


 [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\thema04.csv",  "thema04#csv",  DataAccessMethod.Sequential),
DeploymentItem("thema04.csv"), TestMethod]
        public void Thema_04()
        {
            int num = Convert.ToInt32(TestContext.DataRow["numbers"]);
            int key = Convert.ToInt32(TestContext.DataRow["search"]);
            String expected = Convert.ToString(TestContext.DataRow["found"]);
            String actual = exam.BinarySearch(num, key);
            Assert.AreEqual(expected, actual);
        }
    }
}
```

# ΒΙΒΛΙΟΓΡΑΦΙΑ


Ambler, S. W. (2010). *Introduction to Test Driven Development (TDD).* Ανάκτηση από Agile Data: http://agiledata.org/essays/tdd.html

Beck, K. (1999). Extreme Programming. *Proceedings Technology of Object-Oriented Laguage and Systems* (σσ. 411 - 411). Nancy, France: IEEE Conference Publications.

Copeland, L. (2001, Δεκέμβριος 3). *Computerworld: Extreme Programming*. Ανάκτηση Αύγουστος 4, 2017, από Extreme Programming - Computerworld: https://www.computerworld.com/article/2585634/app-development/extreme-programming.html

Hamill, P. (2005). *Unit Test Frameworks: Tools for High - Quality Software Development.* United States Of America: O'Reilly.

*Wikipedia: Test Driven Development*. (2017, Ιούλιος 12). Ανάκτηση Ιούλιος 15, 2017, από Test Driven Development - Wikipedia: https://en.wikipedia.org/wiki/Test-driven_development

*Wikipedia: The free encyclopedia*. (2017, Αύγουστος 31). Ανάκτηση Σεπτέμβριος 2, 2017, από Extreme Programming: https://en.wikipedia.org/wiki/Extreme_programming

*Wikipedia: Unit Testing*. (2017, Μαϊος 24). Ανάκτηση Ιούνιος 05, 2017, από Unit Testing - Wikipedia: https://en.wikipedia.org/wiki/Unit_testing