Διδακτική Προσέγγιση για Διασφάλιση Ποιότητας Πηγαίου Κώδικα Λογισμικού με Βελτιστοποίηση των Μετρικών του

Μ. Μπασδαβάνος[1], Σ. Δέγγλερη[2]

[1]Τμήμα Τεχνολογίας Πληροφορικής & Τηλεπικοινωνιών, Τ.Ε.Ι. Λάρισας
mgbasdavan@yahoo.gr
[2]Τμήμα Πληροφορικής, ΠΜΣ Τ.Π.Ε. στην Εκπαίδευση, Α.Π.Θ.
sofiadegleri@yahoo.gr

Περίληψη

Το κύριο πρόβλημα για τη διασφάλιση της ποιότητας του πηγαίου κώδικα ενός λογισμικού είναι η έλλειψη συγκεκριμένων μετρήσιμων στόχων αλλά και διαδικασιών μέτρησης. Δεδομένου ότι δεν υπάρχουν επαρκή ιστορικά στοιχεία και συστηματικά εμπειρικά δεδομένα, δεν έχουν αναπτυχθεί ακόμη ικανά εργαλεία και πρότυπα για αυτό το σκοπό. Ένας τρόπος αντιμετώπισης που παρουσιάζεται με την συγκεκριμένη εργασία, σε επίπεδο εκπαιδευτικής προσέγγισης, είναι η αξιοποίηση των μετρικών κώδικα λογισμικού. Ειδικότερα με τη βελτιστοποίηση της τιμής της κυκλωματικής πολυπλοκότητας των μεθόδων αντικειμενοστρεφούς κώδικα, στη φάση της ανάπτυξης, γίνεται προσπάθεια να εξασφαλιστεί κατά το δυνατόν ευκολότερη ανάγνωση, κατανόηση, έλεγχος και εγκυρότητα των αποτελεσμάτων του συγκεκριμένου τμήματος κώδικα. Η διαδικασία χρησιμοποιεί ένα Eclipse Metrics πρόσθετο που επιτρέπει την μέτρηση, εκτός των άλλων μετρικών, και της κυκλωματικής πολυπλοκότητας των μεθόδων του κώδικα, που αναπτύσσονται με κύριο στόχο την επίτευξη της βέλτιστης τιμής της κυκλωματικής πολυπλοκότητας. Αυτό συμβάλλει, σε κάποιο βαθμό, στην ποιότητα του αναπτυσσόμενου κώδικα, χωρίς ασφαλώς να την εγγυάται.
Λέξεις κλειδιά: *ποιότητα λογισμικού, μετρικές κώδικα, κυκλωματική πολυπλοκότητα.*

Abstract

The main problem of quality assurance of software source code is the lack of both specific measurable objectives and measurement procedures. Since there is insufficient historical information and little systematic empirical data, tools and standards have not been yet developed for this purpose. A workaround presented in this paper from an educational perspective, is the use of software code metrics. Specifically, by optimizing the cyclomatic complexity value of object-oriented code methods during the development phase, an attempt is made to ensure easier reading, comprehension, testing and results validity of specific code segment. The procedure uses an Eclipse Metrics plugin that allows, among other metrics, the measurement of code methods cyclomatic complexity, which, in turn, will reach its optimal value through continuing development. It should be pointed out that this process does contribute to the quality of the code being developed, without, however, guaranteeing it 100%.
Keywords: *software quality, code metrics, cyclomatic complexity*.

1. Εισαγωγή

Η έννοια της ποιότητας λογισμικού (software quality) είναι σχετικά αφηρημένη και αρκετά δύσκολο να οριστεί. Για τους λόγους αυτούς η έννοιά της συντίθεται από τα χαρακτηριστικά της που είναι οι παράγοντες ποιότητας (quality factors). Η διασφάλιση της ποιότητας αυτών των παραγόντων αποτελεί βασικό στόχο κατά την αντικειμενοστρεφή σχεδίαση (Booch et al., 2007) και την κύρια διαδικασία ελέγχου της ποιότητας κάθε λογισμικού.

Η παρούσα προσέγγιση παρατηρεί την ποιότητα λογισμικού όχι τόσο από την πλευρά του τελικού χρήστη αλλά από την πλευρά του προγραμματιστή, έμπειρου ή εκπαιδευόμενου, που στην υλοποίησή του οφείλει να εφαρμόσει καλές και δοκιμασμένες πρακτικές. Για αυτό είναι περισσότερο προσιτό και χρήσιμο, για την εν λόγω εργασία, το μοντέλο FCM του McCall από το πρότυπο ISO 9126.

Στο μοντέλο FCM, που περιλαμβάνει 11 παράγοντες ποιότητας, 25 κριτήρια και 41 μετρικές, διακρίνονται παράγοντες όπως η αναγνωσιμότητα (readability), η κατανοησιμότητα (comprehensibility) και η ελεγξιμότητα (testability). Οι παράγοντες αυτοί άπτονται κριτηρίων όπως η απλότητα (simplicity), αυτοπεριγραφικότητα (self-descriptiveness) και συνοπτικότητα (conciseness) (Ξένος, 2003). Στα κριτήρια αυτά έχουν αναφορά διάφορες μετρικές μεταξύ των οποίων και η κυκλωματική πολυπλοκότητα *CC* ή *Vg* (Cyclomatic Complexity ή Conditional Complexity).

Η κυκλωματική πολυπλοκότητα εισάχθηκε ως έννοια και μετρική από τον McCabe για τη μέτρηση της πολυπλοκότητας ενός προγράμματος. Η τιμή της αποτελεί ένα μέτρο εκτίμησης της πολυπλοκότητας ενός τμήματος, δηλ. μίας μεθόδου, αντικειμενοστρεφούς κώδικα. Συνεπώς η μείωση της τιμής της συνεπάγεται περιορισμό της πολυπλοκότητας και αύξηση των δυνατοτήτων ανάγνωσης, κατανόησης και εγκυροποίησης (Καμέας, 2003) του πηγαίου κώδικα, ιδιαίτερα από εκπαιδευόμενους και μη έμπειρους προγραμματιστές.

Η τακτική της βελτίωσης του παραγόμενου κώδικα με γνώμονα τις ανεκτές τιμές των μετρικών του, συνιστά επιπλέον μία αρκετά καλή πρακτική στη διδασκαλία σε όλες τις βαθμίδες της εκπαίδευσης όπου ο αντικειμενοστρεφής προγραμματισμός αποτελεί γνωστικό αντικείμενο.

Στο συγκεκριμένο άρθρο γίνεται χρήση της μετρικής της κυκλωματικής πολυπλοκότητας, ώστε με τη βελτίωση της τιμής της μέσω των αλλαγών στον πηγαίο κώδικα να προκύψουν ευεργετικά αποτελέσματα στους ποιοτικούς παράγοντες, δηλ. την κατανοησιμότητα, την αναγνωσιμότητα και την ελεγξιμότητα. Τελικός στόχος είναι ένας κώδικας απλός, συνοπτικός και αυτοπεριγραφικός (Fenton & Pfleeger, 1997).

Επίσης γίνεται μελέτη ενός εκπαιδευτικού παραδείγματος στο περιβάλλον Eclipse. Στα πλαίσια της μελέτης αυτής, με τη χρησιμοποίηση ενός Eclipse Metrics

πρόσθετου (plugin), ελαττώνεται η κυκλωματική πολυπλοκότητα μίας σχετικά σύνθετης μεθόδου, ώστε ο προκύπτων κώδικας να είναι απλός, εύκολα κατανοήσιμος, γρήγορα ελέγξιμος και αρκούντως αυτοπεριγραφικός, δηλ. χωρίς μεγάλο όγκο σχολιασμών.

Το υπόλοιπο του άρθρου είναι οργανωμένο ως εξής:

Η ενότητα 2 περιγράφει τις μετρικές κυκλωματική πολυπλοκότητα και πολυπλοκότητα διασύνδεσης καθώς και τους τρόπους υπολογισμού των με χρήση του εκπαιδευτικού παραδείγματος της εύρεσης του μέγιστου τριών ακεραίων αριθμών.

Στην ενότητα 3 παρατίθενται τα αποτελέσματα της μέτρησης του κώδικα του εκπαιδευτικού παραδείγματος με το Eclipse Metrics plugin.

Η ενότητα 4 αναφέρει τα συμπεράσματα από τη μελέτη και τις προοπτικές επέκτασής της με ένα σύνολο βασικών μετρικών και με παραδείγματα ευρύτερης κλίμακας.

2. Η Κυκλωματική Πολυπλοκότητα ως Μετρική

Στη βιβλιογραφία υπάρχει ένα πλήθος εναλλακτικών ορισμών της κυκλωματικής πολυπλοκότητας, που πρωτοπροτάθηκε από τον McCabe (McCabe, 1976). Ενδεικτικά αναφέρουμε:

- Μετρά το πλήθος των λογικών αποφάσεων μέσα σε μία απλή μονάδα λογισμικού (π.χ. μέθοδο).
- Δίνει τον αριθμό των απαιτούμενων ελέγχων για μία συγκεκριμένη μονάδα λογισμικού.
- Μετρά ευθέως τον αριθμό των γραμμικά ανεξάρτητων μονοπατιών διαμέσου του πηγαίου κώδικα μίας μονάδας λογισμικού.

Όλοι οι παραπάνω ορισμοί περιγράφουν γλαφυρά την μετρική καθώς και την χρησιμότητά της στην εκτίμηση της ποιότητας του λογισμικού.

Από τα διεθνή εμπειρικά δεδομένα η τιμή της πρέπει να κυμαίνεται εντός συγκεκριμένου διαστήματος. Το κάτω όριο είναι από 2 έως 4, ενώ το άνω όριο από 8 έως 10. Επομένως θα πρέπει τα παραπάνω όρια να λαμβάνονται υπόψη κατά την συγγραφή του πηγαίου κώδικα, προκειμένου να διασφαλίζονται στοιχειωδώς τα κριτήρια και οι παράγοντες ποιότητας λογισμικού.

2.1 Τρόποι Υπολογισμού της Κυκλωματικής Πολυπλοκότητας

α) Η κυκλωματική πολυπλοκότητα βασίζεται ολοκληρωτικά στη δομή του γραφήματος ελέγχου ροής (control flow graph) της εξεταζόμενης μονάδας λογισμικού. Το διάγραμμα ελέγχου ροής περιγράφει τη λογική δομή της μονάδας. Αποτελείται από κόμβους και ακμές. Οι κόμβοι παριστάνουν τις εντολές και οι ακμές τη μεταφορά του ελέγχου μεταξύ των κόμβων. Κάθε πιθανό μονοπάτι εκτέλεσης της μονάδας λογισμικού έχει ένα αντίστοιχο μονοπάτι από τον κόμβο εισόδου μέχρι τον

κόμβο εξόδου του γραφήματος ελέγχου ροής. Ακολουθεί εκπαιδευτικό παράδειγμα, που αφορά την εύρεση του μέγιστου τριών ακεραίων αριθμών:

Παράδειγμα 1

```
public int maximum(){
      int result;
      if(x>y)
            if(y>z)
                  result=x;
            else
                  if(x>z)
                        result=x;
                  else
                        result=z;
      else
            if(x>z)
                  result=y;
            else
                  if(z>y)
                        result=z;
                  else
                        result=y;
      return result;
}
```

Η κυκλωματική πολυπλοκότητα στον εξεταζόμενο κώδικα είναι *CC = 16-12+2 => CC = 6*. Ισοδύναμα υπολογίζεται και από τον τύπο: *CC = p+1*, όπου p το πλήθος των κόμβων που εκφράζουν δυαδικές αποφάσεις, δηλ. κόμβους με διπλές ακμές εξόδου. Άρα *CC = 5 + 1 => CC = 6*.

β) Εναλλακτικός τρόπος υπολογισμού απευθείας από τον κώδικα, χωρίς τη χρήση του γραφήματος ελέγχου ροής, δίνεται από τον τύπο:

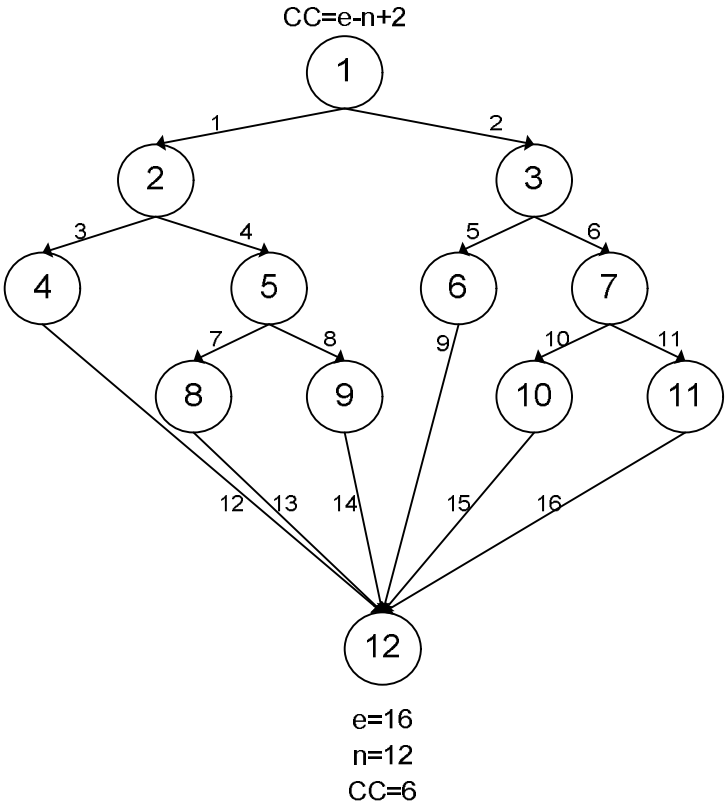
CC = p + 1, όπου *p* το άθροισμα των απλών συνθηκών των εντολών `if` και `while` καθώς και των Boolean τελεστών των σύνθετων συνθηκών. Ειδικά για την εντολή `switch` η συνεισφορά της στο άθροισμα *p* είναι το πλήθος των `case` εντολών συμπεριλαμβανομένης και της `default` περίπτωσης έστω και εάν αυτή δεν εμφανίζεται στον κώδικα.

Έτσι για τον κώδικα του Παραδείγματος 1 προκύπτει:

CC = (1+1+1+1+1)+1 = 5 + 1 => CC = 6.

γ) Έτερος εναλλακτικός τρόπος υπολογισμού από το γράφημα ελέγχου ροής είναι το πλήθος των κύκλων του γραφήματος αυξημένο κατά ένα, δηλ. *CC = c+1 = 5 + 1 => CC = 6*.

Το γράφημα ελέγχου ροής για την παραπάνω μέθοδο είναι το ακόλουθο:



Σχήμα 1: *Γράφημα ελέγχου ροής (Παράδειγμα 1)*

2.2 Απομείωση της κυκλωματικής πολυπλοκότητας

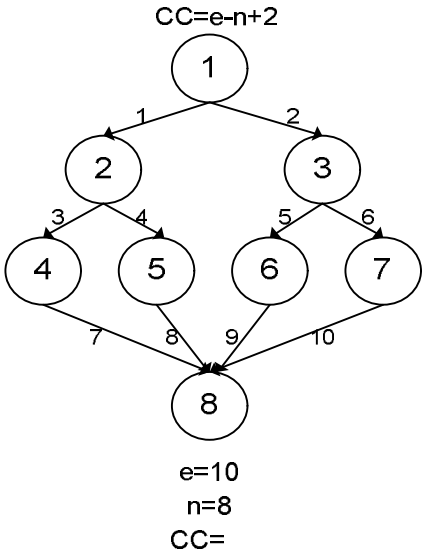
Η βασική μέθοδος απομείωσης της πολυπλοκότητας κώδικα είναι η τροποποίηση του κώδικα της μονάδας λογισμικού ώστε ο τροποποιημένος κώδικας να είναι απλούστερος με ελάττωση του αριθμού των φωλιασμένων εντολών απόφασης. Εάν αυτό δεν επαρκεί για τον περιορισμό της κυκλωματικής πολυπλοκότητας στα ανεκτά όρια τότε εξετάζεται η περίπτωση διάσπασης της μονάδας λογισμικού σε περισσότερες μονάδες. Συγκεκριμένα για το προηγούμενο παράδειγμα προτείνεται σε πρώτη φάση η εξής τροποποιημένη εκδοχή:

Παράδειγμα 2

```
public int maximum(){
      int result;
      if(x>y)
```

```
            if(x>z)
                  result=x;
            else
                  result=z;
      else
            if(y>z)
                  result=y;
            else
                  result=z;
      return result;
}
```

Το γράφημα ελέγχου ροής για την παραπάνω μέθοδο είναι το ακόλουθο:



Σχήμα 2: *Γράφημα ελέγχου ροής (Παράδειγμα 2)*

Η κυκλωματική πολυπλοκότητα στον εξεταζόμενο κώδικα είναι *CC = 10-8+2 => CC = 4*. Ισοδύναμα υπολογίζεται και από τον τύπο: *CC = p+1*, όπου *p* το πλήθος των κόμβων που εκφράζουν δυαδικές αποφάσεις, δηλ. κόμβους με διπλές ακμές εξόδου. Άρα *CC = 3 + 1 => CC = 4*.

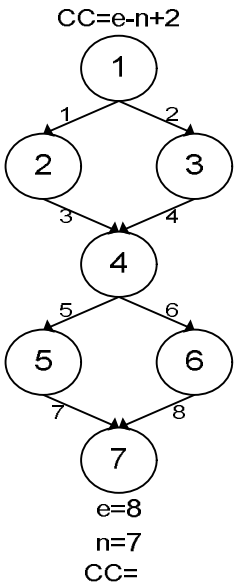
Επομένως η νέα τροποποιημένη εκδοχή του κώδικα εύρεσης του μέγιστου μεταξύ τριών ακεραίων επιφέρει μείωση της τιμής της κυκλωματικής πολυπλοκότητας κατά δύο μονάδες. Τούτο καθίσταται εμφανές τόσο από την πιο απλοποιημένη μορφή του κώδικα όσο και από το απλούστερο γράφημα ελέγχου ροής της νέας έκδοσης του κώδικα.

Μία ακόμη νέα και πιο κατανοητή εκδοχή του παραδείγματος που εξετάστηκε είναι και η παρακάτω:

Παράδειγμα 3

```
public int maximum(){
      int result, presult;
      if(x>y)
            presult=x;
      else
            presult=y;
      if(presult>z)
            result= presult;
      else
            result=z;
      return result;
}
```

Το γράφημα ελέγχου ροής για την παραπάνω μέθοδο είναι το ακόλουθο:



Σχήμα 3: *Γράφημα ελέγχου ροής (Παράδειγμα 3)*

Η κυκλωματική πολυπλοκότητα στον εξεταζόμενο κώδικα είναι *CC = 8-7+2* => *CC = 3*. Ισοδύναμα υπολογίζεται και από τον τύπο: *CC = p+1*, όπου *p* το πλήθος των κόμβων που εκφράζουν δυαδικές αποφάσεις, δηλ. κόμβους με διπλές ακμές εξόδου. Άρα *CC = 2 + 1* => *CC = 3*.

2.3 Πολυπλοκότητα διασύνδεσης

Η πολυπλοκότητα διασύνδεσης IC (interface complexity) είναι η μετρική που εκφράζει το άθροισμα του αριθμού των παραμέτρων και των σημείων επιστροφής (return) μίας μεθόδου. Αυξημένη τιμή της οδηγεί σε παρόμοια προβλήματα με εκείνα της αυξημένης κυκλωματικής πολυπλοκότητας *CC*. Έτσι ενώ στο Παράδειγμα 2 η *IC=1*, στην παρακάτω τροποποιημένη μορφή του η *IC=3+4 => IC=7*.

```
public int maximum(int x, int y, int z){
      if(x>y)
            if(x>z)
                  return x;
            else
                  return z;
      else
            if(y>z)
                  return y;
            else
                  return z;
}
```

3. Αποτελέσματα Μέτρησης Παραδείγματος

Παράδειγμα 1

Το Eclipse Metrics plugin και για τη μέθοδο `maximum()` εμφανίζει *CC=6*.

Metric	T..	Me...	Std....	Max...	Resource causing Maximum	Method
⊞ Number of Overridden Methods (a	0	0	0	0	/Max_3/src/MaxThree.java	
⊞ Number of Attributes (avg/max pe	3	3	0	3	/Max_3/src/MaxThree.java	
⊞ Number of Children (avg/max per	0	0	0	0	/Max_3/src/MaxThree.java	
Number of Classes	1					
⊞ Method Lines of Code (avg/max pe	24	4,8	6,645	18	/Max_3/src/MaxThree.java	maximum
⊞ Number of Methods (avg/max per	5	5	0	5	/Max_3/src/MaxThree.java	
⊞ Nested Block Depth (avg/max per		1	0	1	/Max_3/src/MaxThree.java	MaxThree
⊞ Depth of Inheritance Tree (avg/ma		1	0	1	/Max_3/src/MaxThree.java	
Number of Interfaces	0					
⊟ McCabe Cyclomatic Complexity (av		2	2	6	/Max_3/src/MaxThree.java	maximum
⊟ MaxThree		2	2	6	/Max_3/src/MaxThree.java	maximum
maximum	(6)					
MaxThree	1					
getX	1					
getY	1					
getZ	1					
Total Lines of Code	39					
⊞ Number of Parameters (avg/max p		0,6	1,2	3	/Max_3/src/MaxThree.java	MaxThree
⊞ Lack of Cohesion of Methods (avg/		0,5	0	0,5	/Max_3/src/MaxThree.java	
⊞ Number of Static Methods (avg/ma	0	0	0	0	/Max_3/src/MaxThree.java	
⊞ Specialization Index (avg/max per		0	0	0	/Max_3/src/MaxThree.java	
⊞ Weighted methods per Class (avg/	10	10	0	10	/Max_3/src/MaxThree.java	
⊞ Number of Static Attributes (avg/m	0	0	0	0	/Max_3/src/MaxThree.java	

Εικόνα 1: *Αποτελέσματα Παραδείγματος 1*

Παράδειγμα 2

Η μέθοδος `maximum()` παρουσιάζει μειωμένη τιμή *CC = 4*.

Metric	To...	Me...	Std....	Max...	Resource causing Maximum	Method
Number of Overridden Methods (a	0	0	0	0	/Max3/src/MaxThree.java	
Number of Attributes (avg/max pe	3	3	0	3	/Max3/src/MaxThree.java	
Number of Children (avg/max per	0	0	0	0	/Max3/src/MaxThree.java	
Number of Classes	1					
Method Lines of Code (avg/max pe	18	3,6	4,271	12	/Max3/src/MaxThree.java	maximum
Number of Methods (avg/max per	5	5	0	5	/Max3/src/MaxThree.java	
Nested Block Depth (avg/max per		1	0	1	/Max3/src/MaxThree.java	MaxThree
Depth of Inheritance Tree (avg/ma		1	0	1	/Max3/src/MaxThree.java	
Number of Interfaces	0					
McCabe Cyclomatic Complexity (av		1,6	1,2	4	/Max3/src/MaxThree.java	maximum
MaxThree		1,6	1,2	4	/Max3/src/MaxThree.java	maximum
maximum	4					
MaxThree	1					
getX	1					
getY	1					
getZ	1					
Total Lines of Code	33					
Number of Parameters (avg/max p		0,6	1,2	3	/Max3/src/MaxThree.java	MaxThree
Lack of Cohesion of Methods (avg/		0,5	0	0,5	/Max3/src/MaxThree.java	
Number of Static Methods (avg/ma	0	0	0	0	/Max3/src/MaxThree.java	
Specialization Index (avg/max per		0	0	0	/Max3/src/MaxThree.java	
Weighted methods per Class (avg/	8	8	0	8	/Max3/src/MaxThree.java	
Number of Static Attributes (avg/m	0	0	0	0	/Max3/src/MaxThree.java	

Εικόνα 2: *Αποτελέσματα Παραδείγματος 2*

Παράδειγμα 3

Τα αποτελέσματα της μέτρησης του κώδικα και ειδικά για τη μέθοδο `maximum()` δείχνουν νέα μειωμένη τιμή *CC = 3*.

Metric	To...	Me...	Std....	Max...	Resource causing Maximum	Method
Number of Overridden Methods	0					
Number of Attributes	3					
Number of Children	0					
Method Lines of Code (avg/max pe	16	3,2	3,487	10	/Max3/src/MaxThree.java	maximum
Number of Methods	5					
Nested Block Depth (avg/max per		1	0	1	/Max3/src/MaxThree.java	MaxThree
Depth of Inheritance Tree	1					
McCabe Cyclomatic Complexity (av		1,4	0,8	3	/Max3/src/MaxThree.java	maximum
maximum	3					
MaxThree	1					
getX	1					
getY	1					
getZ	1					
Number of Parameters (avg/max p		0,6	1,2	3	/Max3/src/MaxThree.java	MaxThree
Lack of Cohesion of Methods	0,5					
Number of Static Methods	0					
Specialization Index	0					
Weighted methods per Class	7					
Number of Static Attributes	0					

Εικόνα 3: *Αποτελέσματα Παραδείγματος 3*

Τελικά η αποφόρτιση της πολυπλοκότητας του κώδικα με συνεχείς τροποποιήσεις του, οδηγεί στην απλοποίηση και στην καλύτερη κατανόησή του.

4. Συμπεράσματα και προοπτική

4.1 Συμπεράσματα

Προκύπτει τελικά ότι μείωση της κυκλωματικής πολυπλοκότητας και της πολυπλοκότητας διασύνδεσης οδηγεί σε ευανάγνωστο, κατανοητό, ευκολοελέγξιμο και έγκυρο πηγαίο κώδικα. Το παράδειγμα που χρησιμοποιήθηκε είχε περισσότερο εκπαιδευτικό στόχο για την εφαρμογή καλών πρακτικών προγραμματισμού στη διδακτική διαδικασία αλλά και την επαλήθευση, ότι καλές μετρικές κώδικα είναι αναγκαίες για τη βελτίωση των ποιοτικών χαρακτηριστικών του και τη διασφάλιση της ποιότητάς του.

4.2 Προοπτική

Καθίσταται σαφές ότι βελτίωση όχι μόνο της κυκλωματικής πολυπλοκότητας και της πολυπλοκότητας διασύνδεσης αλλά και άλλων βασικών μετρικών κώδικα, όπως π.χ. η μειωμένη σύζευξη (coupling) και αυξημένη συνοχή (cohesion) (Arisholm, 2002), μπορούν να οδηγήσουν σε προγράμματα πιο συνεκτικά, πιο κατανοητά, εύκολα συντηρήσιμα, ευρύτερα ελέγξιμα και πιο επαναχρησιμοποιήσιμα (Fowler, 2008).

Ασφαλώς η επέκταση της μεθοδολογίας ανάπτυξης λογισμικού με έλεγχο, ώστε οι μετρικές του να βρίσκονται εντός των επιτρεπτών ορίων, απαιτεί την χρησιμοποίηση κώδικα μεγάλης κλίμακας και συνδυαστική χρήση των μετρικών για την επίτευξη βέλτιστου αποτελέσματος (Lanza & Marinescu, 2006).

Βιβλιογραφία

Arisholm, E. (2002). Dynamic coupling measures for object-oriented software. *Software Metrics 2002 Proc. Eighth IEEE Symposium, IEEE Comput. Soc.*, 33-42.

Booch, Gr., Maksimchuk, R., Engle, M., Conallen, J., Huston, K., & Young, B. (2007). *Object Oriented Analysis & Design with Applications*. Pearson Education.

Fenton, N., & Pfleeger, S. (1997). *Software Metrics A Rigorous & Practical Approach*. U.K., London: Thomson Computer Press.

Fowler, M. (2008). *Refactoring: Improving the Design of Existing Code*. Massachusetts: Addison-Wesley.

Lanza, M., & Marinescu R. (2006). *Object-Oriented Metrics in Practice*. Germany, Berlin: Springer-Verlag.

McCabe, T. (1976). A Complexity Measure. *IEEE Transactions in Software Engineering, SE-2, No. 4,* 308-320.

Καμέας, Α. (2003). *Εγκυροποίηση Λογισμικού*. Πάτρα: Ε.Α.Π.

Λιακέας, Γ. (2008). *Εισαγωγή στην Java*. Αθήνα: Κλειδάριθμος.

Ξένος, Μ. (2003). *Διαχείριση και Ποιότητα Λογισμικού*. Πάτρα: Ε.Α.Π.

Χατζηγεωργίου, Α. (2008). *Ανάπτυξη συστήματος βάσει ICONIX*. Πάτρα: Ε.Α.Π.