Ποιότητα Λογισμικού

Μετρικές Αντικειμενοστραφούς Σχεδίασης

Η Μέτρηση στην ζωή μας

- *Οι μετρήσεις βρίσκονται στην καρδιά πολλών συστημάτων που επηρεάζουν σημαντικά την ζωή μας*
  - *Οικονομικά (τιμές, πληθωρισμός, χρηματιστηριακοί δείκτες)*
  - *Ραντάρ (ανίχνευση στόχων)*
  - *Ιατρικά (χοληστερίνη, σάκχαρο, δείκτες αίματος), βοηθούν στη διάγνωση παθήσεων/ασθενειών*
  - *Ατμοσφαιρικά (δείκτες ρύπανσης, καιρικών συνθηκών)*
  - *Βιομηχανία αυτοκινήτων* **(ιπποδύναμη,κατανάλωση, αποστάσεις, «συμπεριφορά»)**
    - **Εξετάζονται διαρκώς σε αγώνες αυτοκινήτων**
- **Δίχως μετρήσεις είναι αδύνατον να λειτουργήσει η τεχνολογία**

Ποιότητα Λογισμικού

- ISO / IEC 9126

  Functionality
  Reliability
  Usability
  Efficiency
  Maintainability
  Portability

Τι είναι η μέτρηση…

"Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules." [Norman Fenton]

Μέτρηση ονομάζεται η διαδικασία κατά την οποία οι αριθμοί και τα σύμβολα συνδέονται με ιδιότητες οντοτήτων του πραγματικού κόσμου έτσι ώστε να τα περιγράφουν σύμφωνα με αυστηρά καθορισμένους κανόνες

Μετρικές Λογισμικού (1/6)

- "You cannot control what you cannot measure“ [Tom De Marco]
- "The degree to which you can express something in numbers is the degree to which you really understand it.“ [Lord Kelvin]
- Για την παρακολούθηση, διαχείριση, ποιοτική μελέτη και βελτίωση ΟΠΟΙΟΥΔΗΠΟΤΕ τεχνικού έργου είναι απαραίτητη η έννοια της μέτρησης
- Η εξαγωγή μέτρων είναι υποκειμενική (π.χ. αξιολόγηση ελκυστικότητας αυτοκινήτου, πολυπλοκότητας λογισμικού)

Μετρικές Λογισμικού (2/6)

- Μέτρο: Ποσοτική ένδειξη αριθμού, διαστάσεων, χωρητικότητας, όγκου κτλ προϊόντος ή διαδικασίας
- Μέτρηση: Διαδικασία υπολογισμού του μέτρου
- Μετρική: Ποσοτική εκτίμηση του βαθμού κατά τον οποίο ένα σύστημα κατέχει ένα χαρακτηριστικό
- Αριθμός λαθών σε ένα πρόγραμμα = ***Μέτρο***
- Συλλογή και καταμέτρηση λαθών = ***Μέτρηση***
- Συσχετισμός λαθών με κάποιο χαρακτηριστικό, π.χ. ποιότητα (πάνω από 100 λάθη -> κακή ποιότητα, κάτω από 10 λάθη => καλή ποιότητα) = ***Μετρική***

Μετρικές Λογισμικού (3/6)

- Μία διαδικασία μέτρησης περιλαμβάνει τις δραστηριότητες:

  Διατύπωση (Καθορισμός μετρικών)
  Συλλογή (Συγκέντρωση δεδομένων)
  Ανάλυση (Υπολογισμός μετρικών)
  Ερμηνεία (Αξιολόγηση μετρικών)
  Ανάδραση (Συστάσεις για βελτίωση)

Μετρικές Λογισμικού (4/6)

- **Αρχές Διατύπωσης Μετρικών:**
  - Οι στόχοι των μετρήσεων πρέπει να καθοριστούν πριν από τη συλλογή δεδομένων
  - Σαφής ορισμός των μετρικών
  - Χρήση μετρικών προσαρμοσμένων στα προϊόντα και τις διαδικασίες

- **Αρχές Συλλογής & Ανάλυσης Μετρικών**
  - Όπου είναι δυνατό η συλλογή και ανάλυση θα πρέπει να αυτοματοποιείται
  - Χρήση αξιόπιστων στατιστικών τεχνικών για διερεύνηση εσωτερικών και εξωτερικών χαρακτηριστικών (π.χ. συσχετισμός πολυπλοκότητας και αριθμού λαθών)
  - Για κάθε μετρική θα πρέπει να επιδιώκεται ο καθορισμός συγκεκριμένων κανόνων ερμηνείας

Μετρικές Λογισμικού (5/6)

Η ιδανική μετρική θα πρέπει να είναι:

- Απλή και υπολογίσιμη
- Εμπειρικά και διαισθητικά πειστική
- Συνεπής και αντικειμενική
- Συνεπής ως προς τη χρήση μονάδων
- Ανεξάρτητη από τη γλώσσα
- Ουσιαστικός μηχανισμός ανάδρασης

Μετρικές Λογισμικού (6/6)

"Validation of a software measure is the process of ensuring that the measure is a proper numerical characterisation of the claimed attribute; this means showing that the representation condition is satisfied." [Norman Fenton]

Μια μέτρηση είναι «καλή» όταν η αριθμητική της τιμή χαρακτηρίζει με ακρίβεια την συγκεκριμένη ιδιότητα του αντικειμένου

Κατηγοριοποίηση μέτρων λογισμικού

- **Διεργασίες**, συλλογές σχετ. δραστηριοτήτων λογισμικού
- **Προϊόντα,** παράγωγα από διεργασίες
- **Πόροι,** που απαιτούνται από μία δραστηριότητα διεργασίας

- Σε κάθε κατηγορία διακρίνουμε τα *χαρακτηριστικά:*
- **Εσωτερικά,** μετρούν την ίδια τη κατηγορία (πχ. εφαρμογή)
- **Εξωτερικά,** μετρούν πως η κατηγορία σχετίζεται με το περιβάλλον, δηλ. την συμπεριφορά της (εκτέλεση εφαρμογής)

Κατηγορίες - Χαρακτηριστικά

Προϊόντα	***Εσωτερικά***	***Εξωτερικά***
Σχεδίαση Κώδικας	Μέγεθος, επαναχρ/ση, σύζευξη, συνοχή Γραμμές κώδικα (LOC), αλγοριθμική πολυπλοκότητα	Ποιότητα, πολυπλοκότητα, συντηρησιμότητα Αξιοπιστία, ευχρηστία, συντηρησιμότητα
Διεργασίες Έλεγχος	Χρόνος, προσπάθεια, αρ. λαθών	Κόστος, σταθερότητα
Πόροι Ομάδες	Μέγεθος, επίπεδο επικοινωνίας, δομή	Παραγωγικότητα, ποιότητα

Μετρικές μεγέθους

- Size of the software produced
- *LOC* - Lines Of Code
- *KLOC* - 1000 Lines Of Code
- *SLOC* – Statement Lines of Code (ignore whitespace)
- Typical Measures:
  - Errors/KLOC, Defects/KLOC, Cost/LOC, Documentation Pages/KLOC

Μετρικές LOC

- Easy to use
- Easy to compute
- Language & programmer dependent

Μετρικές πολυπλοκότητας

- LOC - a function of complexity
- Language and programmer dependent
- Halstead's Software Science (entropy measures)
  - n_1 - number of distinct operators
  - n_2 - number of distinct operands
  - N_1 - total number of operators
  - N_2 - total number of operands

Παράδειγμα

```
if (k < 2)
{
  if (k > 3)
    x = x*k;
}
```

- Distinct operators: if ( ) { } > < = * ;
- Distinct operands: k 2 3 x
- $n_1 = 10$
- $n_2 = 4$
- $N_1 = 13$
- $N_2 = 7$

Halstead's Metrics

- Amenable to experimental verification [1970s]

- Program length: $N = N_1 + N_2$
- Program vocabulary: $n = n_1 + n_2$

- Estimated length: $\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
  - Close estimate of length for well structured programs

- Purity ratio: PR = $\hat{N}/N$

Πολυπλοκότητα προγράμματος

- Volume: V = N $\log_2$ n
  - Number of bits to provide a unique designator for each of the n items in the program vocabulary.

- Difficulty

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2}$$

- Program effort: E=D*V

Object – Oriented Metrics

Chidamber & Kemerer (1991): Πρότειναν την πρώτη σουίτα αντικειμενοστραφών μετρικών

Αρκετές παραλλαγές στη συνέχεια

Βασικές Κατηγορίες Μετρικών:

- *Πολυπλοκότητας*
- *Κληρονομικότητας*
- *Μεγέθους*
- *Σύζευξης*
- *Συνοχής*

Πολυπλοκότητα (1/2)

- Ο αριθμός των μεθόδων και η πολυπλοκότητα των μεθόδων μιας κλάσης, είναι ενδεικτικά του πόσος χρόνος και προσπάθεια χρειάζεται για την ανάπτυξη και την συντήρηση της.
- Όσο μεγαλύτερος είναι ο αριθμός των μεθόδων μιας κλάσης τόσο μεγαλύτερη είναι η εξάρτηση των «παιδιών» της από αυτήν.
- Οι κλάσεις με μεγάλο αριθμό μεθόδων, πιθανότητα στοχεύουν σε συγκεκριμένους τύπους εφαρμογών και μειώνεται η πιθανότητα επαναχρησιμοποίησης τους.

Πολυπλοκότητα (2/2)

- Τρεις μετρικές πολυπλοκότητας

  Cyclomatic Complexity (CC), πολυπλοκότητα μεθόδου

  Weighted Method per Class 1 (WMPC1), πολυπλοκότητα κλάσης

  Weighted Method per Class 2 (WMPC2), πολυπλοκότητα κλάσης

  Response for Class (RFC), πολυπλοκότητα κλάσης

Cyclomatic Complexity

- Από τις παλαιότερες μετρικές, McGabe [1976]

- Αναπαριστά την γνωστική πολυπλοκότητα της κλάσης. Μετρά το πλήθος των πιθανών μονοπατιών σε ένας αλγόριθμο υπολογίζοντας τις διακριτές περιοχές του διαγράμματος ροής, δηλαδή των αριθμό των if, for και while στο σώμα της μεθόδου.

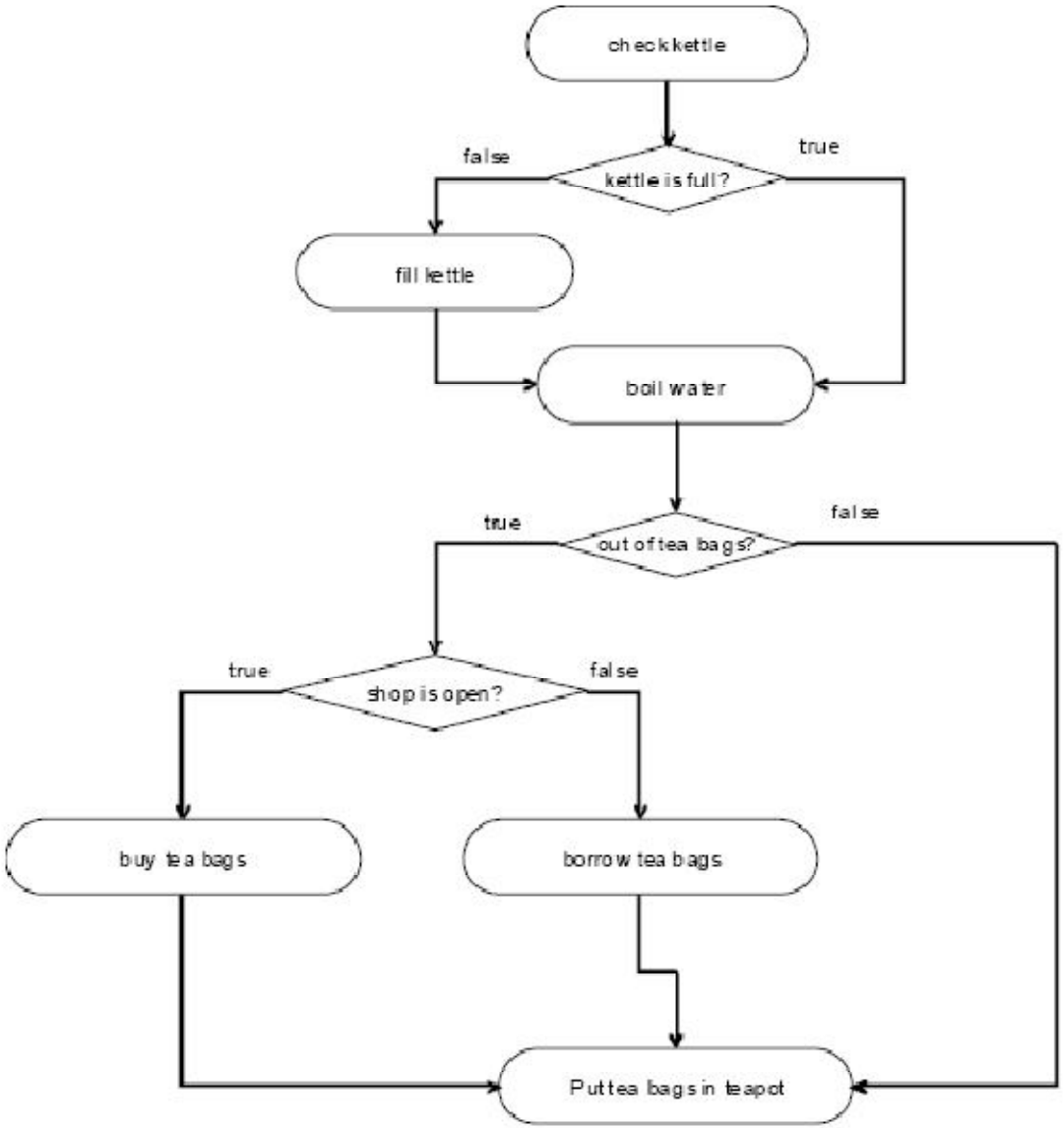
- Υπολογίζεται από το διάγραμμα ελέγχου ροής

  CC = L - N + 2

  L: number of links in the control flow graph

  N:number of nodes in the control flow graph

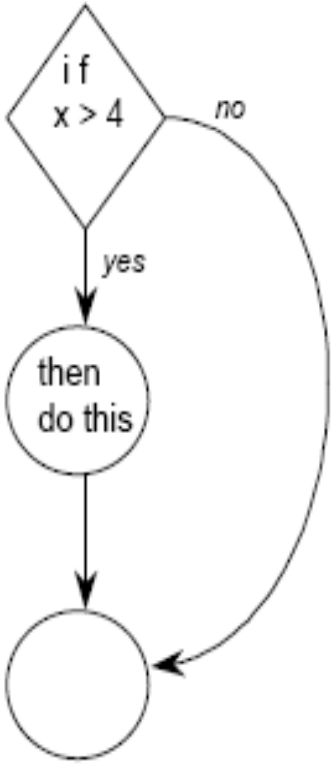
Cyclomatic Complexity



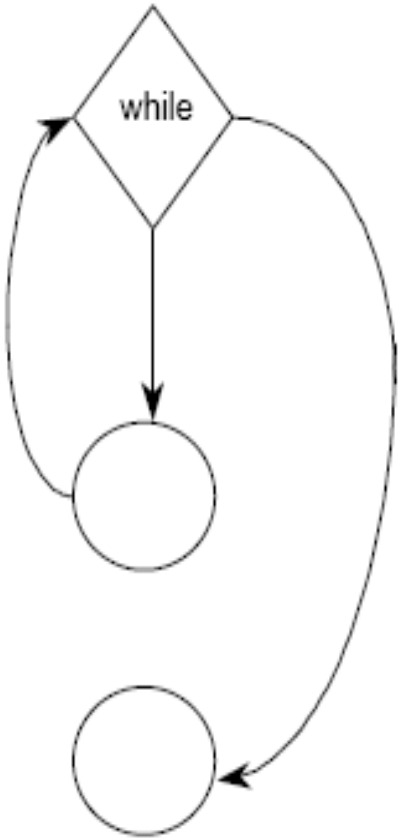
Cyclomatic Complexity

Number of Independent Test Paths => edges - nodes + 2

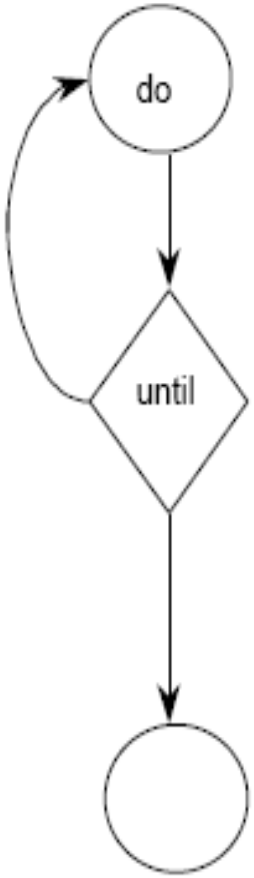
sequence:
1-2+2=1



if / then:
3-3+2=2



while loop:
3-3+2=2



until loop:
3-3+2=2

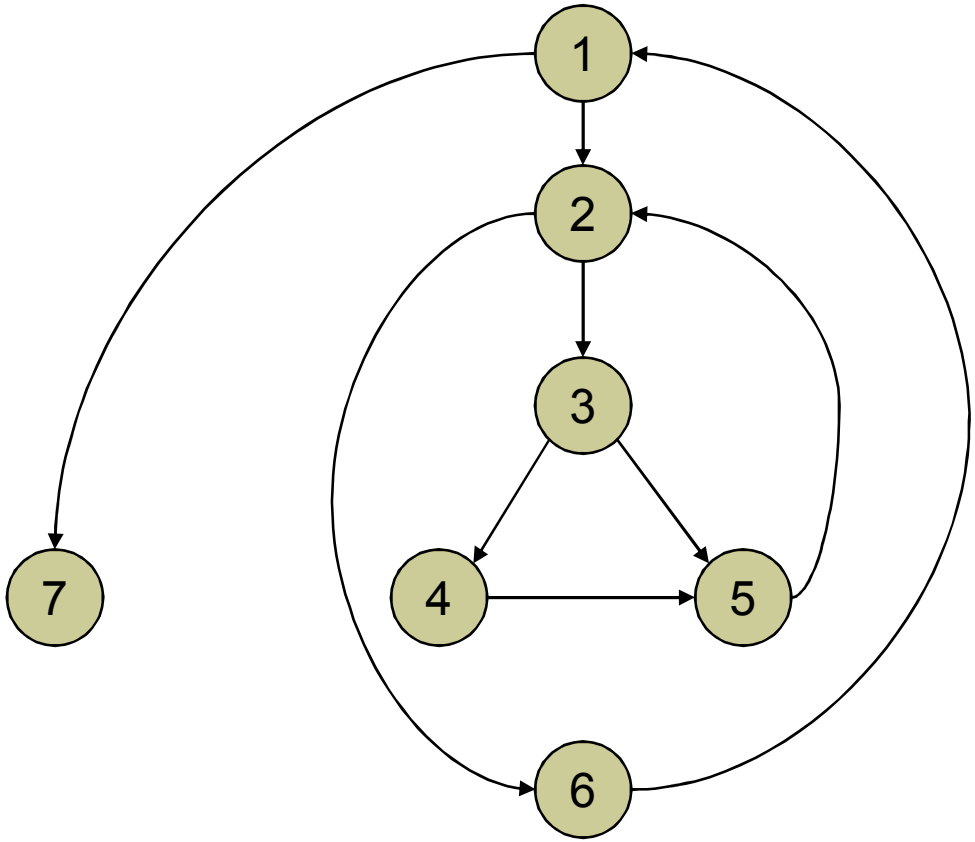
Cyclomatic Complexity

- Set of independent paths through the graph (basis set)
- V(G) = E – N + 2
  - E is the number of flow graph edges
  - N is the number of nodes
- V(G) = P + 1
  - P is the number of predicate nodes

Example

```
  i = 0;
1 while (i<n-1) do
     j = i + 1;
  2 while (j<n) do
    3 if A[i]<A[j] then
      4 swap(A[i], A[j]);
  5 end do;
  6 i=i+1;
7 end do;
```

Flow Graph



Computing V(G)

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4
- Basis Set
  - 1, 7
  - 1, 2, 6, 1, 7
  - 1, 2, 3, 4, 5, 2, 6, 1, 7
  - 1, 2, 3, 5, 2, 6, 1, 7

```
class Company {
Employee* emp[4];
public:
    void setEmployee(Employee* e,int a) {emp[a]=e;}
    void printAll() {
      for (int i=0;i<4;i++) {
            cout<<" Name              = "<<emp[i]->getName();
            cout<<" Type Of Employment = "<<emp[i]->getEmpType();
            cout<<" Type Of Payment    = "<<emp[i]->getPayType();
            emp[i]->calcSalary();
            cout<<endl;
      }
    }
};
```

- setEmployee CC 1
- printAll CC 2

Weighted Method per Class 1

- Μετράει την πολυπλοκότητα μίας κλάσης, με βάση την πολυπλοκότητα των μεθόδων της.
- Η πολυπλοκότητα των μεθόδων μετριέται με χρήση της CC.
- Ως WMPC1 μιας κλάσης ορίζεται ως ο μέσος όρος ή το άθροισμα των CC όλων της των μεθόδων.
- Στη διαδικασία δεν περιλαμβάνονται μέθοδοι που κληρονομούνται από υπερκλάσεις.

```
class Company {
Employee* emp[4];
public:
    void setEmployee(Employee* e,int a) {emp[a]=e;}
    void printAll() {
      for (int i=0;i<4;i++) {
            cout<<" Name              = "<<emp[i]->getName();
            cout<<" Type Of Employment = "<<emp[i]->getEmpType();
            cout<<" Type Of Payment    = "<<emp[i]->getPayType();
            emp[i]->calcSalary();
            cout<<endl;
      }
    }
};
```

WMPC=1.5

Weighted Method per Class 2

- Η συγκεκριμένη μετρική βασίζεται στην υπόθεση ότι μια κλάση με περισσότερες μεθόδους από μία άλλη είναι πιο σύνθετη.
- Επιπλέον, θεωρεί ότι μια μέθοδος με περισσότερες παραμέτρους από μια άλλη είναι και πιο σύνθετη.
- Η μετρική αθροίζει τις μεθόδους και τις παραμέτρους των μεθόδων μιας κλάσης.
- Στη διαδικασία δεν περιλαμβάνονται μέθοδοι που κληρονομούνται από υπερκλάσεις.

Response for a Class (RFC)

Είναι το σύνολο των μεθόδων που μπορούν να κληθούν σε απάντηση ενός μηνύματος προς αντικείμενο κλάσης

"RFC = |RS| where RS is the response set for the class

- "Response set of an object ≡ { set of all methods that can be invoked in response to a message to the object }"

RFC example 1

```
public class A {
  private B aB;
  public void methodA1() {
    return aB.methodB1();
  }
  public void methodA2(C aC) {
    return aC.methodC1();
  }
}
```

RS = { methodA1, methodA2, methodB1, methodC1}

RFC example 2

```
public class A {
  private B aB;
  public void methodA1() {
    return aB.methodB1();
  }
  public void methodA2() {
    return aB.methodB1();
  }
}
```

RS = { methodA1, methodA2, methodB1}

RS = { methodA1, methodA2, methodB1, methodB1}?

Κληρονομικότητα

- Δύο μετρικές κληρονομικότητας

  Depth of Inheritance Tree (DIT)

  Number of Child Classes (NOCC)

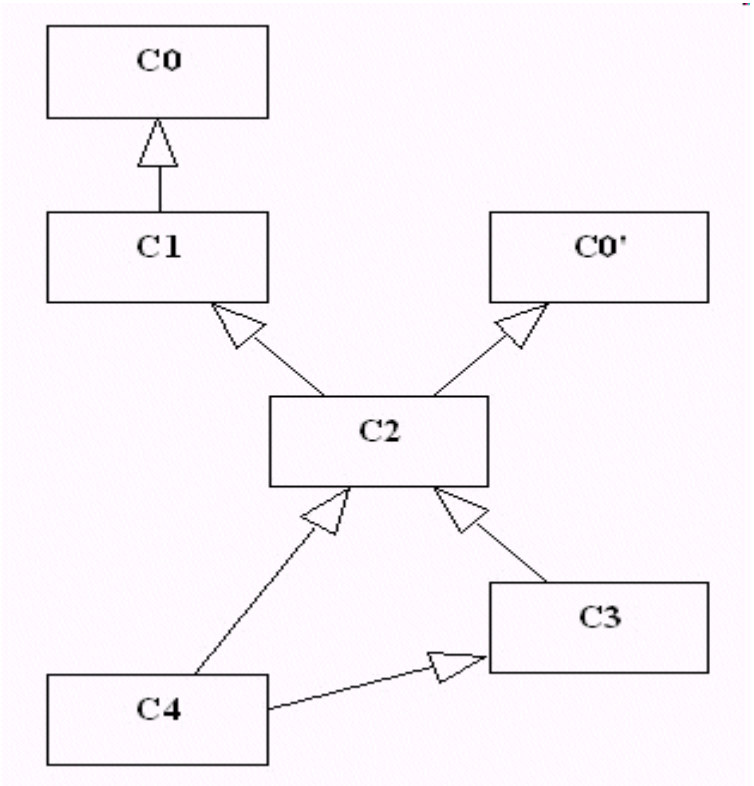
Depth of Inheritance Tree

- Όσο βαθύτερα είναι μια κλάση στην ιεραρχία, τόσο μεγαλύτερος είναι ο αριθμός των μεθόδων που πιθανών να κληρονομεί, γεγονός που κάνει δύσκολη την πρόβλεψη της συμπεριφοράς της.

- Μεγάλα δέντρα κληρονομικότητας καθιστούν μεγάλη σχεδιαστική πολυπλοκότητα, λαμβάνοντας υπ' όψη ότι εμπλέκονται περισσότερες μέθοδοι και κλάσεις.

- Όσο πιο βαθιά βρίσκεται μία κλάση στην ιεραρχία τόσο πιο μεγάλες είναι οι πιθανότητες επαναχρησιμοποίησης μέσω κληρονομημένων μεθόδων.

Depth of Inheritance Tree



DIT (C0) = 0
DIT (C0') = 0
DIT (C1) = 1
DIT (C2) = 2
DIT (C3) = 3
DIT (C4) = 4

Number of Children

- Μετράει τον αριθμό των κλάσεων που κληρονομούν την κλάση υπό εξέταση.
- Μη μηδενική τιμή της μετρικής συνιστά ότι η συγκεκριμένη κλάση επαναχρησιμοποιείται.
- Παρόλα αυτά, η αφαίρεση της κλάσης μπορεί να είναι φτωχή αν υπάρχουν πάρα πολλές υποκλάσεις.
- Επιπλέον, υψηλές τιμές της μετρικής δείχνουν ότι θα χρειαστεί αυξημένος αριθμός ελέγχων για κάθε κλάση – παιδί.

Μέγεθος

- Δύο μετρικές μεγέθους

  Lines of Code (LOC): Μετρά τον αριθμό των γραμμών κώδικα. Σχόλια και κενές γραμμές δεν υπολογίζονται.

  Number of Classes (NOC): Μετρά τον αριθμό των κλάσεων του συστήματος. Υπολογίζεται μόνο σε επίπεδο πακέτου ή συστήματος.

Σύζευξη (1/2)

- Αυξημένα επίπεδα σύζευξης είναι ανεπιθύμητα σε συστήματα αποτελούμενα από υπό-μονάδες και αποτελούν τροχοπέδη στην επαναχρησιμοποίηση.
- Όσο πιο ανεξάρτητο είναι ένα αντικείμενο τόσο πιο εύκολα επαναχρησιμοποιείται.
- Όσο πιο αυξημένη είναι η σύζευξη μεταξύ των αντικειμένων ενός συστήματος, τόσο πιο ευαίσθητο είναι σε αλλαγές σε διάφορα μέρη του σχεδίου. => Δυσκολότερη συντήρηση.

Σύζευξη (2/2)

- Τέσσερις μετρικές σύζευξης

  Coupling Factor (CF), σύζευξη σε επίπεδο συστήματος

  Coupling Between Objects (CBO), σύζευξη σε επίπεδο κλάσης

  Fan Out (FO) , σύζευξη σε επίπεδο κλάσης

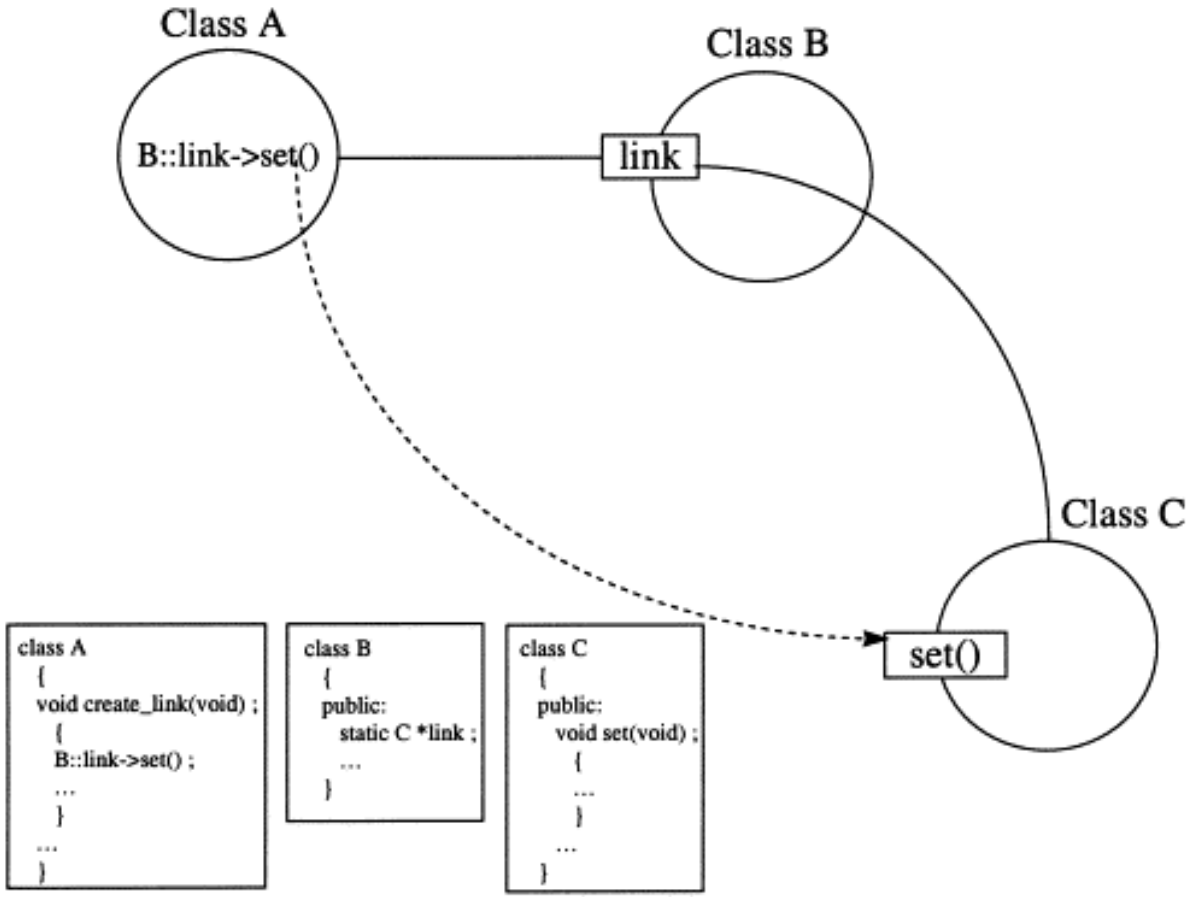
  Fan In (FI) , σύζευξη σε επίπεδο κλάσης

Coupling Factor

- Υπολογίζεται μόνο σε επίπεδο συστήματος
- Υπολογίζεται ως κλάσμα
- Αριθμητής είναι ο αριθμός των μη κληρονομούμενων συζεύξεων.
- Παρονομαστής είναι ο μέγιστος αριθμός πιθανών συζεύξεων στο σύστημα.

Coupling Between Objects

- Μετράει τον αριθμό των κλάσεων με τις οποίες συνδέεται μια κλάση.



Fan Out – Fan In

- Κάθε συσχέτιση που λαμβάνετε υπόψη στον υπολογισμό της CBO μπορεί να «έρχεται» ή να «φεύγει» από την κλάση.
- Ο αριθμός των ακμών που φεύγουν από μια κλάση ονομάζεται Fan-Out
- Ο αριθμός των ακμών που καταλήγουν σε μια κλάση ονομάζεται Fan-In
- Μεγάλο Fan-Out => Η κλάση δεν είναι Αυτάρκης
- Μεγάλο Fan-In => Η κλάση παρέχει μεγάλη λειτουργικότητα

```
class Company {
Employee* emp[4];
public:
    void setEmployee(Employee* e,int a) {emp[a]=e;}
    void printAll() {
      for (int i=0;i<4;i++) {
            cout<<" Name              = "<<emp[i]->getName();
            cout<<" Type Of Employment = "<<emp[i]->getEmpType();
            cout<<" Type Of Payment    = "<<emp[i]->getPayType();
            emp[i]->calcSalary();
            cout<<endl;
      }
    }
};
```

- FI 0
- FO 1
- CBO 1

Συνοχή

- Η συνοχή των μεθόδων σε μία κλάση είναι επιθυμητή από την στιγμή που προωθεί την ενθυλάκωση.
- Η έλλειψη συνοχής υποδηλώνει ότι η κλάση πιθανώς να πρέπει να διασπαστεί σε δύο ή περισσότερες κλάσεις.
- Η έλλειψη συνοχής αυξάνει την πολυπλοκότητα και την πιθανότητα εμφάνισης λαθών κατά την ανάπτυξη.
- Τρεις Μετρικές Συνοχής:
  Lack Of Cohesion of Methods 1 (LOCOM1)
  Lack Of Cohesion of Methods 2 (LOCOM2)
  Lack Of Cohesion of Methods 3 (LOCOM3)

Lack of Cohesion of Methods 1

Θεωρούμε μία κλάση C με μεθόδους M_1, M_2, . . ., M_n

Έστω $\{I_i\}$ το σύνολο των μεταβλητών που χρησιμοποιούνται από τη μέθοδο M_i.

Υπάρχουν n τέτοια σύνολα: $\{I_1\}$, $\{I_2\}$, . . . , $\{I_n\}$

$$P = \{(I_i, I_j) | \, I_i \cap I_j = \varnothing$$

$$Q = \{(I_i, I_j) | \, I_i \cap I_j \neq \varnothing$$

$$LCOM = \begin{cases} |P| - |Q| & if \ |P| > |Q| \\ 0 & otherwise \end{cases}$$

```
public PersonDetails {
private String firstname;
private String surname;
private String street;
private String city;
public PersonDetails() {}                                   //I1 = { }
public setName(String f, String s) {    //I2 = { firstname, surname }
firstname = f; surname = s;
}
public setAddress(String st, String c) {    //I3 = { street, city }
street = st; city = c;
}
public void printAddress() {                //I4 = { street, city }
System.out.println( street);
System.out.println( city);
}
public void printName() {           //I5 = { firstname, surname }
System.out.println( firstname + " " + surname);
}
}
```

Pair (m_i, m_j)	$\mathcal{I}_i \cap \mathcal{I}_j$
PersonDetails, setName	ϕ
PersonDetails, setAddress	ϕ
PersonDetails, printAddress	ϕ
PersonDetails, printName	ϕ
setName, setAddress	ϕ
setName, printAddress	ϕ
setName, printName	$\{\_firstname, \_surname\}$
setAddress, printAddress	$\{\_street, \_city\}$
setAddress, printName	ϕ
printAddress, printName	ϕ
LCOM	8
ignoring constructor	4

```
class Company {
Employee* emp[4];
public:
    void setEmployee(Employee* e,int a) {emp[a]=e;}
    void printAll() {
      for (int i=0;i<4;i++) {
            cout<<" Name              = "<<emp[i]->getName();
            cout<<" Type Of Employment = "<<emp[i]->getEmpType();
            cout<<" Type Of Payment    = "<<emp[i]->getPayType();
            emp[i]->calcSalary();
            cout<<endl;
      }
    }
};
```

- setEmployee - printAll LCOM {1}

Lack of Cohesion of Methods 2

Θεωρούμε μία κλάση C με m μεθόδους M_1, M_2, . . ., M_n

Κάθε μέθοδος προσπελάζει a ιδιότητες, A_1, A_2, . . ., A_a

Έστω $a(M_k)$ = ο αριθμός των ιδιοτήτων που προσπελάζεται από την μέθοδο M_k

Έστω $m(A_k)$ = ο αριθμός των μεθόδων που προσπελάζουν την ιδιότητα A_k

$$LOCOM2 = 1 - \frac{\sum_{i=1}^{a} m(Ai)}{ma}$$

Lack of Cohesion of Methods 3

Θεωρούμε μία κλάση C με m μεθόδους M_1, M_2, . . ., M_n

Κάθε μέθοδος προσπελάζει a ιδιότητες, A_1, A_2, . . ., A_a

Έστω $a(M_k)$ = ο αριθμός των ιδιοτήτων που προσπελάζεται από την μέθοδο M_k

Έστω $m(A_k)$ = ο αριθμός των μεθόδων που προσπελάζουν την ιδιότητα A_k

$$LOCOM_3 = \frac{\left(m - \frac{\sum_{i=1}^{a} m(Ai)}{a} \right)}{m-1}$$