Computer Science Journal of Moldova, vol.**22**, no.1(64), 2014

Evaluating the impact of software metrics on defects prediction. Part 2

Arwa Abu Asad, Izzat Alsmadi

Abstract

Software metrics are used as indicators of the quality of the developed software. Metrics can be collected from any software part such as: code, design, or requirements. In this paper, we evaluated several examples of design coupling metrics. Analysis and experiments follow hereinafter to demonstrate the use and value of those metrics. This is the second part for a paper we published in Computer Science Journal of Moldova (CSJM), V.21, N.2(62), 2013 [19]. We proposed and evaluated several design and code coupling metrics. In this part, we collected source code from Scarab open source project. This open source is selected due to the availability of bug reports. We used bug reports for further analysis and association where bugs are used to form a class for classification and prediction purposes. Metrics are collected and analyzed automatically through the developed tool. Statistical and data mining methods are then used to generalize some findings related to the collected metrics. In addition classification and prediction algorithms are used to correlate collected metrics with high level quality attributes such as maintainability and defects prediction.

Keywords: Design metrics, Object-Oriented Designs, Coupling metrics, software faults.

1 Introduction

Design activity should consider the dependency between classes so that changing in one class should not be propagated to several classes. Quality attributes such as: dependency and modularity can be measured or

evaluated through coupling metrics. Coupling describes how classes are related and dependent on each other. Coupling is considered as one of the fundamental design metrics that aims to minimize coupling among different modules facilitating understanding, maintaining, reusability, modularity and testing tasks of the software and design. In addition, they provide information to the designers regarding the capability of their design to change or to be reused. Low coupling results in components' self-containment, which in turn increases class understandability in isolation. Furthermore, it improves maintainability and increases potentials for reuse. On the other side, high coupling between two classes or components makes it more difficult to understand one of them in separation from the other. Thus, ripple changes are increased due to high dependency among classes. Myers et al., (1974) [18] defined six coupling metrics between pairs of modules. Those are: content, common, external, control, stamp, and data coupling. Eder et al. (1994) [7], Hitz et al. (1995) [10], and Briand et al. (1997) [3] defined OO coupling frameworks. These three frameworks were unified by Briand et al. (1999) to a more formalized framework.

A software product can hardly be free from errors. Maintenance and testing stages involve looking for bugs and fixing them. However, the allocation of potential regions where errors are or can be located plays important role in budget and effort reduction in software projects. In this study, we have used fault proneness as a quality predictor. On the other hand, we have investigated several coupling metrics as possible predictors for fault proneness.

In this paper several design coupling metrics proposed by Briand et al (1999) [4] are assessed using a large set of open source code projects. One of the major design goals is to minimize coupling. Therefore, it is important to be able to develop tools for coupling assessment before or after code development.

2 Coupling Metrics

Coupling is a measure of interconnection among modules. One of major goals in software design is that classes should be loosely coupled.

Therefore, Simple connection between modules will produce more understandable and maintainable software. Loosely coupled software will be less subjective to ripple effect in case of code changes. Myers et al. (1974) [18] defined six procedural coupling types, which are explained in Table 1.

Table 1. Procedural coupling levels

Coupling type	Description
DATA COUPLING	Data coupling occurs when passing pure simple data between two modules by parameters using a simple elementary piece of argument list and every item in the list is used.
STAMP COUPLING	Stamp coupling occurs between modules when passing composite data through parameters using a data structure containing fields, which may or may not be used.
CONTROL COUPLING	Control coupling occurs between modules when data are passed that influence the internal logic of a module (e.g. flags)
COMMON COUPLING	Common coupling occurs when modules communicate sharing global data areas so common coupling is also known as global coupling.
CONTENT COUPLING	Content coupling occurs between two modules if one modifies the internals of other module. In practice, only assembler language allows content coupling. Most object-oriented programming languages do not allow implementing content coupling.

These metrics were realized into object oriented design resulting in many metrics. This research is interested in investigating coupling metrics unified by Briand et al. (1999) [4] framework. Here is a brief preview of coupling metrics that will be measured in this research:

Coupling between object (CBO) (Chidamber & Kemerer, 1994 [6])

Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted as the original definition: "two classes are coupled when methods of one class use methods or instance variables defined by the other class".

Definition 1

CBO = number of classes to which a class is coupled

Response for a Class (RFC) (Chidamber & Kemerer, 1994 [6]) The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class It counts only the first level of calls outside of the class. RFC is simply the number of methods in the set. It regards all methods and properties declared. Calls to properties: Set and Get are all counted separately.

Definition 2

RFC = M + R (First-step measure) where M = number of methods in the class, R = number of remote methods directly called by methods of the class

Message passing coupling (MPC) (Li & Henry, 1993) [13]

The message passing is the number of call statements defined in a class or the number of messages between objects in local method of a class. Thus, this includes only counting method invocations to other classes, and these classes with inheritance relationship have been excluded from counting.

Data Abstraction Coupling (DAC) (Li & Henry, 1993) [13]

Data Abstraction Coupling is the total number of other class types in attribute declarations. It is also referred to as aggregation coupling. DAC does not count primitive types, system types, and inherited types from the base classes. DAC has two variants, which are:

Definition 3

DAC: The number of attributes in a class that have another class as their type (count the repetition of class type)
DAC1: The number of different classes that are used as types of attributes in a class

Information-flow-based coupling (ICP) (Lee et al., 1995) [11]

Information-flow-based coupling is the number of implemented method m of one class plus the number of polymorphically invoked methods of other classes, weighted by the number of parameters of the invoked method.

Definition 4

$$ICP^c(m) = \sum_{m' \in PIM(m) - (M_{NEW}(c) \cup M_{OVR}(c))} (1 + |Par(m')|) \cdot NPI(m, m')$$

Where:
$M_{OVR}(c)$: is the set of overriding methods of class c.
$M_{NEW}(c)$: is the set of non-inherited, non-overriding methods of class c.
$Par(m)$: is the set of parameters of method m.
$PIM(m)$: is the set of Polymorphically Invoked Methods of m.
$NPI(m, m')$: is the Number of Polymorphic Invocations of m' by m.

Briand et al. (1999) [4] redefined ICP as the number of method invocations in one class weighted by the number of parameters of the methods invoked by class methods (the weight is number of parameters +1 but for empirical consideration they refined it to number of parameters only).

Definition 5

$$\frac{Total\ Number\ of\ Method\ Invocations}{Total\ Number\ of\ Method\ Invocations + \sum_{i=0}^{all\ methods} parmOf(m_i)}$$

ICP has two variations, which are:

- H-ICP of just inheritance invocation are considered invocation to method of ancestors classes only
- NIH-ICP non inheritance invocation

ICP is the sum of IH-ICP and NIH-ICP

Definition 6

$$ICP = IH - ICP + NIH - ICP$$

Coupling factor (COF) (Abreu et al., 1995)

COF is defined as: the ratio of the maximum possible number of couplings in the system to the actual number of couplings not imputable to inheritance.

Definition 7

Coupling Factor

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC - 2 \times \sum_{i=1}^{TC} DC(C_i)}$$

$$is\_client(C_c, C_s) = \begin{vmatrix} 1 & iff & C_c \Rightarrow C_s \wedge C_c \neq C_s \\ & & \wedge\neg(C_c \rightarrow C_s) \\ 0 & otherwise & \end{vmatrix}$$

- $is\_client(x, y) = 1$ iff a dependency exists between the client and the server classes. 0 otherwise;
- $(TC2 - TC)$ is the total number of possible dependencies.

Briand suite [3-5]

Briand et al. suggested suite of coupling metrics. The abbreviations used in Briand measurements:

A: Coupling to ancestor classes.

D: Coupling to Descendents.

IC: import coupling, the measure counts for a class C all interactions where C is using another class;

EC – export coupling, counts interactions where class D is the used class.

CA – Class-attribute interaction

CM – Class-method interaction

MM – Method-method interaction

The definitions for CA, CM, and MM are as follows:

Class-attribute interaction (CA) (Briand et al., 1997 [3])

Class-attribute interaction occurs if a class contains an attribute of type another class. It is the number of class-attribute interactions from one class C to another class D.

Definition 8

$$CA(c,d) = \left|a|a \in A_I(c') \wedge T(a) = d\right|.$$

A: is the set of attributes in class c.

$T(a)$: is the type of attribute where the attribute type will be a class.

Class-method interaction (CM) (Briand et al., 1997 [3])

Class-method interaction occurs if a newly defined method of one class has a parameter of type another class.

Definition 9

$$CM(c,d) = \sum_{m \in M_{NEW}(c)} |a|a \in Par(m) \wedge T(a) = d|.$$

$T(a)$: is the type of attribute where the attribute type will be a class.

$Par(m)$: is the set of parameters of method m.

Method-method interaction (MM) (Briand et al., 1997 [3])
Method-method interaction occurs if a method implemented at class c statically invokes a method of class d (newly defined or overriding), or receives a pointer to such a method. The number of method-method interactions from class c to class d.

Definition 10

$$MM(c,d) = \sum_{m \in M_I(c)} \sum_{m' \in M_{NEW}(d) \cup M_{OVR}(d)} (NSI(m,m') + PP(m,m')).$$

$NSI(m,m')$: is the number of static invocations of method m' by m.

M: is the set of methods in a class.

$M_{OVR}(c)$: is the set of overriding methods of class c.

$M_{NEW}(c)$: is the set of non-inherited, non-overriding methods of class c.

Afferent and Efferent Coupling (Martin 2002) [14]
The OO Design package or system metrics of Martin (Afferent and Efferent Coupling), based on fan-in and fan-out metrics, are largely used in the industry and commonly referenced in the academy. Lots of static analysis tools are able to extract them. They can help in understanding how defects are likely to ripple among different classes or components. They measure how much a particular class, method, component etc. is coupled with other components as either calling them or being called by them. The remainder of this paper is organized as follows: Section 3 provides an overview of related work. Section 4 presents goals and approaches. Section 5 presents an experimental section and paper is concluded in Section 6.

3 Related Work

In this section, we will list some examples of related papers of evaluating software design or coupling metrics.

Hitz and Montazeri (1995) [10] distinguished between coupling among objects (CLO) and coupling among classes (CLC). CLC is principally important in maintenance or change dependencies within program. On the other hand, OLC is relevant for runtime-oriented activities like testing and debugging. In addition, they argued that various levels of coupling depend on three attributes: Stability, Access Type and Scope of Access, and each combination of these attributes would have different strength of coupling. Coupling measures the strength of connection between two classes as pairs. Although, coupling was defined as characteristics of pairs of classes, but as a metric, it is the total number of couples that one class has with other classes. Therefore, implicitly all coupling connections are supposed to be of equal strength among a class with others in the system level (Norman et al., 1992). Therefore, in the proposed system coupling was computed using previous definition.

Briand et al. (1999) [4] refined existing framework defined by (Eder et al., 1994 [7]; Hitz and Montazeri, 1995 [10]; and Briand et al. 1997 [3]) into comprehensive and formalized framework for coupling measurements. The framework for coupling consists of six criteria, which are:

1. Type of connection
2. Direction of connection: Fan-in or Fan-out.
3. Granularity of the measure: Domain of the measure and how to count coupling connections.
4. Stability of server: whether a class stable or changing frequently.
5. Direct or indirect coupling.
6. Inheritance: Inheritance-based or non inheritance-based coupling.

Briand et al. (1999) [4] also distinguished two directions of coupling, which are import and export. Export occurs when a class is used as server class in the interaction. On the other hand, import occurs when

a class is used as client class in the interaction. But Lee (2007) [12] specified that Dependency between classes could be in one direction or two directions. A high fan-out represents a class coupling to other classes. High fan-in represents a good level of reuse.

Many object oriented metrics were proposed in the previous studies to predict software quality attributes. One of these attributes is fault proneness. Michael English et al. (2009) [9] summarized the results from 23 papers that evaluated software metrics and their correlation with software fault prone modules. Their evaluation showed that CK and LOC metrics are the most used and evaluated metrics. Emam et al. (2001) [8] used the CK metrics in addition to Briand's coupling metrics to predict faults on a commercial Java system [3-5]. Basili et al. (1996) studied the correlation of fault-proneness with CK metrics for eight student projects. The results showed that WMC, CBO, DIT, NOC and RFC have a significant correlation with faults whereas LCOM didn't have same significant correlation with faults. Tang et al. (1999) [16] found that higher WMC and RFC were found to be associated with fault-proneness on three real time systems. Menzies et al. (2007) [15], on the other hand, evaluated fault proneness for C and Java projects.

There are various types of methods to predict faulty classes such as statistical methods, machine learning methods, etc. However, the trend recently is moved from traditional statistical methods to machine learning methods. Zhou et al. (2006) [17] used logistic regression and machine learning methods to show how OO metrics and fault proneness are correlated. The results showed that WMC, CBO, and SLOC were found to be strong predictors across all severity levels based on the public domain NASA datasets.

4 Goals and Approaches

As described earlier, this is the second part of one paper. In the first part that was published in CSJM, V.21, N.2(62), 2013 [19], we described in details the coupling metrics that we want to investigate. We described them with examples and also described the developed tool to automatically collect those coupling metrics. In this part, we assembled

a case study of a large number of source code.

We will define the terminologies and the abbreviations that are used in later sections. Table 2 shows the terms and the definitions used in this paper as well as the abbreviation of terms.

5 Results and Analysis

In this section, we will present the results from the collection of metrics and their statistical and correlation analysis with bug reports for the same source code. The first section includes descriptive statistics for the collected coupling metrics.

5.1 Descriptive Statistics for Coupling measures

Table 3 presents the descriptive statistics for the coupling measures collected from Scarab project. The columns: Min, Max, Mean, and Std. Deviation are the: minimum value, maximum value, mean value and standard deviation, respectively. $N > 5$ is NO if the count of non-zero values is less than six.

The following observations can be made from Table 3:

- The measures that are counting various relationships through inheritance (i.e. ACAIC, DCAEC, ACMIC, and ACMIC) have all relatively zero values for all columns which means that no extensive use of inheritance (at the attribute and method use, override and reuse) is observed in the Scarab project.
- The largest maximum value is for RFC, which also has the largest mean. This may be explained by the fact that RFC counts method invocations plus the number of methods. MPC has the next maximum mean value as it counts the sent statement.
- All measures with significant variance (i.e. six or more non-zero values – $N > 5$) were subjected to further analysis. Therefore, ACAIC, DCAEC, ACMIC, and DCMEC are removed from further analysis.

Table 2. Terminology Abbreviation

Term	Definition	Abbreviation
Defined methods	Methods declared within class C	M_{def}
Inherited methods	Methods declared within parent class and inherited and (not overridden) within child class C	M_{inh}
Polymorphic method	Methods defined within an interface	M_{poly}
New methods	Methods declared within class C that do not override inherited ones	M_{new}
Overriding methods	Methods declared within class C that override (redefine) inherited ones	M_{over}
Invoked methods	Methods that can be invoked in association with class C	M_{inv}
External calls	Invocations to a method defined in other classes	$Call_{ex}$
Internal calls	Invocations to a method in the same class	$Call_{int}$
Polymorphic calls	Invocations to a method defined in an interface	$Call_{poly}$
Inherited calls	Invocations to a method defined in parent class through object of child class	$Call_{inh}$
Defined attributes	Attributes declared within class C	$Attr_{def}$
Overriding attributes	Attributes declared within class C that overrides (redefines) inherited ones	$Attr_{over}$
Inherited attributes	Attributes inherited (and not overridden) in class C	$Attr_{inh}$
Used attributes	Attributes that can be manipulated in association with class C (those external public attribute from other classes used in method of class C)	$Attr_{use}$
Instance attributes	Attributes of type class	$Attr_{inst}$
Instance parameters	Parameters of type class	arg_{ins}

Table 3. Descriptive statistics for collected metrics

Descriptive Statistics					
Metrics	Min	Max	Mean	Std. Deviation	N>5
CBO	0	53	5.17	8.13	
RFC	0	1098	33.42	85.49	
MPC	0	928	24.56	71.79	
DAC	0	12	.36	1.11	
DAC1	0	11	.33	.99	
ICP	.0	1.6	.46	.39	
ACAIC	0	0	.00	.00	NO
DCAEC	0	0	.00	.00	NO
ACMIC	0	0	.00	.00	NO
DCMEC	0	0	.00	.00	NO
AMMIC	0	32	.21	1.81	
DMMEC	0	44	.21	2.37	
NOC	0	31	.33	2.42	
NDC	0	37	.39	2.87	
NAC	0	4	.39	.70	

5.2 A defect prediction model

In this section, we built a defect prediction model to validate the correlation between coupling metrics and reported bugs. On the other hand, we also measured which coupling metrics have more significant effect in comparison with the rest of collected metrics.

Figure 1 shows the results from Gain Ratio (GR) feature selection method applied on the collected dataset. It can be seen that inheritance metrics have no significant role in bug prediction. CBO, RFC, MPC and ICP are the most significant metrics in defects prediction. In particular, CBO has the most significant effect.

J48graft data mining method is then used on the collected dataset after applying feature selection. Table 4 shows prediction performance

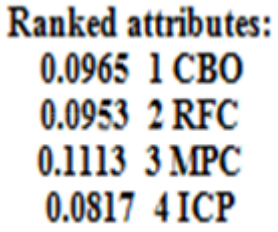

Figure 1. Gain Ratio feature evaluator

metrics: recall, precision, and accuracy. The Table 4 shows the recall, precision and accuracy for faulty (i.e. true case), and for not faulty classes (i.e. false case). The Table 4 shows also the recall, precision and accuracy for overall average of both. *10 cross validation* method is used as the selection for testing and training.

Table 4. Scarab performance metrics with J48graft and 10 cross validation

	Precision	Recall	Accuracy
Not faulty	0.77	0.69	0.69
Faulty	0.61	0.70	0.70
Weighted Avg.	0.70	0.70	0.70

Figure 2 shows the classification tree after applying gain ratio using J48graft.

5.3 Study Limitation

Similar to most experiments conducted in this field one of the limitations is the dependency of the results in one source code project.

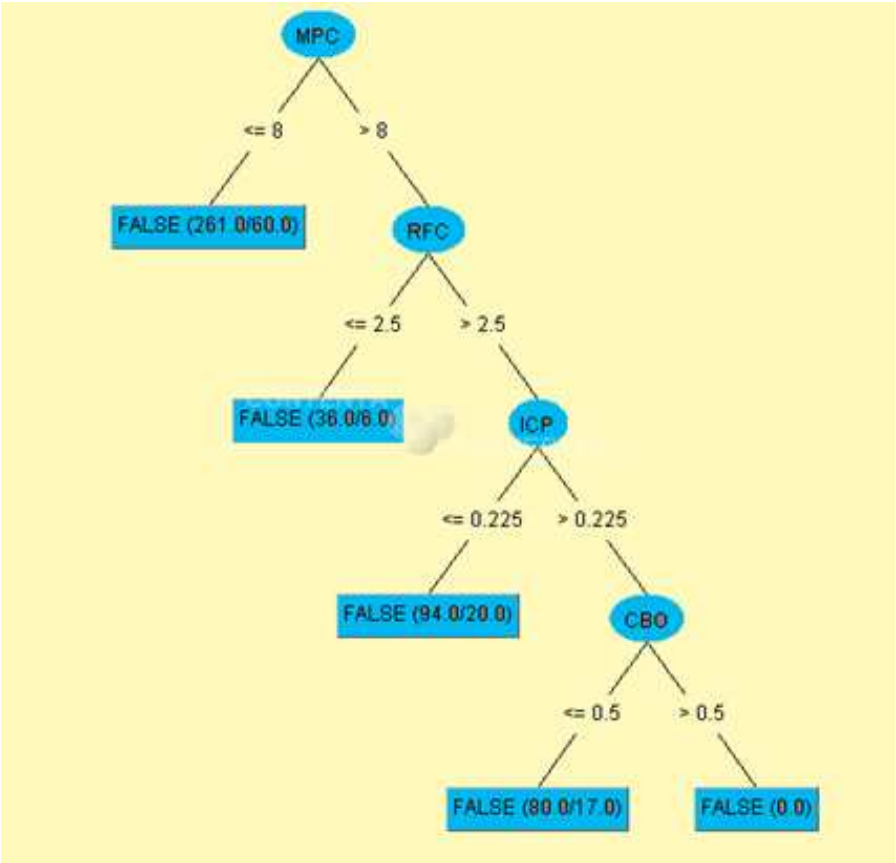

Figure 2. Metrics-bugs-dependent classification tree

This may have a risk of having possibly biased results that may not be generalized to all software source codes especially those that may not share the project same domain, environment, etc.

In addition, the dataset used in this research may not be perfect or complete. Bugs reporting and judgments are also human subjective. In addition, bugs were not classified or categorized based on their seriousness or importance. Those are examples of some of the possible limitations on generalizing the results in this study on other software products.

6 Summary and Conclusion

In this paper, we presented several metrics for evaluating the quality of software design. Those metrics focus on evaluating design coupling quality where software components are expected to be moderately coupled. Experiments and analysis were conducted to demonstrate the value of the proposed and evaluated metrics. Scarab open source is used in the case study due to the availability of bug reports related to the usage of this software. We used such bug reports for classification and prediction where a bug class is formed with values of true or false based on whether the subject class contains bugs or not (based on bug reports). Some coupling metrics such as: CBO, RFC, MPC and ICP showed significant correlation with bugs. This means that those coupling metrics can be used for bugs prediction. Monitoring those metrics early in the software project can help software project managements monitor quality aspects especially those related to bugs.

References

[1] F. Abreu, M. Goul, R. Esteves. *Toward the design quality evaluation of object-oriented software systems*, Proc. Fifth Int'l Conf. Software Quality, Austin, Texas, (1995), pp.44–57.

[2] V. Basili, L. Briand, W. Melo. *A Validation of Object Oriented Design Metrics as Quality Indicators*, IEEE Transactions on Software Engineering, 22, (1996), pp. 751–761.

[3] L. Briand, P. Devanbu, W. Melo. *An investigation into coupling measures for C++*, *Proc. 19th Int'l Conf. Software Eng, ICSE*, (1997), pp. 412–421.

[4] L. Briand, J.W. Daly, J.K. Wust. *A unified framework for coupling measurment in objmct-oriented systems*, IEEE Tranractions on Software Engineering, 25 (1) (1999), pp. 91–121.

[5] L.C. Briand, J.K. Wust, J.W. Daly, D.V. Porter *Exploring the relationships between design measures and software quality in object oriented systems'*, Journal of systems and Software, 51(3), (2000), pp. 254–273.

[6] S. Chidamber, C. Kemerer. *A metrics suite for object oriented design*, IEEE Trans Journal. Software Eng, 20 (6), (1994), pp. 476–493.

[7] J. Eder, C. Kappel, M. Schrefl. *Coupling and Cohesion in Object-Oriented Systems*. Technical Report, Univ. of Klagenlurt, available at ftp://ftp.ifs.uni-inz.ac.at/pub/publications/1993/0293.ps.gz, (1994).

[8] K.E. Emam, W. Melo, J.C. Machado. *The prediction of faulty classes using object-oriented design metrics*, Journal of Systems and Software, 56(1), (2001), pp. 63–75.

[9] M. English, Ch. Exton, I. Rigon, B. Cleary. *Fault Detection and Prediction in an Open-Source Software Project*, In Proceedings of Promise, (2009).

[10] M. Hitz, B. Montazeri. *Measuring coupling and cohesion in object-oriented systems*, Proc. ESEC '95 fifth European Software Eng. Conf., Barcelona, Spain, 1 (4), (1995), pp. 2–10.

[11] Y. Lee, B.S. Liang, S.F. Wu, F.J. Wang. *Measuring the coupling and cohesion of an object-oriented program based on information flow*, Proc. Int'l Conf. Software Quality, Maribor, Slovenia 12, (1995), pp. 81–90.

[12] Y. Lee. *Automated source code measurement environment for software quality*, Doctor Thesis, (2007) Auburn, Alabama, USA.

[13] W. Li, S. Henry. *Object-Oriented metrics that predict maintainability*, J. of Systems and Software, 23 (2), (1993), pp. 111–122.

[14] R.C. Martin. *Agile Software Development, Principles, Patterns, and Practices*, Prentice Hall (2002).

[15] T. Menzies, J.Gneenwald, A. Frank. *Data mining static code attributes to learn defect predictors*, IEEE Trans. on Soft. Eng., 33(1), (2007), pp. 2–13.

[16] M.H. Tang, M.H. Kao, M.H. Chen. *An empirical study on object-oriented metrics*, In Sixth International Software Metrics Symposium, (1999), pp. 242–249.

[17] Y. Zhou, H.H. Leung. *Empirical Analysis of Object-Oriented Design Metrics for Predicting High and low Severity Faults*, IEEE Transactions on Software Engineering, 32(10), (2006), pp. 771–789.

[18] W. Stevens, G. Myers, L.L. Constantine. *Structured design*, IBM Systems Journal, 13 (2), (1974), pp. 115–139.

[19] A.A. Asad, I. Alsmadi. *Design and code coupling assessment based on defects prediction. Part 1*, Computer Science Journal of Moldova (CSJM), V.21, N.2(62), (2013), pp. 204–224.

Arwa Abu Asad, Izzat Alsmadi

Received October 17, 2012

Arwa Abu Asad
Institution: Yarmouk University
Address: CIS department
E–mail: *arwa_abuasad@yahoo.com*

Izzat Alsmadi
Institution : Yarmouk University
Address : CIS department
E–mail: *ialsmadi@yu.edu.jo*