Chhabra JK, Gupta V. A survey of dynamic software metrics. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(5): 1016–1029 Sept. 2010. DOI 10.1007/s11390-010-1080-9

# A Survey of Dynamic Software Metrics

Jitender Kumar Chhabra and Varun Gupta

*Department of Computer Engineering, National Institute of Technology, Kurukshetra-136119, India*

E-mail: jitenderchhabra@rediffmail.com; varun3dec@yahoo.com

Received June 12, 2008; revised April 3, 2010.

**Abstract** Software metrics help us to make meaningful estimates for software products and guide us in taking managerial and technical decisions. However, conventional static metrics have been found to be inadequate for modern object-oriented software due to the presence of object-oriented features such as polymorphism, dynamic binding, inheritance and unused code. This fact motivates us to focus on dynamic metrics in place of traditional static metrics. Moreover, dynamic metrics are more precise than static metrics as they are able to capture the dynamic behaviour of the software system during measurement. These dynamic metrics are usually obtained from the execution traces of the code or from the executable models. In this paper, advantages of dynamic metrics over static metrics are discussed and then a survey of the existing dynamic metrics is carried out. These metrics are characterized into different categories such as dynamic coupling metrics, dynamic cohesion metrics. Towards end of the paper, potential research challenges and opportunities in the field of dynamic metrics are identified.

**Keywords** metrics, object-oriented programming, software engineering

## 1 Introduction

Software metrics are the units of measurement that are used to characterize software engineering products (design, source code etc.), software engineering processes (analysis, design, coding, testing etc.) and software engineering professionals (the efficiency of an individual tester, or the productivity of an individual designer). If used properly, software engineering metrics allow us to quantitatively define the degree of success or failure, for a product, a process, or a person, make meaningful and useful managerial and technical decisions, and make quantified and meaningful estimates[1]. Thus, incorporating metrics into development plans is a simple step towards creating better systems.

The most popular and time-honoured software metrics have been LOC (lines of code) and Cyclomatic Complexity[2]. These measures were originally defined for procedural programs and later incorporated for object-oriented systems. The LOC metric is a measure of a size of a module and Cyclomatic Complexity measures logical complexity of a module. Software metrics proposed and used for procedural paradigm have been found inadequate for object-oriented software products[3] mainly because of the distinguishing features of the object-oriented paradigm such as classes, encapsulation, inheritance and polymorphism. Chidamber and Kemerer[4] addressed the need for new and modified metrics for object-oriented systems by introducing a set of metrics. Their metrics suite has been a subject of discussion for many years and the authors themselves and other researchers have continued to improve or add to the "CK" metric suite. Other major metrics suites proposed for object-oriented systems are MOOD metrics[5], Lorenz and Kidd metrics[6], Briand *et al.* metrics[7], Harrison *et al.* metrics[8] and Bansiya *et al.* metrics[9]. Static metrics focus on static properties of the software and a number of static metrics have been proposed in literature for the measurement of coupling[4,7,10-16], cohesion[4,11-12,17-27] and other attributes of object-oriented software using design or source code of the software, which are static in nature. The static metrics are able to quantify various aspects of the complexity of design or source code of a software system, but their ability to accurately predict the dynamic behaviour of an application is as yet unproven. Traditional static metrics alone may be insufficient in evaluating the dynamic behaviour of an application at runtime, as its behaviour will be influenced by the execution environment as well as the complexity of the source code. Object-oriented features such as polymorphism, dynamic binding, inheritance and common presence of unused ("dead") code in commercial software, render the static metrics imprecise, as they do not precisely reflect the runtime situation of the software[28]. For instance, target method of a

Survey

polymorphic call depends on the run time type of the object receiving the call. In programs that employ inheritance, this target may change[29]. Moreover, the complex dynamic behaviour of many real-time applications motivates us to focus on dynamic metrics in place of traditional static metrics.

Dynamic metrics are the class of software metrics that capture the dynamic behaviour of the software system and are usually obtained from the execution traces of the code or from the executable models. Major dynamic metrics proposed have been for the measurement of coupling[28-40], cohesion[41-43], and complexity[34,44-46].

In this paper, need and benefits of having dynamic metrics for software systems are discussed and major contributions of the paper are briefly described as follows:

- a comparison between static and dynamic metrics is done;
- study of existing dynamic metrics categorized into different classes is carried out;
- different dynamic metrics belonging to same class are compared and their relationships with external quality attributes are examined;
- research challenges and opportunities available in the field of dynamic metrics are highlighted;

The remainder of this paper is organized as follows. In Section 2 advantages of dynamic metrics over static metrics are discussed and in Section 3 various dynamic coupling metrics proposed in literature are studied and compared on the basis of their relations with quality attributes. Section 4 examines and compares the existing metrics for the measurement of dynamic cohesion and Section 5 describes the dynamic complexity metrics and their relations with quality attributes. Section 6 discusses miscellaneous dynamic metrics and Section 7 discusses the relatively new concept of pseudo dynamic metrics. Section 8 identifies the potential research directions and challenges in the area of dynamic metrics and Section 9 concludes the paper.

## 2 Advantages of Dynamic Metrics over Static Metrics

In this section, we will discuss benefits of having dynamic metrics in comparison to their static counterparts. Static measures are obviously simpler to collect because there is no need to run the software. Moreover, to obtain dynamic metrics, code or simulation models of the software system are needed, which are available very late in the software development lifecycle. Static metrics are widely used due to the fact that they are easier to obtain, especially at the early stages of software development. However, the potential benefits of dynamic metrics collected by executing the program outweigh the complexity and cost of measuring them.

Static metrics satisfy the purpose of judging the quantity attributes like size and complexity of the software artefacts. But they are less precise than dynamic metrics in measurement of the quality attributes of software such as reliability, testability, as static metrics are evaluated only by means of static inspection of the software artefacts. The quality of software systems is more dependent on the runtime behaviour than the potential characteristics implied by the static analysis of the software system. Dynamic metrics are computed on the basis of the data collected during actual execution of the system, and thus directly reflect the quality attributes (performance, error rates etc.) of the software in its operational mode. Moreover, static metrics deal with the structural aspects of a software system, whereas runtime metrics also deal with the behavioural aspects of the system. For example, according to the results of a controlled experiment conducted by Briand *et al.*, static coupling measures may be insufficient to explain discrepancies in changeability for object-oriented designs[47]. Moreover, static metrics are somewhat constrained in their ability to deal with inheritance, polymorphism and dynamic binding issues since the runtime types of field access and method invocation are not known. However, dynamic metrics are capable to deal with such issues. The major differences between static and dynamic metrics are listed in Table 1.

**Table 1.** Comparison Between Static and Dynamic Metrics

| Static Metrics | Dynamic Metrics |
|---|---|
| Simpler to collect | Difficult to obtain |
| Available at the early stages of software development | Accessible very late in software development lifecycle |
| Less accurate than dynamic metrics in measuring qualitative attributes of software | Suitable for measuring quantitative as well as qualitative attributes of software |
| Deal with the structural aspects of the software system | Deal with the behavioral aspects of the system also |
| Inefficient to deal with dead code and OO features such as inheritance, polymorphism and dynamic binding | Dynamic metrics are capable to deal with all object-oriented features and dead code |
| Less precise than dynamic metrics for the real-life systems | More precise than static metrics for the real-life systems |

Major benefit of using dynamic metrics in software engineering is their ability to more precisely measure the internal attributes of software like coupling, complexity etc., which have direct impact on quality factors of software such as reliability, testability, reusability, maintainability, performance, error-rates. In subsequent sections, different dynamic metrics proposed in literature till date are discussed and presented into different categories depending on their types.

## 3 Dynamic Coupling Metrics

Dynamic coupling metrics are used to measure actual coupling taking place between a pair of objects or classes at runtime in a software system. Dynamic coupling metrics are measured at object level and can be aggregated to class or system level. Moreover, dynamic coupling measures can be defined at different stages of software development lifecycle like design-time or coding-time. In subsequent sub-sections, we will discuss and then compare different types of dynamic coupling metrics proposed in literature.

### 3.1 EOC and IOC Metrics

Yacoub *et al.*[34] propose object level dynamic coupling measures, Export Object Coupling (EOC) and Import Object Coupling (IOC) based on executable object-oriented design models and these models are generated using Real-Time Object Oriented Modelling (ROOM) language[48]. The design models used to collect the coupling measures are a kind of sequence diagrams[49] that allow execution simulation. The EOC or IOC count the number of messages sent between two distinct objects $o_i$ and $o_j$ in a given ROOM sequence diagram $x$ (in opposite directions), divided by the total number of messages exchanged in $x$. Thus, the result of each metric is the percentage that reflects the "intensity" of interactions between two objects in a particular direction relative to the total number of object interaction in $x$. These metrics are defined within a scenario scope, i.e., measurements are calculated for parts of the design model that are activated during the execution of a specific scenario triggered by an input stimulus. Then, these metrics can be extended to have an application scope, i.e., for all scenarios. These metrics are presented in Table 2.

For instance, in a simple scenario $x_1$ where $o_1$ sends three messages ($m_1$, $m_2$ and $m_3$) to $o_2$ and $o_2$ sends two messages ($m_4$ and $m_5$) to $o_1$, then $EOCx_1(o_1, o_2) = (3/5) \times 100\% = 60\%$ and $IOCx_1(o_1, o_2) = (2/5) \times 100\% = 40\%$.

### 3.2 Arisholm Metrics Suite

The concept of import and export coupling given by Yacoub *et al.* is extended by Arisholm[29] to take into consideration the direction as well as class level, where he proposes 12 dynamic coupling measures for object-oriented software divided along three orthogonal dimensions: direction, mapping and strength. Out of these 12 metrics, six metrics are defined at object level and other six are defined at class level. Each dynamic coupling metric name (e.g., IC_OC) starts with either IC or EC to distinguish between *import coupling* and *export coupling* based on *direction* of the coupling. *import coupling* counts the messages sent from an object or class, whereas *export coupling* counts the messages received by an object or class. The next letter indicates the mapping level ('O' for Object and 'C' for Class). Mapping level here defines the granularity level at which coupling is being measured by the concerned metric. The last letter in the name of a particular metric denotes the *strength* of coupling as shown in Table 3. Here, *strength* of coupling measures the amount of association between the two objects. The amount of association between the objects may be quantified at three levels of granularity: 1) *Dynamic messages* ($D$): the number of times each message is sent from one object to another; 2) *Distinct method invocations* ($M$): the number of distinct method invocations between two objects; 3) *Distinct classes* ($C$): the number of distinct classes involved in association between the objects. The twelve metrics proposed by Arisholm are defined in Table 3.

**Table 2.** EOC and IOC Metrics

| Metric | Description | Definition |
|---|---|---|
| Export Object Coupling | $EOC_x(o_i, o_j)$, the export coupling for object $o_i$ w.r.t. object $o_j$, is the percentage of the number of messages sent from $o_i$ to $o_j$ w.r.t the total number of messages exchanged during the execution of the scenario $x$. | $EOC_x(o_i, o_j) = \frac{\lvert\{M_x(o_i,o_j) \mid o_i,o_j \in O \wedge o_i \neq o_j\}\rvert}{MT_x} \times 100$ where $M_x(o_i, o_j)$ is the number of messages sent from $o_i$ to $o_j$ and $MT_x$ is the total number of messages exchanged during the execution of the scenario $x$. |
| Import Object Coupling | $IOC_x(o_i, o_j)$, the import coupling for object $o_i$ w.r.t. object $o_j$, is the percentage of the number of messages received by object $o_i$ from object $o_j$ w.r.t. the total number of messages exchanged during the execution of the scenario $x$. | $IOC_x(o_i, o_j) = \frac{\lvert\{M_x(o_j,o_i) \mid o_i,o_j \in O \wedge o_i \neq o_j\}\rvert}{MT_x} \times 100$ where $M_x(o_j, o_i)$ is the number of messages received by object $o_i$ from object $o_j$ and $MT_x$ is the total number of messages exchanged during the execution of the scenario $x$. |

**Table 3.** Dynamic Coupling Metrics by Arisholm

| Metric Name | Description | | | Definition |
|---|---|---|---|---|
| | Direction of Coupling | Mapping Level | Strength of Coupling | |
| IC_OD | Import coupling (IC) | Object (O) | Dynamic messages (D) | This measure counts the total number of messages sent from one object to other objects. |
| IC_OM | | | Distinct methods (M) | This measure counts the number of distinct methods invoked from one object to other objects. |
| IC_OC | | | Distinct classes (C) | This measure counts the number of distinct server classes used by the methods of the given object. |
| IC_CD | | Class (C) | Dynamic messages (D) | This measure counts the total number of messages sent by all methods in all objects of a class. |
| IC_CM | | | Distinct methods (M) | This measure counts the number of distinct methods invoked by all methods in all the objects of a class. |
| IC_CC | | | Distinct classes (C) | This measure counts the number of distinct server classes used by all methods of all objects of a class. |
| EC_OD | Export coupling (EC) | Object (O) | Dynamic messages (D) | This measure counts the total number of messages received by one object from other objects. |
| EC_OM | | | Distinct methods (M) | This measure counts the number of distinct methods received by an object. |
| EC_OC | | | Distinct classes (C) | This measure counts the number of distinct client classes that in a given object are being used. |
| EC_CD | | Class (C) | Dynamic messages (D) | This measure counts the total number of messages received by all methods of all objects of a class. |
| EC_CM | | | Distinct methods (M) | This measure counts the number of distinct methods received by all methods of all objects of a class. |
| EC_CC | | | Distinct classes (C) | This measure counts the number of distinct client classes that in all objects of a given class are being used. |

**Table 4.** Dynamic Coupling Metrics by Mitchell and Power

| Metric | Description | Definition |
|---|---|---|
| Dynamic CBO for a class | This metric is a direct translation of the C&K CBO metric, except it is defined at runtime. | No. couples of a class with other classes at runtime |
| Degree of dynamic coupling between two classes at runtime | No. times a class $A$ accesses methods or instances variables from a class $B$ as a percentage of the total number of methods or instance variables accessed by $A$. | $\frac{\text{No. times a class } A \text{ accesses methods or instance variables from a class } B \text{ at runtime}}{\text{Total No. times a class } A \text{ access any methods or instance variables}} \times \frac{100}{1}$ |
| Degree of dynamic coupling within a given set of classes | This metric is an extension of above metric, to indicate the level of dynamic coupling occurring within a given set of classes. | $\frac{\text{Sum of No. accesses to methods or instance variables outside each class}}{\text{Sum of total No. accesses from these classes}} \times \frac{100}{1}$ |
| $R_{\mathrm{I}}$ | Runtime import coupling between objects | No. classes from which a given class accesses methods or instance variables at runtime |
| $R_{\mathrm{E}}$ | Runtime export coupling between objects | No. classes which access methods or instance variables from a given class at runtime |
| $RD_{\mathrm{I}}$ | Runtime import degree of coupling | $\frac{\text{No. accesses a class makes}}{\text{Total No. accesses}}$ |
| $RD_{\mathrm{E}}$ | Runtime export degree of coupling | $\frac{\text{No. accesses made to a class}}{\text{Total No. accesses}}$ |

Arisholm *et al.*[31] extend the work done by Arisholm[29] by formally defining the dynamic coupling measures in an operational form and validating them theoretically as well as empirically.

### 3.3 Mitchell and Power Metrics Suite

As discussed above, Arisholm *et al.* measure the amount of import and export coupling between objects

at different levels, whereas Mitchell and Power[32-33] propose to measure degree of import and export coupling between objects. Mitchell and Power define a set of dynamic coupling metrics as given in Table 4.

The first three metrics in Table 4 are defined on the basis of static coupling metric, CBO[4]. They are designed to apply to an application at runtime and provide a means to evaluate class level coupling. The next four metrics ($R_{\mathrm{I}}$, $R_{\mathrm{E}}$, $RD_{\mathrm{I}}$, $RD_{\mathrm{E}}$) in Table 4 are defined to evaluate object level dynamic coupling. The metrics $RD_{\mathrm{I}}$ and $RD_{\mathrm{E}}$ are an improvement over metrics $R_{\mathrm{I}}$ and $R_{\mathrm{E}}$ as they are normalized and may be more useful in comparing classes of different sizes.

Mitchell and Power in [33] examine the relationship between static and dynamic coupling metrics in the context of the influence of instruction coverage. The main measures used in this study are the static coupling metric (CBO), six dynamic metrics (IC_CD, IC_CM, IC_CC, EC_CD, EC_CM, EC_CC) proposed by Arisholm *et al.*[29] and instruction coverage measure ($I_c$). The instruction coverage measure $I_c$ corresponds to the Java bytecode instructions. The results of study indicate strong influence of coverage measures on the correlation between static and dynamic metrics.

### 3.4 Dynamic Coupling Metric (DCM)

Another dimension to measure dynamic coupling is initiated by Hassoun *et al.*[35-37], where they target to measure the influence of one object on others over a period of time, instead of import/export coupling concept. Hassoun *et al.* propose a dynamic coupling metric DCM for measuring object level coupling for systems built on meta-level architectures (declarative control languages that allow one to write specifications of program behaviour). This metric is derived from the study that coupling between two objects can be defined in terms of time during which one object influences the other. Two objects are said to be coupled if either one of them could influence the history of the other. The history of an object is defined as the sequence of its states in time. The Dynamic Coupling Measure of an object ($P$) during a time period $\Delta t$, denoted by $DCM(P)|\Delta t$, is defined as sum over all program execution steps and sum over the total number of objects, ($O_i$), coupled to object $P$:

$$DCM(P)|\Delta t = \sum_j \sum_i f_i(t_j) g_p(|O_i|)$$

where $i = 0, 1, 2, \cdots$, the number of coupled objects, $\sum_i$ is the sum over the set of objects coupled to $P$, $\Delta t$ is an ordered sequence of program execution steps $\langle t_0, .., t_j, .., t_n \rangle$, and $\sum_j$ is the sum over the program execution steps.

Here, $f_i(t_j)$ assumes values 1 or 0 depending on whether coupling of the $i$-th object is live at $t_j$ or not and $g_p(|O_i|)$ denotes the complexity measure of the $i$-th object coupled to object $P$.

At the system level, the coupling measure in a time interval $\Delta t$ is the sum of all measures defined in the above formula over all the objects of the system, i.e.,

$$DCM(\text{system})|\Delta t = \sum_{P}^{\text{all-system-objects}} DCM(P).$$

The DCM metric can be used to measure the coupling of a particular object or the entire system at runtime.

Hassoun *et al.*[37] conclude that reflective systems (systems whose behaviour and structure can change during program execution) exhibiting the same behaviour as corresponding non-reflective systems have less coupling, fewer objects and more interactions.

### 3.5 Zaidman *et al.*'s Work

Zaidman *et al.*[38-40] propose a variation of import coupling and use the same for the purpose of program comprehension. Authors consider the following properties of a coupling metric in order to be useful for the purpose of program comprehension:

- Since, software engineers try to comprehend the software at the class-level only. While selecting dynamic coupling measures, only those metrics which are defined at class-level need to be considered.
- All classes external to the actual project (e.g., library classes), have no direct influence on the program comprehension process.
- Only those classes that have a prominent role within the system's architecture need to be considered and such classes are expected to give orders to other classes, i.e., tell them what to do and what to give in return. As such, these classes are expected to request the services of other classes. This suggests that direction of coupling needs to be taken into consideration is the "import coupling" for the purpose of program comprehension.

Out of twelve metrics proposed by Arisholm[29], two metrics IC_CM and IC_CC adhere to the criteria set out as above, namely: working at the class-level and measuring import coupling. Authors also use a variation of IC_CC metric and refer it as IC_CC′. This metric differs from IC_CC metric in the sense that IC_CC metric is targeted more towards finding the number of class-collaborations, while IC_CC′ retrieves the number of method-collaborations. Consider the following example:

A class having one method calls two distinct methods of second class and one method of third class.

IC_CC is calculated as two (number of distinct classes called) and is calculated as three (number of distinct methods called).

These dynamic coupling measures allow us to identify the most need-to-be-understood classes in a system. Detecting these classes very early in the program comprehension process allows the end user to pay attention towards these classes and start exploring the software system directly from there. Authors experiment with various dynamic coupling metrics and also compare direct and indirect coupling solutions. To simulate the indirect coupling, they use the HITS web-mining algorithm[50]. Their experiments show that taking indirect coupling into account delivers better results for program comprehension.

### 3.6 Comparison of Dynamic Coupling Measures and Their Relations with Quality Attributes

In above subsections, a number of different metrics have been presented for measuring dynamic coupling. These metrics are defined at different stages of software development and are useful for different purposes. First type of metrics, EOC and IOC are defined at design-time and are derived from dynamic design models (models depicting execution scenarios). EOC and IOC metrics can affect many of the quality attributes such as maintainability, understandability, reusability and error-propagation. Objects having higher values of EOC or IOC would be more critical to changes due to maintenance and are more likely to export or import these maintenance changes to other objects. Moreover, objects having higher values of EOC or IOC metrics are harder to understand since their dynamic behaviour tightly depends on each other. Objects with higher EOC or IOC are less reusable because they strongly depend on each other and are more likely to be used together. Further, objects with high EOC or IOC are more likely to be a source of error propagation since errors are more likely to propagate from the faulty source to the destination object as a result of the frequent messages exchanged between them.

However, these metrics do not give a precise depiction of the actual runtime situation as they are calculated during the early design stage of a program. Further, EOC and IOC metrics do not comply with the coupling properties for object-oriented software systems described in the axiomatic framework given by Briand *et al.*[10] and these metrics do not account for inheritance and polymorphism. Second type, Arisholm's dynamic coupling metrics are defined at actual runtime and quantify the flow of messages between objects at runtime. These metrics adhere to the theoretical framework for coupling measures proposed in [10] for object-oriented software systems. It has been shown that these metrics are complementary to simple size measures and static coupling measures. Moreover, authors show that dynamic coupling measures capture different properties than static coupling measures, though some degree of correlation exists between them. Authors also demonstrate that dynamic export coupling measures are good indicators of change proneness of software systems. However, Arisholm *et al.* define and study dynamic coupling measures as stand-alone metrics and do not consider the effect of code coverage measures on the proposed measures in detail.

Third type, Mitchell and Power metrics are an extension of static CBO coupling metric as defined in Table 4. The Dynamic CBO and Degree of Dynamic Coupling between classes metrics are proposed to quantify the external complexity of a class at runtime. Degree of Dynamic Coupling within a given set of classes metric is defined to determine external complexity within a group of classes. These metrics are directly related to testability and maintainability. The greater the level of coupling present, the more rigorous testing needs to be done and the greater the Dynamic CBO is for a class, maintenance is more difficult as the class will be more sensitive to changes in other classes with which it is coupled. Authors[28] show that the runtime metrics, $R_{\mathrm{I}}$, $R_{\mathrm{E}}$, $RD_{\mathrm{I}}$ and $RD_{\mathrm{E}}$ capture different properties than the static metrics although some degree of correlation does exist. Authors in [33] successfully demonstrate that dynamic coupling metrics might be better interpreted in the context of coverage measures, rather than as stand-alone software metrics and for this purpose, they use dynamic coupling metrics proposed by Arisholm[29].

Fourth type, DCM is a dynamic coupling measure for systems built on meta-level architectures and is calculated during the actual program execution. The DCM metric can be used to predict the runtime complexity of the system. The value of DCM metric in an object-oriented system has relation with quality of the system in terms of software maintenance. It may help system engineers to decide on the appropriate software components to be used in production and maintenance phase. Classes with high object couplings need more attention and consequently induce higher maintenance cost. Thus, these types of classes should be assigned to more experienced developers. Further, DCM metric can be used to compare systems' coupling at runtime and can also be used as a means of comparing runtime coupling of a system at different stages of its development. Knowledge of amount of coupling at runtime can also be helpful in making decisions on re-engineering and re-factoring. Zaidman *et al.* use two already existing

dynamic coupling metrics; IC_CM and IC_CC defined by Arisholm[29] and propose a metric IC_CC′, which is a variation of IC_CC metric. These dynamic coupling measures are quite useful in identification of key classes for comprehension process in a software system. Authors' work clearly indicates that dynamic coupling metrics and dynamic analysis with its goal-oriented strategy, pay dividends when used for program comprehension purposes.

The different types of dynamic coupling metrics as discussed above have been found to be indicators of external quality attributes as given in Table 5.

**Table 5.** Relations of Dynamic Coupling Metrics with Quality Attributes

| Metric | Quality Attributes |
|---|---|
| EOC & IOC Metrics | Maintainability, understandability, reusability & error-propagation |
| Arisholm Metrics | Change proneness |
| Mitchell & Power Metrics Suite | External complexity, testability & maintainability |
| DCM Metric | Maintainability |
| Zaidman *et al.* Metrics | Program comprehension |

## 4 Dynamic Cohesion Metrics

Despite extensive research work conducted in the measurement of static cohesion[4,17-27], only a few metrics have been proposed for the measurement of cohesion at runtime.

### 4.1 Gupta *et al.* Metrics

Gupta *et al.*[41] re-define module cohesion metrics SFC (Strong Functional Cohesion) and WFC (Weak Functional Cohesion) originally proposed by Bieman and Ott[18,51]. Gupta *et al.*[41] initiate the dynamic cohesion measurement using program execution based approach on the basis of dynamic slicing (dynamic slice is the set of all statements whose execution had some effect on the value of a given variable). They use dynamic slices of outputs to measure module cohesion. They state that module cohesion metrics based on static slicing approach have got some inadequacies in cohesion measurement. The static measures significantly overestimate the levels of cohesion present in the software. Their approach addresses the drawbacks of static cohesion metrics by considering dynamic behaviour of the programs and designing metrics based on dynamic slices obtained through program execution. The dynamic cohesion metrics are defined on the basis of common definitions, common use and common definition-use pairs in dynamic slices, which facilitate more precise cohesion measurement than existing static metrics.

Authors define SFC as module cohesion obtained from common def-use pairs of each type common to the dynamic slices of all the output variables and WFC as module cohesion obtained from def-use pairs of each type found in dynamic slices of two or more output variables.

### 4.2 Mitchell and Power Metrics

The proposal of Gupta *et al.* is an extension of Bieman's static cohesion and CK's LCOM (Lack of COhesion Metric)[4] is the base used for defining dynamic cohesion metrics by Mitchell and Power[42-43]. LCOM is an inverse metric that measures lack of cohesion in a class. Suppose a class contains $n$ methods, $m_1, \ldots, m_n$, and let $\{I_i\}$ be the set of instance variables referenced by method $m_i$. Two disjoint sets can be defined as:

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

Here, $P$ is the number of pairs of methods having no common instance variables and $Q$ is the number of pairs of methods having common instance variables. $LCOM$ is defined as:

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

**Table 6.** Dynamic Cohesion Metrics by Mitchell and Power

| Metric | Description | Definition |
|---|---|---|
| Runtime Simple LCOM ($R_{\mathrm{LCOM}}$) | $R_{\mathrm{LCOM}}$ is an extension of the static LCOM and considers instance variables that are actually accessed at runtime. | $R_{\mathrm{LCOM}} = \begin{cases} \lvert P^R\rvert - \lvert Q^R\rvert, & \text{if } \lvert P^R\rvert > \lvert Q^R\rvert, \\ 0, & \text{otherwise.} \end{cases}$ Here, $P^R = \{(I_i^R, I_j^R) \lvert I_i^R \cap I_j^R = \varnothing\}$ and $Q^R = \{(I_i^R, I_j^R) \lvert I_i^R \cap I_j^R \neq \varnothing\}$ where $\{I_i^R\}$ is the set of instance variables used by method $m_i$ at runtime. |
| Runtime Call-Weighted LCOM ($RW_{\mathrm{LCOM}}$) | Runtime Call-Weighted LCOM ($RW_{\mathrm{LCOM}}$) is defined by weighing each instance variable by the number of times it is accessed at runtime. | $RW_{\mathrm{LCOM}} = \begin{cases} \lvert P^{\mathrm{W}}\rvert - \lvert Q^{\mathrm{W}}\rvert, & \text{if } \lvert P^{\mathrm{W}}\rvert > \lvert Q^{\mathrm{W}}\rvert, \\ 0, & \text{otherwise,} \end{cases}$ where $P^{\mathrm{W}} = \varnothing$, if $\{I_1\}, \ldots, \{I_n\} = \varnothing$. Here, $P^{\mathrm{W}} = \sum_{1 \leqslant i,j \leqslant n} \{(N_i + N_j) \lvert I_i \cap I_j = \varnothing\}$ and $Q^{\mathrm{W}} = \sum_{1 \leqslant i,j \leqslant n} \{(N_i + N_j) \lvert I_i \cap I_j \neq \varnothing\}$, where $N_i$ is the number of times, method $M_i$ dynamically accesses instance variables from the set $\{I_i\}$ and $N_j$ is the number of times, method $M_j$ dynamically accesses instance variables from the set $\{I_j\}$. |

Mitchell and Power propose two metrics Runtime Simple LCOM ($R_{\mathrm{LCOM}}$) and Runtime Call-Weighted LCOM ($RW_{\mathrm{LCOM}}$) described in Table 6 based on above defined LCOM metric. First metric is a direct translation of the LCOM metric to take into account only those instance variables that are actually accessed at runtime. Second metric is an extension of the Runtime Simple LCOM, modified to take into account the total number of accesses made to an instance variable by a method of the class.

### 4.3 Comparison of Dynamic Cohesion Measures and Their Relations with Quality Attributes

Only two kinds of dynamic cohesion metrics exist in the literature till date: one by Gupta *et al.* and the other by Mitchell *et al.* Gupta *et al.* propose dynamic cohesion metrics for procedure-oriented programs whereas Mitchell and Power define dynamic cohesion metrics for object-oriented programs. Dynamic cohesion metrics proposed by Gupta *et al.* are important for software restructuring during software maintenance. Restructuring of existing software is a form of preventive maintenance that is essential when the software undergoes new releases. The results of the study conducted by Mitchell and Power indicate that proposed runtime cohesion metrics can provide informative qualitative analysis of a program and complement existing static cohesion metrics. These dynamic cohesion metrics can be used to indicate external software quality attributes such as reusability, maintainability and can be helpful in redesigning of classes in object-oriented software systems.

## 5 Dynamic Complexity Metrics

Software complexity metrics have been frequently used as indicators for software quality in terms of external factors such as testability and maintainability[2,4]. Most of the complexity metrics proposed in literature deal with the program at rest. However, the complexity of a program also depends on its execution environment and thus should be measured dynamically. Some dynamic complexity metrics have been defined as measures of complexity of the software at execution time, which are discussed in the following subsections.

### 5.1 Munson and Khoshgoftar Metrics Suite

Munson and Khoshgoftar[45-46] estimate the dynamic complexity based on relative complexity of a module defined by them in [44]. The relative complexity of a module is obtained after classifying various complexity metrics in few independent complexity domains and then mapping these domains to a single metric, i.e., relative complexity, which characterizes the complexity of each module as a single value. Authors define dynamic complexity of a component as the product of the static relative complexity of the component and the probability of execution of that component. The probability of execution of components is based on actual traces of execution of the software obtained using profiling tools. Further, authors also present dynamic complexity metrics to quantify the dynamic complexity of sub-systems and systems. A system's dynamic complexity in a given environment is determined by its source code complexity and its operational profile in that environment. An operational profile characterizes a software system's environment in terms of the possible evaluations of input variables along with the probabilities of these inputs in that environment[52]. Let $\rho_i'$ be the relative complexity of module $i$. Then, the average complexity of the system having $n$ modules is given by:

$$\rho' = \frac{1}{n}\sum_{i=1}^{n}\rho_i'.$$

However, in operation some modules would be referenced with greater probability than others. Thus, a system's dynamic complexity may be represented as follows:

$$d_p = \sum_{i=1}^{n} p_i \times \rho_i'$$

where $p_i$ is the probability of reference to module $i$ in the operational environment and $\rho_i'$ is the relative complexity of module $i$. Thus, dynamic complexity of a system may be high in an environment that exercises complex functions having high probability of reference.

### 5.2 Yacoub *et al.* Metrics

Yacoub *et al.*[34] define dynamic complexity metrics using ROOMcharts[48] specifications and use the simulation reports to calculate dynamic measurements. The ROOMcharts are extensions of state-diagrams and are used for behavioural specifications in real-time object modelling and simulation. The proposed dynamic complexity metrics extend authors' previous work[53] for measuring operational complexity[46] of modules based on Petri Net models. The operational complexity of objects is based on the static McCabe's cyclomatic complexity[2] obtained from a control flow graph. For each scenario $x$, a subset of the ROOMchart model of an object $o_i$ is executed in terms of state entries, state exits, and fired transitions. This subset of the ROOMchart model is translated into control flow graph. Then, it is possible to calculate the cyclomatic complexity of the executed path for each object $o_i$ for a scenario $x$, i.e.,

$ocpx_x(o_i)$. Using the probabilities of execution scenarios (or operation profiles), a measure of the operational complexity of the object from the simulation report is obtained. Using these reports, the complexity of each object $o_i$ for each scenario $x$ can be obtained and the probabilities of scenarios can be used to calculate the operational complexity of the object as given below:

$$OCPX(o_i) = \sum_{x=1}^{|X|} PS_x \times ocpx_x(o_i).$$

Thus, dynamic complexity metrics guide the process of identifying complex objects and trace these objects to classes from which they are instantiated. As a result, classes can be graded based on the complexity of their instantiated objects and more efforts for implementation and testing will be higher devoted to classes having higher dynamic complexity.

### 5.3 Comparison of Dynamic Complexity Metrics and Their Relations with Quality Attributes

Munson and Khoshgoftar propose dynamic complexity metrics primarily for procedure-oriented programs whereas Yacoub *et al.* define dynamic complexity metrics for object-oriented designs. Another difference in their approach is that metrics proposed by Munson *et al.* are obtained from the code as well as executable design models whereas metrics given by Yacoub *et al.* can be collected only from design models. Dynamic complexity metrics proposed by Munson *et al.* are found to influence the reliability of software systems. A component that is fault-prone according to static complexity metrics may not be failure-prone in a given environment, because the most complex component may never be executed in a given environment. Since failures, not faults determine quality of the software from user's perspective, the detection of failure-prone components in the system becomes more important. Therefore, dynamic complexity measures can be more helpful in the detection of failure prone components in software and thus, dynamic complexity metrics more accurately predict reliability of the software system and testing effort required for the system. Moreover, dynamic complexity metrics can be used to rank various environments or executions. The environments that produce higher dynamic complexity place a greater stress on the system or subsystem under consideration. Thus, the dynamic complexity metrics assist in rating the effectiveness of various test cases and selection of test cases for components under consideration. Dynamic complexity metrics proposed by Yacoub *et al.* are useful to measure the quality of object-oriented designs. These metrics can be used to rank classes at design-time based on the complexity of their instantiated objects. The identification of highly complex classes at early phase of software development lifecycle can be helpful in estimation of testing, verification and validation efforts required for the system. However, to use the proposed dynamic complexity metrics early in the development phase, the application has to be modelled as executable design models. The extra effort required in developing executable design models is justifiable for many complex real-time systems where deadlines are required to be examined prior to developing the system.

## 6 Miscellaneous Dynamic Metrics

In this section, we will discuss various types of dynamic metrics, which do not belong to a particular class and have not been explored in much detail in literature. A brief overview about each of them is given in the following subsections.

### 6.1 Dynamic Metrics for GUI Programs

Graphical User Interfaces (GUIs) make the software easier to use from user's perspective. However, they increase the overall complexity of the software since GUI programs unlike conventional software are event-based systems. The special characteristics of a GUI program suggest that traditional methods of statically evaluating its complexity may not be suitable as a static analysis of the source code only measures what may happen when the program is executed whereas a dynamic analysis attempts to quantify what actually happens. A dynamic analysis of a GUI application may be problematic, as GUIs cannot be executed in the batch and thus, it becomes necessary to define a new methodology for collecting dynamic trace data of a GUI program.

Mitchell and Power[54] outline a new technique for collecting dynamic trace information from Java GUI programs and a number of simple runtime metrics are proposed. A number of simple metrics defined by authors give an approximation of attributes of a program such as its size, coupling, cohesion and memory use. These metrics are described in Table 7.

The meth.ob metric is a measure of size of a GUI program. The exPubMet.Ob metric gives an estimation of level of coupling present in a GUI program. It can be easily perceived that the GUI program has a greater external public methods called per object ratio than the non-GUI programs. The priMet.ob metric shows that simple programs devote a greater proportion of their method access to the internal working of their classes than the GUI program. This is illustrated by the fact that they have higher private methods per object ratio than the GUI programs. The GUI programs reveal

**Table 7.** Dynamic Metrics for GUI Programs

| Metric | Description | Definition |
|---|---|---|
| meth.ob | Measure of the program size. | $\frac{\text{Total No. methods called}}{\text{Total No. objects created}}$ |
| exPubMet.Ob | Measure of the level of coupling within a program at runtime. | $\frac{\text{No. external public methods called}}{\text{Total No. objects created}}$ |
| priMet.ob | Measure of the level of cohesiveness within a program. | $\frac{\text{No. private methods called}}{\text{Total No. objects created}}$ |
| meth.inst | This gives an estimation of the memory use of the methods. | $\frac{\text{No. methods called}}{\text{Kilo byte code instructions executed}}$ |
| ob.inst | This gives an indication of how memory hungry the program is overall. | $\frac{\text{No. objects called}}{\text{Kilo byte code instructions executed}}$ |

higher values of meth.inst metric, i.e., higher method to kilo byte code execution ratio in comparison to simple programs. The ob.inst metric exhibits a similar value, as expected for small and simple programs. The above analysis may have a number of useful applications:

• this work may be useful in the quantification of different software testing strategies for GUI applications;

• this approach will promote to investigate how inheritance affects the runtime performance of a program;

• it will help in defining metrics to quantify various aspects of an object-oriented application such as polymorphism, concurrency and dynamic binding.

### 6.2 Dynamic Metrics for Risk Assessment

Yacoub *et al.*[55] present a risk assessment methodology that is based on dynamic metrics obtained from UML specifications during the early stages of the software development lifecycle, specifically at the architecture level. This work is based on authors'[34] earlier work done on dynamic complexity and dynamic coupling metrics, which can be obtained from simulating design specifications. The complexity of components of real-time applications can be assessed at runtime using dynamic metrics. The dynamic metrics are used to account for the fact that a fault in a frequently executed component will frequently manifest itself into a failure. Severity analysis is performed using Failure Mode and Effect Analysis (FMEA)[56] with the aid of simulation runs to study the effect of a failure. The FMEA technique is an efficient way to identify all possible failure modes and their consequential effects on the system. In this method, failure modes of architecture elements are identified and their effects are recorded. Then, these failure modes are ranked according to the degree of severity of their effects and the worst-case effect on the system is recognized.

In this particular work, authors combine severity and complexity factors to develop heuristic risk factors for the architecture components and connectors. Based on component dependency graphs and using analysis scenarios, they develop a risk assessment model and a risk analysis algorithm that aggregates risk factors of components and connectors at the architectural level. This work shows a method to analyse the overall risk factor of the architecture as the function of the risk factors of its constituting components and connectors.

### 6.3 Requirements-Based Dynamic Metric (RBDM)

Cleland-Hunang *et al.* propose dynamic metrics for predicting the volume of data that would flow across a network in a distributed system[57]. This prediction is driven by requirement specifications and captures dynamic metrics by defining typical usage patterns in terms of scenarios. Scenarios are then mapped to architectural components, and dataflow across inter-partition links is estimated. These dynamic metrics are available early in the system lifecycle. The method provides the means of validating architectural designs early during development, which directly addresses well-recognized problems such as correctly partitioning a system for distribution. Despite the fact that the metric results match actual measurements fairly closely, it is believed that it is still extremely difficult to consistently predict typical runtime dataflow. The difficulty lies in the fact that users' interactions with the system may vary dramatically making it hard to predict typical user behaviour.

The dynamic metrics proposed by Yacoub *et al.*[34] are also useful, but these metrics report levels of interactions only relative to the overall interaction of the scenario or the system. In contrast, RBDM predict actual runtime measurements within a realistic context, and can be applied at a much earlier stage than the other currently used methods.

### 6.4 Dynamic Metrics for Object Clustering

Cho *et al.*[58] propose dynamic coupling and cohesion

metrics, which can be used to cluster objects logically. The proposed dynamic metrics are measured by the number of messages passed between functions contained in objects at runtime. The clustering of objects reduces network traffic and the load of client/server so that it can improve system performance. In client/server applications, the quality of object clustering plays a key role in determining the overall performance of the system. Thus, a set of objects with higher exchange of number of messages between objects should be grouped into a single cluster so that each cluster can have a higher cohesion. As a result, the overall message traffic among objects can be minimized. Various dynamic as well as static metrics are used in order to measure the dynamic message traffic and to tune up the system performance. The proposed object-oriented design metrics mainly deal with static coupling and cohesion, and they only consider the basic class relationships such as association, inheritance, and composition. Thus, these metrics are not appropriate for measuring the traffic load of object messages, which is closely related to the system performance. Authors propose a set of metrics that consider the relevant weights on various class relationships and estimate the static and dynamic message flow among the objects at the detailed level of member functions. By applying these metrics along with UML[49], it is believed that clusters can be defined more efficiently and systematically, yielding high performance distributed applications.

## 7 Pseudo Dynamic Metrics

As discussed above, despite being dynamic metrics more accurate than static metrics, sill they are more difficult and costlier to obtain than static measures. This gap between static and dynamic metrics can be narrowed down by using the concept of pseudo dynamic metrics. Pseudo dynamic metrics are the static metrics adjusted to reflect the operational profile of the expected usage[59]. Operational profiles can be estimated from the analysis of source code or UML design specifications[49] and do not require execution traces. Once the static metrics and the operational profiles are available, the pseudo dynamic metrics can be estimated. Pseudo metrics can be collected earlier than dynamic metrics as they depend on operation profile, which can be obtained from UML specifications in the design phase. It is well known that the impact of any software metric is extremely high when used in early stages of the software development lifecycle, when it is more viable for changing and modifying the product under development. The methodology for estimating pseudo dynamic metrics involves the following steps:

1) obtain the static metrics for the components;

2) determine the operation profile for all the concerned components;

3) adjust the static metrics based on operation profile to get pseudo dynamic metrics.

Pseudo dynamic metrics correlate with their dynamic counterparts and hence can be used as early indicators of software quality attributes such as reusability, reliability, testability, maintainability etc.

## 8 Potential Research Directions

From the above detailed discussions of different types of dynamic metrics, it can easily be said that the field of dynamic metrics is wide open for researchers. Some of the possible research directions in the area of dynamic metrics are listed below:

• From above discussions on dynamic metrics, it can be easily observed that there are no major metrics available at runtime for the measurement of testability of the software systems. Thus, future research may involve investigating the role of dynamic metrics in the area of software testing. These measures can guide about the effectiveness of software testing strategies and can also contribute to better judgement of the testability of the code in its operational environment.

• As mentioned in Section 2, dynamic metrics have the advantage of being more precise, but they are difficult to measure in comparison to static ones. Thus, there is clear opportunity for researchers in hybrid approach where the dynamic analysis results are augmented by static information for collection of metrics data. This hybrid approach can combine static as well as dynamic approaches to make the measurement process more cost-effective.

• Since the main difficulty arises in collection process of dynamic metrics, it becomes important to devise some techniques so as to make the process of collection of the dynamic metrics easy. We have found the aspect-oriented approach as one such technique, which has the potential of computing various metrics at runtime such as dynamic coupling, dynamic cohesion. Going in further details of aspect-oriented approach is beyond the scope of this paper.

• As some dynamic metrics have been successfully used for the purpose of program comprehension[38-40], another possible direction of research can be to use dynamic metrics for identifying the key classes as well as the key collaborations among these classes in a system. These key classes and key collaborations can be useful in better understanding the system for the purpose of clustering and the measurement of amount of actual coupling among them.

• From the above study, it can easily be observed that very few metrics have been proposed for measuring

cohesion at runtime. Hence, measurement of dynamic cohesion of object-oriented programs based on associations taking place at runtime or dynamic/hybrid slicing can be a promising area for future research.

• Since pseudo dynamic metrics can easily be measured from operation profiles, which is obtained from UML diagrams, the UML based representation helps in calculating various metrics for different purposes with a low cost. Such metrics will have benefits of both dynamic and static measures. Defining such pseudo dynamic metrics for the measurement of different software attributes is another promising research opportunity readily available to researchers.

• As described above, there are a number of dynamic metrics proposed for the measurement of internal attributes of the software. But, impact of these metrics is not well studied on external attributes of software applications. Goseva-Popstojanova[60] has studied the impact of dynamic metrics on the identification of failure prone components of the software applications being used by NASA. Identification of the most problematic parts of software holds enormous potential for reducing the cost and improving the quality of software. Thus, study of the impact of several dynamic metrics like dynamic coupling metrics on the external attributes of the software applications is another good alternate for future research.

• It can be easily observed from the survey of different works conducted by different authors that, till date, dynamic metrics have been measured and validated using only small software systems. In future research in this direction, large-scale industrial systems need to be undertaken to evaluate the usefulness of the proposed dynamic metrics in real-life scenarios.

## 9 Conclusions

An extensive study of the dynamic metrics, proposed till date, has been reported in this paper. A comparison of dynamic metrics with static metrics is carried out by the authors and observations clearly indicate that dynamic metrics are more precise than their static counterpart, but at the same time, are more difficult to collect. The need and usefulness of dynamic metrics for object-oriented systems has also been identified and pointed out. Although lots of dynamic metrics have been proposed in the literature to measure coupling, cohesion, complexity, but very few metrics exist for external attributes. Our survey has further pointed out that little work has been done till now to use the pseudo dynamic metrics and hybrid approach of static as well as dynamic metrics, although they have tremendous scope. Based on the survey of existing dynamic metrics, we have tried to reveal potential research challenges and opportunities existing in the field of dynamic metrics. Moreover, by realizing the importance of dynamic metrics and their clear edge over their corresponding static metrics, we can conclude that focus of future research in the field of software metrics is going to be on the dynamic metrics and on the study of impact of dynamic metrics on different software quality attributes.

## References


[1] Henderson-Sellers B. Software Metrics. Prentice Hall, Hemel Hempstaed, UK, 1996.

[2] McCabe T. A complexity metric. *IEEE Transactions on Software Engineering*, 1976, 2(4): 308-320.

[3] Fenton N, Neil M. Software metrics: Successes failures and new directions. *Journal of Systems and Software*, 1999, 47(2/3): 149-157.

[4] Chidamber S R, Kemerer C F. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 1994, 20(6): 467-493.

[5] Abreu F B. The MOOD metrics set. In *ECOOP'95 Workshop on Metrics*, Åarhus, Dénmark, Aug. 7-11, 1995.

[6] Lorenz M, Kidd J. Object-Oriented Software Metrics: A Practical Guide. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[7] Briand L C, Devanbu W, Melo W. An investigation into coupling measures for C++. In *Proc. the 19th International Conference on Software Engineering* (*ICSE 1997*), Boston, USA, May 17-21, 1997, pp.412-421.

[8] Harrison R, Counsell S, Nithi R. Coupling metrics for object-oriented design. In *Proc. the 5th International Software Metrics Symposium Metrics*, Bethesda, USA, Mar. 20-21, 1998, pp.150-156.

[9] Bansiya J, Etzkorn L, Davis C, Li W. A class cohesion metric for object-oriented designs. *Journal of Object-Oriented Programming*, 1999, 11(8): 47-52.

[10] Briand L C, Daly J W, Wust J K. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 1999, 25(1): 91-121.

[11] Lee Y S, Liang B S. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proc. International Conference on Software Quality*, Maribor, Slovenia, Oct. 17-20, 1995, pp.81-90.

[12] Allen E, Khoshgoftaar T. Measuring coupling and cohesion, an information-theory approach. In *Proc. the IEEE International Symposium on Software Metrics*, Boca Raton, USA, Nov. 4-6, 1999, pp.119-127.

[13] Eder J, Kappel G, Schrefl M. Coupling and cohesion in object-oriented systems. Tech. Rep. 2/93, Department of Information Systems, University of Linz, Austria, 1993.

[14] Li W, Henry S. Maintenance metrics for the object-oriented paradigm. In *Proc. the First International Software Metrics Symposium*, May 21-22, Baltimore, USA, 1993, pp.52-60.

[15] Hitz M, Montazeri B. Measuring coupling and cohesion in object-oriented systems. In *Proc. International Symposium on Applied Corporate Computing*, Monterrey, Mexico, Oct. 1995, pp.25-27.

[16] Alexander R T, Offutt J. Coupling-based testing of O-O programs. *Journal of Universal Computer Science*, 2004, 10(4): 391-427.

[17] Briand L C, Daly J W, Wüst J. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 1998, 3(1): 65-117.

[18] Ott L M, Bieman J M, Kang B K. Developing measures of class cohesion for object-oriented software. In *Proc. the*

*7th Annual Oregon Workshop on Software Metrics*, Oregon, USA, 1995.

[19] Ott L M, Bieman J M. Program slices as an abstraction for cohesion measurement. *Journal of Information and Software Technology*, 1998, 40(11/12): 691-699.

[20] Kang B K, Bieman J M. Design-level cohesion measures, derivation, comparison, and applications. In *Proc. the 20th Computer Software and Applications Conference*, Seoul, Korea, Aug. 21-23, 1996, pp.92-97.

[21] Chae H S, Kwon Y R. A cohesion measure for classes in object-oriented systems. In *Proc. the 5th International Software Metrics Symposium*, Bethesda, USA, 1998, pp.158-166.

[22] Chae H S, Kwon Y R, Bae D H. A cohesion measure for object-oriented classes. *Software Practice and Experience*, 2000, 30(12): 1405-1431.

[23] Chae H S, Kwon Y R. Improving cohesion metrics for classes by considering dependent instance variables. *IEEE Transactions on Software Engineering*, 2004, 30(11): 826-832.

[24] Chen Z, Zhou Y, Xu B, Zhao J, Yang H. A novel approach to measuring class cohesion based on dependence analysis. In *Proc. International Conference on Software Maintenance*, Montreal, Canada, Oct. 3-6, 2002, pp.377-384.

[25] Zhou Y L, Wen L, Wang J, Chen Y, Lu H, Xu B. DRC: A dependence relationships based cohesion measure for classes. In *Proc. the Tenth Asia-Pacific Software Engineering Conference* (*APSEC 2003*), Chiang Mai, Thailand, Dec. 10-12, 2003, pp.215-233.

[26] Zhou Y, Lu J, Lu H, Xu B. A comparative study of graph theory-based class cohesion measures. *ACM SIGSOFT Software Engineering Notes*, 2004, 29(2): 1-6.

[27] Wang J, Zhou Y, Wen L, Chen Y, Lu H, Xu B. DMC: A more precise cohesion measure for classes. *Information and Software Technology*, 2005, 47(3): 167-180.

[28] Mitchell A, Power J F. An empirical investigation into the dimensions of run-time coupling in Java programs. In *Proc. the 3rd Conference on the Principles and Practice of Programming in Java*, Las Vegas, USA, Jun. 16-18, 2004, pp.9-14.

[29] Arisholm E. Dynamic coupling measures for object-oriented software. In *Proc. the Eighth IEEE Symposium Software Metrics* (*METRICS 2002*), Ottawa, Canada, Jun. 4-7, 2002, pp.33-42.

[30] Arisholm E. Empirical assessment of changeability in object-oriented software [Ph.D. Dissertation]. University of Oslo, 2001.

[31] Arisholm E, Briand L C, Foyen A. Dynamic coupling measures for object-oriented software. *IEEE Transactions on Software Engineering*, 2004, 30(8): 491-506.

[32] Mitchell A, Power J F. Using object-level run-time metrics to study coupling between objects. In *Proc. ACM Symposium on Applied Computing*, Santa Fe, USA, Mar. 13-17, 2005, pp.1456-1462.

[33] Mitchell A, Power J F. A study of the influence of coverage on the relationship between static and dynamic coupling metrics. *Science of Computer Programming*, 2006, 59(1/2): 4-25.

[34] Yacoub S, Ammar H, Robinson T. Dynamic metrics for object-oriented designs. In *Proc. the 5th International Software Metrics Symposium*, Boca Raton, USA, Nov. 4-6, 1999, pp.50-61.

[35] Hassoun Y, Johnson R, Counsell S. A dynamic runtime coupling metric for meta-level architectures. In *Proc. European Conference on Software Maintenance and Reengineering* (*CSMR 2004*), Tempere, Finland, Mar. 24-26, 2004, pp.339-346.

[36] Hassoun Y, Johnson R, Counsell S. Empirical validation of a dynamic coupling metric. Technical Report, BBKCS-04-03, School of Computer Science and Information Systems, Birkbeck College, University of London, UK, March, 2004.

[37] Hassoun Y, Counsell S, Johnson R. Dynamic coupling metric-proof of concept. *IEE Proc. Software*, 2005, 152(6): 273-279.

[38] Zaidman A, Demeyer S. Analyzing large event traces with the help of coupling metrics. In *Proc. the 5th International Workshop on OO Reengineering*, Oslo, Norway, Jun. 14-18, 2004.

[39] Zaidman A, Bois B D, Demeyer S. How webmining and coupling metrics can improve early program comprehension. In *Proc. the 14th International Conference on Program Comprehension* (*ICPC 2006*), Athens, Greece, Jun. 14-16, 2006, pp.74-78.

[40] Zaidman A. Scalability Solutions for Program Comprehension through Dynamic Analysis [PhD. Dissertation]. University of Antwerp, 2006.

[41] Gupta N, Rao P. Program execution based module cohesion measurement. In *Proc. the 16th International Conference on Automated on Software Engineering* (*ASE 2001*), San Diego, USA, Nov. 26-29, 2001, pp.144-153.

[42] Mitchell A, Power J F. Run-time cohesion metrics for the analysis of Java programs. Technical Report, Series No. NUIM-CS-TR-2003-08, National University of Ireland, Maynooth, Co. Kildare, Ireland, 2003.

[43] Mitchell A, Power J F, Run-time cohesion metrics: An empirical investigation. In *Proc. the International Conference on Software Engineering Research and Practice*, Las Vegas, USA, Jun. 21-24, 2004, pp.532-537.

[44] Munson J C, Khoshgoftaar T M. Measuring dynamic program complexity. *IEEE Software*, 1992, 9(6): 48-55.

[45] Khoshgoftaar T M, Munson J C, Lanning D L. Dynamic system complexity. In *Proc. Software Metrics Symposium*, Baltimore, USA, May 21-22, 1993, pp.129-140.

[46] Munson J C, Khoshgoftaar T M. Software Metrics for Reliability Assessment. Handbook of Software Reliability Engineering, Michael Lyu (ed.), McGraw-Hill, 1996, pp.493-529.

[47] Arisholm E, Sjøberg D I K, Jørgensen M. Assessing the changeability of two object-oriented design alternatives — A controlled experiment. *Empirical Software Engineering*, 2001, 6(3): 231-277.

[48] Selic B, Gullekson G, Ward P. Real-Time Object Oriented Modeling. John Wiley & Sons, Inc., 1994.

[49] Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language Users Guide. Addison-Wesley, 1998.

[50] Kleinberg J M. Authoritative sources in a Hyperlinked environment. *Journal of the ACM*, 1999, 46(5): 604-632.

[51] Bieman J M, Ott L M. Measuring functional cohesion. *IEEE Transactions on Software Engineering*, 1994, 20(8): 644-657.

[52] Musa J D, Iannino A, Okumoto K. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, Inc., 1987.

[53] Ammar H H, Nikzadeh T, Dugan J. A methodology for risk assessment of functional specification of software systems using coherent petri nets. In *Proc. the Fourth International Software Metrics Symposium* (*Metrics 1997*), Albuquerque, USA, Nov. 5-7, 1997, pp.108-117.

[54] Mitchell A, Power J F. An approach to quantifying the run-time behaviour of Java GUI applications. In *Proc. Winter International Symposium on Information and Communication Technologies*, Cancun, Mexico, Jan. 5-8, 2004, pp.1-6.

[55] Yacoub S, Ammar H, Robinson T. A methodology for architectural-level risk assessment using dynamic metrics. In *Proc. 11th Int. Symp. Software Reliability Eng*, San Jose, Oct. 8-10, 2000, pp.210-221.

[56] RIAC document: Procedures for performing failure mode effects and criticality analysis. US MIL-STD-1629, Nov. 1974, US MIL-STDJ629A, Nov. 1980, US MIL_STD_1629A/Notice 2, Nov. 1984.

[57] Cleland-Hunang J, Chang C K, Kim H, Balakrishnan A. A requirements-based dynamic metric in object-oriented systems. *Proc. the 5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, Aug. 27-31, 2001, pp.212-219.

[58] Cho E S, Kim C J, Kim S D, Rhew S Y. Static and dynamic metrics for effective object clustering. In *Proc. Asia-Pacific Software Engineering Conference*, Taipei, China, Dec. 2-4, 1998, pp.78-85.

[59] Gunnalan R, Shereshevsky M, Ammar H H. Pseudo dynamic metrics. In *Proc. the Third ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt, Jan. 3-6, 2005, pp.117-iiv.

[60] Goseva-Popstojanova K. The impact of dynamic metrics on identification of the failure prone parts of the software. Lane Dept. of Computer Science and Electrical Engineering, West Virginia University, Morgantown, USA, June 2006.

**Jitender Kumar Chhabra** Ph.D., is working as an assistant professor in Department of Computer Engineering, National Institute of Technology, Deemed University, Kurukshetra-136119, INDIA. He received his B.Tech. degree in computer engineering as 2nd rank holder and M.Tech. degree in computer engineering as Gold Medalist, both from Regional Engineering College (now N. I. T.), Kurukshetra. He completed his Ph.D. degree in software metrics in GGS Indraprastha University, Delhi, INDIA. He has been teaching in N.I.T. Kurukshetra since last 15 years. He has also worked in collaboration with international software companies like Hewellet-Packard & Tata Consultacy Services. He has published more than 50 research papers in various international and national journals and conferences. He is reviewer of many reputed international journals. He has authored a widely circulated Schaum-Series book on Programming with C along with Byron S Gottfried from McGraw Hill Publications. He has been awarded with many prizes and awards including International Educator of the years 2005, 2008, All India Open Debate Winner, Best Teacher Award, etc. His areas of interest include software engineering, database system, data structure, procedural and object oriented programming.

**Varun Gupta** is pursuing his Ph.D. degree in software engineering in Department of Computer Engineering, National Institute of Technology, Deemed University, Kurukshetra-136119, INDIA. He obtained his B.Tech. degree in computer science & engineering from Guru Nanak Dev University, Amritsar in 1999 and M.E. degree in software engineering from Thapar Institute of Engineering and Technology, Patiala, Deemed University, in 2003. He worked as a lecturer in Department of Computer Science & Engineering, RIMT Institute of Engineering and Technology for 4 years. Presently, he is working as assistant director in Directorate of Information Technology, PSEB, Patiala. His areas of interest include software engineering, object-oriented design & development, and aspect-oriented software development.