Journal of Computer Science 4 (11): 903-909, 2008
ISSN 1549-3636

Towards A Dynamic Object-Oriented Design Metric Plug-in Framework

Chuan Ho Loh and Sai Peck Lee
Department of Software Engineering, Faculty of Computer Science and Information Technology,
University of Malaya, 50603 Kuala Lumpur, Malaysia

Abstract: Problem Statement: The evolution of software is made difficult by the need to integrate new features with all previously implemented features in software applications. **Approach:** present study introduced a general-purpose, platform-independent object-oriented design metric plug-in framework called jmetric intended to help building scalable, extendable object-oriented design metric plug-ins. jmetric seeks to address problem by providing the plug-in developer a structured way to separately develop and incrementally integrate independent object-oriented design metrics as plug-ins to a domain specific object-oriented design metrics framework. jmetric was engineered to provide functional building blocks to accelerate the adding, removing and updating of object-oriented design metric plug-ins in tools such as Eclipse, JDeveloper, NetBeans, JBuilder and other Java-based tools. Dependency injection is heavily used in jmetric to accelerate the adding, removing and updating of object-oriented metrics plug-ins. We studied several commonly used integrated development environments and software metrics tools to identify the extendibility of the tools to provide additional object-oriented design metric functionalities as plug-ins. **Results:** We demonstrate a tool called jmetric tool that had developed as a reference implementation to validate the plug-in capabilities of jmetric. **Conclusion:** Extending other tools such as Eclipse, JDeveloper and NetBeans to include metric functionalities is possible by wiring plug-ins through dependency injection mechanism in jmetric.

Key words: Design metric, object-orientation, plug-in framework

INTRODUCTION

Measurement is recognized as a key element of any engineering process. We use measures to assess the quality of an engineered product. Examples are analysis models, design models and other software artifacts. The IEEE-computer society, with the support of a consortium of industrial sponsors, has published a guide to the Software Engineering Body of Knowledge and, throughout this guide, measurement is pervasive as a fundamental engineering tool[1]. A few researchers such as Harrison, Counsell and Nithi, Chidamber and Kemerer and Lorenz and Kidd have proposed a number of design metrics for object-oriented systems. Examples are weighted methods per class, depth of inheritance tree, number of children, coupling between object classes, response for a class, lack of cohesion in methods, class size, number of operations overridden by a subclass, number of operations added by a subclass and method inheritance factor[3,4,7,11,14,16]. Most of the software metric tools such as Resource Standard Metrics, Essential Metrics, Krakatau Metrics, CodeReports, DeepCover and Together ControlCenter are based on a closed architecture relying heavily on proprietary constraints such as vendor specific APIs. As such, we introduce jmetric, an open architecture framework intended to help building scalable, extendable object-oriented design metric plug-ins. jmetric allows the adding, removing and updating (i.e., upgrading and swapping) of plug-ins easily through subclassing the required abstract classes. Hence, object-oriented metrics such as depth of inheritance tree, number of public attributes and public method density are implemented as add-ons to tools such as jmetric tool and Eclipse.

MATERIALS AND METHODS

We have studied several commonly used software metrics tools and categorized the tools into standalone tools and IDEs. Table 1 and 2 show the tools that we have studied.

Corresponding Author: Chuan Ho Loh, Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia

Table 1: Standalone tools

Tool name	Tool vendor
Krakatau professional	Power software
Essential metrics	Power software
J style	Man machine systems
Resource standard metrics	M squared technologies
Java metrics	Semantic designs
C# Metrics	Semantic designs
McCabe IQ developers edition	McCabe software
Visual studio team system (Development edition and team suite)	Microsoft
METRIC	Software research
CMT Java	Test well
CMT++	Test well

Table 2: IDEs

IDE name	IDE vendor
Eclipse IDE for Java EE developers	Eclipse
Eclipse classic	Eclipse
Eclipse IDE for Java developers	Eclipse
Eclipse IDE for C/C++ developers	Eclipse
Visual studio standard edition	Microsoft
Visual studio professional edition	Microsoft
Visual studio team system (team suite)	Microsoft
J Developer studio edition	Oracle
J Developer J2EE edition	Oracle
J Developer Java edition	Oracle
Net Beans IDE	Net beans

Standalone tools: We analyzed the standalone tools based on five distinctive attributes as shown in Table 3: Supported platforms, supported languages, supported configuration management (CM) integration, supported IDEs integration and architecture type. Most of the tools such as Krakatau Professional, Essential Metrics, JStyle and McCabe IQ Developers Edition are based on a closed architecture and thus rely heavily on proprietary constraints. We have discovered only a few software metrics tools such as Resource Standard Metrics and CMT++ provide integration on IDEs such as Visual Studio. Tools such as Resource Standard Metrics integrates with many version control systems such as Subversion, CVS, Vault, Perforce, ClearCase, Team Foundation Server, AccuRev, StarTeam, Git and CMVC.

IDEs: We analyzed the IDEs based on two distinctive attributes as shown in Table 4: Supported platforms and supported languages. IDEs such as Visual Studio Team System (Development Edition and Team Suite) commercially off-the-shelf supports software metrics (referred to as code metrics): Lines of Code, Depth of Inheritance, Cyclomatic Complexity, Class Coupling and Maintainability Index. Most of the IDEs such as Eclipse IDE for Java EE Developers, Eclipse Classic, Eclipse IDE for Java Developers, Eclipse IDE for C/C++ Developers and NetBeans IDE lack of software metrics functionalities. All the IDEs are based on open architecture and thus allow the adding, removing and updating of object-oriented design metrics as plug-ins directly into the IDEs. As such, the IDEs allow object-oriented design metrics to be implemented as plug-ins into the IDEs.

Motivation: Based on our classification schemes shown in Table 3 and 4, we believed having a good object-oriented design metric plug-in framework in place allows metric plug-in developers to spend more time concentrating in implementing specific object-oriented design metrics rather than dealing with low-level details. This leads to a distinctive breakdown of jmetric at the design stage into frozen spots and hot spots.

Frozen spots: Frozen spots represent the overall architecture of jmetric (i.e., its basic components and the relationships between them). These remain unchanged (i.e., frozen) in any instantiation of jmetric.

Hot spots: Hot spots represent those parts where specific object-oriented design metrics are implemented independently as plug-ins.

In our study, we intend to design a lowly coupled object-oriented design metric plug-in framework. As such, the design of jmetric is based on the Hollywood Principle: "Don't call us, we'll call you."[13]. A good object-oriented design metric plug-in framework also needs to include techniques to decouple high-level modules from low-level modules. The dependency injection principle is employed to fulfill such objective. jmetric used the interface injection heavily in which alternative implementations of a specified plug-in can be easily integrated at runtime into jmetric via a configuration file. There are three ways in which an object can reference an external module, according to the pattern used to provide the dependency:

- Type 1 or interface injection, in which the exported module provides an interface that its users must implement in order to get the dependencies at runtime
- Type 2 or setter injection, in which the dependent module exports a setter method that the framework uses to inject the dependency
- Type 3 or constructor injection, in which the dependencies are provided through the class constructor[10,15]

Table 3: Standalone Tool Classification Scheme

Tool name	Supported platforms	Supported platforms	Supported CMs integration	Supported IDEs integration	Architecture type	
Krakatau professional	Windows, solaris	C, C++, Java	-	-	Closed	
Essential metrics	Windows, solaris	C, C++, Java	-	-	Closed	
J style	Windows	Java	-	-	Closed	
Resource standard metrics	Windows, mac OS X, linux, unix Closed	C, C++, Java, visual C # .net	All CMs	Visual Studio, Eclipse, J Builder		
Java Metrics	Windows	Java	-	-	Closed	
C # Metrics	Windows	Visual C #.net	-	-	Closed	
McCabe IQ developers edition	All platforms	Ada, ASM86/95, C, visual C# .net visual C++ net, C++, cobol, Fortran, Java, JSP, perl, PL1, visual basic, visual basic net	-	-	Closed	
Visual studio team system (development suite)	Windows C++ .net, visual edition and team	J script, visual. basic .net	- C# net , visual	-	Open	
METRIC	Unix	C, C++, Ada, Fortran	-	-	Closed	
CMT Java solaris	Windows, linux,	java	-	-	Closed	HP-UX,
CMT++ solaris	Windows, linux,	C, C++	-	Visual Studio	Closed	HP-UX,

Table 4: IDE classification scheme

IDE name	Supported platforms	Supported languages
Eclipse IDE for java EE developers	Windows, Mac OS X, linux	Java
Eclipse classic	Windows, Mac OS X, linux	Java
Eclipse IDE for java developers	Windows, Mac OS X, linux	Java
Eclipse IDE for C/C++ developers	Windows, Mac OS X, linux	C, C++
Visual studio standard edition	Windows	Jscript, visual C++ .net, visual C# .net , Visual basic .neT
Visual studio professional edition	Windows	Jscript, visual C++ .net, visual C# .net , visual basic .nET
Visual studio team system (team suite)	Windows	Jscript, visual C++ .NET, visual C# .net , visual basic .net
Jdeveloper studio edition	Windows, Mac OS X, linux	Java
Jdeveloper J2EE edition	Windows, Mac OS X, linux	Java
Jdeveloper java edition	Windows, Mac OS X, linux	Java
Netbeans IDE	Windows, Mac OS X, linux, solaris	Java, C, C++, ruby

Dependency injection is very effective at assembling loosely coupled plug-ins and at configuring these plug-ins. Specifically, jmetric uses dependency injection to:

- Inject the same dependency into multiple plug-ins
- Inject different implementations of the same dependency
- Inject the same implementation in different configurations

Jmetric high-level architecture: We have developed a framework and a tool (i.e., a reference implementation of jmetric) known as jmetric and jmetric tool respectively to predict metrics such as number of operations overridden by a subclass tree, number of operations added by a subclass, percent public and protected, number of children and depth of the inheritance tree. Figure 1 shows the high-level architecture of jmetric.

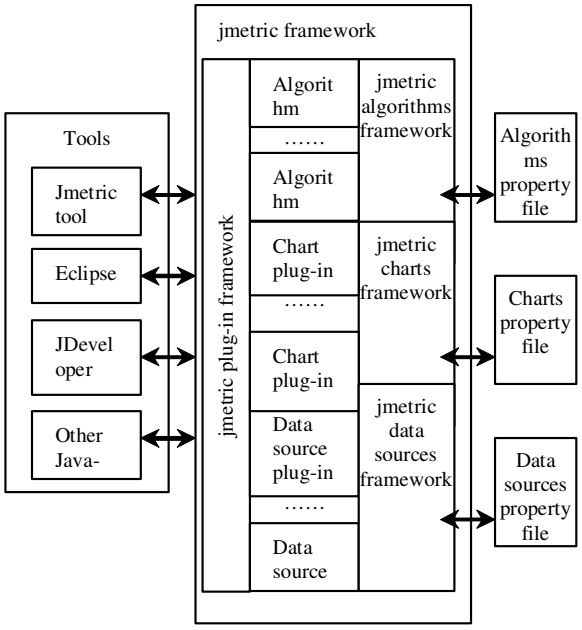

Fig. 1: jmetric high-level architecture

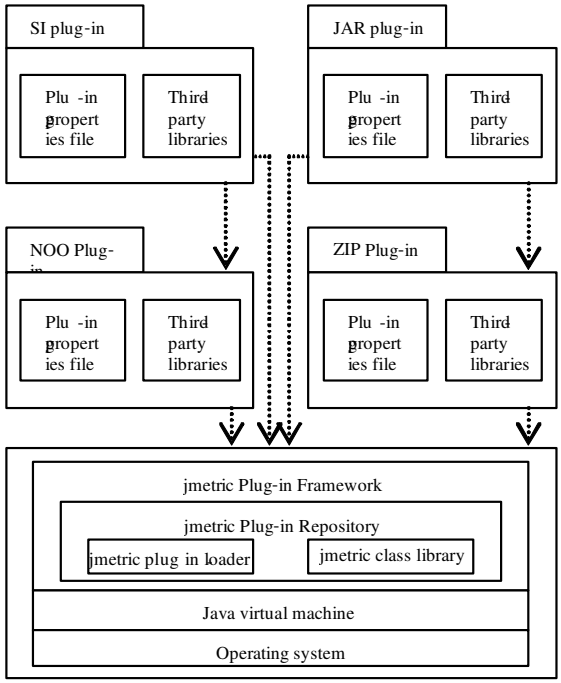

Fig. 2: jmetric layered architecture

jmetric plug-in framework is a general-purpose framework intended to help building scalable, extendable Jmetric components. The plug-in framework provides a runtime engine that dynamically registers, unregisters, loads and unloads plug-ins. The plug-in framework is engineered in such a way that additional application-specific plug-ins can be created in order to meet application-specific requirements. Generally, the framework consists of abstract and concrete classes that describe specific object-oriented design metrics via plug-ins. Any instantiation of jmetric consists of composing and subclassing the existing abstract classes.

There are typically three types of plug-ins, namely jmetric algorithm plug-ins, jmetric chart plug-ins and jmetric data source plug-ins. A jmetric plug-in is a structured component that describes itself to the jmetric plug-in framework through standard Java property files.

Jmetric algorithm plug-ins: A jmetric algorithm plug-in is a structured component that describes a metric. Examples of jmetric algorithms are depth of inheritance tree and number of operations overridden by a subclass tree.

Jmetric chart plug-ins: A jmetric chart plug-in is a structured component that describes the indexes of a metric graphically. Examples of jmetric chart plug-ins are scaled chart, bar chart and line chart.

Jmetric data source plug-ins: A jmetric data source plug-in is a structured component that describes a data source, for example, a directory, a JAR file (i.e., Java Archive), a WAR file (i.e., Web Application Archive) and a ZIP file (i.e., Compressed File Archive).

Jmetric tool, Eclipse, JDeveloper and other Java-based tools are considered as application systems. The application systems rely heavily on the plug-in framework to interact with specific types of plug-ins: jmetric algorithm plug-ins and jmetric chart plug-ins.

Jmetric layered architecture: Figure 2 shows the jmetric layered architecture. Each jmetric plug-in may rely on services provided by other jmetric plug-ins and each may in turn provide services on which other jmetric plug-ins may rely. The SI (i.e., Specialization index) plug-in relies on NOO (i.e., Number of operations overridden by a subclass). The specialization index provides an indication of the degree of specialization for each of the subclasses in an object-oriented system[16]. The metric relies on NOO metric as one of the numerators of SI metric. JAR plug-in relies on ZIP plug-in as JAR files are based on ZIP file format specification.

Jmetric runs on any Java Virtual Machine (JVM) that adheres to the JVM specification published by Sun. Different reference implementations of JVMs supported by jmetric are Sun JVM, Oracle JVM and IBM JVM.

The JVM parameters such as –Xmsn (i.e., to specify the initial size, in bytes, of the memory allocation pool) and -Xmxn (i.e., to specify the maximum size, in bytes, of the memory allocation pool) can be configured accordingly to tune the performance of jmetric on different operating systems.

Jmetric repository consists of jmetric plug-in loader and jmetric class library. The jmetric plug-in loader implements the runtime engine that dynamically discovers and activates jmetric plug-ins. The jmetric class library is a standard library of classes and interfaces that brings together a large number of object-oriented design metrics functions. The library is designed as the foundation on which jmetric plug-ins such as jmetric algorithm plug-ins and jmetric data source plug-ins are built.

Jmetric maintains a registry of available plug-ins and the functions they provide via variation points (also known as extension points) and variants (also known as extensions). To simplify distribution of plug-ins, a jmetric plug-in may be packaged as single JAR file that is unpacked transparently in runtime.

Jmetric Plug-in High Level Architecture: Figure 3 shows the jmetric plug-in high-level architecture. A jmetric plug-in (also known as add-in, add-on or snap-in) is a structured component that describes itself to jmetric through a standard configuration file (i.e., a standard Java properties file)[2,5,6,12]. jmetric maintains a registry of available plug-ins and the functions they provide through extension points (also known as variation points) and extensions (variants). The jmetric plug-ins are designed based on nine distinctive design characteristics: complexity (i.e., a low effort is required in understanding an operational jmetric plug-in), flexibility (i.e., a low effort required in modifying an operational jmetric plug-in), maintainability (i.e., a low effort is required in maintaining an operational jmetric plug-in), reusability (i.e., a low effort is required in reusing an operational jmetric plug-in), testability (i.e., a low effort is required in testing an operational jmetric plug-in), operability (i.e., a low effort is required in operating an operational jmetric plug-in), traceability (i.e., a low effort is required in tracing an operational jmetric plug-in), interoperability (i.e., a low effort is required in interoperating an operational jmetric plug-in among different tools).

A jmetric plug-in may itself be augmented by different kinds of extensions. jmetric plug-ins provide different types of slots (i.e., extension points) that extensions (i.e., other jmetric plug-ins) can plug into. A jmetric plug-in (e.g., NOO plug-in) allows three types of slots (i.e., algorithms, charts and data sources) upon which any number of a jmetric extender plug-in (e.g., SI plug-in) can be plugged.

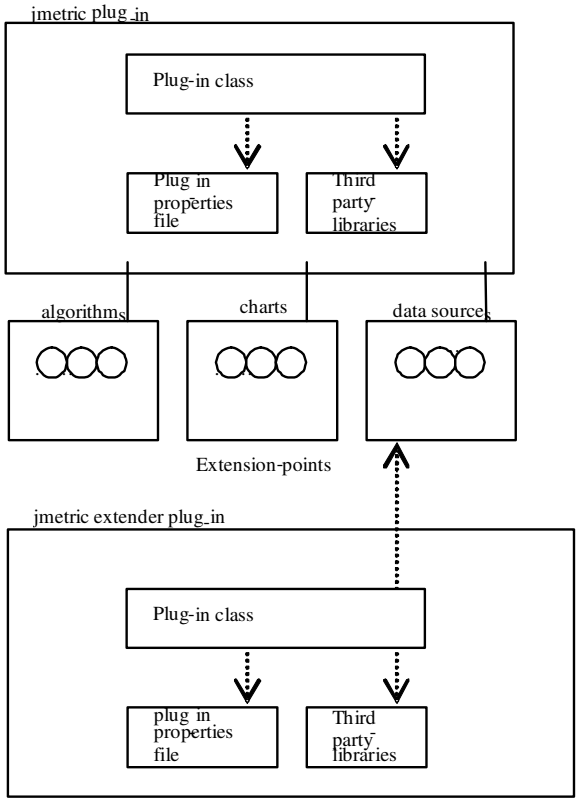

Fig. 3: jmetric plug-in high-level architecture

Variability mechanisms are used in conjunction with each other to help building a small, coherent number of plug-ins[9]. Through variability mechanisms, the components of jmetric can be combined in a variety of ways to produce the desired plug-ins effectively and efficiently. jmetric implemented the inheritance (i.e., via virtual operation extension point type), configuration (i.e., via configuration item slot extension point type) and parameters (i.e., via parameter extension point type) variability mechanisms.

RESULTS

We have developed a tool called jmetric tool as a reference implementation to illustrate the capabilities of the jmetric. A reference implementation is, in general, an implementation of a specification to be used as a definitive interpretation for that specification. During the development of the conformance test suite, at least one relatively trusted implementation of each interface is necessary to discover errors or ambiguities in the specification and validate the correct functioning of the test suite[8].

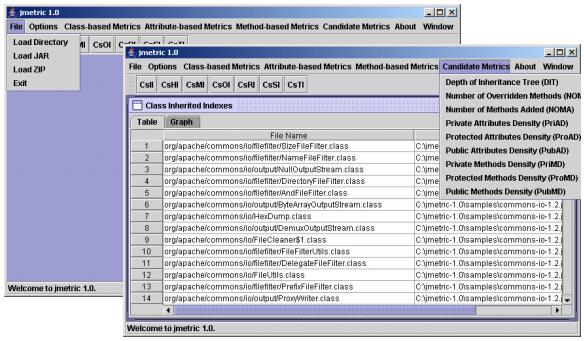

Fig. 4: jmetric tool

The jmetric reference implementation, jmetric tool (also known as jmetric sample implementation or jmetric model implementation) is a software example of jmetric plug-in specification intended to be used to help object-oriented design metric plug-in developers to implement their own version of plug-in specification.

Figure 4 shows the jmetric tool that we have developed to validate the plug-in capabilities of jmetric, namely jmetric algorithm plug-ins, jmetric chart plug-ins and jmetric data source plug-ins. jmetric tool can be used to predict a number of metrics proposed by other researchers such as depth of inheritance tree, number of overridden methods, number of methods added, private attributes density and protected attributes attributes density. The tool categorizes object-oriented design metrics into four categories: Class-based metrics, attribute-based metrics, method-based metrics and candidate metrics. These are referred to as jmetric algorithms plug-in as shown in Fig. 1. The jmetric data source plug-ins such as JAR plug-in and ZIP plug-in are located under the File menu. Lastly, the jmetric chart plug-ins are located under the Graph tab of each metric.

DISCUSSION

Standalone tools: Most of the software metrics tools such as Krakatau Professional and Essential Metrics lack of a general-purpose, platform-independent plug-in-based metric framework. Standalone tools such as JStyle, Krakatau Professional, Essential Metrics and McCabe IQ Developers Edition rely heavily on proprietary APIs. Extending the functionalities of these tools is not likely to be possible without the need to access vendor-specific native APIs. jmetric provides a Java-language alternative to these vendor-specific APIs through plug-ins. The plug-ins allow third-party tools (i.e., standalone software metrics tools, IDEs and other types of tools) to interface with jmetric.

IDEs: IDEs such as Eclipse IDE for Java EE Developers, Eclipse Classic, Eclipse IDE for Java Developers and NetBeans lack of software metrics functionalities. A few IDEs such as Visual Studio Team System (Development Edition and Team Suite) provides software metrics functionalities (referred to as code metrics) commercially off-the-shelf. Examples of code metrics provided by the IDE are Class Coupling, Depth of Inheritance and Cyclomatic Complexity. Extending the IDE to include other metrics is difficult as the IDE lacks of a domain specific framework that consists of specialized framework components on design metrics. jmetric accomplishes the goal to bridge between the IDEs and jmetric via jmetric plug-ins as most IDEs provide a way to extend the functionalities of the IDEs through plug-ins..

CONCLUSION

The primary objective of the research is to design and develop a lightweight object-oriented design metric framework that helps to assemble metrics into tools such as Eclipse and NetBeans via plug-ins. Underlying the framework is a common pattern to perform the wiring of plug-ins via dependency injection. Dependency injection is heavily used to accelerate the adding, removing and updating of object-oriented metrics plug-ins in tools such as Eclipse and JDeveloper. jmetric plug-ins are activated (i.e., loaded) lazily upon instantiating the framework but not deactivated (i.e., unloaded) when they are no longer required. This potentially causes the memory footprint to grow when more functionality are injected into the framework. In future, we intend to address this issue via a scheduling algorithm. Other future work includes extending the framework to other object-oriented programming languages such as Visual C# .NET and Visual Basic .NET through a service-oriented architecture. We also intend to investigate the possibilities to build an object-oriented metric domain specific language through Eclipse Modeling Framework and Domain Specific Language Tools for Visual Studio. Through the domain specific language, we plan to build an exemplary visual tool to provide a code generation facility via modeling based on a structured data model such as XML.

REFERENCES

1. Alain, A., A. Sellami and W. Suryn, 2003. Metrology, measurement and metrics in software engineering. Proceedings of the 9th International Symposium on Software Metrics, Sep. 03-05, IEEE Computer Society, Washington, DC., USA., pp: 2. http://portal.acm.org/citation.cfm?id=943758.

2. Bako, B., A. Borchert, N. Heidenbluth and J. Mayer, 2006. Linearly ordered plugins through self-organization. Proceedings of the International Conference on Autonomic and Autonomous Systems, July 16-18, IEEE Computer Society, Washington DC., USA., pp: 25-25. DOI: 10.1109/ICAS.2006.1.
3. Chidamber, S.R. and C.F. Kemerer, 1994. A metrics suite for object-oriented design. IEEE Trans. Software Eng., 20: 476-493. DOI: 10.1109/32.295895.
4. Chidamber, S.R., D.P. Darcy and C.F. Kemerer, 1998. Managerial use of metrics for object-oriented software: An exploratory analysis. IEEE Trans. Software Eng., 24: 629-639. DOI: 10.1109/32.707698.
5. Chatley, R., S. Eisenbach and J. Magee, 2003. Modelling a framework for plugins. Proceedings of the Workshop on Specification and Verification of Component-Based Systems, Sep. 03-11, Springer-Verlag, Londong, UK., pp: 49-57. http://pubs.doc.ic.ac.uk/ModellingPluginFramework/.
6. Chatley, R., S. Eisenbach and J. Magee, 2004. Magicbeans: A platform for deploying plugin components. Proceedings of 2nd International Working Conference on Component Deployment, May 3, Springer-Verlag, London, UK., pp: 97-112. DOI: 10.1007/b98000.
7. Churcher, N.I. and M.J. Shepperd, 1995. Towards a conceptual framework for object-oriented software metrics. ACM Software Eng. Notes, 20: 69-75. http://portal.acm.org/citation.cfm?id=224163.
8. Eric, D., E. Fong and A., Goldfine, 2003. Requirements for GSC-IS reference implementations.
9. Frank, P., A. Schnieders, J. Weiland and M. Weske, 2005. Variability mechanisms for process models, PESOA-Report No. TR 17/2005. http://www.pesoa.de/pages/Publications/Fachberichte062005/PESOA_TR_17-2005.pdf.
10. Gamma, E., R. Helm, R. Johnson and J. Vlissides, 1994. Design Patterns: Elements of Reusable Object-Oriented Software. 1st Edn., Addison Wesley, USA., ISBN: 0-201-63361-2.
11. Harrison, R., S.J. Counsell and R.V. Nithi, 1998. An evaluation of the mood set of object-oriented software metrics. IEEE Trans. Software Eng., 24: 491-496. DOI: 10.1109/32.689404.
12. Johannes, M., I. Melzer and F. Schweiggert, 2002. Lightweight plug-in-based application development. Proceeding of the International Conference NetObjectDays on Objects, Components, Architectures, Services and Applications for a Networked World, Oct. 07-10, Springer-Verlag, London, UK., pp: 87-102. http://portal.acm.org/citation.cfm?id=648033.744228.
13. Larman, C., 2002. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd Edn., Prentice Hall PTR., USA., ISBN: 10: 0130925691, pp: 656.
14. Lorenz, M. and J. Kidd, 1994. Object-Oriented Software Metrics. 1st Edn., Prentice Hall, USA., ISBN: 10: 013179292X, pp: 146.
15. Martin F., 1996. Analysis Patterns: Reusable Object Models. 1st Edn., Addison Wesley Professional, USA., ISBN: 10: 0201895420, pp: 384.
16. Roger S. P., 2007. Software Engineering: A Practitioner's Approach. 6th Ed., McGraw-Hill, USA., ISBN: 10: 0077227808.