Available online at www.elixirpublishers.com (Elixir International Journal)

Advanced Engineering Informatics

Elixir Adv. Engg. Info. 34 (2011) 2557-2564

ISSN: 2229-712X

# Software metrics – a survey to procedure oriented, object oriented and web based metrics

Rakesh Kumar[1] and Gurvinder Kaur[2]

[1]Department of Computer Science and Applications, Kurukshetra University Kurukshetra.

[2]Guru Nanak Khalsa Institute of Technology and Management Studies, Yamuna Nagar.

**ARTICLE INFO**

**Article history:**

Received: 24 March 2011;

Received in revised form: 23 April 2011;

Accepted: 28 April 2011;

**Keywords**

Software Metrics,
Software Measurement,
Procedure Oriented Metrics,
Object Oriented Metric,
Web Based Metrics.

**ABSTRACT**

Without the help of measurement quality software cannot be built. Measurement is an essential aspect for achieving the basic management objectives of prediction, progress, and process improvement. Software measurement is the raw data associated with various elements of the software process and product. It acts as a quantitative basis for the development and validation of models for a software development process. The major goal of software metrics is identification and measurement of the essential parameters affecting the software development. An oft-repeated phrase by De Marco holds true; "You can't manage what you can't measure!" [DEM86]. All process improvement must be based on measuring where you have been, where you are now, and properly using the data to predict where you are heading. Collecting good metrics and properly using them always leads to process improvement! This paper gives an exhaustive overview of metrics used in software development in different language paradigms. Metrics are classified as procedure oriented, object oriented and web oriented metrics. The object oriented metrics are further classified into Chidamber and MOOD metrics. In an object-oriented system, traditional metrics which are generally applied to the methods comprising the operations of a class is also highlighted. Web based objects further classified as multimedia files, web building blocks, scripts and links are also described.

## Introduction

Metrics are the units of measurement taken on a particular item or process. Software engineering metrics are the units of measurement that are used to characterize: a) Processes, e.g., activities of problem definition, analysis, designing b) Product, e.g., designs models, source code and test cases c) People, e.g., Efficiency and productivity of the designer, programmer and tester. Many software metrics have been proposed in the literature [CHI94] [FEN96].

For these metrics to be widely accepted, empirical studies of the use of these metrics as quality indicators are required. According to lord Kelvin, a physicist, Software Metrics is "When you can measure what you are speaking about, and can express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."

## Classification of metrics

According to [MIL88] and [MOE93], metrics can be classified by different aspects as procedure oriented metrics, object oriented metrics and web based metrics.

Procedural metrics are those which measure the properties related to software developed in procedural programming languages. Object oriented development requires analysis, design and implementation in software metrics.

Web based metrics are those which represent the size of web applications.

**Tele:**
**E-mail addresses: rsagwal@rediffmail.com, allagh.gurvinder@gmail.com**

## Procedural metrics

Procedural metrics are those metrics which are used to measure the properties related to software developed in procedural programming languages. They are organized around a view of the software in which the individual procedural or subprogram is the most significant unit. These metrics consists of: a) Lines of Code (LOC):The Lines of Code measures the volume of code that is used to compare or estimate projects that use the same language, and is coded using the same coding standards. [CON86] Several ways are given below through which the lines can be counted depending upon what is counted and what line of count can be achieved. i) *Physical lines (LINES):* This is the simplest line count. Each line ends with a line break, usually CR+LF. LINES counts every line, be it a code, a comment or an empty line. ii) *Logical lines of code (LLOC):* The number of logical lines can be counted by LLOC. Where two or more lines are joined with the "_" line continuation sequence, they count as one line. b) McCabe's Cyclomatic complexity: MCCabe developed a measure of the complexity of a program known as cyclomatic complexity. [CAB76] The cyclomatic complexity (CC) of a graph (G) is computed by the formula: CC(G) = Number (edges) : Number (nodes) + 1. This metric is used as an ease of maintenance metric, a quality metric, it can measure the minimum effort and best areas of concentration for testing. c) Knot metric: The Knot metric was designed for a method of modeling control flow in Fortran programs. The programmers draw lines, in source code text, to indicate how different instructions would alter program flow. For these program complexity was used. The knot metric was used to measure program complexity by counting the intersections of control flow lines in the program models [TAI84]. For any structured procedural program, the knot complexity measure can be computed as the sum of intersecting program jumps. A program jump from line A to line B, denoted as an ordered pair (A, B), intersects with another program jump (X, Y) if either of the following conditions are satisfied: 1)min(A,B) < min(X,Y) < max(A,B) AND max(X,Y) > max(A,B) 2) min(A,B) < max(X,Y) < max(A,B) AND min(X,Y) < min(A,B).

These intersecting program jumps are known as knots. d) Function Point Analysis (FP) by Allan J. Albrecht was an objective and structured technique to measure software size by quantifying its functionality provided to the user based on the requirements and logical design.

This technique breaks the system into smaller components so that they can be better understood and analyzed. It divides the system into five basic components namely external inputs, external outputs, queries, logical master files and interface files. [ALB83]. FP can be calculated as: FP = count total *[0.65 + 0.01*∑Fi] where: FP = Total number of adjusted function points, count total = the sum of all user inputs, outputs, inquiries, files and external interfaces to which have been applied the weighting factor and Fi = a complexity adjustment value. e) Fan-In Fan-Out Complexity:

Henry and Kafura [HEN81][SOM92] proposed a method that identify the number of calls to the module and the number of calls from a module. The procedure's complexity (C) is then defined as: $C = L \times (\text{Fan-in} \times \text{Fan-out})^2$ where L is any measure of module length such as LOC or V(G), fan-in = the number of calls to the module and fan-out = the number of calls from the module. C is compared with Halstead's *E* metric and McCabe's cyclomatic complexity in [HEN81]. f) Bang Metrics: Demacro's Bang Metric is used to predict the application size based on the analysis model [DEM98]. This metric is calculated by using certain algorithms and data primitives from a set of formal specifications for the software. g) Halstead Software Science: Maurice Halstead developed a quantitative measure of complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. [HAL77].

The number of unique operators and operands (n1 and n2) as well as the total number of operators and operands (N1 and N2) can be calculated by collecting the frequencies of each operator and operand token of the source program. He identified a set of metrics for several aspects of programs and software production effort. He proposed program vocabulary (n), program length (*N*), and volume (*V*) metrics and effort (*E*) and development time (*T*). (i) The vocabulary size of a program (n) is the sum of the number of unique operators and operands: n = n1 + n2 (ii) The length N of a program can be defined as the total usage of 'all' operators appearing in the implementation plus the total usage of 'all' operands appearing in the implementation. N = N1+ N2 (iii) The program volume (V) is the information contents of the program, measured in mathematical bits. It can be calculated as: V = N * log2(n) (iv) The Halstead Effort estimates the amount of work that it would take to recode a particular method. The effort (E) is proportional to the volume and to the difficulty level of the program. E = V * D (v) The time (T) of a program is proportional to the effort. T = E / 18

**Object-Oriented Metrics**

Object Oriented Metrics are the metrics for analyzing OO language as an indicator of quality attributes. "Object-oriented design is a method of design encompassing the process of object oriented decomposing and a notation for depicting both logical and physical as well as static and dynamic models of the system under design"[SHE95].

Inheritance, association, aggregation, polymorphism and message passing are some of the objet-oriented mechanisms. Chidamber also defined a large number of object-oriented metrics which included Weighted Methods per Class (WMC), Number of Children (NOC), and Response for Class (RFC) [CHI94]. Metrics also further categorized for measuring size such as number of methods, number of attributes, number of classes, measuring coupling such as direct class coupling, number of dependencies in and coupling factor.

The Object Oriented metrics are further divided into two sub-categories: Intra and Inter Class Metrics. Intra class metrics are the metrics which measure characteristics related to one class, such as Weighted Methods per Class (WMC), Number of Children (NOC), and Depth of Inheritance Tree [CHI94]. Inter class metrics are those that measure features between a set of classes such as Coupling Factor (CF) and Method Hiding Factor (MHF) [ABR94].

OO metrics gather quantitative measurement of their product and processes for possible improvement as well as estimation for the software project. OO Software Development Methodologies divides the development cycle into 4 phrases, Analysis, Design, Implementation and Testing. OO metrics has different aims and purpose for different phrases of the development:

1) Analysis phrase: In the analysis phrase, use case models are used to capture the functional requirements of a software project. Measurement helps in estimate of cost, schedule and resource required in the software project. In 1993, Gustav Karner of Objectory (now Rational Software) used the "Use Case Points" method for sizing and estimating projects. 2) Design and Implementation Phrase: In this phrase metrics evaluate the design to highlight possible inappropriate design to the system designer. Possible metrics used to determine inappropriate design includes Coupling between Object Classes (CBO), Response for a Class (RFC), Depth of Inheritance Tree (DIT), and Number of Children (NOC). 3) Testing Phrase: Estimation of software testing efforts can be derived from use case model from the analysis phrase. This assists the project manager to estimate the test effort needed for the project.

**Metrics for analysis**

Chidamber - Kemerer (CK) [CHI94] and MOOD[1,2] [ABR95] [ABR01]suites are two suites of metrics in object-oriented.

**Chidamber & kemerer object-oriented metrics suite:** The Chidamber & Kemerer metrics suite originally consists of 6 metrics calculated for each class: WMC, DIT, NOC, CBO, RFC and LCOM. 1) *Weighted Methods Per Class (WMC):* Number of methods defined in class is known as WMC. [CHI93][BAS96]. It measures the complexity of an individual class. A class with more member functions is more complex and therefore results to errors. Larger the number of methods in a class, greater is the potential impact on children as children inherit all the methods defined in a class. 2) *Depth of Inheritance Tree (DIT):* The depth of a class within the inheritance hierarchy is defined as the maximum length from the class node to the root/parent of the class hierarchy tree and is measured by the number of ancestor classes. [CHI94] 3) *Number of Children (NOC):* This is the number of direct descendants (subclasses) for each class. [CHI93] [BAS96] 4) *Coupling between Object Classes (CBO):* It is the number of classes to which a class is coupled. [CHI93][BAS96][WHI97][BRI96][HOU] 5) *Response for a Class (RFC and RFC´):* The response set of a class is a set of methods that can be executed in response to a message received by an object of that class. RFC = M + R and RFC' = M + R' where M = number of methods in the class R = number of remote methods directly called by methods of the class R' = number of remote methods called, recursively through the entire call tree. [CHI93][BAS96][WHI97][BRI96] 6) *Lack of Cohesion of Methods (LCOM):* The 6th metric is the number of disjoint/non-intersection sets of local methods.

**Mood And Mood2 Metrics:** Fernando Brito e Abreu defined the MOOD metrics for designing a summary of the overall quality of an object-oriented project. The original MOOD metrics suite consists of 6 metrics. The MOOD2 metrics were added later.

**i) MOOD metrics suite:** The metrics suite includes 6 metrics: MHF, AHF, MIF, AIF, PF and CF.a) *Method Hiding Factor (MHF) & Attribute Hiding Factor (AHF):* MHF measure how methods can be encapsulated in a class and AHF measure the encapsulation of variables in a class. b) *Method Inheritance Factor (MIF) & Attribute Inheritance Factor (AIF):* MIF is the inherited methods/total methods that are available in a class whereas AIF is the inherited attributes/total attributes which are available in a class. 3) *Polymorphism Factor (PF):* Polymorphism Factor measures the degree of method overriding in the class inheritance tree. Also, PF = number of actual method overrides / maximum number of possible method overrides. 4) *Coupling Factor (CF):* Coupling Factor is the factor that measures the actual couplings among classes in relation to the maximum number of possible couplings. [BRI96]

**ii) MOOD2 metrics Suite:** The MOOD2 metrics set is a later addition by the author of the MOOD metrics set. This includes OHEF, AHEF, IIF, PPF. [ABR01]. a) *Operation/Attribute Hiding Effectiveness Factor (OHEF & AHEF):* OHEF are the classes that access operations. AHEF are classes that access attributes. b) *Internal inheritance factor (IIF):* It is the class that inherits a visual basic class or all the classes that inherit something is known as IIF. If there is no inheritance then IIF=0. ***Parametric polymorphism factor (PPF):*** This metric is the percentage of the classes that are parameterized. [ABR95][ABR01] [MIS03][ABR96][BRI96]. Some measures that can be computed using PPF are: 1) *Coupling:* Stevens defined coupling as "the measure of the strength of association established by a connection from one module to another [MYE74]. 2) *Cohesion:*

"Cohesion measures the degree of connectivity among the elements of a single class or object" [SHE95].3) *Encapsulation:* It is packaging or binding together a collection of items like low : level (records and arrays) and mid : level encapsulation (subprograms as procedures, functions, subroutines and paragraphs) 4) *Factoring Effectiveness:* This is equivalent to the number of unique methods by total number of methods 5) *Application Granularity:* This equals total number of objects / total function points. 6) *Methods per class:* Average number of methods per object class = Total number of methods / Total number of object classes.

**Object-Oriented Specific Metrics**

Different metrics were applied to the concepts of classes, coupling, and inheritance as:

**Class Metrics:** Different types of class Metrics are: 1) *Attribute Hiding Factor (AHF):* Ratio of sum of inherited attributes in all classes to the total number of available classes attributes [MOR88]. 2) *Class Cohesion (CCO)*: This measures the relations between the classes [CHI91]. 3) *Class Entropy Complexity (CEC):* It helps in measuring the complexity of classes based on information content [DAV97]. 4) *Comment Lines per Method (CLM):* This measures the percentage of comments in methods [LOR94]. 5) *Data Access Metric (DAM):* It is the ratio of the number of private attributes to the total number of attributes declared in the class [DAV97]. 6) *Function Oriented Code (FOC):* Percentage of non object–oriented code used in a program can be measured. [LOR94].7) *Internal Privacy (INP):* It refers to the use of accessory functions even within a class [CHI94]. 8) *Measure of Attribute Abstraction (MAA)*: This is ratio of the number of attributes inherited by a class to the total number of attributes in the class [DAV97]. 9) *Measure of Functional Abstraction (MFA)*: It is the ratio of the number of methods inherited by a class to the total number of methods accessible by members in the class [DAV97].10) *Number of Abstract Data types (NAD)*: Number of user-defined objects used as attributes in a class that are necessary to instantiate an object instance of the class [DAV97]. 11) *Number of Class Methods in a class (NCM)*: It weighs the measures in a class but not in instances [LOR94].12) *Number of Instance Variables in a class (NIV):* It measures relations of a class with other objects of the program [LOR94]. 13) *Number Of Ancestors (NOA)*: It is the total number of ancestors of a class [KOL93]. 14) Number of Public Attributes (NPA): It counts the number of attributes declared as public in a class [DAV97]. 15) *Number of Parameters per Method (NPM):* It is the average number of parameters per method in a class [DAV97]. 16) *Number of Reference Attributes (NRA):* It counts the number of pointers and references used as attributes in a class [DAV97]. 17) *Percentage of Commented Methods (PCM):* It is the percentage of commented methods [LOR94]. 18) *Public Data (PDA):* It counts the accesses of public and protected data of a class [CAB94]. 19) *Percent of Potential Method uses actually Reused (PMR):* It is the percentage of the actual method uses [MOR88]. 20) *Percentage of Public Data (PPD):* It is the percentage of the public data of a class [CAB94]. 21) *Weighted Class Size (WCS):* It is the number of ancestors plus the total class method size [KOLE93].

**Method Metrics** – The Method metrics can be categorized as: 1) *Average Method Complexity (AMC):* It is the sum of the cyclomatic complexity of all methods divided by the total number of methods [MOR88]. 2) *Average Method Size (AMS):* It measures the average size of program methods [LOR94]. 3) *MAX V(G) (MAG):* It is the maximum cyclomatic complexity of the methods of one class M. 4) *Method Complexity (MCX):* It relates complexity with the number of messages [LOR94][DRE89].

**Coupling Metrics:** The Coupling Metrics can be classified as following: 1) *Class Coupling (CCP):* It measures connections between classes based on the messages they exchange [BOY93]. 2) *Coupling Factor (CFA):* It is the ratio of the maximum possible number of couplings in the system to the actual number of couplings not imputable to inheritance [HAR98].

**Inheritance Metrics:** The Inheritance Metrics are: 1) *FAN:IN (FIN):* It is the number of classes from which a class is derived and high values indicates excessive use of multiple inheritance [CAB94]. 2) *Class Hierarchy Nesting Level (HNL):* It measures the depth in hierarchy that every class is located [LOR91]. 3) *Method Reuse Metrics (MRE):* It indicates the level of methods reuse [ABR94]. 4) *Number of Methods Inherited (NMI):* It measures the number of methods a class inherits [LOR94]. 5) *Number of Methods Overridden (NMO):* It is the number of methods needed to be re-declared by the inheriting class [LOR94]. 6) *Percent of Potential Method uses Overridden (PMO):* It is the percentage of the overridden methods [MOR88]. 7) *Ratio between Depth and Breadth (RDB):* It is the ratio between the depth and the width of the hierarchy of the classes [BEL99]. 8) *Reuse Ratio (RER):* It is the ratio of the number of super-classes divided by the total number of classes [ROS97]. 9) *Specialization Index (SIX):* It measures the type of specialization [LOR94]. 10) *Specialization Ratio (SPR):* It is the ratio of the number of subclasses divided by the number of super-classes [ROS97].

**System Metrics:** The System metrics includes: 1) *Average Depth of Inheritance (ADI):* It is computed by dividing the sum of nesting levels of all classes by the number of classes [DAV97]. 2) *Average Number of Ancestors (ANA):* It determines the average number of ancestors of all the classes [DAV97]. 3) *Application Granularity (APG)*: It is the total number of objects divided by the total number of function points [MOR88]. 4) *Association Complexity (ASC):* It measures the complexity of the association structure of a system [KOL93]. 5) *Category Naming (CAN):* It divides classes into semantically meaningful sets [CHI94]. 6) *Number of time a Class is Reused (CRE):* It measures the references in a class and the number of the applications that reuse this class [LOR94][ABR94]. 7) *Functional Density (FDE):* It is the ratio of LOC to the function points [FEN97]. 8) *Number of Classes Thrown away (NCT):* It measures the number of times a class is rejected until it is finally accepted [LOR94][WES92]. 9) *Number Of Hierarchies (NOH):* It is the number of distinct hierarchies of the system [KOL93].10) *Object Library Effectiveness (OLE):* It is the ratio of the total number of object reuses divided by the total number of library objects [MOR88]. 11) *Problem Reports per Class (PRC):* It measures defect reports on this class [LOR94]. 12) *Percent of Reused Objects Modified (PRO):* It declares the percentage of the reused objects that have been modified [BEL99].13) *System Reuse (SRE):* It declares the percentage of the reuse of classes [ABR94].

**Lorenz and Kidd Object-Oriented Metrics**

Lorenz and Kidd also proposed some metrics for the quantification of software quality assessment [LOR94]. They introduced 11 metrics applicable to class diagrams. These metrics were classified into three categories: 1) Class size metrics quantifies an individual class. This includes: a) *Number of Public Methods (NPM)*: This is used to count the number of public methods in a class. b) *Number of Methods (NM):* In this the total number of methods in a class counts all public, private and protected methods. c) *Number of Public Variables per class (NPV):* This metric counts the number of public variables in a class. d) *Number of Variables per class (NV):* The total number of variables including public, private and protected variables. 2) Class Inheritance metrics which gives quality of the classes with the use of inheritance. These metrics can be categorized into: a) *Number of Methods Inherited (NMI):* This metric measures the number of methods inherited by a subclass b) *Number of New Methods (NNA):* A method is defined as an added method in a subclass if there is no method of the same name in any of its super classes. 3) Class Internals metrics which shows the general characteristics of classes. a) *Average parameters per Method (APM):* This is defined as the total number of parameters in a class / Total number of methods b) *Specialization Index (SIX):* The specialization index measures to what extent subclasses redefine the behavior of their super classes.

**Web Application Metrics**

Estimation of the size of web applications is a new problems for cost analysts as hypertext languages (html, xml, etc.), multimedia files (audio, video, etc.), scripts (for animation, bindings, etc.) and web building blocks (active components like ActiveX and applets, building blocks like buttons and objects like shopping carts, and static components like DCOM and OLE) are employed in such applications. Improved size estimating techniques are therefore needed to address the shortfall. Else, the size estimates that we use to drive our cost models will be flawed.

Using Halstead software science [HAL77] Web Objects as a new metrics was developed for representing the size of web applications. Web Objects are an extension of function points. There are four additional types of objects incorporated into web applications: a) *Multimedia files:* These are developed to incorporate audio, video and images into applications. These helps in creating web pages; creating video for the web (MPEG-1&2 files); creating publishable documents for the web; and creating, editing and enhancing complex images for both clients and servers. b) *Web building blocks*: These develop web-enabled fine-grained component and building block libraries and any wrapper code required to either instantiate or integrate them. These make use of the additional active (ActiveX, applets, agents, guards, etc.), fine-grained static (COM, DCOM, OLE, etc.) and course-grain reusable (shopping carts, buttons, logos, etc.) building blocks that are acquired or developed to incorporate into web applications for both client and server. c) *Scripts:* these are developed to link html/xml data and generate reports automatically; query ODBC- compliant databases via prompts; integrate and animate applications via predefined logic (via GIF); and direct dynamic web content per customizable pallets, masks, windows and commands (streaming video, real-time 3D, special effects, motion, guided workflow, batch capture, etc.) for both clients and servers. d) *Links (xml, html and query language lines):* These link the applications, integrate them together dynamically and bind them to the database and other applications in a persistent manner.

**Web Application Metrics**

There are some of the web application metrics as described below: 1) HTTP Content: These explain about the HTTP Content metrics, their descriptions, and user actions. a) *Average Connect Time:* This metric measures the average connect time for all pages in the transaction. This can be calculated as: Total Connect Time / Number of Connections Made. The Connect Time is one of the phases of a transaction that helps to isolate and fix response time problems. b) *Average First Byte Time:* This metric measure the average First Byte Time for all pages in the transaction. This metric can be computed as: Total First Byte Time / Number of Requests Made. The First Byte time is one of the phases of a transaction that helps to isolate and fix response time problems. c) *Average Response Time:* The Average Page Response metric calculates the average response time of the pages within a single transaction. This is calculated as: Total Transaction Time / Number of Pages in the Transaction. d) *Beacon Name:* The beacon name is the name of the beacon for which the current metric data is being collected. e) *Broken Count:* This metric measures the number of errors encountered when displaying content for the pages accessed by the transaction. f) *Computed Response Time:* This metric represents the estimated response time for a client to fetch all the pages in a transaction. g) *Connect Time:* Connect Time is the first phase of a transaction and represents the time it takes for a connection to the Web server established for all requests. Each transaction is broken into individual phases by Enterprise Manager. h) *Content Time:* It is the amount of time taken to transfer page content to the browser. Page content includes images and style sheets, as opposed to the HTML coding for the page. i) *First Byte Time:* This is the total time taken between the last byte of the request sent and the first byte of the response received by the server for all requests made. j) *HTML Bytes:* This metric provides information about the amount of data transferred during the selected transaction. k) *HTML Content:* This metric serves as a container for a set of metrics that provide information about the content of the Web pages. l) *HTTP Response:* This metric is a container for a set of metrics used to measure the performance of the Web Application transactions. m) *HTML Time:* It is the amount of time taken to transfer the HTML coding of the page to the browser. n) *Page Content Bytes:* This metrics provides information about the amount of data transferred during the selected transaction. o) *Redirect Time*: This represents the total time of all redirects within a transaction. p) *Slowest Response Time:* This metric indicates the maximum response time measured for a particular page within a transaction. q) *Status:* This metric returns a value of 1 if the selected beacon was successfully able to run the transaction for this Web application target. r) *Status Description:* If the beacon is unable to run the transaction successfully, this metric returns a description of the error that prevented the transaction from running. s) *Total Bytes:* This metric provides information about the amount of data transferred during the selected transaction. t) *Total Response Time:* This metric calculates total transaction time by assuming all contents of a page are fetched in a serial manner. u) *Transaction Name:* It is the name of the transaction for which the current metric data is collected. v) *Transfer Rate:* This indicates how quickly data is being transferred from the Web server to the client browser. This is computed as: Total Kilobytes Received / Total Transaction Time. w) *Web Application:* Enterprise Manager can be used to view performance and availability metrics for the Web applications. 2) HTTP Step Group: The following describes the HTTP Step Group metrics, their descriptions, and user actions. a) *[HTTP Step Group] Broken URL Count:* This metrics measures the number of errors encountered when displaying content for the pages accessed by the step group. b) *[HTTP Step Group] Connect Time:* Connect Time is the total time spent in the transaction connecting to the server. There may be multiple connections made during a transaction. c) *[HTTP Step Group] First Byte Time:* This metric measure the First Byte Time, which is the total time taken between the last byte of the request sent and the first byte of the response received by the server for all requests made. d) *[HTTP Step Group] First Byte Time per Page:* This is the First Byte Time divided by the number of pages in the step group. e) *[HTTP Step Group] HTML Time:* This metrics measures the HTML Time, which is the amount of time it takes to transfer the HTML coding of the page to the browser. f) *[HTTP Step Group] Non-HTML Time:* This is the amount of time it takes to transfer the non-HTML content such as images to the browser. g) *[HTTP Step Group] Perceived Slowest Page Time:* It is the amount of time that a web browser takes to play the slowest page in a step group. h) *[HTTP Step Group] Perceived Time per Page:* It is the average amount of time that a Web browser takes to play each page in the step group. i) *[HTTP Step Group] Perceived Total Time:* It is the amount of time a Web browser takes to play the step group. j) *[HTTP Step Group] Redirect Time:* It represents the total time of all redirects within a step group. The time taken to redirect the request affect the overall response time of the page. k) *[HTTP Step Group] Status:* It indicates whether the Web transaction was successful. l) *[HTTP Step Group] Status Description:* If the beacon is unable to run the transaction successfully, this metrics returns a description of the error that prevented the transaction from running. m) *[HTTP Step Group] Time per Connection:* This is the Connect Time divided by the number of connections made while playing a step group. n) *[HTTP Step Group] Total Time:* Indicates the overall time spent in processing the step group. o) *[HTTP Step Group] Transfer Rate (KB per second):* The transfer rate indicates how quickly data is being transferred from the Web server to the client browser. This is computed as: Total Kilobytes Received / Total Transaction Time. 3) HTTP Transaction: The lists the HTTP Transaction metrics, their descriptions, and user actions. a) *[HTTP Transaction] Connect Time:* Connect Time is the total time spent in the transaction connecting to the server. b) *[HTTP Transaction] First Byte Time:* This metric measure the First Byte Time, which is the total time taken between the last byte of the request sent and the first byte of the response received by the server for all requests made. c) *[HTTP Transaction] First Byte Time per Page:* This is the First Byte Time divided by the number of pages in the transaction. d) *[HTTP Transaction] HTML Time:* This metric measures the HTML Time, which is the amount of time it takes to transfer the HTML coding of the page to the browser. e) *[HTTP Transaction] Non-HTML Time:* This is the amount of time it takes to transfer the non-HTML content such as images to the browser. f) *[HTTP Transaction] Perceived Slowest Page Time:* This is the amount of time a web browser takes to play the slowest page in the transaction. g) *[HTTP Transaction] Perceived Time per Page:* This is the average amount of time a Web browser takes to play each page in the transaction. h) *[HTTP Transaction] Perceived Total Time:* This is the amount of time that a web browser takes to play the transaction. i) *[HTTP Transaction] Redirect Time:* Redirect time represents the total time of all redirects within a transaction. The time taken to redirect the request affect the overall response time of the page. j) *[HTTP Transaction] Status:* Indicates whether the Web transaction was successful. k) *[HTTP Transaction] Status Description:* If the beacon is unable to run the transaction successfully, this metric returns a description of the error that prevented the transaction from running. l) *[HTTP Transaction] Time per Connection:* This is the Connect Time divided by the number of connections made while playing a transaction. m) *[HTTP Transaction] Total Time:* It indicates the overall time spent to process the transaction. n) *[HTTP Transaction] Transfer Rate (KB per second):* The transfer rate indicates how quickly data is being transferred from the Web server to the client browser. This is computed as: Total Kilobytes Received / Total Transaction Time. 4) HTTP User Action: The following section lists the HTTP User Action metrics, their descriptions, and user actions. a) *[HTTP Step] Connect Time:* Connect Time is the total time spent in the transaction connecting to the server. b) *[HTTP Step] First Byte Time:* This metric measure the First Byte Time, which is the total time taken between the last byte of the request sent and the first byte of the response received by the server for all requests made. c) *[HTTP Step] First Byte Time per Page Element:* This is the First Byte Time divided by the number of step elements. d) *[HTTP Step] HTML Time:* This metric measures the HTML Time, which is the amount of time it takes to transfer the HTML coding of the page to the browser. e) *[HTTP Step] Non-HTML Time:* This is the amount of time it takes to transfer the non-HTML content such as images to the browser. f) *[HTTP Step] Perceived Slowest Page Element Time:* This is the amount of time that a Web browser takes to play the slowest step element. g) *[HTTP Step] Perceived Time per Page Element:* The average amount of time that it would take a Web browser to play each page in a step. h) *[HTTP Step] Perceived Total Time:* The amount of time that takes a Web browser to play the step element. i) *[HTTP Step] Redirect Time:* Redirect time represents the total time of all redirects within a step. The time taken to redirect the request affects the overall response time of the page. j) *[HTTP Step] Status:* Indicates whether the Web transaction was successful. k) *[HTTP Step] Status Description:* If the beacon is unable to run the transaction successfully, this metric returns a description of the error that prevented the transaction from running. l) *[HTTP Step] Time per Connection:* This is the Connect Time divided by the number of connections made while playing a step element. m) *[HTTP Step] Total Time:* Indicates the overall time spent in processing the step.

This includes all the phases of the transaction, including Connect Time, Redirect Time, First Byte Time, HTML Time, and Non-HTML Time. n) *[HTTP Step] Transfer Rate (KB per second):* The transfer rate indicates how quickly data is being transferred from the Web server to the client browser. This is computed as: Total Kilobytes Received / Total Transaction Time. o) *[HTTP Step] URL:* This is the URL associated with the step. 5) HTTP Raw: This lists the HTTP Raw metrics, their descriptions, and user actions. a) *HTTP Raw Connect Time:* This is the total time spent in the transaction connecting to the server. There may be multiple connections made during a transaction. b) *HTTP Raw First Byte Time:* This is the First Byte Time divided by the number of pages in the step, step group, or transaction. c) *HTTP Raw HTML Time :* This metric measures the HTML Time, which is the amount of time it takes to transfer the HTML coding of the page to the browser. d) *HTTP Raw Non-HTML Time:* This is the amount of time it takes to transfer the non-HTML content such as images to the browser. e) *HTTP Raw Perceived Slowest Page / Page Element Time:* The amount of time that it would take a web browser to play the slowest page in the step, step group, or transaction. f) *HTTP Raw Perceived Time per Page / Page Element:* The average amount of time that take a Web browser to play each page in the step, step group, or transaction. g) *HTTP Raw Perceived Total Time:* Indicates the overall time spent to process the step, step group, or transaction. This includes all the phases of the step / step group / transaction, including Connect Time, Redirect Time, First Byte Time, HTML Time, and Non-HTML Time. h) *HTTP Raw Redirect Time:* Redirect time represents the total time of all redirects within a transaction. The time taken to redirect the request can affect the overall response time of the page. i) *HTTP Raw Status:* Indicates whether the Web transaction was successful. j) *HTTP Raw Status Description:* If the beacon is unable to run the step, step group, or transaction successfully, this metric returns a description of the error that prevented the transaction from running. k) *HTTP Raw Time Per Connection:* This metric measures the average connect time for all pages in the transaction. This is calculated as: Total Connect Time / Number of Connections Made. l) *HTTP Raw Transfer Rate (KB per second):* The transfer rate indicates how quickly data is being transferred from the Web server to the client browser. This is computed as: Total Kilobytes Received / Total Transaction Time. m) *HTTP Raw Total Time:* Indicates the overall time spent to process the step, step group, or transaction. This includes all the phases of the transaction, including Connect Time, Redirect Time, First Byte Time, HTML Time, and Non-HTML Time. n) *HTTP Raw URL:* This is the URL in which scripting on the page enhances content navigation.

The web metrics can also be classified using the web quality model. Ramler et al. [RAM02] defined a cube structure in which three basic aspects when testing a web site. Another cube was also proposed by Ruiz [RUI03] which composed of the aspects taken for the evaluation of web site quality, features, life-cycle processes and quality aspects, which can be considered orthogonal. The model was reviewed by basing the features dimension on aspects relevant to the web found in the literature [CAL04]. Using this version of WQM, web metrics were classified.

**Architecture of Web Metrics**

The architecture of Web Metrics uses an intermediate abstraction layer to decouple language parsing from metrics extraction process.

A set of intuitive relations is generated that a separate analyzer uses as an input to compute metrics. Relations are described among language entities, such as classes, interfaces, methods and attributes. The metrics can be calculated directly by performing SQL queries.

The architecture consists of three main components (Figure below) 1) Parser: In this each parser contains a grammar parser, a symbol table, and supporting classes. Syntax of a particular language can be recognized by the grammar parser and is written in JavaCC. JavaCC is a tool that generates parsers source code in Java, given grammar as an input. It uses a top down LL(k) parsing algorithm [AHO86].

The symbol table and all supporting classes are written in Java. The Common Interface module is shared among all parsers and it provides a standard API to report relations on the database. 2) Database: It stores the relation set which represents the source code. The implementation is based on the open source DBMS MySQL 3) Analyzer: It calculates the metrics querying the relation set with the SQL language.

Each metric is a class which implements the Measure interface. This is implemented according to the Strategy design pattern [GAM95] by the way in which the analyzer calculates each metric.

New metrics can be developed by adding a new class implementing this interface. Metrics calculated are reported through an XML file.

This file can be transformed with XSLT into any kind of report. The XML file is transformed into a simple HTML page including the following metrics: CK Metrics Suite, LOC and McCabe Cyclomatic Complexity average per method.

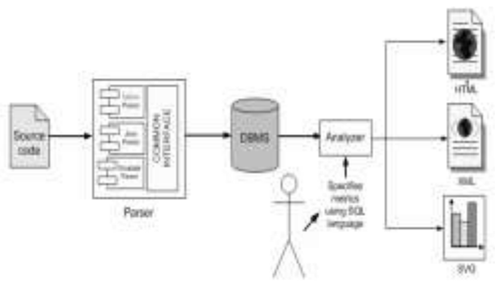

**Figure Architecture overview**

## Conclusion

In this paper the main concepts of software measurement is highlighted. Many metrics with different language paradigms have been invented in software industry as stated. Most of these have been defined and then tested only in a limited environment. Subsequent attempts should be carried out to test or use the metrics. The impact of implementing a web metric measurement tool on a website along the following two components should be established: programming code and website performance. While the former deals with the amount of programming effort it would take to integrate the code to run a web metric measurement tool with existing website code, the latter focuses on website performance issues that might arise from running a data collection tool on a website.

## References

1. [ALB83] Albrecht A.J. & J. Gaffney, *“Software function, source lines of code and development effort prediction”,* IEEE Trans. on Software Engg, SE:9, Vol. 6, 1983, pp 639:648.
2. [AHO86] Aho A.V., R. Sethi, J.D. Ullman *“Compilers: Principles, Techniques and Tools,”* Addison-Wesley, 1986.
3. [ABR94] Abreu F.B. & R. Carapuca, *“Candidate Metrics for Object Oriented Software within a Taxonomy Framework”,* Journal of Systems and Software, Vol. 26, No. 1, 1994.
4. [ABR95] Abreu F.B., Miguel Afonso Goulão & Rita Esteves : *“Toward the Design Quality Evaluation of Object-Oriented Software Systems”,* act as de 5th International Conference on Software Quality, Austin, Texas, EUA, Outubro, 1995, pp.44:57.
5. [ABR96] Abreu, Fernando B, Rita, E., Miguel, G. : *“The Design of Eiffel Program: Quantitative Evaluation Using the MOOD metrics”,* Proceeding of TOOLS’96 USA, Santa Barbara, California, July 1996
6. [ABR01] Abreu F.B.: *“Using OCL to Formalize Object-Oriented Metrics Definition*”, Report ES007/01 of the S/W Engg. Group, INESC, Portugal, 2001.
7. [BOY93] Boyd N., *“Building object–oriented frameworks”,* The Smalltalk report, Volume 3(1), 1993, pp 1:16.
8. [BAS96] Basili V., L.C. Braind, W.L. Melo, *“A Validation of Object-Oriented Design Metrics as Quality Indicators”,* IEEE Trans. on S/W Engg, SE:22, Vol. 10, 1996.
9. [BRI96] Briand L.C., John W. Daly, and Jurgen Wust: *“A Unified Framework for Coupling Measurement in Object-Oriented Systems.”* Fraunhofer Institute for Experimental Software Engineering. Kaiserslautern, Germany. 1996.
10. [BEL99] Bellin D., M. Tyagi, M. Tyler, *“Object Oriented Metrics: An Overview”*, Web Publication, 1999.
11. [BRI99] Briand L.C., John W. Daly, and Jurgen Wust. *“A unified framework for coupling measurement in object-oriented systems.”* IEEE Transactions on Software Engineering, 25(1), Jan./Feb. 1999, 91–121.
12. [CON86] Conte S., H. Dunsmore, and V.Shen, *"Software Engineering Metrics And Models, 1ST edition."*, Benjamin/Cummings, Menlo Park, CA. , 1986.
13. [CHI91] Chidamber S.R., Chris F. Kemerer, *“Towards A Metrics Suite For Object Oriented Design,”* OOPSLA’91, 1991, pp. 197:211.
14. [CHI93] Chidamber S.R., Chris F. Kemerer, *“A Metrics Suite For Object Oriented Design,”* M.I.T. Sloan School of Management, 1993.
15. [CHI94] Chidamber S.R. and C.F. Kamerer, “*A metrics Suite for Object-Oriented Design.”* IEEE Trans. S/W Engg, vol. SE:20, no.6, 1994, pp. 476:493.
16. [CAL04] Calero, C. , Ruiz, J., and Piattini, M. “ *A Web Metrics Survey Using WQM, “* Proceedings ICWE04, LNCS 3140, Springer: Verlag Heidelberg, July 2004, pp.147:160.
17. [DEM86] DeMarco, Tom, Controlling Software Projects, Yourdon Press, New York, 1986.
18. [DRE89] Dreger J.B., *“Function point analysis”,* Prentice-Hall, 1989.
19. [DAV97] Davis C. & J. Bansiya, *“Using QMOOD++ for object-oriented metrics”,* Dr. Dobb’s Journal, December 1997.
20. [DEM98] DeMarco, Tom and Boehm, Barry W. *“Controlling Software Projects: Management, Measurement, and Estimates”,* Prentice Hall PTR/Sun Microsystems Press, March 1998, pp. 80:91.
21. [FEN96] Fenton N. & S.L. Pfleeger, “*Software Metrics: A Rigorous and practical approach”*. International Thomson Computer Press, 1996.
22. [FEN97] Fenton N. & S.L. Pfleeger, *“Software Metrics: A Rigorous & Practical Approach”,* Second edition, 1997, International Thomson Computer Press.
23. [GAM95] Gamma E., R. Helm, R. Johnson, and J. Vlissides. *“Design Patterns: Elements of Reusable Object-Oriented Software”.* Addison-Wesley, 1995.
24. [HOU] Houari A. Sahraoui, Robert Godin, Thierry Miceli: “*Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation?”* http://www.iro.umontreal.ca/~sahraouh/papers/ICSM00.pdf
25. [HAL77] Halstead M.H., *"Elements of Software Science, 1st edition"*, Elsevier North-Holland, ISBN: 0444002057, 1977.
26. [HEN81] Henry S. & D. Kafura, *“Software structure metrics based on information flow”,* IEEE Trans. on S/W Engg., SE:7, Vol. 5, 1981, pp. 510:518.
27. [HAR98] Harrison R., S.J.Counsell, and R.V.Nithi, *“An Evaluation of MOOD set of Object Oriented Software Metrics”.* IEEE Trans. Software Engineering, vol. SE:24, no.6, June 1998, pp. 491:496.
28. [KOL93] Kolewe R., *“Metrics in Object-Oriented Design and Programming”,* Software Development, October 1993.
29. [LOR91] Lorenz M., *“Real world reuse”,* Journal of object–oriented programming. Volume 6, 1991.
30. [LOR94] Lorenz, Mark & Kidd Jeff: *“Object-Oriented Software Metrics”*, Prentice Hall, 1994.
31. [MYE74] Myers, G., Stevens, W., and Constantine, L., *"Structured Design,"* IBM Systems Journal, vol. 13, 1974, pp. 60:73.
32. [CAB76] McCabe. *A Complexity Measure.* IEEE Transactions on Software Engineering, Vol. Se:2, No. 4, December 1976, pp. 308:320
33. [MIL88] Mills, Everald E. *Software Metrics*, SEI Curriculum Module SEI: CM:12:1.1, Carnegie Mellon University.

34. [MOR88] Morris K.L., *"Metrics for Object-Oriented Software Development Environments",* Master thesis, M.I.T., 1988.
35. [MOL93] Moller, Handbuch der Informatik, R.Oldenbourg Verlag, "*A high level description of useful metrics and their use within the development process with startup guidelines and examples".*
36. [CAB94] McCabe & Associates, McCabe *"Object Oriented Tool User's Instructions"*, 1994.
37. [MIS03] Misra & Bhavsar: "*Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality."* Springer-Verlag 2003. [ROS97] Rosenberg, Linda H., *"Metrics for Object Oriented Environments",* EFAITP/AIE Third Annual Software Metrics Conference, December, 1997.
38. [RAM02] Ramler, R., Weippl, E., Winterer, M., Schwinger, W. and Altmann, J., *"A quality-driven approach to web testing",* Ibero-american Conference on Web Engineering, ICWE 2002, Santa Fe, Vol. 1, 2002, pp. 81:95.
39. [RUI03] Ruiz, J., Calero, C. and Piattini, M., *"A three-dimensional web quality model",* Proceedings of the International Conference on Web Engineering (ICWE 2003), LNCS 2722, 2003,pp. 384:5.
40. [SOM92] Sommerville, Ian, *"Software Engineering,"* Addison-Wesley Publishing Company, 1992.
41. [SHE95] Shepperd M., N. Churcher,: Comments on *"A metrics suite for object-oriented design''*. IEEE Trans on S/W Engineering 21, 1995, pp. 263:265.
42. [TAI84] Tai K., *"A Program Complexity Metric Based on Data Flow Information in Control Graphs,"* Proc. 7th Int'l Conf. Software Eng., IEEE Press, 1984, pp. 239:248.
43. [WES92] West M., *"An investigation of C++ metrics to improve C++ project estimation",* IBM internal paper, 1992.
44. [WHI97] Whitney R.: Course material. CS 696: "*Advanced OO*." Docs 6 & 8, Metrics. Spring Semester, 1997. San Diego State University.
45. http://www.eli.sdsu.edu/courses/spring97/cs696/notes/metrics/metrics.html,http://www.eli.sdsu.edu/courses/spring97/cs696/notes/metrics2/metrics2.html

**About the authors:**

Dr. Rakesh Kumar received his PhD in Computer Science and M.C.A from Kurukshetra University, Kurukshetra, Haryana. He is currently Reader at the Department of Computer Science & Applications, Kurukshetra University. His current research focuses on programming languages, information retrieval systems, software engineering, and Artificial Intelligence.

Ms. Gurvinder Kaur is Lecturer in Computer Science at Guru Nanak Khalsa Institute of Technology and Management Studies, Yamuna Nagar. She has done her postgraduates degrees in Master of Science (M. Sc.) in Information Technology, Master of Computer Applications (M.C.A) from the Maharishi Dayanand University Rohtak and M.Phil (Computer Science) from Madurai Kamaraj University. At present she is pursuing PhD in Computer Science from Kurukshetra University.