OBJECT-ORIENTED METRICS – A SURVEY

M. Xenos[*1,2], D. Stavrinoudis[*2], K. Zikouli[*3] and D. Christodoulakis[*2,3]

[*1] Hellenic Open University
16 Saxtouri Str & Agiou Andreou, GR 262 23 Patras GREECE
Tel. (+30 61) 361423, Fax (+30 61) 361410

[*2] Computer Technology Institute
61 Riga Feraiou Str, GR 262 21 Patras GREECE
Tel. (+30 61) 960336, Fax (+30 61) 997783

[*3] Patras University
Computer Engineering and Informatics Department,
Rio, GR 265 00 Patras GREECE
Tel. (+30 61) 997216

SUMMARY

This paper presents the results derived from our survey on metrics used in object–oriented environments. Our survey includes a small set of the most well known and commonly applied traditional software metrics which could be applied to object–oriented programming and a set of object–oriented metrics (i.e. those designed specifically for object–oriented programming). These metrics were evaluated using existing meta–metrics as well as meta–metrics derived from our studies, based mostly on the practitioner’s point of view, and emphasising applicability in three different programming environments: Object Pascal, C++ and Java. The results can be of great assistance to quality engineers in selecting the proper set of metrics for their software projects.

List of keywords: Software quality, software metrics, object–oriented programming, object–oriented metrics.

1. INTRODUCTION

Numerous software metrics related to software quality assurance have been proposed in the past and are still being proposed. Several books presenting such metrics exist, such as Fenton’s [1], Shepperd’s [2] and others. Although most of these metrics are applicable to all programming languages, some metrics apply to a specific set of programming languages. Among metrics of this kind, are those that have been proposed for object–oriented programming languages.

Nowadays, a quality engineer can choose from a large number of object–oriented metrics. The question posed is not the lack of metrics but the selection of those metrics which meet the specific needs of each software project. A quality engineer has to face the problem of selecting the appropriate set of metrics for his software measurements. A number of object–oriented metrics exploit the knowledge gained from metrics used in structured programming and adjust such measurements so as to satisfy the needs of object–oriented programming. On the other hand, other object–oriented metrics have been developed specifically for object–oriented programming and it would be pointless to apply them to structured programming.

All metrics used (metrics proposed for both structured programming that can be applied to object–oriented programming and pure object–oriented metrics) are briefly presented in section 2. In section 3, the meta–metrics used and the evaluation criteria are discussed, while in section 4, the results from the evaluation of the metrics are presented. Finally, in section 5, research suggestions and future work are presented.

2. THE METRICS

The metrics presented and evaluated in this paper are both 'pure' object–oriented metrics and metrics proposed for structural programming that could also be applied to object–oriented programming. Measurements using some of these metrics have been automated in our laboratory [3,4] in order to facilitate collection of measurement results. Many of the proposed metrics have not been included in this survey. The reason for this is that either they have very similar characteristics / slide variations to metrics included in this survey or that they re-apply a traditional metric to object–oriented programming.

The aim of this paper is not to mention all the existing metrics or fully present the mentioned metrics but to make the reader aware of their existence and offer references for further reading. The metrics are presented in the following sections: Section 2.1 presents a set of well-known (popular) traditional metrics that can be applied to object–oriented programming, while section 2.2 presents metrics designed especially for object–oriented programming. For facilitating the presentation we use a 3–letters code for each metric (e.g. AML for a metric named "average module length") followed by the metric name, a brief definition and its references.

2.1 Traditional metrics

The metrics presented hereinafter have been selected from the most well known software metrics that have been proposed and could be easily applied to object–oriented programming as well. They are sorted alphabetically according to their codes.

AML	(**A**verage **M**odule **L**ength) measures the average module size [5].
BAM	(**B**inding **A**mong **M**odules) measures data sharing among modules [6,7].
CCN	(**C**yclomatic **C**omplexity **N**umber) measures the number of decisions in the control graph [8].
CDF	(**C**ontrol flow complexity and **D**ata **F**low complexity) is a combine metric based on variable definitions and cross-references [9].
COC	(**C**onditions and **O**perations **C**ount) counts pairs of all conditions and loops within the operations [10].
COP	(**Co**mplexity **P**air) combines cyclomatic complexity with logic structure [11].
COR	(**Co**upling **R**elation) assigns a relation to every couple of modules according to the kind of coupling [12].
CRM	(**C**ohesion **R**atio **M**etrics) measure the number of modules having functional cohesion divided by the total number of modules [13,14,15].
DEC	(**De**cision **C**ount) offers a method to measure program complexity [16].
DSI	(**D**elivered **S**ource **I**nstructions) counts separate statements on the same physical line as distinct and ignores comment lines [17].
ERE	(**E**xtent of **Re**use) categorises a unit according the lever of reuse (modifications required) [18].
ESM	(**E**quivalent **S**ize **M**easure) measures the percentage of modifications on a reused module [17,19].

EST	(**E**xecutable **St**atements) counts separate statements on the same physical line as distinct and ignores comment lines, data declarations and headings [17].
FCO	(**F**unction **Co**unt) measures the number of functions and the source lines in every function [20,21].
FUP	(**Fu**nction **P**oints) measures the amount of functionality in a system [22].
GLM	(**Gl**obal **M**odularity) describes global modularity in terms of several specific views of modularity [23].
IFL	(**I**nformation **Fl**ow) measures the total level of information flow between individual modules and the rest of a system [24]
KNM	(**Kn**ot **M**easure) is the total number of crossing points on control flow lines [25].
LOC	(**L**ines **O**f **C**ode) measures the size of a module [1,16].
LVA	(**L**ive **Va**riables) deals with the period each variable is used [26].
MNP	(**M**inimum **N**umber of **P**aths) measures the minimum number of paths in a program and the reachability of any node [27].
MOR	(**Mor**phology metrics) measure morphological characteristics of a module, such as size, depth, width and edge-to-node ratio [15].
NLE	(**N**esting **Le**vels) measures the complexity as depth of nesting [28].
SSC	(composite metric of **S**oftware **S**cience and **C**yclomatic complexity) combines software science metrics with McCabe’s complexity measure [29].
SSM	(**S**oftware **S**cience **M**etrics) are a set of composite size metrics [30].
SWM	(**S**pecification **W**eight **M**etrics) measure the function primitives on a given data flow diagram [31].
TRI	(**Tr**ee **I**mpurity) determines how far a graph deviates from being a tree [32].
TRU	(**Tr**ansfer **U**sage) measures the logical structure of the program [33].

2.2 Object–Oriented Metrics

The metrics presented hereinafter have been selected from metrics proposed specifically for object–oriented measurements and cannot be applied to another programming style. This is a small fraction of the most well-know metrics analysed in our laboratory (due to the space limitations of this paper). The categories chosen to present the metrics are not defining a metrics classification but used simply to ease the presentation and sometimes a metric may fall in more than one category. The metrics presented are: class related metrics, method related metrics, inheritance metrics, metrics measure coupling and metrics measure general (system) software production characteristics. They are sorted alphabetically, according to the codes, as follows.

2.2.1 CLASS METRICS

AHF	(**A**ttribute **H**iding **F**actor) is the ratio of the sum of inherited attributes in all system classes under consideration to the total number of available classes attributes [34].
CCO	(**C**lass **Co**hesion) measures relations between classes [35].
CEC	(**C**lass **E**ntropy **C**omplexity) measures the complexity of classes based on their information content [36].
CLM	(**C**omment **L**ines per **M**ethod) measures the percentage of comments in methods [37].
DAM	(**D**ata **A**ccess **M**etric) is the ratio of the number of private attributes to the total number of attributes declared in the class [36].
FOC	(**F**unction **O**riented **C**ode) measures the percentage of non object–oriented code that is used in a program [37].
INP	(**In**ternal **P**rivacy) refers to the use of accessory functions even within a class [38].
LCM	(**L**ack of **C**ohesion between **M**ethods) indicates the level of cohesion between the methods [38].

MAA	(**M**easure of **A**ttribute **A**bstraction) is the ratio of the number of attributes inherited by a class to the total number of attributes in the class [36].
MFA	(**M**easure of **F**unctional **A**bstraction) is the ratio of the number of methods inherited by a class to the total number of methods accessible by members in the class [36].
MHF	(**M**ethod **H**iding **F**actor) is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration [39].
NAD	(**N**umber of **A**bstract **D**ata types) is the number of user-defined objects used as attributes in a class that are necessary to instantiate an object instance of the class [36].
NCM	(**N**umber of **C**lass **M**ethods in a class) measures the measures available in a class but not in its instances [37].
NIV	(**N**umber of **I**nstance **V**ariables in a class) measures relations of a class with other objects of the program [37].
NOA	(**N**umber **O**f **A**ncestors) is the total number of ancestors of a class [40].
NPA	(**N**umber of **P**ublic **A**ttributes) counts the number of attributes declared as public in a class [36].
NPM	(**N**umber of **P**arameters per **M**ethod) is the average number of parameters per method in a class [36].
NRA	(**N**umber of **R**eference **A**ttributes) counts the number of pointers and references used as attributes in a class [36].
PCM	(**P**ercentage of **C**ommented **M**ethods) is the percentage of commented methods [37].
PDA	(**P**ublic **Da**ta) counts the accesses of public and protected data of a class [41].
PMR	(Percent of **P**otential **M**ethod uses actually **R**eused) is the percentage of the actual method uses [34].
PPD	(**P**ercentage of **P**ublic **D**ata) is the percentage of the public data of a class [41].
RFC	(**R**esponse **F**or a **C**lass) is the number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class [42].
WCS	(**W**eighted **C**lass **S**ize) is the number of ancestors plus the total class method size [40].
WMC	(**W**eighted **M**ethods per **C**lass) is the sum of the weights of all the class methods [43].

2.2.2 METHOD METRICS

AMC	(**A**verage **M**ethod **C**omplexity) is the sum of the cyclomatic complexity of all methods divided by the total number of methods [34].
AMS	(**A**verage **M**ethod **S**ize) measures the average size of program methods [37].
MAG	(**MA**X V(**G**)) is the maximum cyclomatic complexity of the methods of one class [41].
MCX	(**M**ethod **C**omple**x**ity) relates complexity with the number of messages [37,44].

2.2.3 COUPLING METRICS

CBO	(**C**oupling **B**etween **O**bjects) counts the number of classes a class is coupled with [1].
CCP	(**C**lass **C**ou**p**ling) measures connections between classes based on the messages they exchange [45].
CFA	(**C**oupling **Fa**ctor) is the ratio of the maximum possible number of couplings in the system to the actual number of couplings not imputable to inheritance [39].

2.2.4 INHERITANCE METRICS

AIF	(**A**ttribute **I**nheritance **F**actor) is the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes for all classes [46].
DIT	(**D**epth of **I**nheritance **T**ree) measures the number of ancestors of a class [46,47].
FEF	(**F**actoring **Ef**fectiveness) is the number of unique methods divided by the total number of methods [34].

FIN	(**F**AN-**IN**) is the number of classes from which a class is derived and high values indicate excessive use of multiple inheritance [41].
HNL	(Class **H**ierarchy **N**esting **L**evel) measures the depth in hierarchy that every class is located [48].
MIF	(**M**ethod **I**nheritance **F**actor) is the ratio of the sum of the inherited methods in all classes to the total number of available methods for all classes [39].
MRE	(**M**ethod **Re**use metrics) indicate the level of methods reuse [49].
NMI	(**N**umber of **M**ethods **I**nherited) measures the number of methods a class inherits [37].
NMO	(**N**umber of **M**ethods **O**verridden) is the number of methods need to be re-declared by the inheriting class [37].
NOC	(**N**umber **O**f **C**hildren) is the total number of children of a class [50].
PFA	(**P**olymorphism **Fa**ctor) is the ratio of the actual number of possible different polymorphic situations of a class to the maximum number of possible distinct polymorphic situations for this class [51].
PMO	(Percent of **P**otential **M**ethod uses **O**verridden) is the percentage of the overridden methods [34].
RDB	(**R**atio between **D**epth and **B**readth) is the ratio between the depth and the width of the hierarchy of the classes [52].
RER	(**Re**use **R**atio) is the ratio of the number of superclasses divided by the total number of classes [50].
SIX	(**S**pecialisation **I**nde**x**) measures the type of specialisation [37].
SPR	(**Sp**ecialisation **R**atio) is the ratio of the number of subclasses divided by the number of superclasses [50].

2.2.5 SYSTEM METRICS

ADI	(**A**verage **D**epth of **I**nheritance) is computed by dividing the sum of nesting levels of all classes by the number of classes [36].
ANA	(**A**verage **N**umber of **A**ncestors) determines the average number of ancestors of all the classes [36].
APG	(**Ap**plication **G**ranularity) is the total number of objects divided by the total number of function points [34].
ASC	(**As**sociation **C**omplexity) measures the complexity of the association structure of a system [40].
CAN	(**Ca**tegory **N**aming) divides classes into semantically meaningful sets [38].
CRE	(Number of time a **C**lass is **Re**used) measures the references in a class and the number of the applications that reuse this class [37,49].
FDE	(**F**unctional **De**nsity) is the ratio of LOC to the function points [1].
NCT	(**N**umber of **C**lasses **T**hrown away) measures the number of times a class is rejected until it is finally accepted [37,53].
NOH	(**N**umber **O**f **H**ierarchies) is the number of distinct hierarchies of the system [40].
OLE	(**O**bject **L**ibrary **E**ffectiveness) is the ratio of the total number of object reuses divided by the total number of library objects [34].
PRC	(**P**roblem **R**eports per **C**lass) measures defect reports on this class [37].
PRO	(**P**ercent of **R**eused **O**bjects Modified) declares the percentage of the reused objects that have been modified [52].
SRE	(**S**ystem **Re**use) declares the percentage of the reuse of classes [49].

3. EVALUATION: GOALS AND METHOD

In this section, the object–oriented metrics previously presented are briefly evaluated. The goal of such evaluation is not to criticise such metrics or to compare them but to aid the practitioner in selecting an appropriate set of metrics suitable for his particular case. Although all the presented metrics can be off value during a specific object–oriented measurements program, many may not fit into a specific measurements program based on the particular user’s needs.

Meta-metrics have been used for the evaluation of metrics [16] since 1986. Desirable characteristic of software metrics have been proposed at a higher level of abstraction by Weyuker [54]. For our survey, 7 different meta-metrics that cover all aspects of the measurements procedure have been used. The letters in parenthesis after the meta-metric name is used to facilitate and shorten future reference in this paper to the corresponding meta-metrics.

- **Measurement scale** (**MS**). The values assigned to a metric could be on various scales. Such scales, according to Stevens [55], are: nominal, ordinal, interval, ratio and absolute. As expected, metrics on nominal or ordinal scale could not be used as easily as metrics on ratio or absolute scale.
- **Measurements independence** (**MI**). The ability of a metric to always offer the same result (measurement) for the same measured unit is important. Metrics that may have various interpretations from different users could not easily be used in surveys of historical data. For example, measurements of LOC need clarification on how lines of code have been measured but measurements of the number of separate classes need no further clarification and are similar to all measurement programs.
- **Automation** (**AU**). The effort required to automate a metric varies. Some metrics are relatively easy to implement (e.g. LOC), some more difficult (e.g. HNL) and some could not be implemented due to their nature (e.g. PMO).
- **Value of implementation** (**VI**). This examines if the result of the metrics is dependent on the implementation. If the metric is implementation independent, it means that we may have measurements in early stages which will hold regardless of the implementation language and programming style. On the other hand, if a metric is dependent on the implementation, it will offer us a way to evaluate the success of the implementation and to take corrective action where needed.
- **Monotonicity** (**MO**). This applies to a metric, if for two units embedded in a third unit the metric’s value for the third unit is always equal or greater than the sum of the metric’s values for the two smaller units.
- **Simplicity** (**SI**). This meta-metric examines how simply a metric is defined related to the simplicity of the metric’s definition, how easily this definition could be understood and facilitate actions in the development plan.
- **Accuracy** (**AC**). This examines if the metric actually measures what is supposed to be measured and how the metric is finally related to the abstract software characteristics or factors are to be measured.

The results of the evaluation of the metrics presented in section 2, using the aforementioned meta-metrics are presented in the following section.

4. EVALUATION RESULTS

In the tables of this section, the results from evaluating the metrics are illustrated. The table rows represent metrics referenced by their assigned 3-letter codes, while table columns represent meta-metrics referenced by their code letters.

Metric	MS	MI	AU	VI	MO	SI	AC
AHF	+	+	+	+	-	+	+
CCO	+	-	+	+	+	=	+
CEC	+	-	-	+	-	-	-
CLM	+	+	+	-	+	+	+
DAM	+	+	+	+	-	+	+
FOC	+	-	+	+	+	+	+
INP	+	+	=	+	+	+	+
LCM	+	+	=	+	+	=	+
MAA	+	+	+	+	-	+	+
MFA	+	+	+	+	-	+	+
MHF	+	+	+	+	-	+	+
NAD	+	-	+	+	+	+	+
NCM	+	+	+	+	+	+	+
NIV	+	+	+	+	+	+	+
NOA	+	+	+	+	+	+	+
NPA	+	+	+	+	+	+	+
NPM	+	+	+	+	+	+	+
NRA	+	+	+	+	+	=	-
PCM	+	+	+	-	+	+	+
PDA	+	+	-	+	-	+	+
PMR	+	+	-	+	-	=	+
PPD	+	+	+	+	-	+	+
RFC	+	+	+	+	+	+	+
WCS	+	+	+	+	+	=	-
WMC	+	-	+	+	+	+	+

Table 1. Class metrics

Examining measurement scale (MS) we use two symbols “+” and “-”. The “+” characterise metrics that offer results on absolute, ration and interval scale, “-” characterise metrics on nominal and ordinal scale. According to the measurements’ independence, (MI) we use “+” for metrics that are always measured in the same way and “-” for metrics that their data collection may vary according to each case. On the automation (AU) easiness, we use “+” for metrics automated easily, “=” for metrics that require significant effort to automate and “-” for metrics that cannot be automated.

Metric	MS	MI	AU	VI	MO	SI	AC
AMC	+	+	=	+	+	=	+
AMS	+	-	+	+	+	+	+
MAG	+	+	=	+	+	=	+
MCX	+	+	+	+	+	=	+

Table 2. Method metrics

Metric	MS	MI	AU	VI	MO	SI	AC
CBO	+	+	+	+	+	+	+
CCP	+	+	+	+	+	+	+
CFA	+	-	-	+	-	=	-

Table 3. Coupling metrics

For the value of implementation (VI), “+” denotes that the metric is dependent on the implementation while “-” that it is not. The existence “+” or lack “-” of monotonicity (MO) is presented in a similar manner. For the value of simplicity (SI) three symbols are used: “+” for very well defined metrics, “=” for fairly defined metrics and “-” for metrics that are difficult to understood, interpreted and related to external software characteristics. Finally, the symbols “+” and “-” are used respectively for accuracy (AC).

Metric	MS	MI	AU	VI	MO	SI	AC
AIF	+	+	+	+	-	+	+
DIT	+	+	=	+	-	+	+
FEF	+	-	+	+	-	+	+
FIN	+	-	+	+	+	+	+
HNL	+	+	=	+	-	+	+
MIF	+	+	+	+	-	+	+
MRE	+	+	+	+	-	+	-
NMI	+	+	+	+	+	+	+
NMO	+	+	+	+	+	+	+
NOC	+	+	=	+	+	+	+
PFA	+	-	-	+	-	-	-
PMO	+	-	-	+	-	-	-
RDB	+	+	=	+	-	+	+
RER	+	+	+	+	-	+	-
SIX	-	+	=	+	+	+	+
SPR	+	+	+	+	-	+	-

Table 4. Inheritance metrics

Metric	MS	MI	AU	VI	MO	SI	AC
ADI	+	+	=	+	-	+	+
ANA	+	-	=	+	-	+	+
APG	+	-	-	+	-	+	+
NCT	+	+	=	+	+	+	+
NOH	+	-	=	+	+	+	+
OLE	+	+	-	+	-	-	-

ASC	+	+	+	+	+	=	+
CAN	-	-	-	-	-	+	-
CRE	+	+	-	+	+	+	+
FDE	+	-	+	+	-	-	-
PRC	+	+	+	+	+	+	+
PRO	+	+	-	+	-	=	+
SRE	+	+	-	+	-	+	+

Table 5. System metrics

The reader must note that the results presented in this section may be used in order to aid the selection of the appropriate metrics for his case and not to assess the metrics. Some metrics appropriate for a case may not be suitable for another. We must also point out that, for each metrics, a set of boundaries (expected value limits in between them the metric corresponds to ‘proper’ software) have been proposed derived by our experiments. However, the presentation of such boundaries may only be presented in an extended report and not in a survey paper.

5. SUGGESTIONS AND FUTURE WORK

The above results can be used in order to determine when and how each of the above metrics can be used according to the quality characteristics a practitioner wants to emphasise. Moreover, various constrains can be set on the expected results of these measurements in order to improve the quality of a software program.

Future goals include the publication of an extended report (already available in Greek) that will include over 200 metrics and their suggested boundaries. Another future goal is the definition of a minimum set of metrics applicable to the majority of cases and particular sets of metrics applicable to specific (defined) programming cases and to develop a framework that facilitates measurements of all metrics (we currently have independent tools for some metrics).

REFERENCES

[1] N. Fenton & S.L. Pfleeger, “Software Metrics: A Rigorous & Practical Approach”, Second edition, 1997, International Thomson Computer Press.

[2] M.J. Shepperd & D. Ince, “Derivation and Validation of Software Metrics, Clarendon Press, Oxford, UK, 1993.

[3] M. Xenos & D. Christodoulakis, “An Applicable Methodology to Automate Software Quality Measurements”, IEEE Software Testing and Quality Assurance International Conference, New Delhi, 1994, pp. 121-125.

[4] M. Xenos, P. Thanos & D. Christodoulakis, “QSUP: A supporting environment for preserving quality standards”, Proceedings of the 5th International Conference on Software Quality, Dundee, 1996, pp 146-155.

[5] B.W. Boehm, J.R. Brown, J.R. Kaspar et al., “Characteristics of Software Quality”, TRW Series of Software Technology, 1978.

[6] V.R. Basili & A.J. Turner, “Iterative enhancement: a practical technique for software development”, IEEE Transactions on Software Engineering, SE-1, Volume 4, 1975, pp 390-396.

[7] S. Henry & D. Kafura, “Software structure metrics based on information flow”, IEEE Transactions on Software Engineering, SE-7, Volume 5, 1981, pp 510-518.

[8] T.J. McCabe, “A complexity measure”, IEEE Transactions on Software Engineering, SE-2, Volume 4, 1976, pp 308-320.

[9] E.I. Oviedo, “Control flow, data flow and program complexity”, Proceedings of the IEEE Computer Software and Applications Conference, 1980, pp 146-152.

[10] W.J. Hansen, "Measurement of program complexity by the pair (cyclomatic number, operation count)", ACM SIGPLAN Notices 13, Volume 3, 1978, pp 29-33.

[11] G.J. Mayers, "An extention to the cyclomatic measure of program complexity", ASM SIGPLAN Notices 12, Volume 10, 1977, 61-64.

[12] N.E. Fenton & A. Melton, "Deriving structurally based software measures", Journal of Systems and Software, 12, 1990, pp 177-187.

[13] J.M. Bieman & L.M. Ott, "Measuring functional cohesion", IEEE Transactions on Software Engineering, SE-20, Volume 8, 1994, pp 254-259.

[14] A. Macro & J. Buxton, "The Craft of Software Engineering", Addison-Wesley, 1987.

[15] E. Yourdon & L.L. Constantine, "Structured Design", Prentice Hall, 1979.

[16] S.D. Conte, H.E. Dunsmore & V.Y. Shen, "Software engineering metrics and models", Benjamin/Cummings, 1986, pp 62-70.

[17] B.W. Boehm, "Software engineering economics", Englewood Cliffs, Prentice-Hall, 1981.

[18] Software Productivity Consortium, "Software Measurement Guidebook", International Thomson Computer Press, 1995.

[19] S.M. Thebaut, "The saturation effect in large-scale software development: its inpact and control", Ph.D. Thesis, Purdue University, 1983.

[20] C.P. Smith, "A software science analysis of programming size" Proceedings of the ACM national Computer Conference, 1980, pp 179-185.

[21] S.N. Woodfield, V.Y. Shen & H.E. Dunsmore, "A study of several metrics for programming effort", The Journal of Systems and Software, Volume 2, 1981, pp 97-103.

[22] A.J. Albrecht & J. Gaffney, "Software function, source lines of code and development effort prediction", IEEE Transactions on Software Engineering, SE-9, Volume 6, 1983, pp 639-648.

[23] H.L. Hausen, "Yet another model of software quality and productivity", Measurement for Software Control and Assurance, Littlewood, Elsevier, 1989.

[24] S. Henry & D. Kafura, "Software structure metrics based on information flow", IEEE Transactions on Software Engineering, SE-7, Volume 5, 1981, pp 510-518.

[25] M.R. Woodward, M.A. Hennell, D. Hedley, "A measure of control flow complexity in program text", IEEE Transactions on Software Engineering, SE-5, Volume 1, 1979, pp 45-50.

[26] H.E. Dunsmore & J.D. Gannon, "Data referencing: an empirical investigation", IEEE Computer, Volume 12, 1979, pp 50-59.

[27] N.F. Schneidewind & H. Hoffmann, "An experiment in software error data collection and analysis", IEEE Transactions on Software Engineering, SE-5, Volume 3, 1979, pp 276-286.

[28] J.C. Zolnowski & D.B. Simmons, "Taking the measure of program complexity", Proceedings of the National Computer Conference, 1981, pp 329-336.

[29] A.L. Baker & S.H. Zweben, "A comparison of measures of control flow complexity", IEEE Transactions on Software Engineering, SE-6, Volume 6, 1980, pp 506-512.

[30] M.H. Halstead, "Elements of software science", Elsevier, 1977.

[31] T. DeMarco, "Controlling Software Projects", Yourdon Press, 1982.

[32] D.C. Ince & S. Hekmatpour, "An approach to automated software design based on product metrics", Software Engineering Journal, 3(2), 1988, pp 53-56.

[33] M.R. Woodward, M.A. Hennell & D. Hedley, "A measure of control flow complexity in program text", IEEE Transactions on Software Engineering, SE-5, Volume 1, 1979, pp 45-50.

[34] K.L. Morris, "Metrics for Object-Oriented Software Development Environments", Master thesis, M.I.T., 1988.

[35] R.S. Chidamber & C.F. Kemerer, "Towards a metrics suite for object–oriented design", Proceedings of the OOPSLA '91 Conference, 1991, pp 197-211.

[36] J. Bansiya & C. Davis, "Using QMOOD++ for object-oriented metrics", Dr. Dobb's Journal, December 1997.

[37] M. Lorenz & J. Kidd, “Object–Oriented Software Metrics”, Prentice Hall, 1994.

[38] S.R. Chidamber & C.F. Kemerer, "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, SE-20, Volume 6, 1994, pp 476-493.

[39] R. Harrison, “An Evaluation of the MOOD Set of Object-Oriented Software Metrics”, IEEE Transactions on Software Engineering, SE-24, Volume 6, 1998.

[40] R. Kolewe, “Metrics in Object-Oriented Design and Programming”, Software Development, October 1993.

[41] T.J. McCabe, L. A. Dreyer et al., “Testing an object-oriented application”, Journal of the Quality Assurance Institute, October 1994, pp 21-27.

[42] R. Sharble & S. Cohen, “The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods”, Software Engineering Notes, Volume 18, No 2, 1993, pp 60-73.

[43] McCabe & Associates, “McCabe Object-Oriented Tool User’s Instructions”, 1994.

[44] J.B. Dreger, “Function point analysis”, Prentice-Hall, 1989.

[45] N. Boyd, “Building object–oriented frameworks”, The Smalltalk report, Volume 3(1), 1993, pp 1-16.

[46] V.R. Basili, L.C. Briand, W.L. Melo, “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering, SE-22, Volume 10, 1996.

[47] T.K. Shih, C.M. Chung, C.C. Wang, W.C. Pai, “Decomposition of inheritance hierarchy DAGs for object-oriented software metrics”, Workshop on ECBS, 1997.

[48] M. Lorenz, “Real world reuse”, Journal of object–oriented programming. Volume 6, 1991.

[49] F. Abreu & R. Carapuca, “Candidate Metrics for Object Oriented Software within a Taxonomy Framework”, Journal of Systems and Software, Volume 26, No 1, 1994.

[50] L.H. Rosenberg & L.E. Hyatt, “Software Quality Metrics for Object-Oriented Environments”, Crosstalk Journal, April 1997.

[51] F. Abreu, “MOOD - Metrics for Object-Oriented Design”, OOPSLA’94 Workshop, 1994.

[52] D. Bellin, M. Tyagi, M. Tyler, “Object Oriented Metrics: An Overview”, Web Publication, 1999.

[53] M. West, “An investigation of C++ metrics to improve C++ project estimation”, IBM internal paper, 1992.

[54] E. Weyuker, “Evaluating software complexity measures”, IEEE Transactions on Software Engineering, Volume 3, 1988.

[55] S.S. Stevens, “On the theory of scales of measurement”, Science, Volume 103, 1947, pp 677-709.