Towards the Formalization of Software Measurement by Involving Network Theory

GORDANA RAKIĆ, ZORAN BUDIMAC, MILOŠ SAVIĆ, MIRJANA IVANOVIĆ, University of Novi Sad

Complex network theory is based on a graph theory and statistical analysis. Software network is a sub-class of complex network and is usually represented by directed graphs representing relationships between software entities. Software metrics is a (numerical) measure that reflects some property of a software product or its specification. The goal of this paper is to set relationship between particular software metrics and corresponding measures from complex networks theory, including software networks. That relationship will help in discovering potentially new and useful metrics that are based on complex networks. Furthermore, it will narrow the gap between two similar research areas that is often too big. The specific goal of this paper is to present the method how the relationships can be established.

Categories and Subject Descriptors: **D.2.8 [Software Engineering]:** Metrics - *Complexity measures*; *Product metrics*

General Terms: Measurement, Standardization

Additional Key Words and Phrases: Software networks, Software metrics

1. INTRODUCTION

Real-world complex networks are used to model real-world and evolving systems in order to better understand them [Albert and Barabási 2002, Boccaletti et al. 2006, Newman 2003, Newman 2010]. They are most naturally represented as undirected or directed graphs denoted as G = (*N, L*) or G(N, K) = (*N, L*) or G(N, K) or $G_{N, K}$ where:

- *N* is defined as $\{n_1, n_2,\ldots,n_N\}$ - a set of nodes (vertices, points);
- *L* is defined as $\{l_1, l_2,\ldots,l_K\}$ - a set of links (edges, lines), i.e., a set of pairs of elements of *N*;
- N is a total number of nodes;
- K is a total number of links.

Besides representation in a form of graph, the complex network theory also provides a set of techniques for statistical analysis.

Software networks are a sub-class of complex networks and they are directed graphs representing relationships between software entities, .e.g. packages, classes, modules, methods, procedures, etc. They can be observed as a static representation of software code and design, and be used in analysis of the quality of software development process and software product with particular application in a field of large-scale software systems.

In software networks it can differentiated between horizontal and vertical dimension. Horizontal dimension includes: low level (i.e., static call graph (SCG) /method collaboration network (MCN), function uses global variable (FUGV)), middle level (i.e., class/interface/module, etc. collaboration network, e.g. CCN - class collaboration network), and high level (i.e., package collaboration network). Vertical dimension usually reflects dependencies between software entities represented by hierarchy tree (e.g. package-class/interface-method/field).

This work was supported by the Serbian Ministry of Education, Science and Technological Development through project *Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support*, no. OI174023. Author's address: G. Rakić, Z. Budimac. M. Savić, M. Ivanović, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, email: {goca, zjb, svc, mira}@dmi.uns.ac.rs.

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

Software metric [Kan 2002] is defined as a function which maps some property of the software (e.g., whole software product, one piece of a software product, and specification of a software product) into a numerical value. The clear advantage of software networks is that they are based on well-understood, intuitive, and theoretically sound basis - graphs. The disadvantage is that the calculation of even the simplest measurements requires the construction of potentially huge graphs - see figure 1 for example [Savić et al. 2014]. The advantage of software metrics is that measurements can be often calculated in a simple way. The disadvantage is that they are based on various definitions and input data (e.g., source code, software architecture, graphs) and can often give different results for the same input data [Novak and Rakić 2010, Rakić and Budimac 2010]. For example, cyclomatic complexity [McCabe 1996], although precisely defined, can be calculated in at least two different ways: one by constructing the Control Flow Graph (CFG) and then finding linearly independent paths through it; and the second one: by counting decision points directly from the source code. For a characteristic example of an output of one software metrics tool see figure 2 [Bothe, 2012].

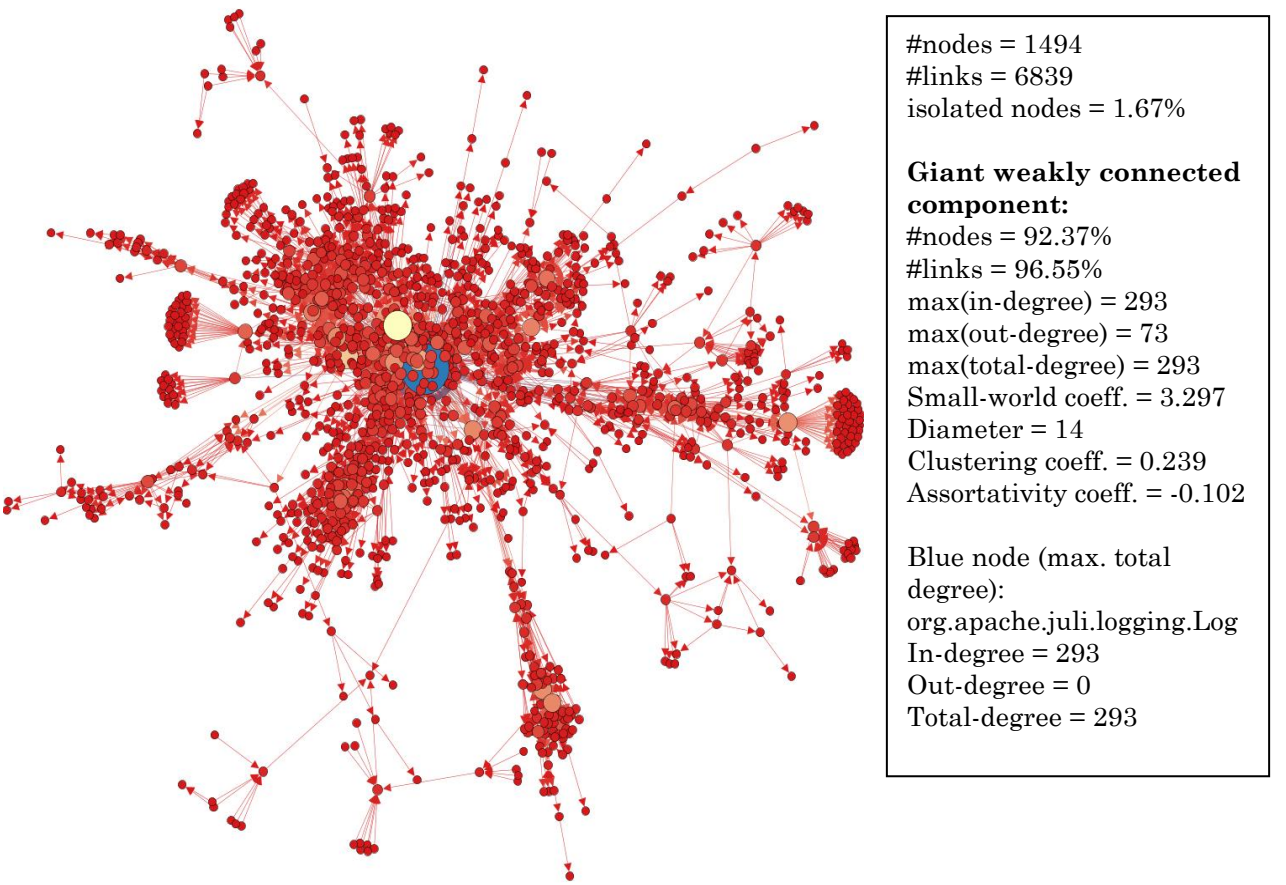
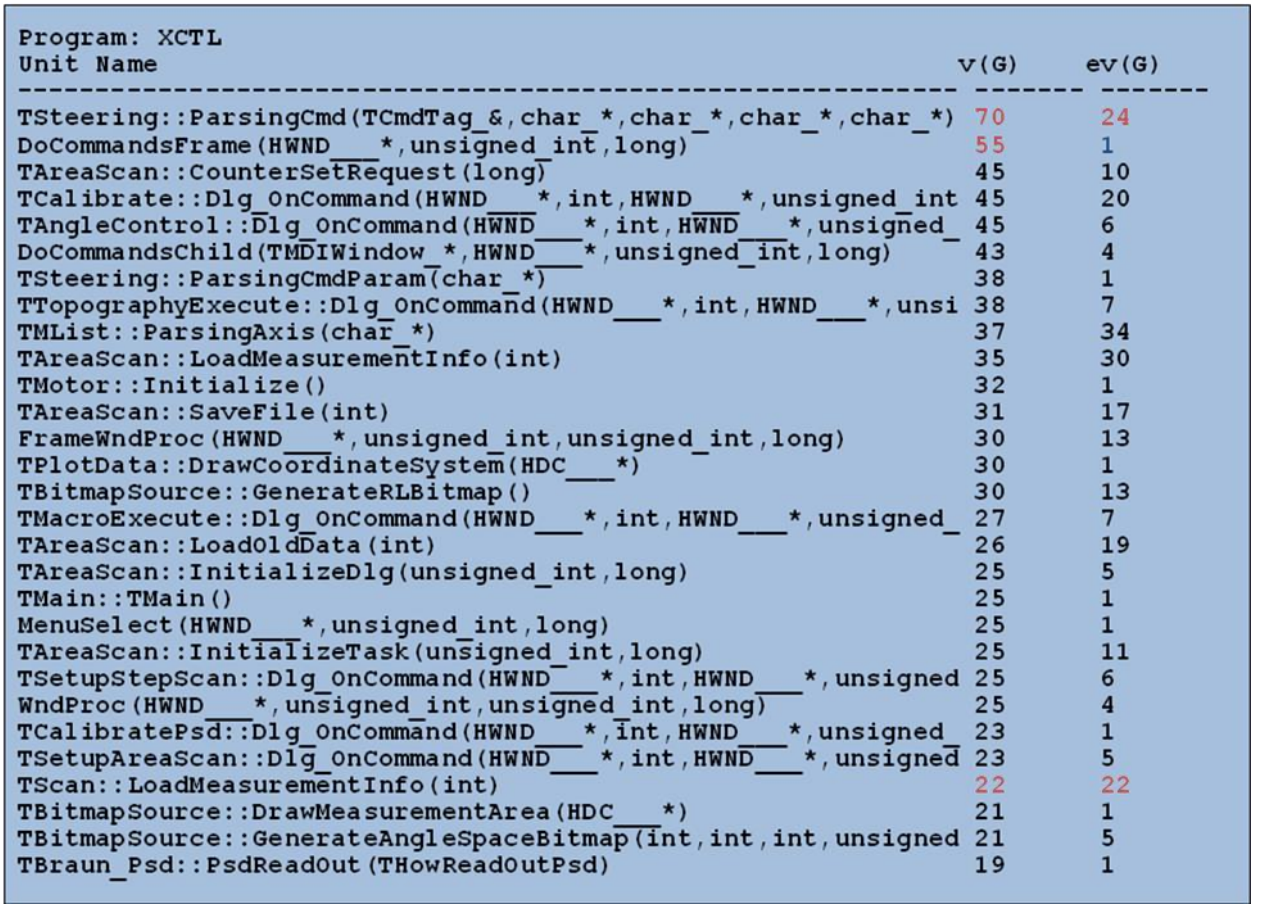


Fig. 1. Class collaboration network (CCN) (as part of a software network) for "tomcat-7.0.29 (java soft.)" with some numerical values.



```
Program: XCTL
Unit Name                                                    v(G)    ev(G)
------------------------------------------------------------ ------- -------
TSteering::ParsingCmd(TCmdTag_&,char_*,char_*,char_*,char_*) 70      24
DoCommandsFrame(HWND___*,unsigned_int,long)                  55      1
TAreaScan::CounterSetRequest(long)                           45      10
TCalibrate::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_int 45      20
TAngleControl::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 45      6
DoCommandsChild(TMDIWindow_*,HWND___*,unsigned_int,long)     43      4
TSteering::ParsingCmdParam(char_*)                           38      1
TTopographyExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsi 38      7
TMList::ParsingAxis(char_*)                                  37      34
TAreaScan::LoadMeasurementInfo(int)                          35      30
TMotor::Initialize()                                         32      1
TAreaScan::SaveFile(int)                                     31      17
FrameWndProc(HWND___*,unsigned_int,unsigned_int,long)        30      13
TPlotData::DrawCoordinateSystem(HDC___*)                     30      1
TBitmapSource::GenerateRLBitmap()                            30      13
TMacroExecute::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 27      7
TAreaScan::LoadOldData(int)                                  26      19
TAreaScan::InitializeDlg(unsigned_int,long)                  25      5
TMain::TMain()                                               25      1
MenuSelect(HWND___*,unsigned_int,long)                       25      1
TAreaScan::InitializeTask(unsigned_int,long)                 25      11
TSetupStepScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 25      6
WndProc(HWND___*,unsigned_int,unsigned_int,long)             25      4
TCalibratePsd::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned_ 23      1
TSetupAreaScan::Dlg_OnCommand(HWND___*,int,HWND___*,unsigned 23      5
TScan::LoadMeasurementInfo(int)                              22      22
TBitmapSource::DrawMeasurementArea(HDC___*)                  21      1
TBitmapSource::GenerateAngleSpaceBitmap(int,int,int,unsigned 21      5
TBraun_Psd::PsdReadOut(THowReadOutPsd)                       19      1
```

Fig. 2. Values for Cyclomatic complexity (v(g)) for a program in experimental physics.

The goal of this paper is to set a method that would map one kind of measurements to the other. This would help to bring together researchers from both fields to hopefully adopt the common measurements, common definitions, and the common language between two research directions. Furthermore, due to a stronger theoretical background of software networks, it would achieve stronger formalisation of "classical" software metrics. Final consequence would be a stronger formal support to software quality control in general.

The rest of the paper is organized as follows. The second section overviews some of the most used metrics to illustrate similarities and differences between them. The third section describes our method of unifying several sets of metrics and gives a short example. Fifth section introduces related work, while the sixth one concludes the paper.

2. BACKGROUND

This section provides an overview of the most used software measurements, divided in several categories.

2.1 Network analysis measures

Measures frequently used in analysis of complex networks are related to the connectivity, distance, centrality, and clustering of nodes [Albert and Barabási 2002; Boccaletti et al. 2006; Newman 2003, 2010].

2.1.1 Connectivity

The most basic topological characteristic of a node is its degree – the number of links incident to the node. In the case of directed network we can distinguish between the in-degree (the number of in-coming links, fan-in) and the out-degree (the number of out-going links, fan-out) of a node. The connectivity of nodes in a regular network can be described by one number – the average degree. For a non-regular network the connectivity of nodes can be expressed by the degree distribution which is the probability $P(k)$ that randomly selected node has degree equal to k.

2.1.2 Distance

The distance between two nodes is defined as the length of the shortest path connecting them. A majority of real-world networks possess the *small-world* property. Having the small-world property means that the average distance between nodes in a network is a small value, much smaller than the number of nodes in the network. The harmonic mean of distances between nodes in the network reflects the communication efficiency of the network. The largest distance among nodes in a network is also known as the diameter of the network.

2.1.3 Centrality

Centrality measures rank nodes in a network with respect to their topological importance. The basic metrics of node importance are betweenness centrality, closeness centrality and eigenvector centrality. The betweenness centrality of a node is the extent to which the node is located on the shortest path connecting two arbitrary selected nodes. If the node is positioned on a large number of shortest paths then it has a vital role to the overall connectivity of the network. Consequently, nodes having high betweeness centrality are in position to maintain and control the spread of information across the network.

2.1.4 Clustering

A community or cluster in a network is a subset of nodes that are more densely connected among themselves than with the rest of the network. The quality of a partition of a network into communities is usually quantified by the Girvan-Newman modularity measure. The modularity measure accumulates the difference between the number of links in a community with the expected number of links among nodes constituting the community in a random network with the same degree distribution.

2.2 Software Metrics

Software metric can be defined as numerical value which reflects some property of a software development processes and software products [N. Fenton, 1996]. The mostly used categories of metrics are size, complexity and structure metrics, while structure metrics mainly reflect important aspects of product design. Classical software metrics are the oldest ones and the mostly used. They reflect size and complexity of source code. Structure is reflected by design metrics. This area of software metrics has expanded with development of object-oriented approach when different set of design metrics were introduce. Each of these sets usually contains also some newly introduced size and complexity metrics but they are always derived from some classical metric.

2.2.1 Classical size and complexity metrics

There are three mostly used classical software metrics:

- *Lines of Code (LOC*) family counts the lines of a program with or without comments, empty lines, etc.
- *Cyclomatic Complexity (CC)* counts linearly independent paths through a program
- *Halstead Metrics (H)* map complexity of a program to a number of operands and operators in it.

2.2.2. Design Metrics & Object Oriented Metrics

There are 4 generally accepted families of design and object oriented metrics. These families contain many intersections and there are still no universally accepted set of designing and object-oriented metrics.

- *Lorenz Metrics* [M. Lorenz, 1994] consist of :
  - *Average Method Size - AMS* based on LOC of every method.
  - *Average Number of Methods per Class – ANMC*
  - *Average Number of Instance Variables per Class - ANIVC*
  - *Class Hierarchy Nesting Level*
  - *Number of Subsystem-to-Subsystem Relationships - NSSR* is more general than CBO (see below)
  - *Number of Class-to-Class Relationships in Each Subsystem - NCCR* is analogous to CBO
  - *Instance Variable Usage - IVU*
  - *Average Number of Comment Lines (per Method) - ANM* based on LOC family of metrics
  - *Number of Problem Reports per Class*
  - *Number of Times Class is Reused*
  - *Number of Classes and Methods Thrown Away*

- *Morris metrics* [K. Morris, 1989] consist of :
  - *Methods per Class*
  - *Inheritance Dependencies*
  - *Degree of Coupling Between Objects* defined as total number of links / total number of objects
  - *Degree of Cohesion of Objects* = total number of incoming links / total number of objects
  - *Object Library Effectiveness* = total number of reusing object / total number of objects
  - *Factoring Effectiveness* = number of unique methods / total number of methods.
  - *Degree of Reuse of Inheritance Methods* is expressed as the percentage of really reused methods with respect to potentially reusable methods.
  - *Average Method Complexity* is based on CC metrics
  - *Application Granularity* also uses function point method for establishing the cost of a software project.

- *C&K metrics* [S. Chidamber and C. Kemerer. 1994] consist of
  - *Weighted Methods per Class (WMC)* is a sum of CC metrics for all methods of a class.
  - *Depth of Inheritance Tree (DIT)* is analogous to the *Class Hierarchy Nesting Level* (see above).
  - *Number of Children (NOC)* is the number of immediate sub-classes of a class.
  - *Coupling Between Object Classes (CBO)* is the number of other classes with which a class under consideration is connected.
  - *Response for a Class (RFC)* is a number of methods that can be called by the object of class under investigation.
  - *Lack of Cohesion in Methods (LCOM)* is the relationship between the number of methods in a class and the usage of variables in those methods.

- *MOOD metrics (Metrics for object oriented design)* [F. B. Abreu, 1995] consist of:
  - *Method Hiding Factor (MHF)* representing encapsulation
  - *Attribute Hiding Factor (AHF)* representing encapsulation
  - *Method Inheritance Factor (MIF)* representing inheritance
  - *Attribute Inheritance Factor (AIF)* representing inheritance
  - *Polymorphism Factor (PF)* representing polymorphism
  - *Coupling Factor (COF)* representing coupling

3. DEMONSTRATION OF THE MAPPING METHOD

We informally describe the method of uniting and harmonizing metrics from the two world in the following way:

a) Take one complex network measure / software metrics measure,

b) consider its possible meaning in software metrics measurements / complex network measurements,
c) define relation,
d) test relation if necessary.

For example,
a) we can observe the node degree in a complex network (or in any directed graph), with an original meaning that an in-degree (fan-in) represents the number of incoming links into a node, while an out-degree (fan-out) represents the number of outgoing links from the node.
b) semantics in the "software world" (e.g. dependency graphs or software architecture) can be easily found in (software) class collaboration network. If a class *C* is represented by a node then class references (in and out) are highly related to in- and out-degree in a complex network.
c) Then we can set the following relations.

$$In(C) = (A \in In(C) \text{ iff } A \rightarrow C) \quad (1)$$
$$Out(C) = (A \in Out(C) \text{ iff } C \rightarrow A) \quad (2)$$
$$Fan\text{-}in(C) = |In(C)| \quad (3)$$
$$Fan\text{-}out(C) = |Out(C)| \quad (4)$$
$$CBO(C) = Fan\text{-}in(C) + Fan\text{-}out(C) - |In(C) \cap Out(C)| \text{ or} \quad (5)$$
$$CBO(C) = Fan\text{-}in(C) + Fan\text{-}out(C) \text{ iff } (In(C) \cap Out(C)) = \varnothing \quad (6)$$

where: $a \rightarrow b$ denotes that there is a link from a to b, *In(C)* is the set of nodes that references *C*, *Out(C)* is the set of nodes referenced by *C*) and CBO(C) is "Coupling Between Object Classes" metrics from C&K set of metrics.

d) Relations that are presented in this paper are formally provable (or understandable from their basic explanations), therefore the (statistical) testing on these relations is not necessary. Otherwise, a statistical proof should be provided by using any tools available, but the SSQSA framework (Set of Software Quality Static Analyzers) [G. Rakić et al., 2013] would provide the most reliable results because it is based on a common language- and metrics-independent internal structure. Furthermore, SSQSA already implements many of described metric algorithms, therefore it is easy to run most of the needed statistical test.

4. RELATED WORK

In [M. Lanza and R. Marinsku, 2006] a numerous observations using classical and object-oriented metrics are given. However, no relations to complex (software) networks are given. [L. Šubelj and M. Bajec, 2012] discussed how some of the features and measures in general complex networks can be used to reason about the quality of software. For example, the nodes (software entities) with high betweenness are influential and an extra care should be paid to their maintenance. Similarly software entities with high *Fan-in* values are often used and are thus candidates for reuse, while those with high *Fan-in* are complex ones. However, no relationships between those and other (classical and object-oriented) metrics were discussed.

5. CONSLUSION AND FURTHER WORK

Our goal is to join two researches in the field of general software measurements: "classical" software metrics (incl. object-oriented one) and complex (software) networks. Many metrics in both fields are analogous to each other, some are the same (but with different names and different description mechanisms) and some are (at the first sight) unrelated (figure 3).

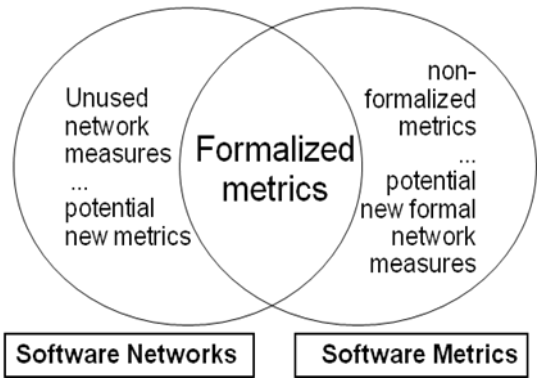


Fig. 3. The relationship between two kinds of software measurements.

By using the method that we proposed in this paper we hope to establish a solid and formalized intersection of both metrics. In doing so we are based on (software) networks that already have a formal background, while "classical" metrics are often described textually. Additional contribution may be achieved after mapping of all possible measures. The rest of measure may potentially be defined as new software metric derived from network measure, or wise versa.

REFERENCES

F. B. Abreu. 1995. Design Quality Metrics for Object-Oriented Software Systems. ERCIM News No.23 - October 1995 - INESC. http://www.ercim.org/publication/Ercim_News/enw23/abreu.html

R. Albert and A.-L. Barabási. 2002. Statistical mechanics of complex networks, *Rev. Mod. Phys.* 74 (1), pp. 47–97, DOI:http://dx.doi.org/10.1103/RevModPhys.74.47.

S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D. Hwang. 2006. Complex networks: structure and dynamics, *Phys. Rep.* 424 (45), pp. 175–308, DOI:http://dx.doi.org/10.1016/j.physrep.2005.10.009.

K. Bothe. 2012. Lecture notes for Software engineering course, Humboldt University Berlin.

S. Chidamber and C. Kemerer. 1994. A Metrics Suite for Object Oriented Design, *IEEE Trans. Software Eng.*, Vol 20, no 6, June, pp. 476-493.

N. Fenton, S. L. P. (1996). Software Metrics: A Rigorous and Practical Approach. Thomson Computer Press

M. Lanza and R. Marinescu. 2006. *Object-oriented metrics in practice*, Springer, Berlin, ISBN10 3-540-24429-8.

M. Lorenz. 1994. *Object-oriented software metrics: a practical guide*, Prentice Hall, USA, ISBN:0-13-179292-X.

S. H. Kan. 2002. *Metrics and models in software quality engineering*, Addison-Wesley Longman Publishing Co., Inc., 2002.

J. McCabe. 1976. A Complexity Measure, *IEEE Transactions on Software Engineering*, pp. 308–320.

K. Morris. 1989. *Metrics for object-oriented software development environments*, Sloan School of Management, MIT, Boston, USA, 135 pages.

M.E.J. Newman. 2003. The structure and function of complex networks, *SIAM Rev.* 45, pp. 167–256, DOI:http://dx.doi.org/10.1137/S003614450342480.

M. E. J. Newman. 2010. *Networks: An Introduction*, Oxford University Press Inc., New York, NY, USA, 2010.

J. Novak and G. Rakić. 2010. Comparison of software metrics tools for .NET. In *Proc. of 13th International Multiconference Information Society-IS*, Vol A, pages 231–234, 2010.

G. Rakić and Z. Budimac. 2010. Problems in systematic application of software metrics and possible solution. In *Proc. of The 5th International Conference on Information Technology (ICIT)*, Amman, Jordan, 2010.

G. Rakić, Z. Budimac, Z., and M. Savić, 2013. Language independent framework for static code analysis. In *Proc. of the Balkan Conference in Informatics* 2013, BCI '13, Thessaloniki, Greece, September 19-21, 2013, pp. 236–243.

M. Savić, G. Rakić, Z. Budimac, M. Ivanović. 2014. A language-independent approach to the extraction of dependencies between source code entities, *Information and Software Technology* 56, Elsevier, DOI:10.1016/j.infsof.2014.04.011, pp.1268-1288.

L. Šubelj and M. Bajec. 2012. Software systems through complex networks science: Review, analysis and applications, In Proc. of First International Workshop on Software Mining, ACM, pp. 9-16.