AN ANALYSIS ON COMPLEXITY MEASUREMENTS

Agung Fatwanto[1]

Abstrak

Upaya untuk menemukan dan mengeliminasi kesalahan akan sangat berguna bagi proses pengembangan software. Langkah preventif ini diharapkan dapat menurunkan kerusakan potensial pada sistem software yang dikembangkan. Tingkat kompleksitas dipercaya memiliki kaitan dengan kesalahan dalam pengembangan software. Beberapa studi telah dilakukan untuk mengetahui korelasi antara tingkat kesalahan dengan kerusakan software. Tulisan ini berusaha menganalisis beberapa metrik kompleksitas untuk mengetahui nilai guna metrik. Sebuah klasifikasi untuk metrik kompleksitas dan kriteria yang dapat dipakai sebagai meta-metrik didefinisikan dan diterapkan sebagai alat analisis. Juga telah dilakukan evaluasi pada salah satu metrik kompleksitas yang paling memenuhi kualifikasi.

Keywords: complexity, metrics, measurements, software, defects.

A. Introduction

Minimizing defects rate is one of the main goal on managing software system development. As size of the project increase, it is believed that the software being build become more complex. Some preliminary studies noted that software complexity have significant correlation with errors and defects rate.

[1] Dosen Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta

Research on complexity has been done for several years. The main point of those studies were tried to find any correlation between some complexity attribute and the resulting defects rate. There are three types of works on the measurements of complexity: (i) Finding the relation between size and complexity. Generally, these types of works try to utilize size of program, such as Line of Code (LOC) and Function Point (FP) as a predictor of defects. Some research were aiming to proof the conventional thinking that larger program size would impact to the higher complexity, and finally result on the greater of defects rate. Surprisingly the result is not as what have been commonly thinking. On some works which have been done in FORTRAN,[2] Pascal, PL/S, and Assembly language[3], found that size of program modules have negative relationship with complexity. Another study on Ada[4] and data collected from the development of AS/400[5], shown that LOC has a curvilinear relation with defects rate. The explanation of these phenomena could be described in this manner: as the size of program modules become lower, it could be more modules need to build the system. Therefore, more interface need to be acknowledged. Since the interface complexity levels are independent over the module size, small module would have to deal with same level of interface complexity as larger module. And since smaller modules have to face with more interface, so smaller modules mean higher interface complexity. On the other way, if the size of module becomes larger, it will be more parts of the internal module need to be handled, therefore the complexity is rise. It is explain the curvilinear relationship between LOC and defects. The previous study by Withrow found that modules with around 250 LOC have the lowest defects rate.[6] (ii) Analyzing the mechanism of program. This type of study analyzes the internal mechanism of a program within module. There are two well known works of this type. The first work was study of

[2] V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation", in *Communication of the ACM*, 1984, p.42-52.

[3] V. Shen, T. Yu, and L. Paulsen, "Identifying Error-prone Software – An Empirical Study", in *IEEE Transaction on Software Engineering*, 1985, p. 317-324.

[4] C. Withrow, "Error Density and Size in Ada Software", in *Software Engineering Process Group*, 1990, p.26-30.

[5] S. H. Kan, *Metrics and Models in Software Quality Engineering 2nd Edition.*, (Boston, MA: Addison-Wesley, 2003).

[6] Ibid p.26-30.

cyclomatic complexity, which later known as McCabe cyclomatic complexity.[7] This study was trying to relate the decision structure of a program and complexity. On research conducted by Craddock, it is known that cyclomatic complexity is a better predictor of defects than LOC at the low-level design and code inspections[8]. The second work was study on the vocabulary of a program, which named Halstead's software science.[9] Computation of complexity on this study relies on counting the number of operator and operand which build the program. Another study of this type was knot count.[10] Knot count measures the number of crossing edge from the control flow operator to its destination which can not be avoided. There also a study on entropy-based software complexity.[11] This research shows that complexity of a program is inversely proportional with its operator's information content. (iii) Viewing the structure of a software system to find out the whole system complexity. According to this, software system complexity depends on the structure of modules which build up the system and the interaction between them. A software system could always be seen as a building block which made up from a lot of materials and constructed in a somewhat architectural scheme. Analysis of the raw materials and the architectural design may give a good view on the complicatedness of that building. Analogically, this complexity analyzes the architectural complexity of a building. To mention some studies on this type of works are: invocation complexity,[12] stability measurements,[13] data flow complexity,[14] information flow metrics,[15] system partitioning measure,[16]

---

[7] T. J. McCabe, "A Complexity Measure", in *IEEE Transaction on Software Engineering*, 1976, p. 308-320.

[8] L. L. Craddock, "Analyzing Cost-of-Quality, Complexity, and Defect Metrics for Software Inspections", in *Technical Report TR07.844 IBM Rochester*, 1987.

[9] M. H. Halstead, *Elements of Software Science*, (New York: Elsevier North Holland, 1977).

[10] M. R. Woodward, M. A. Hennel, and D. Hedley, "A Measure of Control Flow Complexity in Program Text", in *IEEE Transaction on Software Engineering*, 1979.

[11] W. Harrison, "An Entropy-Based Measure of Software Complexity", in *IEEE Transaction on Software Engineering*, 1992.

[12] C. L. McClure, "A Model for Program Complexity Analysis", in *Proceeding IEEE Third International Conference on Software Engineering*, 1978, p.149-157.

[13] S. S. Yau, and J. S. Collofello, "Some Stability Measures for Software Maintenance", in *IEEE Transaction on Software Engineering*, 1979, p.545-552.

[14] E. I. Oviedo, "Control Flow, Data Flow and Program Complexity", in *Proceeding IEEE COMPSAC*, 1980, p.146-152.

[15] S. M. Henry, and D. Kafura, "Software Structure Metrics Based on Information Flow", in *IEEE Transaction on Software Engineering*, 1981, p.510-518.

structured designs quality,[17] hybrid information flow metrics,[18] system complexity model,[19] object-oriented metrics,[20] object-oriented metrics suite,[21] metric for object-oriented design,[22] and functional complexity.[23] This type of complexity measurements are categorized as structured design complexity, and become the scope of this paper. As a constraint, this paper will give a focus in analyzing structured design complexity on metric which most aligned with some criteria describe in section C.

B. A Brief Description of Structured Design Complexity

According to *Oxford Advance Learner's Dictionary*, complexity means "state of being complex", and complex means "difficult to understand or explain because there are many different parts".[24] Meanwhile, IEEE defines software complexity as "the degree to which a system or component has a design or implementation that is difficult to understand and verify"[25]. Measurements on structured design complexity realize that each module on a software system has to be treated as a constructor of a system, so the complexity of a whole system will depend on the complexity of each module and the interaction between them.

---

[16] L. A. Belady, and C. J. Evangelisti, "System Partitioning and It's Measure", in *Journal of System and Software*, 1981, p.23-39.

[17] D. A. Troy, and S. H. Zweben, "Measuring the Quality of Structured Design", in *Journal of System and Software*, 1981, p.113-120.

[18] S. M. Henry, and C. Selig, "Predicting Source-Code Complexity at the Design State", in *IEEE Software*, 1990, p.36-44.

[19] D. N. Card, and W. W. Agresti, "Measuring Software Design Complexity", in *Journal of System and Software*, 1988, p.185-197.

[20] M. Lorenz, and J. Kidd, *Object-Oriented Software Metrics*, (USA: Prentice-Hall, 1994).

[21] S. R. Chidamber, S. R., and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", in *IEEE Transaction on Software Engineering*, 1994, p.476-493.

[22] F. Abreu, "MOOD – Metrics for Object Oriented Design", in OOPSLA'94 Workshop, 1994.

[23] D. A. Tran-Cao, A. Abran, and G. Levesque, "Functional Complexity Measurement", in *Proceeding International Workshop on Software Measurement*, 2001, p.173-181.

[24] *Oxford Advance Learner's Dictionary*, 1991.

[25] *IEEE Computer Society: IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990.

C. Some Criteria on Software Metrics

Kafura and Henry recommend four criteria for a practical and powerful software metrics system:[26] (i) metrics have to be applicable for a large-scale system. On handling such kind of system, metrics should be automatic for speed and accuracy. The metrics also have to be applicable at the design phase in order to minimize the evaluation cost. (ii) metrics must considering change in the structure of the system, so it should be complete and sensitive for that change. Any alteration and their magnitude which important for the system should be reflected. It also should sensitive to the implicit of data connection on the system. (iii) metrics have to be validated on real large-scale system. Metrics which only verified on small-scale projects would be unreliable in dynamic nature. (iv) metrics must be robust. Metrics measurement should applicable to all class of system structures, such as single module complexity, interface properties, and should be able to signaling where part of the system which need to be modify. It should also be interpretable, so it can shows causes and solution for structural complexity problems.

Xenos and his colleagues offer seven meta-metrics for the software metrics evaluation.[27] (i) Measurement Scale. It classify metrics in term of different values on various measurements scale which defined by Steven.[28] It is nominal, ordinal, interval, ratio, and absolute. (ii) Measurements Independence. This criteria categorize metrics based on its outcome when applied at the equal measurement unit. The measurements result could be similar or different. It also examined the level of interpretation ambiguity of such metrics. (iii) Automation. This meta-metric measuring the degree of effort to make a metric become automatic. (iv) Value of Implementation. It observed the independency of a metric over implementation. An independent metric could be use at early stage of development. On the other hand, a dependent metrics is able to assess the result of the implementation, so it can explain the success and unsuccessful factors and how to

[26] D. Kafura, and S. M. Henry, "Software Quality Metrics Based on Interconnectivity", in *Journal of System and Software*, 1981, p.121-131.

[27] M. Xenos, D. Stavrinoudis, K. Zikouli, and D. Christodoulakis, "Object-Oriented Metrics – A Survey", in *Proceeding of the FESMA 2000*, 2000.

[28] S. S. Stevens, "On the Theory of Scales of Measurement", in *Science*, 1947, p.677-709.

address it. (v) Monotonicity. A metric will have good level of monotonicity if its value is always equal or greater than the total value of its constituent metrics which assembling it. (vi) Simplicity. It look at the level of simplicity of a metric regard to their definition, understandable, and its supporting for future plan. (vii) Accuracy. These criteria observe the relation of a metric to its characteristics or factors being measured.

Getting idea from those two metrics criteria and considering the current technology and practice in software development, it would be important to propose a new criterion for software metrics: (i) Predictive. It could be implemented at the early stage of development time. One intention of using complexity metrics is to estimate the complexity of a software design. Minimizing errors as early as possible is less expensive rather than fixing the defects in later stage. A good prediction of complexity at design phase would help software engineer to reduce the potential defects rate of the system being developed. (ii) Guidance. *Ability in guiding the software designer to simplify the complexity of* software system. A system will absolutely have inherent complexity which embedded on them. But a poor design would increase the level of complexity. Good complexity metrics will be able to show which part of module need to be re-engineered in order to gain simplest complexity. (iii) Comprehensive. Computing complexity considering to the whole system. An effort to simplify complexity on module basis, can be justifiable by metrics which just considering the modular complexity. Unfortunately, simplifying modular complexity will be meaningless in regard to the complexity of the whole system. This could be happen because a simplifying effort to a single module would lead to the rising of complexity level of other parts of the system. Additionally, complexity of whole system depends on the interaction between their parts. (iv) Computable. Could be implemented easily, accurately, fast, and engaging little effort. All of those criteria will be achieved easily by any metrics for small-scale software project. For a large-scale project, it absolutely need automation process in order to align with the above criteria. It have to be mathematically proofed that such kind of metrics will be computable.

D. Assessing Complexity Metrics

Using the last propose criteria, it would be able to evaluate the above complexity metrics in regard to their usefulness in the current practices. Apply with the first criteria, which is predictive, it is clearly seen that the first two type of complexity metrics are not fulfilling. Both size and internal mechanism metrics have could only be assessed after the coding phase. Different with the previous metric, the third type of metrics is used at design phase. These metrics calculate the complexity of software system at higher level than the first two ones. It could be used as predictors of complexity and later errors and defects rate for the next development phase. From this point, it just structured design complexity that satisfying the predictive criteria. Employing the second criteria, it is known that size metrics does not meet with it. According to some studies, size metrics just giving signal about how large of a module should be in order to get low complexity. Unfortunately, most of those studies only work on data gathered from the project with single language and similar environment. It would be different if it have to apply at diverse software background. Meanwhile, the last two type of complexity metrics fulfill this criteria. Internal mechanism complexity metrics can give explicit guidance how to minimize complexity on a code level. On the similar way, structured design complexity could guiding software designer to avoid unnecessary complexity that would happen during design phase. That is why, internal mechanism and structured design metrics comply with guidance criteria. Utilized the third criteria, comprehensive, again the first two type of complexity metrics would not fulfilling it. Both the two metrics only observe software at the code level. It can not employed on the design level when software is seen as a system consist of several component. In the meantime, structured design complexity was developed concerning software design. It really built based upon platform for design measurement. Hence, it used for evaluating design of software system, and therefore comply with the third criteria. Almost all of complexity metrics mentioned here, satisfying the fourth criteria. Except for some structured complexity metrics, all other metrics are computable and would be able to make automation on it. Table 1 summarize this evaluation of the three type of metrics.

In recent practices, Object-Oriented Design (OOD) has been widely used for developing software system in academics, industries

and even governmental area. The newly emerging methodology, Model Driven Architecture (MDA) from the Object management Group (OMG), have also began been using worldwide in many different application and environment. Two of those things have a common platform on the way of developing software system. Building software system using both OOD and MDA, will always based on the structured design methodology. Therefore, structured design complexity would be helpful to give guidance in today development process.

Table 1.
Characteristics of the Three Type of Complexity Metrics

Complexity Metrics Type	Predictive	Guidance	Comprehensive	Computable
Size Metrics	not	not	Not	yes
Procedural Complexity Metrics	not	yes	Not	yes
Structured Complexity Metrics	yes	yes	Yes	yes*

* not for all metrics

This paper will only analyze on information flow metrics, hybrid information flow metrics, and especially focusing on system complexity metrics. Those three metrics were chosen simply because beside their use are widely spreading in software engineering practice, they are easily computable and automable, and also they have relations to some extent. Moreover, some other metrics were not adequately validated yet (such as data invocation complexity, stability measurements, data flow complexity, and functional complexity) which are not satisfy the Kafura and Henry's third criteria. While the others are not simple and easily computable (such as system partitioning measurements, structured design quality, and object oriented metrics) which are not comply with the third and sixth criteria from Xenos and his colleagues'.

E. Review on System-level Complexity Metrics

Viewing software system as a building, it will possible to analyze the architectural complexity of the building. A system always consist from components and there exist interaction between those components. Complicatedness of a system would depend upon three factors:

1. internal complexity of the constructing components,
2. interaction complexity between components within a system, and

3. the message contents which pass from one component to another.

On a software system, those three matters become internal module (procedural) complexity, structural complexity, and data (parameter) complexity. Figure 1 describes the graphical view of a software system.

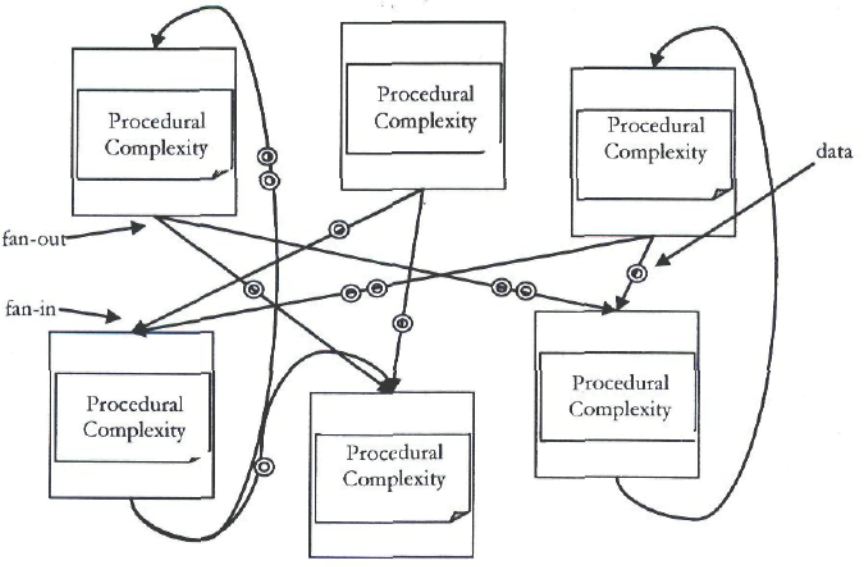

Figure 1. Graphical View of Software System

From this abstract description, it already been defined some parameter on structural complexity. Number of request to a given module is fan-in for that module, while the number of call to another module its fan-out. Based on these parameters, this paper will analyze system complexity metrics which had examined as the most aligned metrics based on software metrics criteria. Although this metrics was assembled for procedural purpose in nature, it still meaningful if its use on systematical framework. That could be happened since this metric was constructed based on the platform of coupling proposed by Myers, Yourdon, and Constantine.[29]

Before analyze system complexity metrics, it is important to describe two preliminary studies on complexity which becomes the basis for constructing that metrics. Those studies were done by Henry,

[29] G. J. Myers, *Composite Structured Design*, (U.K.: Van Nostrand Reinhold, 1978).

Kafura, and Selig. The first study done by Henry and Kafura formulate information-flow complexity as:[30]

$$C_p = (fan\_in \times fan\_out)^2 \quad (1)$$

where C_p is the complexity of module p. This complexity have a significant correlation with defects ($r^2 = 0.95$) when they validate it on their subsequent study with UNIX operating system project. That correlation is higher compare with McCabe complexity ($r^2 = 0.89$) and only slightly less than Halstead's software science ($r^2 = 0.96$) when validated using same data. That study also shown that there is high correlation between McCabe and Halstead's ($r^2 = 0.84$), but there are less correlation between information flow and the other two metrics with $r^2 = 0.38$ for McCabe and $r^2 = 0.35$ for Halstead's[31]. However, in some later study, it is known that it only fan-out who has a significant relation with defects rate. One example which shows that there is no relation between fan-in and defects rate was study on AS/400 system where the system was written in PL/MI with around 70 KLOC[32]. A main drawback from this complexity formulation in the context of systematic view is this formulation only takes the inter-module relationship into its account. The abandoning data complexity on the computation of system complexity will give misleading prediction if its implemented at the data-intensive system. Meanwhile, a simplification effort guided by this formulation would only be minimizing the number of fan-in and fan-out. Suppose there is a module with 1 fan-in and n fan-out. According to the formulation, the complexity of that module is simply n^2. If a reconstruction done to that module, and all the procedural routine on another module are collected to one module, so the fan-out for this module become zero. Applying the above formula, the complexity of the module become zero. That absolutely does not make any sense, since the internal module complexity become higher.

---

[30] S. M. Henry, and D. Kafura, "Software Structure Metrics Based on Information Flow", in *IEEE Transaction on Software Engineering*, 1981, p.510-518.

[31] D. Kafura, and S. M. Henry, "Software Quality Metrics Based on Interconnectivity", in *Journal of System and Software*, 1981, p.121-131.

[32] S. H. Kan, *Metrics and Models in Software Quality Engineering 2nd Edition*, (Boston, MA: Addison-Wesley, 2003).

To overcome the inclusiveness problem of internal module complexity on the complexity formula, Henry and Selig conducted another study. This new formula composes both internal mechanism and structural complexity into account to formulate the system complexity. They proposed a new complexity formulation which they called Hybrid Information-Flow Metric[33]:

$$HC_p = C_{ip} \times (fan\_in \times fan\_out)^2 \qquad (2)$$

where HC_p is hybrid-complexity of module p and C_{ip} is the internal complexity of module p. In this study, Henry and Selig validate their proposed metrics using data from 27 projects which consist of 981 modules. That data had excluding around 10 percents of outliers. Unfortunately, Henry and Selig did not validate correlation between this complexity and defects rate. Since their work was not intended to find out the relation between complexity and defects, simply they just observed the relation between this hybrid complexity with the refinement level of program for specification purposes. Hence, even this formulation has include the procedural complexity of module, this formulation still in complete since its not employing data complexity to count the overall system complexity. Therefore, this formula would not suitable if it is applied on data-intensive software system.

Another study on complexity at systematic level was conduct by Card and Agresti. In that study, they propose system complexity metrics. According to them, complexity of a module rely on the interaction of that module with another modules in the system and the complexity of input-output (data) which pass to and from that given module. They construct a formula for this complexity as[34]:

$$C = \frac{\sum_{i=1}^{n} f^2{}_i}{n} + \frac{\sum_{i=1}^{n} \frac{v}{f_c + 1}}{n} \qquad (3)$$

---

[33] S. M. Henry, and C. Selig, "Predicting Source-Code Complexity at the Design State", in *E Software*, 1990, p.36-44.

[34] D. N. Card, and W. W. Agresti, "Measuring Software Design Complexity", in *Journal of System and Software*, 1988, p.185-197.

where:

C = software system design complexity,
f = number of fan-out,
v = number of I/O unit, and
n = number of module in the system

On this study, they found that design complexity had a significant correlation with defects rate ($r^2 = 0.83$). Followed with subsequent study conducted by Card and Glass, they derived a regression formula for error rate based on system complexity with[35]:

Error rate = -5.2 + 0.4 x complexity **(4)**

Since the above formulation was derived from regression analysis, its always open to debate. The result of formulation which derived from a regression analysis will always depend on the data that had been used, and its just observed a particular substance. That formulation would probably not fit if its implemented to another environment, using different language, and at different application purposes. To make it robust, a proposed formulation should validated on various environment employing large-scale data.

In summary, this complexity metrics looks quite good and applicable for computing complexity at system design level. It satisfies almost all of the meta-metric criteria mentioned in section 3. According to this formula, complexity of a module is depended on the number of interaction of this module with another module within system and number of data which come into that module. Additionally, a whole system complexity is depended on the total relative complexity of its constructing module. The main obstacle of this formulation is quite similar with the two previous metrics. According to this formula, since the complexity of a module is related straightly to its fan-out and I/O unit, therefore minimizing module complexity could be done by gathering as much process as possible into one module, so it will reduce the number of fan-out and I/O unit. It is obviously very illogical because a central module where all the process gathered would be very complex. Another scenario can also be used to describe the weakness of this metrics. Suppose there is a module which have no fan-out and just

[35] D. N. Card, and Robert L. Glass, *Measuring Software Design Quality*, (Englewood Cliffs, N.J.: Prentice-Hall, 1990).

have one I/O unit that passes to that module. The absolute complexity value of this module will be only one. When this value divided by number of modules of the whole system, it will become very small. Even if this module has very complex procedure within it, this complexity formulation did not take this into account.

Look at to those three metrics, it is clearly seen that they have common weaknesses. First, they didn't considering all of the three factors on system-level complexity. One explanation why it could happen because one method used to derive the formulation is regression analysis. Using that tool, if there are two variables which have high correlation between each other, it can just use one variable as a predictor, since the other factor is already represented. From the study by Card and Glass, it found that procedural complexity had a significant correlation with I/O unit. Therefore, they just use I/O unit for computing formulation, since the internal procedure complexity has represented by the I/O unit. This first weakness leads to the second weakness. The developing of all the three metrics above is lack of theoretical basis. They looks like did not build with strong mathematical platform, and conventional thinking on system complexity, for instance, the formula on system complexity metrics. On the computation on data complexity, the number of I/O unit is divided by f+1. The addition of fan-out with one just to anticipate if there is a module with zero fan-out, so it would lead to infinity on data complexity. So, that formula did not have a rational judgment on the relationship with data complexity. A formulation should be based on the rational thinking about a matter being formulate. The third weakness is all of those metrics did not carry out complexity in comprehensive and integrative way. Reducing complexity should be seen as the whole system interest. There are two essential aspects on system-level complexity. (i) Given that system-level complexity rely on three factors mentioned above, an effort in reducing complexity in one part of system, would lead to increasing complexity at another part within this system. That is one of main logical basis on system complexity. (ii) Another rational basis which should also be considered is there always remains inherent complexity on any kind of system. Consequently, any formulation on complexity should not result null or zero value.

F. Summary and Future Work

Metrics on complexity measurement can be classified as: size related metrics, procedural complexity, and structured design complexity. In order to be useful for recent practice in software development project, a complexity metrics should satisfy these following criteria: predictive, giving guidance, comprehensive, and computable. Using the above criterion and another meta-metrics mentioned in section C[36] it clearly seen that system complexity metric is the one who most aligned with those criteria, hence it is worth to analyze. However, it still needs to be explored another form of complexity measurements, especially on system-level complexity metrics which satisfy the two aspect of rational basis. This work could be done using graph theory as a foundation for developing the *formulation concept*.

[36] D. Kafura, and S. M. Henry, "Software Quality Metrics Based on Interconnectivity", in *Journal of System and Software*, 1981, p.121-131.

REFERENCES

Abreu, F., "MOOD – Metrics for Object Oriented Design", in *OOPSLA'94 Workshop*, 1994.

Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation", in *Communication of the ACM*, January 1984, p. 42-52.

Belady, L. A., and C. J. Evangelisti, "System Partitioning and It's Measure", in *Journal of System and Software*, 2, 1981, p. 23-39.

Card, D. N., and W. W. Agresti, "Measuring Software Design Complexity", in *Journal of System and Software*, 8, 1988, p.185-197.

Card, D. N., and Robert L. Glass, *Measuring Software Design Quality*, Englewood Cliffs, N.J.: Prentice-Hall, 1990.

Chidamber, S. R., and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", in *IEEE Transaction on Software Engineering*, Vol. 20, No. 6, June 1994, p. 476-493.

Craddock, L. L., "Analyzing Cost-of-Quality, Complexity, and Defect Metrics for Software Inspections", in *Technical Report TR07.844, IBM Rochester*, Minn., April 1987.

Halstead, M. H., *Elements of Software Science*, New York: Elsevier North Holland, 1977.

Harrison, W., "An Entropy-Based Measure of Software Complexity", in *IEEE Transaction on Software Engineering*, Vol. 18, No. 11, November 1992.

Henry, S. M., and C. Selig, "Predicting Source-Code Complexity at the Design State", in *IEEE Software*, March 1990, p.36-44.

Henry, S. M., and D. Kafura, "Software Structure Metrics Based on Information Flow", in *IEEE Transaction on Software Engineering*, Vol. SE-7, 1981, p.510-518.

IEEE Computer Society, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990.

Kafura, D. and S. M. Henry, "Software Quality Metrics Based on Interconnectivity", in *Journal of System and Software*, Vol. 2, 1981, p.121-131.

Kan, S. H., *Metrics and Models in Software Quality Engineering 2nd Edition*, Boston, MA: Addison-Wesley, 2003.

Lo, B., "Syntactical Construct Based APAR Projection", in *IBM Santa Teresa Laboratory Technical Report*, California, 1992.

Lorenz, M. and J. Kidd, *Object-Oriented Software Metrics*, USA: Prentice-Hall, 1994.

McCabe, T. J., "A Complexity Measure", in *IEEE Transaction on Software Engineering*, Vol. 2, No. 4, December 1976, p. 308-320.

McClure, C. L., "A Model for Program Complexity Analysis", in *Proceeding IEEE Third International Conference on Software Engineering*, May 1978, p.149-157.

Myers, G. J. *Composite Structured Design*, Wokingham, U.K.: Van Nostrand reinhold, 1978.

Oviedo, E. I., "Control Flow, Data Flow and Program Complexity", in *Proceeding IEEE COMPSAC*, Chicago, IL, November 1980, p. 146-152.

———, *Oxford Advance Learner's Dictionary*, U.K.: Oxford University,1991.

Shen, V., T. Yu, S. Thebaut, and L. Paulsen, "Identifying Error-prone Software – An Empirical Study", in *IEEE Transaction on Software Engineering*, Vol. SE-11, No. 4, April 1985, p. 317-324.

Stevens, S. S., "On the Theory of Scales of Measurement", in *Science*, Vol. 103, 1947, p. 677-709.

Tran-Cao, D., A. Abran, and G. Levesque, "Functional Complexity Measurement", in *Proceeding International Workshop on Software Measurement*, Montreal, Quebec, August 2001, p.173-181.

Troster, J., "Assessing Design-Quality Metrics on Legacy Software", in *Software Engineering Process Group, IBM Canada Ltd. Laboratory*, North York, Ontario, September, 1992.

Troy, D. A., and S. H. Zweben, "Measuring the Quality of Structured Design", in *Journal of System and Software*, 2, 1981, p.113-120.

Withrow, C., "Error Density and Size in Ada Software", in *Software Engineering Process Group, IBM Canada Ltd. Laboratory*, North York, Ontario, January 1990, p. 26-30.

Woodward, M. R., M. A. Hennel, and D. Hedley, "A Measure of Control Flow Complexity in Program Text", in *IEEE Transaction on Software Engineering*, SE-5, Issue 1, January 1979.

Xenos, M., D. Stavrinoudis, K. Zikouli, and D. Christodoulakis, "Object-Oriented Metrics – A Survey", in *Proceeding of the FESMA 2000*, Madrid: Federation of European Software Measurement Association, 2000.

Yau, S. S., and J. S. Collofello, "Some Stability Measures for Software Maintenance", in *IEEE Transaction on Software Engineering*, Vol. SE-5, 1979, p. 545-552.

Yourdon, E. and L. L. Constantine, *Structured Design*, Englewood Cliff, N.J.: Prentice-Hall, 1979.