УДК 004.053
DOI: 10.18523/2617-3808.2020.3.12-16

A. Hlybovets, O. Shapoval

INVESTIGATION OF THE RELATIONSHIP BETWEEN SOFTWARE METRICS MEASUREMENTS AND ITS MAINTAINABILITY DEGREE

The goal of this work is to practically apply methods of empirical engineering software, algorithms for data collection and data analysis. The results include software measurement, analysis and selection of direct and indirect metrics for research and identification of dependencies between direct and indirect metrics. Based on the received results, there were built dependencies between software metrics and software expertise properties were selected by individual variation.

For measurement results analysis there were used primary statistical analysis, expert estimations, correlation and regression analysis. Expert estimation is the dominant strategy when estimating software development effort. Typically, effort estimates are over-optimistic and there is a strong over-confidence in their accuracy. Primary data analysis is the process of comprehending the data collected to answer research questions or to support or reject research hypotheses that the study was originally designed to evaluate. Correlation analysis gives possibility to make some conclusions about which metrics and expert estimations are much coupled, and which are not. Regression analysis involves both graphical construction and analytical research and gives an ability to make a conclusion about which metrics and expert estimations are the most coupled. Analyzing regression lines for metrics of normal and nonnormal distributions give an ability to identify pairs of 'metric – expert estimation'.

There have been calculated and measured metrics relations for defining relation of such quality attributes as Understandability and Functionality Completeness. Understandability expresses the clarity of the system design. If the system is well designed, new developers are able to understand easily the implementation details and quickly begin contributing to the project. Functionality Completeness refers to the absence of omission errors in the program and database. It is evaluated against a specification of software requirements that define the desired degree of generalization and abstraction.

Relationship between metric and expertise includes building direct relationships between the metric and expertise, indirect metrics and expertise. Additionally, it has been determined whether they have common trends of the relationship between those direct metrics and expert estimates, indirect metrics and expert estimates. The practical results of this work can be applied for software measurements to analyze what changes in the code (affecting given metric) will cause increasing or decreasing of what quality attribute.

Keywords: Software quality attributes, Understandability, Functionality Completeness, Direct Metrics, CYC, NOM, NOC, CALL, FOUT, Indirect Metrics, AMW, ATFD, BOvR.

INTRODUCTION

One of the main difficulties during software architecture developing is to evaluate the available design options and choose the best ones. Additionally, even after the right decisions were made during initial phase of the system development, it is crucial to control quality of produced changes into the system afterwards. One of the main reason of problems with depicted needs is that developers are unclear what criteria they should use to make design decisions and why. Some developers rely on their previous engineering experience and personal preferences in methods, technologies, tools, and patterns. The problem is that each member of the development team has its preferences, opinions and assumptions. As a result, it can be difficult to reach consensus on the team and agree on some decisions. The debate about subjective opinions and preferences can not only damage the relationship between colleagues but may not necessarily lead to a software architecture optimized to achieve business goals.

APPROACH DESCRIPTION

To overcome potential organizational and collaboration issues and define objective properties of the software, the development team and the client should agree about a defined set of quality attributes of the system and approach of results measurement. Within systems engineering, quality attributes are realized non-functional requirements used to evaluate the performance of a system. They are usually Architecturally Significant Requirements that require

© *A. Hlybovets, O. Shapoval,* 2020

architects' attention. From the perspectives of system support possibilities, next software quality attributes are important:

Understandability property

Software developers and accompanying persons should read and understand the source programs and other types of program documents in their work. The clarity of software documents is therefore important. To understand the level of comprehensibility of the software, we need to answer the following questions [7]. When programmers try to reuse a software developed by other programmers, the difficulty in understanding the system limits reuse. It is not easy to measure the comprehensibility of software, because understanding is an internal process of people. There is the notion of the "intelligibility integral" as a model for measuring software intelligibility, which can first extract the best software intelligibility value from higher weight factors. In other words, we have given an integrated dimension to measuring software comprehensibility through the literature on this subject [9].

Functionality completeness property

Completeness refers to the absence of omission errors in the program and database. It is evaluated against a specification of software requirements that define the desired degree of generalization and abstraction (selective omission). "Data completeness" is a measurable error between the database and the specification [7]. Even highly generalized databases can be "complete data" if they contain all the objects described in the specification. A database is a "complete model" if its specification is appropriate for the application. Validity is a measure of the attribute of the accuracy of software functionality following the specification of requirements. Each attribute must have a specific domain and range. Program validation is the process of determining whether the values of program processes are sufficiently accurate, complete, and logically compatible with the intended use of the data. Verification often consists of several steps, including logical checks, accuracy estimates, and error analysis.

After the definition of expected software quality attributes, it's is required to come up with a set of objective points that may answer to the question of what degree of quality attributes compliance the system is capable. In this task accomplishing we may use the results of Empirical software engineering, which is a part of software engineering that focuses on gathering evidence, through measurements and experiments involving software systems (software products, processes, and resources). For example, the degree of fulfillments of defined software quality attributes as understandability and functionality completeness may be based on next metrics.

- **Lines of code** is a software metric used to measure the size of a program by counting the number of lines in the text of the program source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to evaluate the performance or programming effort after a software release. There are two main types of SLOC measures: physical SLOC and logical SLOC. The specific definitions of these two indicators differ, but the most common definition of physical SLOC is the number of lines in the source code of the program, including comment lines. Blank lines are also included unless the lines of code in the section contain more than 25 % blank lines. In this case, empty lines exceeding 25 % are not considered in the lines of code.
- **Number of methods i**s used to calculate the average number of operations of all classes in a class. A class must have several, but not excessive, operations. This information is useful in identifying a lack of primitiveness in class operations (prohibition of reuse) and in classes that are slightly larger than data types. For a class, this is a simple count of the number of operations. For a package, this is the average number of operations per package class. This value should remain between 3 and 7. This will mean that the class has operations, but not too many. A value greater than 7 may indicate the need for further object-oriented decomposition, or that the class does not have a consistent goal. A value of 2 or less indicates that this is not a class, but just a data construct [6].
- **Number of Direct Descendants** (NDD) This metric is the number of direct descendants (subclasses) for each class. It is believed that classes with many children are difficult to modify, and, as a rule, they require additional testing due to the effect of changes on all children. They are also considered more complex and error-prone, because a class with many children may need to provide services in more contexts and therefore should be more flexible [8].
- **Height of Inheritance Tree** (HIT) for a class or structure is the number of base classes (including the System.Object class, so DIT> = 1). Types for which HeigthOfInheritance is above 6 may be difficult to maintain. However, this is not the rule,

because sometimes your classes can inherit from level classes that are of high importance for the depth of inheritance. For example, the average depth of inheritance for wireframe classes derived from System.Windows.Forms.Control [8].

- **Fan-out** (FOUT) – the number of classes to call, this is calculated as the sum of the FANOUT metric (that is, the classes from which the operations call the methods) for all user operations. This metric provides raw information about distributed activity calls in classes. Branching is a measure of the ability of an electronic logic gate output to control multiple inputs of other logic gates of the same type. In most designs, the logic gates are connected to form more complex circuits, and usually, one output of the logic element is connected to several inputs of the logic element. The technology used to implement logic gates typically allows direct connection of gate inputs without the need for additional interface circuits [6].
- **Average Method Weight** (AMW) – the average statistic complexity of all methods in a class.
- **Access to foreign data** (ATFD) is the number of external classes from which this class accesses attributes directly or through access methods. Because ATFD measures how many external attributes are used by a class, it is clear that the higher the ATFD for a class, the higher the likelihood that the class is (or is about to become) a class of God. It quantifies one of the key disharmonies of identity distortion, that is, the rude use of attributes from other classes. As you can see, this is again one of the reasons why the method is considered disharmonious [6].
- **Base Class Overriding Ratio** (BOvR) – the number of methods of the measured class that override methods from the base class, divided by the total number of methods is the class. Quantifies the degree of overriding and specialization of base class methods [1].
- **Tight Class Cohesion** (TCC) – the relative number of method pairs of a class that access in common at least one attribute of the measured class. Is the relative number of methods directly connected via accesses of attributes [6].
- **Weight of Class** (WOC) – the number of ‘functional’ public methods divided by the total number of public members.

ANALYSIS RESULTS

In order to make sure that the picked metrics satisfy the expected software quality attributes we can use a series statistical analysis tools like primary statistical analysis, expert estimations, correlation and regression analysis.

Primary data analysis is the process of comprehending the data collected to answer research questions or to support or reject research hypotheses that the study was originally designed to evaluate. The choice of data analysis methods depends on the type of data being collected, quantitative or qualitative.

After the primary statistical analysis is completed a conclusion can be made about which metrics and expert estimations belongs to normal distribution and which not.

Analyzing coefficient of excess and skewness, I can assume that direct metric HIT, indirect metrics TCC WOC and all expert estimations (reliability and understandability) belong to normal distribution [1]. Figure 1 depicts an example of measurement results of project measurements for Lines of Code metric per classes. As we can see, most of the classes (about 4250) have for about 1000 lines of code.

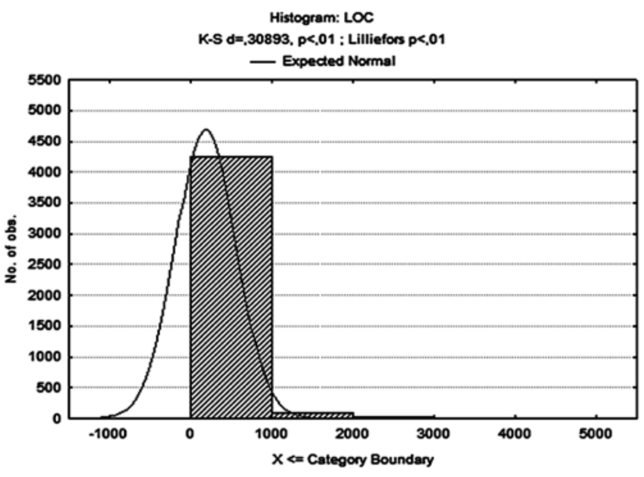

Fig. 1. LOC Histogram

It the same time, most of that have an amount of methods for about 1 to 50 (Figure 2). Which does not correlate nicely with may postulates of good software engineering.

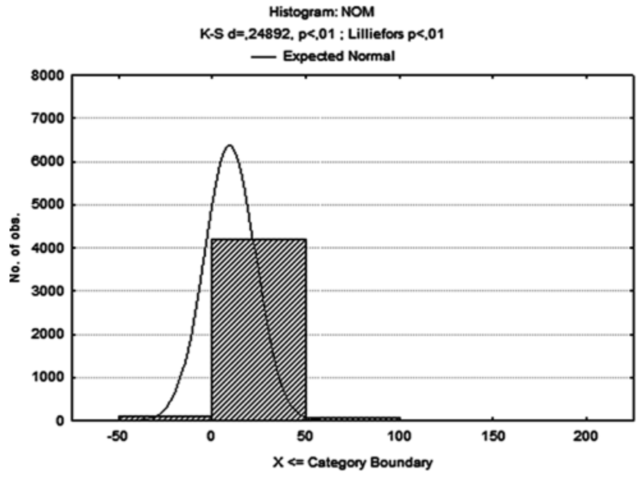

Fig. 2. NOM Histogram

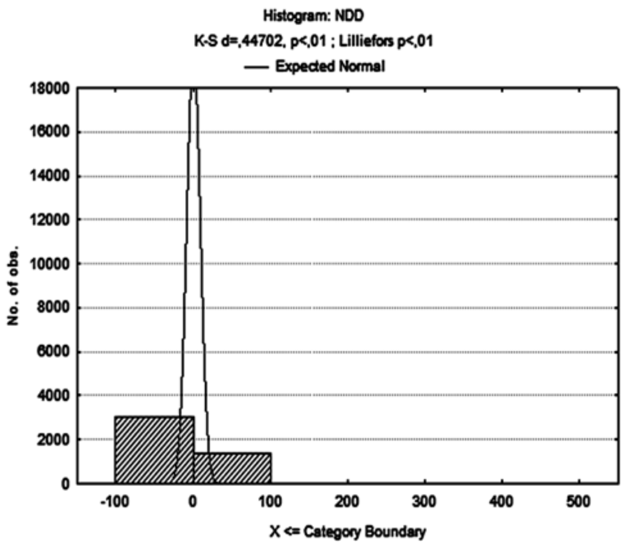

Fig. 3. NDD Histogram

The Average number of direct descendants is some cases may be in the range up to 100 (Figure 3).

Correlation analysis gives possibility to make some conclusions about which metrics and expert estimations are much coupled, and which are not. Analyzing correlation coefficient and Spearman coefficient, I chose several meanings that have most tight relation. For example, TCC and WOC metrics are the use with normal distribution.

Regression analysis is the last stage in the study on the dependence of metrics and expert estimates. It is carried out only under the condition that the variance of the dependent variable (expert assessment) must remain constant when changing the value of the argument (metric), i.e. first, the variance of the expert assessment is determined for each accepted value of the metric. Next, the regression is identified. It involves both graphical construction and analytical research. Graphical construction begins with the definition of the correlation field. If the correlation field is elliptical, a linear regression relationship is inferred. Next, the construction of linear regression and its evaluation. If the constructed points of the correlation field fall into a circle, it is concluded that there is no dependence. If the correlation field does not fit into a circle or ellipse but has a different form, then a conclusion is made about the nonlinear dependence in the regression line [4].

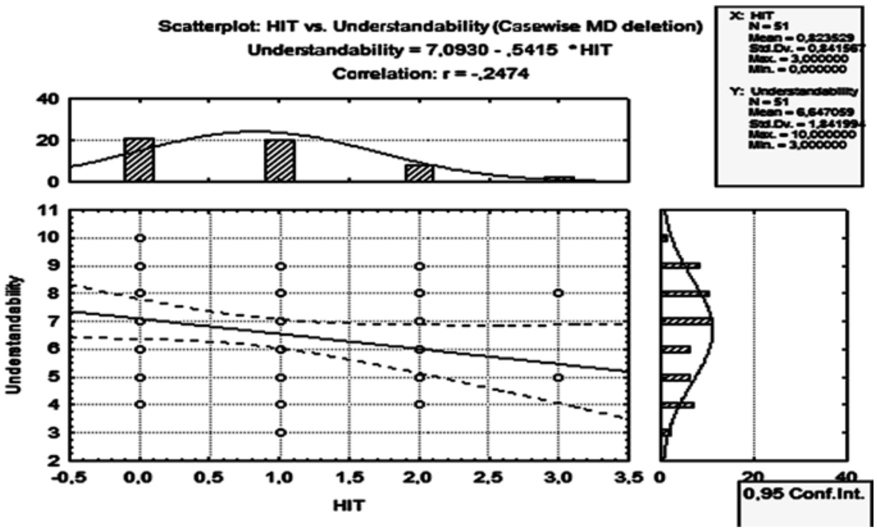

Fig. 4. Regression Line: HIT – Understandability

Conclusions

Empirical software engineering is a set of actions to gain knowledge to better understand aspects of software development. The result of the action is several statements about a certain list of problems. These statements are answers to questions and confirm or refute hypotheses.

Identify four main areas of empirical research:

- Research related to the technical part (research of the development environment, research of the used software methods in development, research of the software product itself);
- Research of development processes of each developer individually (everything that the developer does);
- Research of possibilities of integration of software products made by developers;
- Study of interaction, coordination of the development team (for example, whether the team works as a single mechanism or simply as separate developers who interact at some related stages).

As the results of the work, there can be named tuples of 'metric – expert estimation' that are most coupled: HIT – Understandability (for normal distribution – linear dependence), LOC – Understandability, BOvR – Understandability, BOvR – Functional (for unnormal distribution – nonlinear dependence). The practical results of this work can be applied for software measurements to analyze what changes in the code (affecting given metric) will cause increasing or decreasing of what software property.

References

1. Ebert, C., Dumke, R., Bundschuh, M., & Schmietendorf, A. (2005). *Best Practices in Software Measurement: How to use metrics to improve project and process performance*. Springer-Verlag Berlin Heidelberg.
2. Evolutionary architecture and emergent design: Emergent design through metrics. (2010). *IBM Deloper Website*. Ford N. Retrieved from https://www.ibm.com/developerworks/library/j-eaed6.
3. Fenton, N., & Pfleeger, Sh. (1996). *Software Metrics: A Rigorous and Practical Approach*. Cambridge: Cambridge University Press.
4. Kan, S. (2002). *Metrics and Models in Software Quality Engineering*. Second Edition. Addison Wesley.
5. Laird, L., & Brennan, C. (2006). *Software Measurement and Estimation: A practical approach*. New Jersey: John Wiley & Sons.
6. Lanza, M., & Marinescu, R. (2006). *Object-Oriented Metrics in Practice*. London: Springer-Verlag Berlin Heidelberg.
7. Mens, T., & Demeyer, S. (2008). *Software Evolution*. Berlin: Springer-Verlag Heidelberg.
8. Shull, F., Singer, J., & Sjoberg, D. (2008). *Guide to Advanced Empirical Software Engineering*. London: Springer-Verlag Limited.
9. Sommerville, I. (2016). *Software Engineering*. Pearson Education Limited.

Глибовець А. М., Шаповал О. О.

ДОСЛІДЖЕННЯ ЗАЛЕЖНОСТІ МІЖ ЗНАЧЕННЯМИ МЕТРИК ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА МОЖЛИВІСТЮ ЙОГО СУПРОВОДУ

У роботі розглянуто один із методів контролю якості програмного забезпечення на основі виділення необхідних атрибутів і метрик для його аналізу та супроводу, а також підходи до їх підбору на основі первинного, кореляційного та регресійного аналізу даних.

Ключові слова: атрибути якості програмного забезпечення, зрозумілість, повнота функціональності, прямі метрики, LOC, BOvR.

Матеріал надійшов 10.06.2020

Creative Commons Attribution 4.0 International License (CC BY 4.0)