Identifying Software Metrics Thresholds for Safety Critical System

Štěpán Cais and Petr Pícha
University of West Bohemia in Pilsen
scais@kiv.zcu.cz, ppicha@kiv.zcu.cz

ABSTRACT

The quality of software is very important for every kind of software system. Without quality software, there will be more accidents and system failures. As the responsibility of the software increases, the quality becomes more and more important. This is even more crucial for the case of safety critical software, the failure of which can endanger human lives. For system quality analysis, the software metrics can be used. Although software metrics are well-known and widely used, there is still a discussion about identifying their thresholds and usage. Furthermore, there is a lack of information about software metrics thresholds, particularly about thresholds of safety critical software. This puts evaluators of safety critical systems into a position where they neither have any data for comparison, nor do they know suitable methods for evaluation. This article describes evaluation of software metrics of one safety critical system and provides information about measured data and data collection and evaluation method. Finally, the determined thresholds are presented for future comparison.

KEYWORDS

Software metrics, software metrics thresholds, heavy-tailed data, real-time system analysis

1 STUDY CONTEXT

1.1 Metrics and thresholds

Software metrics are well-known means for software quality analysis [1]. Using software metrics one can measure the software quality from different points of views, search for potential problems within the software, identify its parts which should be examined or control the whole system development. The broad usage of software metrics is demonstrated by the high number of existing metrics themselves as well as tools used for their measuring [2].

However, the usage of metrics themselves is only part of the software measuring and evaluating process because once we obtain or calculate the values of our specific set of metrics we also need to determine when does the value represent a positive (or at least tolerable) characteristic of the measured software, and when does this value become a sort of warning sign, i.e. an indication of some unsatisfying or unwanted aspect of the software. The boundaries between these two states or sets of values (or any number of shades of grey in between them) are commonly known as thresholds.

We can sort software metrics to categories according to their usage. One of them is the program metrics used for measuring inner characteristics of an inspected system (i.e. number of lines of code, number of methods etc.). A subset of these metrics is a collection of object-oriented metrics the goal of which is to measure the object-oriented software. There is a huge amount of object-oriented metrics and many of them were the subject of research and examination in the past [3]. Nevertheless, the problem of their actual usage does not lie only in the extensive amount of existing object-oriented metrics and therefore in selecting the appropriate subset to measure given software. Even if just a limited number of commonly known metrics is used, the result of their application does not have to be unequivocal. Unfortunately, even though there is a considerable effort to make the view on software metrics clear, there still is no widely

used and unified view on threshold values of software metrics.
This lack of consistent view is especially obvious in the field of safety-critical software systems. The recently conducted research in this area shows that measuring metrics of different system types brings different threshold values. The results of measuring can be influenced by measurement tool [4], [5], programming language [5], [6], or by the category of software [7].

1.2 Safety Critical Systems

Software quality is a very important aspect of software development, which is constantly being emphasized. One of the areas of the software development field where the demands on the quality are the highest is the safety critical software development. As far as this type of software is concerned, quality is a crucial aspect. Any failure in such software can damage freight or endanger lives of people [8]. A particular case of safety critical software can be the hard real-time system which enables the functioning of the train security system or the aircraft control system.

1.3 Goals of this study

The goal of this paper is to find threshold values for a given set of software metrics measuring one particular object-oriented safety critical system and afterwards evaluate the system using the found threshold values.

1.4 Studied system

The system this paper concerns itself with is used in railway industry. It is responsible for safe functioning of various devices, in which it serves as a base platform for their own functionality. In a very simplified sense, it can be seen as a hard-real time system, which enables special algorithms to run on their respective devices. The software is written in C++. All software development is being done by experienced team and according to certain coding standards. The whole process of development is being constantly validated and verified. The source code is regularly checked by automatic tools which evaluate compliance with coding standards and compute current values of specified object oriented metrics.

1.4 Research Questions

From the goals of this study mentioned above the following research questions were derived:

1) How to find metrics thresholds values for safety critical system?
2) What will be the results of evaluation of our particular system using these metrics threshold values?

These questions will be answered in the conclusion of this paper.

2 STUDY LIMITATIONS

According to [6], there are two ways to determine the software metrics thresholds: by the statistical methods or by the widely accepted threshold values for particular metric. As for our type of analyzed system these widely accepted threshold values do not exist, and so we find them by using statistical methods.
Statistical methods are based on the principle of derivation of threshold values by collecting data from available systems. Characteristically, we measure various metrics over various systems. Afterwards, we employ statistical methods and derive threshold values. However, this approach has several pitfalls, which have to be considered.

2.1 General limitations and constrictions and their resolution

Above all, there are differences between evaluated software projects. They can differ in programming language used (Java, C, C++ etc.), in the particular software type (game,

office software, real-time system etc.), in size (100 LOC – 100 MLOC) etc. Due to this fact, it is necessary to take the programming language, the project type and its size into consideration.
Another significant finding can be seen in [4] where it is shown that different software metrics measuring tools can evaluate the same projects with different threshold values. This is caused particularly by ambiguous definition of software metrics. For example, the computation of the DIT metric (Depth of Inheritance Tree) for Java language can be done with or without including the Object class. As long as different tools are used, the result of the measurement can be different, because each tool can implement different technique of computation.
As shown in [4], the choice of measuring tool can negatively influence the set of classes assigned for inspection. While one tool can identify set A as the set of classes most critical for inspection, another tool can pick set B. In the worst case scenario the intersection of these sets can be empty. If this happens, the staff responsible for class review can misguidedly define the set of classes for inspection.
Another issue is the large number of existing object-oriented software metrics. It is a common occurrence that for different projects there are often different sets of metrics used for evaluation. We can also see different metrics used for measuring in different studies. This makes it difficult to collect data for analysis and evaluation of metrics thresholds, if we want to compare our measured data with others.
If we sum up previous paragraphs, for proper usage of statistical methods for evaluating threshold values of software metrics we need to have at our disposal only data from similarly typed and sized software systems coded in the same programming language, measured by the same tool, and with the same set of metrics applied. In case we cannot reach previous requirements, it is probable that the evaluation of software metrics on the grounds of statistical methods is not going to be precise and it could result in misleading conclusions.

2.2 Specific limitations of measuring safety-critical software

As described before, the absence of widely accepted threshold values for the type of systems similar to the one we try to evaluate forces us to use the statistical methods for their determination and therefore we need data from other sufficiently similar systems for analysis and comparison. Unfortunately, in the particular case of safety critical systems we are in an even more difficult situation. There are almost no data published from measuring this kind of software. Safety critical systems are not usually open-source and it is practically impossible to get their source code for measuring. Considering this fact, determination of threshold values of safety critical systems by collecting sufficient amount of data and applying statistical methods is almost impossible.
In this situation, determination of thresholds for safety critical system which is a subject of this study is more than likely to be tricky. It is ill-advised to compare measured values with data from other studies, because there are nearly no results from desirable systems. We also cannot mine data from accessible sites for our own measurement, because there is almost no suitable software for comparison. On the other hand, we still have source code of the analyzed system and tools for measuring. Even if we do not have adequate data for comparison, we can still evaluate our system and determine extremes in measured data. We can apply statistical methods on our data and establish local abnormalities. On the basis of measured data we can define a set of classes showing unusual measured values which can be afterwards analyzed more rigorously.

3 STUDY DESIGN

In the next few sections we describe and justify the following steps taken in the course of this study:

• Particular and appropriate set of metrics selection.
• The process of data collection, analysis and evaluation (metric values and threshold determination) including measuring our particular system using the threshold values found.
• Tools selection and usage details.

3.1 Metrics used

For evaluation of our system we used the following metrics: CBO, DIT, LCOM, NOC, NOM, RFC, SLOC 1, SLOC 2, and v(g) (the abbreviations meaning described lower). This set of metrics was chosen as sufficiently suitable for evaluation of the system which has to meet the EN ISO 50128. The exact process of choosing the appropriate set of metrics was the subject of an internal procedure within the owner company of the studied system and will not be described in this paper. Simple description of these metrics follows, for more detailed information see [9].

CBO (Coupling Between Objects) In Understand C++ API has name CountClassCoupled. The metric counts the number of other classes coupled to the analyzed one. Class X is coupled to class Y if class X uses a data, type or member from class Y.

DIT (Depth of Inheritance Tree)
Understand C++ API names this metric as MaxInheritanceTree. This metrics describes the depth of a class within the inheritance hierarchy. The value equal the number of nodes on the way up from the class node to the root of the inheritance hierarchy; the root has DIT value of 0.

LCOM (Lack of Cohesion in Methods)
In Understand C++ API has name PercentLackOfCohesion. It counts percentage of all inner methods of the class, which use particular instance variables. The final value is 100% minus the average value of percentage of all class instance variables.

NOC (Number of Children)
In Understand C++ API has name CountClassDelivered. The number of immediate subclasses of a given class.
NOM (Number of Methods)
Understand C++ API names this metric as CountDeclMethod. It represents number of all methods of a given class (including inherited methods).

RFC (Response for Class)
In Understand C++ API has name CountDeclMethodAll. This metrics has the same meaning as NOM, but includes also the inherited methods.

SLOC1 (Source Lines of Code - Function)
The API function has name CountLineCode. The metric counts the number of lines of code except for the lines containing comment only. We use SLOC1 for counting code lines of functions.

SLOC2 (Source Lines of Code - Class)
We use the same API function as in SLOC1 (CountLineCode), but for measuring lines of code of classes.

v(g) (Strict Cyclomatic Complexity)
In Understand C++ API has name CyclomaticStrict. Counts possible paths through the program functions during execution.

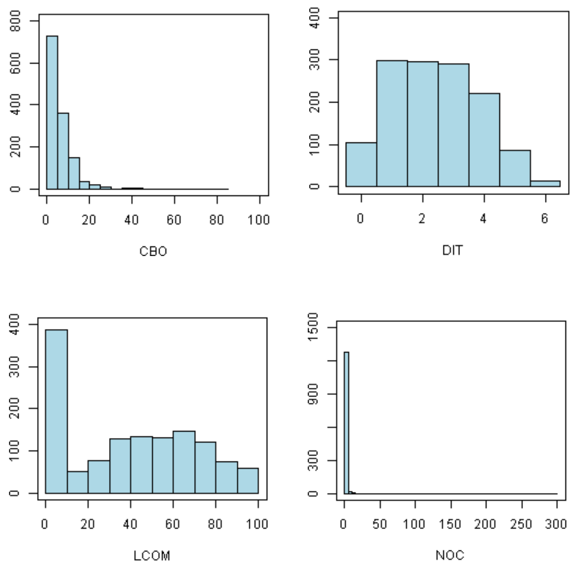
3.2. Data collection and analysis

The data collection and analysis in this study has several phases. First, we need to collect the values of the metrics themselves for the studied system. After that, some statistics and graphic representation are most likely to be necessary for the data to be prepared for further analysis. Then, the analysis of the results is performed to categorize the metrics, since further steps of threshold determination may vary in regard to their specific characteristics. Finally, we identify the actual threshold values. The step-by-step description below refers to the way the data from every individual metric is processed in the course of this study.

The metric values measurement through all the classes of the studied system can be easily automated by an appropriate software tool (or tools) and this automation is very desirable as it minimizes the risk of human error. The outcome of this automated tool should include the specific values as well as their graphic representation, most likely histograms, and some descriptive statistics.

Histograms can be used as a visual representation of the data probability distribution (i.e. normal, exponential, etc.) in a form of a graph. In this graph particular for our purposes the x axis values represent possible or observed values of a particular metric for individual classes and the y axis values show the number of classes or functions with the same results.

The usage of descriptive statistics is one of several possibilities how to evaluate measured data and gain some overall view of the dataset.



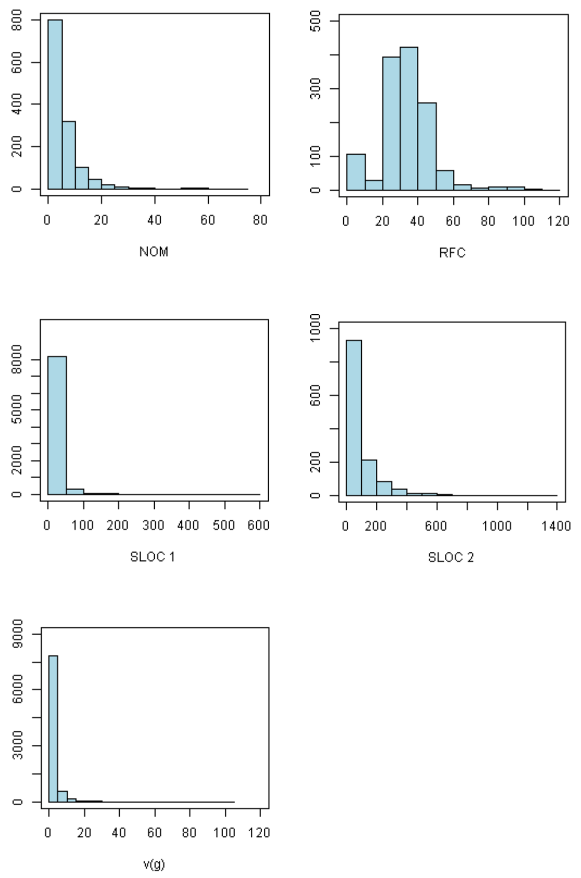


Figure 1. Histograms of measured metrics, the Y axis represents frequency of X values.

3.3 Measured descriptive statistics

The descriptive statistics suitable for the purposes of this study are:

• **arithmetic mean** - sum of all the values divided by the number of the values;

• **standard deviation** - shows how significant divergence from the mean exists in the dataset;

• **modus** - the most frequent value in the data set;

• **median** - represents the value located in the middle of the spread of the values;

• **kurtosis** - gives information about whether the data distribution has a peak or not;

• **skewness** - gives information about the symmetry of the data distribution;

• **minimum** and maximum – extreme values in the dataset.

The skewness and kurtosis statistics are probably the most difficult to imagine. As an example, normal distribution has the skewness equal zero. Distribution with a positive skewness has longer right tail and the mean is placed off to the right side of the peak value. On the other hand, distribution with a negative skewness has longer left tail and the mean is placed off to the left side. As we can see in [10], data with skewed distribution can have maximal values considerably different from the mean.
The simplified view on the kurtosis can be "how curvy the final function is". While the positive kurtosis indicates peaked data distribution, the negative kurtosis can be a sign of a flat data distribution.

3.4 Probability distribution issue

The so far collected and computed data may not suffice, as there is a different approach to determination of the threshold values with regard to the particular probability distribution that the data from an individual metric show. The analyses of information in studies [3] and [10] show that measured data of some software metrics do not follow normal distribution. Relatively often we can see that measured data approach heavy-tailed distribution and are right-skewed. Normal distribution data should have the skewness and kurtosis around zero [11]. If measured data shows skewness, the distribution does not need to be spread in the close range around the mean value [10]. In this type of distribution, the frequency of high values for random variable is very low whereas frequency of low values is very high [3]. In [10], we can find this information about right-skewed data "This skewness in metrics data affects the interpretation and usage of these metrics in evaluating software quality. Such metrics are not always well characterized by their descriptive statistics such as mean, standard deviation, minimum, maximum, and quartiles".
As far as we consider this argument, it is necessary to primarily determine the characteristic of measured data and continue to analyze metric values with regard to the results.
Therefore, before determining the threshold values, we first need to categories the metrics according to their histograms and descriptive statistics into two separate groups: the ones showing heavy-tail probability distribution, and the ones with the normal distribution.
With the heavy-tail distributed metrics the approximation and determination of their actual distribution seems appropriate. As described before, the presence of heavy-tail distribution can be observed from the descriptive statistics and histograms but again some automated software tool to help us approximate and determine the actual type of probability distribution should minimize the human error factor, make the whole process more manageable and confirm the observed conclusions.

3.5 Data categorization

Finally, after the categorization of metrics through their respective probability distribution, we should have all the input data necessary for the actual threshold value determination.
In [6], a range of typical values is used for metrics evaluation. This range represents typical values of particular metrics for which its lower and upper boundaries and extreme values are defined. For computation of typical values and upper and lower boundaries the arithmetic mean and deviation are used. If a measured value of a metric is 50% higher than the highest value of the interval between upper and lower boundary then the value is evaluated as an extreme value. This approach is suitable in the case where the data have normal distribution. Because data with normal distribution are centralized around their mean values, we can look at their values as determinative.
In [3], the technique of separating values into three categories was used for assessing the thresholds values of heavy-tailed data. The

categories are assigned the names of “good”, “regular”, and “bad” and each category represents particular range of values.
The “good” category represents range, which contains the most frequent values. The values with low frequency of occurrence, but which are still considered as not being rare belongs to the “regular” category. The “bad” category contains values, the occurrences of which are rare. This technique is appropriate for our heavy-tailed data as well.
Applying the above mentioned process on safety critical system will result in getting the set of metric values of each class. By selecting the classes, values of which cross threshold values, we get set of classes fit for review.

3.6 Tools used

For our analysis, three tools were used. First, the tool Understand C++ 3.1 (www.scitools.com) was used for measuring all the metrics values for all the classes of the studied system. We chose this tool mostly for these reasons: the Understand C++ can measure many software metrics and contains sufficiently deep description about how metrics are counted and also enables to compute all of measured metrics of C++ source code.
Second, for simplification and it´s widely usage, the Microsoft (MS) Excel 2003 was chosen for computation of the descriptive statistics and selection of classes for inspection.
Ultimately, the EasyFit tool (www.mathwave.com) was used for the analysis to fit the measured data to various probability distributions. This tool was chosen because of its simple usage and huge amount of probability distributions it can evaluate.

3.7 Execution

Here we sum up the actual process of data collection and analysis in the course of the study execution.
The data from source code were gathered by Understand C++. For any measured metric its own MS Excel file with results was created. After all metrics were measured, The MS Excel functions for computing all the descriptive statistics were used. For gaining better insight on data representation, we created a histogram for each measured metric. If descriptive statistics and histograms evinced heavy-tail data distribution, the EasyFit tool was used for data distribution fitting. In the end, MS Excel was used for selecting classes for review.

4 RESULT PRESENTATION AND FURTHER ANALYSIS

As we can see in table 1 and table 2, all metrics except RFC and LCOM have mean >= median >= mode. This can be seen as a sign of the right skewness in the data [10].
The peaked characteristic of data shows the positive kurtosis in all the measured metrics. Metrics CBO, NOC, NOM, SLOC1, SLOC2, and v(g) have mean values notably smaller than maximal value, which also shows the skewness of data distribution. The kurtosis values for DIT and LCOM are negative, so we can consider DIT and LCOM as having a flat distribution.

Table 1. Measured descriptive statistics of CBO, DIT, LCOM, NOC and NOM.

	CBO	**DIT**	**LCOM**	**NOC**	**NOM**
Arithmetic mean	6,28	2,41	39,64	0,92	6,42
Median	5	2	41	0	4
Modus	2	1	0	0	3
Standard deviation	6,16	1,43	31,57	8,22	6,8
Kurtosis	22,62	-0,71	-1,26	932,91	23,99
Skewness	3,14	0,21	0,08	28,62	3,96
Minimum	0	0	0	0	0
Maximum	82	6	100	274	75

Table 2. Measured descriptive statistics of RFC, SLOC1, SLOC2, v(g).

	RFC	SLOC 1	SLOC 2	v(g)
Arithmetic mean	33,85	14,92	94,74	2,9
Median	33	7	55	1
Modus	39	5	19	1
Standard deviation	14,91	26,67	120,8	5,47
Kurtosis	3,19	116,13	20,37	94,53
Skewness	0,66	8,56	3,62	8,01
Minimum	0	1	3	1
Maximum	106	582	1361	103

In the given histograms the right skew of the values of CBO, NOC, NOM, SLOC1, SLOC2, and v(g) is noticeable as we can see high concentration of values in the left side of the histograms. The visual observation confirms the results of descriptive statistics, where for the given set of metrics mean >= median >= mode and also relatively high values of the skewness and the kurtosis exist.
Different results can be found in the data from DIT, LCOM and RFC. The data from DIT do not show any concentration of values on the right side of the histogram. The shape of their histograms and also the results from the descriptive analysis evince the similarity with normal distribution (negative skewness and low values of kurtosis).
The LCOM data have a peak at zero; the rest of the histogram corresponds with the normal data distribution. The descriptive statistics suggest the LCOM data to be flat and non-skewed (the mean >= median >= mode does not hold here).
We can see high concentration of values around the mean in the RFC histogram. The results from the descriptive statistics show that the RFC data have relatively low values of the skewness and kurtosis. In the RFC data the relation mean >= median >= mode is not valid.
For these reasons, we will not analyze the data from these three metrics (DIT, LCOM and RFC) in regard to the existence of heavy-tail. The threshold values for these three metrics will be established from the mean and the standard deviation according to the method used in [6].

4.1 Heavy-tail analysis

The existence of heavy-tail was analyzed for the data from metrics CBO, NOC, NOM, SLOC1, SLOC2, and v(g). To determine the best probability distribution, we considered results from the EasyFit tool and the visual representation of the data probability functions. In figures 2 – 7 we can see the results of fitting function.
The data from the CBO metric can be characterized by generalized Pareto distribution. This data distribution belongs to heavy-tail probability distributions [12].

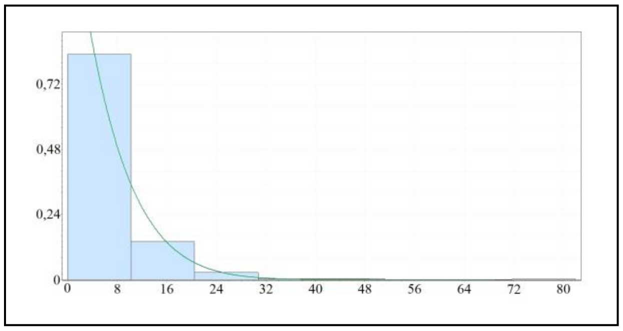


Figure 2. CBO with generalized Pareto distribution.

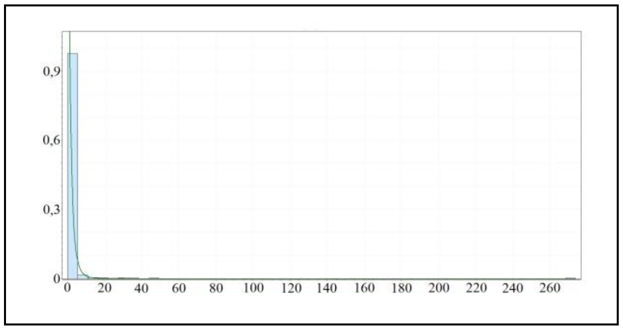


Figure 3. NOC with Weibull distribution.

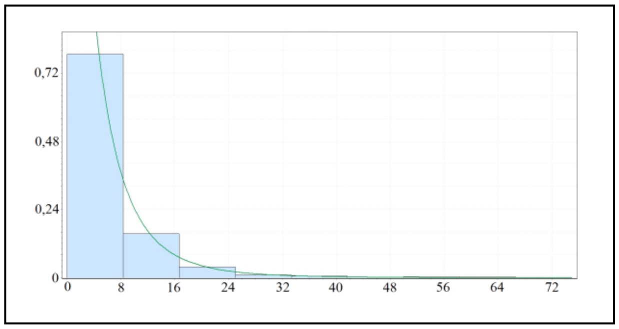


Figure 4. NOM with generalized Pareto distribution.

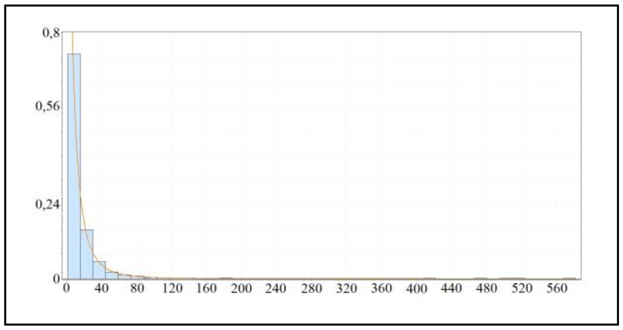


Figure 5. SLOC1 with lognormal distribution.

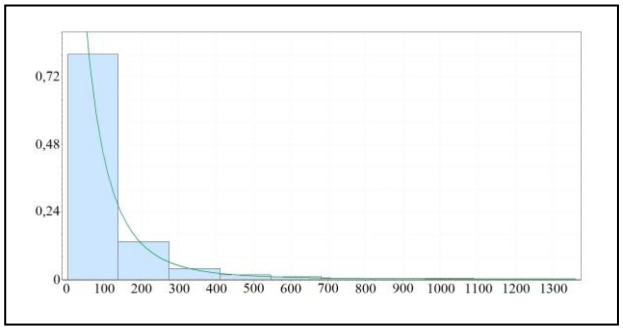


Figure 6. SLOC2 with generalized Pareto distribution.

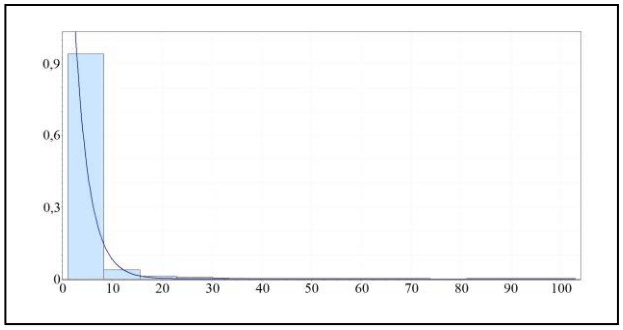


Figure 7. v(g) with exponential distribution.

This distribution also fitted data from NOM and SLOC2 metrics. The Weibull distribution characterized very well the data of the NOC metric. The Weibull distribution also belongs to heavy-tail probability distributions [3]. The data from the SLOC1 metrics were well fitted by the Lognormal distribution, while the data from the v(g) metrics fitted the Exponential distribution. Both of these distributions are considered to be heavy-tailed [13].

From these results we can see that all the data from the metrics CBO, NOC, NOM, SLOC1, SLOC2, and v(g) can be approximated by heavy-tailed distributions. According to [14], if data are heavy-tailed, their mean is not representative. Because of this, it is not convenient to use mean for establishing the threshold values. As a result, the thresholds values for these metrics were assessed according to the method used in [3] by separating values to three categories.

4.2 Identifying thresholds values for metrics with heavy-tail data distribution

We used the method form [3] for identifying thresholds values from our data. We established three categories for the metrics CBO, NOC, NOM, SLOC1, SLOC2, and v(g) with the names “good”, “regular” and “bad”. The values for these categories were drawn from data and histogram analysis.

The values of the CBO metric are most frequently spread in the range 0 – 15. Almost 90% of all values were lower than 10. It is clear to see from the plots that values between 16 and 25 were presented much less frequently, but still with better probability then values higher than 25. For the CBO metric the value ranges for the categories were designated as: 0 – 15 good, 16 – 25 regular, 25+ bad.

The most frequent value in the NOC metric is 0. From the definition of the NOC metric, this value represents classes with no children. There are almost 95% of all classes without children. Classes with 1 to 5 children occur much less frequently, but we can still see their presence in the plots. The probability of the occurrence of the class with more than 5 children is very low. For the NOC metric we assigned the three categories as: 0 good, 1 – 5 regular, 6+ bad.

The NOM metrics has the most values in the range 0 – 10. Almost 90% of all the values are positioned in this range. The values in the interval 10 – 25 are presented with lower frequency, but are still apparent in the plots. The values higher than 25 can be considered as rare. We designed the three categories accordingly: 0 – 10 good, 11 – 25 regular, 26+ bad.

The values of the SLOC1 metric are most frequent in the range 0 – 10. We can find approximately 90% of all the values in this interval. To the next category we can put the values in the range 11 – 69. The values higher than 69 can be find very seldom. We established the three categories as 0 – 10 good, 11-69 regular, 70+ bad.

The SLOC2 values are most frequently spread in the interval 1 – 200. The values can be found in this range with almost 95% probability. The values 201 – 400 are still evident in the data, but not so often as data from the previous category. The values over 400 are rare. The three categories are therefore: 1 – 200 good, 201 – 400 regular, 401+ bad.

The most frequent value in the data from the v(g) metric is 1 and probability of this value is almost 70%. Almost 25% of all the values from this metrics are in the range 2 – 15. The values over 15 occur very seldom. For the v(g) metric we established the three categories as: 1 good, 2 – 15 regular, 16+ bad.

4.3 Identifying thresholds values for metrics without heavy-tail data distribution

The data from the metrics DIT, LCOM, and RFC do not evince heavy-tail distribution. To identify thresholds values for these metric we used method showed in [6]. For each of these metrics we used results from the descriptive statistics – the mean and the standard deviation – for calculating the threshold values. The first threshold value corresponds with the mean and represents the most typical value. The second threshold value is calculated as a sum of the mean and the standard deviation. This second value represents high, but still acceptable set of values. The third threshold value is calculated as a multiplication of the second threshold value by the coefficient 1.5 [6]. The third threshold value is considered as an extreme and should not be present in the data.

The typical value (mean) of the DIT metrics is 2, the second threshold value corresponds to 4 and extreme value is 6. The LCOM metric has the typical value equal to 40, the high – but still not extreme – value is determined as 72. We calculated the extreme value of LCOM as 108. For the RFC the typical value is 34, the high value is 49 and the extreme value is considered to be 74.

5 SYSTEM EVALUATION

As we mentioned in the beginning, the results of static analysis can highlight potential bad manners in code and can help people interested in system maintaining to find possible threads in code. The results of the system measuring can show classes and pieces of code, which should be put on a review.

After we found thresholds of our system, we aimed at evaluating the source code in “the light” of determined thresholds values. We used MS Excel files with measured values and MS Excel functions for identifying classes, which should be reviewed, because they evince abnormal values.

We aimed only at the category “bad”, because classes with the worst evaluation could be the most harmful for the system. For the CBO metric, we identified 24 classes, which were in the category over 25. The NOC metric showed that 27 classes should be reviewed. Similarly, the NOM metrics detected 25 classes for review. Most classes for review showed the SLOC2 metrics – 35. From the non-heavy tailed metrics, the DIT identified 15 classes for review, while RFC identified 24. The metrics specialized in functions also detected possible problems – the v(g) metric detected 187 functions while the SLOC1 metric showed 212 functions. All these classes and functions were

reported as possible thread and passed to the persons responsible for system maintaining.
The calculated extreme value from the LCOM metric is 108. However, as we can see in the histogram of LCOM measured values, there is no class with the LCOM metric higher than 100. This result is influenced by the special shape of the data from LCOM metric. As we can see in the histogram of LCOM, the data are concentrated in the low values, but with higher values, they start to have characteristic of normal distributed data. Because of this behavior, the extreme values were identified as over the maximum value and there is no class which would overstep this threshold.

6 CONCLUSION

The goal of this study was to determine the threshold values for given metrics for safety critical system. We have been analyzing values of the CBO, DIT, LCOM, NOC, NOM, RFC, SLOC1, SLOC2, and v(g) metrics. Data for these metrics were collected with the use of the Understand C++ tool.
Our analyzed system does not correspond by its function and robustness to free and accessible application. Because of this, we could not compare our measured data with the data from other published studies nor with the data measured from any free-to-download software from the internet. In addition, the data from similar software are almost impossible to gather. For all these reasons we did not use statistical data from any other previous studies. Therefore the first research question is answered by our described approach only partially for the specific instance of having only one safety critical system and nothing to comparing it with.
We analyzed the measured data for the existence of heavy-tail and according to results we categorized our data into two groups. The first group contains metrics with potential heavy-tail data distribution, the second without it. With regard to the results from the descriptive statistics and histograms, we assigned the CBO, NOC, NOM, SLOC1, SLOC2, and v(g) metrics to the first group, leaving the DIT, LCOM, and RFC metrics to be classified into the second group. For both groups we establish the thresholds values. For the first group we identified the threshold values from the data and the plot characteristics by using method from [3]. The threshold values for the second group were determined from the descriptive statistics with the method used in [6]. The final results answering our second research question are shown in the table 3. We found that for each metrics there are no more than 2% percent of all classes in need of a review. This percentage seems to be relatively small, although the severity of the problems in identified classes will be the aim of the following deeper study.

Table 3. Identified thresholds of measured metrics. The good, regular and bad are valid for heavy-tailed data.

	Heavy tail	**Good / Typical value**	**Regular / High value**	**Bad / Extreme value**
CBO	yes	0 - 15	16 - 25	25+
DIT	no	2	4	6
LCOM	no	40	72	108
NOC	yes	0	1 - 5	6+
NOM	yes	0 - 10	11 - 25	26+
RFC	no	34	49	74
SLOC 1	yes	0 - 10	11 - 69	70+
SLOC 2	yes	1 - 200	201 - 400	401+
v(g)	yes	1	2 - 15	16+

7 DISCUSSION

In this study, we were evaluating special type of software system, which was written in C++ and does not correspond by its function and responsibility to standard computer software. For the threshold evaluation, the modern tools and techniques were used.
We identified threshold values for our system and then applied them to uncover possible unsecure parts of the code. The thresholds values of metrics detected in average 25 classes

and 200 functions, which should be reviewed. We advised these classes to the person responsible for system maintaining.

The threshold values for this type of system have not been published yet and thus our results can be used for comparison for any future evaluation of analogous software. We evaluated relatively decent amount of software metrics (9 metrics) and for the metric measuring, we used software with precisely described computation of these metrics.

Although we applied modern techniques to threshold identification, our study still should be viewed as evaluation of one software system and our results should not be taken as a dogma. Our techniques can be applied to another kind of software for which it is hard to gather data, but it is necessary to bear in mind, that the thresholds represent only local data and for general usage, the broader comparison should be used. We present our data as a base for possible future comparison with other studies and we hope our results can bring more light into the thresholds of safety critical software metrics.

As a future endeavor, we would like to study differences between metrics thresholds measured in this study and data from a system with at least some degree of similarity. We would like to compare our results with measurement of thresholds of open-source operational systems (which appear to be the most similar to our software).

ACKNOWLEDGEMENTS

The work was supported by the UWB grant SGS-2013-029 Advanced Computer and Information Systems.

REFERENCES

[1] S. Kaur, S. Singh and H. Kaur, “A Quantitative Investigation Of Software Metrics Threshold Values At Acceptable Risk Level,” International Journal of Engineering Research & Technology, vol. 2, March 2013, www.ijert.org.

[2] T.L. Alves, C. Ypma and J. Visser, “Deriving metric thresholds from benchmark data,” Software Maintenance, 2010 IEEE International Conference, vol. 1, pp.12-18, September 2010.

[3] K.A.M. Ferreira, M.A.S. Bigonha, R.S. Bigonha, L.F.O. Mendes and H.C. Almeida, “Identifying thresholds for object-oriented software metrics,” Journal of Systems and Software, vol. 85, pp.244-257, February 2012.

[4] R. Lincke, J. Lundberg, and W. Löwe “Comparing software metrics tools,” Proceedings of the 2008 International Symposium on Software Testing and Analysis, pp.131-142, July 2008.

[5] S. Herbold, J. Grabowski, S. Waack, “Calculation and optimization of thresholds for sets of software metrics,” Empirical Software Engineering, vol. 16, pp.812-841, December 2011.

[6] M. Lanza, R. Marinescu and S. Ducasse, Object-Oriented Metrics in Practice. Springer-Verlag New York, Inc., Secaucus, NJ, 2005.

[7] L.B.L. De Souza and M.D.A. Maia, “Do software categories impact coupling metrics?,” Proceedings of the 10th Working Conference on Mining Software Repositories, pp.217-220, May 2013.

[8] N.G. Leveson, Safeware: System Safety and Computers. ACM, New York, NY, USA, 1995.

[9] www.scitools.com/documentsmetricsList.php, cited 20.1.2014

[10] R. Shatnawi and Q. Althebyan, “An Empirical Study of the Effect of Power Law Distribution on the Interpretation of OO Metrics,” ISRN Software Engineering, vol. 2013, 2013.

[11] T. Tamai and T. Nakatani, “Analysis of software evolution processes using statistical distribution Models,“ Proceedings of the International Workshop on Principles of Software Evolution, pp.120-123, May 2005.

[12] R.W. Katz, “Do Weather or Climate Variables and Their Impacts Have Heavy-Tailed Distributions?” Proceedings of 13th Symposium on Global Change and Climate Variations (American Meteorological Society), 2001.

[13] J.C. Gardiner, “Modeling heavy-tailed distributions in healthcare utilization by parametric and Bayesian methods,” Proceedings of SAS Global forum 2012, paper 418, April 2012.

[14] P. Oliviera, H. Borges, M.T. Valente and H.A.X. Costa, “Metrics-based Detection of Similar Software,” International Conference on Software Engineering and Knowledge Engineering, pp.447-450, 2013.