Software Metrics Visualization

Vira Liubchenko
Department of Software Engineering, Odesa Polytechnic National University, Shevchenko Avenue 1, Odesa, Ukraine
lvv@op.edu.ua

Keywords: Metrics, Software, Visualization, Data, Diagrams, Analysis, Decision Making, Effectiveness.

Abstract: Software engineering is an empirical field of study. To support managerial and technical decision-making, the engineer needs numerical measures closely connected with different software metrics. Visual representation of numerical data improves the effectiveness of human data processing and shows insights that humans may miss. This paper aims to provide a systematic review of the approaches for software metrics visualization and define the possible recommendation for their use. The study is based on the literature review of the papers from two text collections – IEEE Xplore and ACM Digital Library – and the scientometric database Scopus. After merging and filtering, the final set of publications contains 16 papers. Our study showed that there were the metrics used significantly more often; among them are lines-of-code, cyclomatic complexity, coupling, and cohesion. We were not able to identify such leaders for visualization means. Instead, there was a tendency to combine different metrics on one chart or dashboard to provide the whole process picture. Based on the results of empirical studies reported in the literature, we offered an analysis of simple charts' properties and recommendations on their use for support decision-making in the software engineering process.

1 INTRODUCTION

It is well known that software engineering is an empirical field of study. Therefore, the numerical indicators are extremally essential and helpful for process monitoring and artifact evaluation.

Software metrics are an invaluable tool for measuring the progress and performance of software development projects. By providing a quantitative measure of the various aspects of a project, they can be used to inform decision-making, identify areas needing improvement, and track progress over time. Furthermore, software metrics are versatile and can be adapted to fit the needs of any given project. By carefully selecting the right metrics, software developers can gain insight into their project performance and make better-informed decisions.

However, each metric measures the distinctive feature or artifact of the development process. Therefore, the decision maker (e.g., architect, team lead, project manager) should simultaneously analyze the metrics set to get a realistic picture of the current situation in the development process.

To support decision-makers, we follow the idea of visualizing software metrics. Usually, the visualization can display a large amount of information in one chart, so the visualization can be used to present complex data dependencies typically found in software artifacts. As a result, the decision-maker can see the patterns, which allows the detection of known and unknown problems or opportunities in the software project.

This paper aims to provide a systematic review of the approaches for software metrics visualization and define the possible recommendation for their use.

To reach our goal, we focused on the two research questions:

RQ1: What are the most frequently used software metrics?

RQ2: What kind of visualization means are used most often?

The contributions of this study are twofold. First, we built a reference list of the most popular software metrics for visualization. Second, we gave a brief analysis of using visualization means.

The rest of the paper is structured as follows. First, in Section 2, we explain our study design and the methodology we followed. Section 3 outlines the publication trends of the papers we gathered. Section 4 presents the metrics and visualization means identified during our review and discusses our results. Some recommendations concerning chart use are considered in Section 5. Finally, we close the study by summarizing the conclusions in Section 6.

2 STUDY DESIGN

The study aimed to answer RQ1 and RQ2 based on the relevant papers. The steps of the search and selection process we followed are depicted in Figure 1.

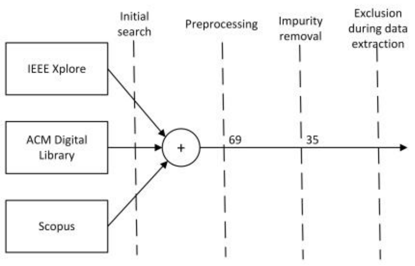

Figure 1: Papers selection methodology.

Initial search. We use two text collections – IEEE Xplore and ACM Digital Library – and the scientometric database Scopus as data sources for the study. The keywords for the queries were “software,” “metrics,” and “visualization” with connectors AND. We apply this query to all metadata at IEEE Xplore and ACM Digital Library and to “Article title, Abstract, Keywords” at Scopus.

Preprocessing. We formed the initial set of papers at this stage by merging the papers from three sources. Next, we removed the duplicates from the set. The issue of copies was caused by using the database Scopus; most of the papers found there had been collected in IEEE Xplore and ACM Digital Library. After the preprocessing, the papers set collected 69 papers.

Impurity removal. We performed an impurity removal per set of papers because many side papers matched the query but were irrelevant to the research purpose. We can distinguish irrelevant papers into three groups: concerned software structure visualization (tree structure hierarchy, software maps, trace messages, ontology-based visualization), presentation software statements and code coloring, and repository footprints. Some of these papers used the metrics but did not visualize them. We removed all these papers to make a coherent set of papers around our purpose. After removing them, we got 35 papers for the subsequent analysis.

Exclusion during data extraction. Working on the papers study, we found some papers that were not as relevant as expected. There were three types of such papers: presented the evolution of one framework and described the same concepts, introduced new metrics, and described an approach connected with building artificial entities (e.g., city, feather), which presented code and were based on metrics using. Therefore, we removed those papers from our final set. As one can see in the Reference Section, the final set of publications contains 16 papers.

3 PUBLICATION TRENDS

Now we present some publication trends found.

Figure 2 shows the distribution of publication years in the working set of papers obtained after impurity removal.

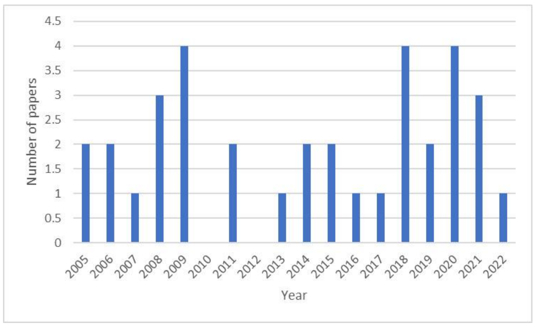

Figure 2: Distribution of publication years in the working set of papers.

For comparison, Figure 3 shows the distribution of publication years in the final set of publications.

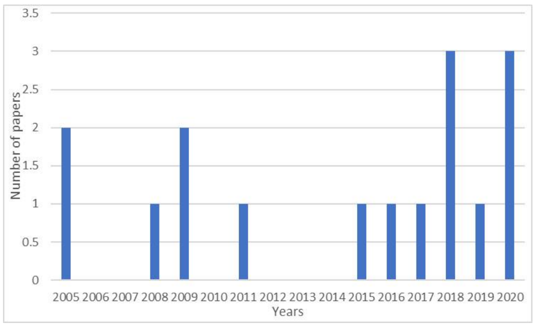

Figure 3: Distribution of publication years in the final set of papers.

We can see that the scientific interest in the topic was not very intensive. Although interest in empirical software engineering methods, based on metrics use, is consistently higher. In Figure 4, we show the statistics of Google Trends on request "software metrics" for the last three years.

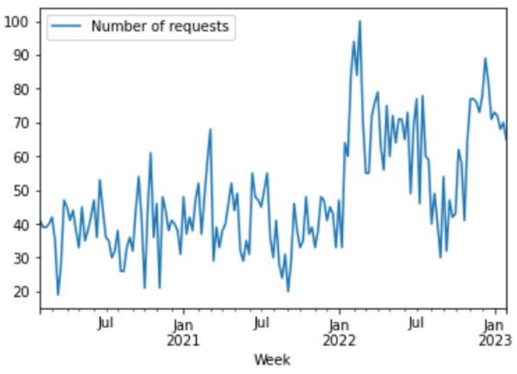

Figure 4: Interest over time to request "software metrics" (data from Google Trends).

However, we think the software metrics visualization issue was significantly underestimated. Different metrics reflect different aspects of the development process, and metrics understanding is a complex task for decision-makers. To get the whole picture, decision-makers should study many metrics simultaneously. Visualization can support and simplify the use and understanding of software metrics.

4 STUDY RESULTS

Our study showed that there were the metrics used significantly more often. We were not able to identify such leaders for visualization means. Let us present our results in detail.

4.1 Software Metrics

In the beginning, we should point out that often papers did not define the metrics precisely pointed only, e.g., "cohesion," when the figures were shown the concrete metrics. Some papers also presented a visualization approach in a metrics-independent way and did not mention concrete metrics. The distribution of metrics tackled in the papers is shown in Table 1.

Our study showed two prominent leaders of popularity: lines-of-code (LOC) and cyclomatic complexity. Many empirical studies demonstrated the dependency between LOC and the characteristics of the development process and software performance. Cyclomatic complexity is highly related to implementation and testing characteristics. We suppose these dependencies caused the high frequency of this metrics pair use.

Table 1: Software metrics tackled by the papers.

Metrics	Papers
Lines-of-code	[1] [2] [3] [4] [5] [6] [7] [8] [9]
McCabe's cyclomatic complexity	[2] [3] [5] [8] [9] [10]
Coupling and cohesion measures	[3] [4] [8] [10] [11] [12]
Fan-in, fan-out	[1] [3] [7] [8]
Depth of inheritance tree	[4] [8] [10]
Number of children	[4] [10]
Encapsulation measures	[4] [10]
Number of methods	[4]
Parameter number	[8]
Method length	[8]
Number of bugs	[6]
Passage rates of unit testing	[13]
Maintenance Index	[5]

We can see a focus on coupling and cohesion, as it is known that it is essential to keep low coupling and high cohesion in software design. But we should point out that both are generalizations of different metrics.

We can also see the attention to modularity connected with fan-in, fan-out, and encapsulation measures. As for coupling and cohesion, the generalization of different concrete metrics was used.

Most of the metrics relate to code features. However, researchers also pay attention to metrics specific to other stages of software engineering, e.g., testing and maintenance.

4.2 Visualization Means

Unlike software metrics, all papers clearly defined the type of visualization means they described. And unlike software metrics, there was no leader of popularity. The distribution of visualization means in the papers is shown in Table 2.

A large part of the working set of papers (the set after impurity removal) was dedicated to the issue of structure representations. Most often, the researchers used a dependency chart which reflects software structure with arc diagrams. To present the measures of software unit features, they used the size and color of nodes or provided the data in table form.

Table 2: Visualization means tackled by the papers.

Visualization means	Papers
TreeMap	[1] [6] [8]
Visual encoding	[4] [7]
Line chart	[5] [16]
Kiviat diagram	[14][15]
Rainbow-color mapped constant-size bars	[2]
Timeline and multilevel timeline	[3]
Metrics value distribution	[4]
HotSpot view	[8]
Scatter Plot	[8]
Toxicity Chart	[8]
Colored graph with variable size of nodes	[9]
Weighted digraph	[13]
Parallel coordinates plots	[10]
RadViz	[10]
Radial stacked bar chart	[11]
Zoomable circle packing graph	[12]
Bar chart	[16]
Punch card	[16]

A pretty exotic way of visualization was proposed in [2]. The approach combined the metric-lens technique (showing numerical values of metrics as colored bars) with UML diagrams (showing system structure). The authors claimed that the approach effectively helps understand the relations between metrics and structure at a finer level than the UML diagrams alone.

The significant issue concerning visualization is the need for simultaneous demonstration of different metrics. One of the proposed solutions was Kiviat diagrams. These diagrams are suited to present multivariate data, such as the feature vectors extracted from several source code releases and release history data. For similar purposes (visualizing source code metrics), Kiviat diagrams have also been used by related visualization approaches and tools.

The visual encoding approach combines different metrics as different parameters of geometry shapes. It supports the simplicity of the comparison between different software units. However, each researcher proposed their configuration. It does not look like a unified approach will be offered.

Simple line charts are reported not only as visualization means. For example, in [5], authors described line charts used for monitoring metrics from the time perspective and their prediction. In [10], the authors described using coordinates line plots multivariate software metrics (Figure 5) in parallel with the RadViz technique (Figure 6). Linking the two approaches was applied to detect outliers, which could indicate bad smells in software systems. They complement each other in identifying data patterns, clusters, and outliers.

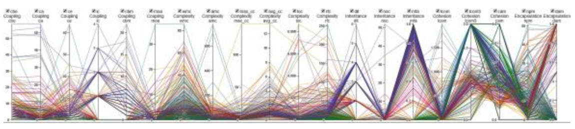

Figure 5: Parallel coordinates view [10].

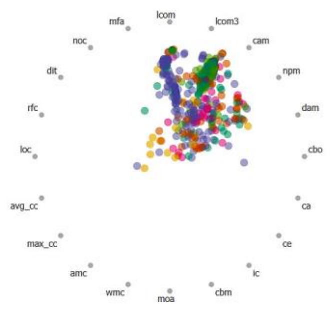

Figure 6: RadViz view of exploring noteworthy outlier patterns in detail concerning a focused set of metrics [10].

Finally, we should notice the idea of metrics dashboards in [8].

4.3 Discussion

There are many well-known software metrics. However, in most of the analyzed works, an attempt to combine some metrics was made. For example, two papers from the working set were devoted to studying the properties of indicators that combine several well-known metrics. As we already mentioned, many publications used visual encoding with geometric shapes, the parameters of which are determined by various metrics. In dependency charts, two metrics were also considered when designating vertices.

This confirms that a single metric is not valuable for analysis and decision-making because it reflects only one characteristic of an artifact or process. To provide a more comprehensive picture, multiple metrics need to be combined. It causes the search for different approaches to combine different metrics on one chart or dashboard.

Let us remember that analyzing software metrics is not an end; it supports engineering or management decisions. Software metrics are used in various applications, such as detecting low-quality code, finding design weaknesses, or estimating work progress. Therefore, the concept of a good visualization of software metrics depends mainly on their application.

In this context, it might be appropriate to consider structure diagrams in conjunction with metrics visualization. However, in our opinion, this approach does not cover all needs. Software types are becoming more and more diverse, which leads to the introduction of additional characteristics and, accordingly, additional metrics. For example, for software systems of artificial intelligence (AI-based software systems), such characteristics are interpretability, scalability, safety, fairness, staleness, etc. None of these metrics can be associated with the structural components of the system. However, for these characteristics, it can also be helpful to study them together and their behavior over time.

Software diversification leads to the introduction of new, specialized metrics requiring adequate visualization. For example, one uses Koopman Spaghetti Factor (*KSF*) for embedded software. *KSF* is calculated as

$$KSF = SCC + 5 \cdot Globals + SLOC / 20,$$

where *SCC* is the strict cyclomatic complexity, *Globals* is the global variables count, and *SLOC* is the number of non-comment source code lines.

As we can see, *KSF* combines three different metrics into a single metric. Therefore, it could be helpful while visualizing *KSF* to provide a granular representation of the three components of the metric.

Another example is model quality metrics in AI-based software systems. The common practice uses four metrics – accuracy, recall, precision, and F-score. Usually, their values for different models are shown in tabular form. It is necessary to compare the quadruples of metrics for different models to solve the problem of choosing the best or acceptable model. Under these conditions, using bar charts is more convenient than working with four numeric values from 0 to 1.

Summing up, we should note that metrics visualization is undoubtedly valuable for software engineering. Using metrics can make it possible to detect outliers and other deviations, predict the development of a process, and so on. Therefore, next, we provide an analysis of the charts' properties and recommendations on their use.

5 EFFECTIVE VISUALIZATION

Visualization capabilities of different charts may significantly affect the effectiveness of understanding and interpretation of the presented data by the decision-maker. Respectively, they also affect the quality and efficiency of decision-making. Note that software metrics are quantitative. Therefore, it is unnecessary to consider use cases for nominal and ordinal measurements.

In [17], there was pointed out that contextual information can serve as an essential input to developing and evaluating effective visualizations. The authors identified four principal contextual factors affecting visualization effectiveness: problem, stakeholder, purpose, and time.

The problem category concerns the problem situation to be supported and potential solutions. In the software metrics case, the problem is usually in the artifacts quality evaluation, efforts and bugs prediction, quality in use assurance, etc. It requires visualization to present the data clearly and comfortably. The stakeholder category involves any stakeholder-related aspects that affect the design of a visualization. In our case, the stakeholders are IT-friendly decision-makers who need easily readable and understandable information. The purpose category includes contextual information about what a visualization stakeholder is trying to achieve through applying the visualization in a particular domain. In our case, the purpose is to support decision-making by providing as much relevant information as can. The time category contains temporal information associated with decisional problems, stakeholders, and purposes. In our case, the interest in temporal information is restricted by prediction tasks only.

In the described context, using only the simplest charts, such as line charts, bar charts, scatterplots, or pie charts, is advisable. The focus of data presentation is mapping data values to graphical representations.

In [18], five recommendations were formulated. We analyzed their application for the software metrics visualization.

G1. Use bar charts for finding clusters. The bar charts have a better overall performance in terms of time, accuracy, and user preferences for finding clusters. Clustering is helpful, for example, for modularity evaluation. For such purposes, the alternative could be a scatterplot. But it restricts analysis to only two metrics simultaneously.

G2. Use line charts for finding correlations. The line charts performed better in terms of time, accuracy, and user preferences. As in the previous case, the possible alternative is the scatterplot, which restricts analysis by two metrics. Studying correlation with the line chart provides some additional effects; for example, [10] demonstrated the identification of bad smells in code with line charts.

G3. Use scatterplots for finding anomalies. The scatterplots have high accuracy and speed and are highly preferred by users for this task. For purposes of software development, it could be helpful for outlier detection. The crucial important issue is the choice of metrics pair. We can apply dimensional reduction for scatterplot drawing. However, we should remember the importance of a clear interpretation of visual representation. If the decision-maker needs sense interpretation for the outliers, it could be impossible in the artificial feature space.

G4. Avoid line charts for tasks requiring readers to precisely identify a specific data point's value. The fact that the axes' values were drawn at uniform intervals makes it difficult to identify the value of a specific data point precisely. Anyway, such tasks are usually not relevant for decision-making in software engineering.

G5. Avoid using tables and pie charts for correlation tasks. As it was noticed in G2, for the correlation study, the more appropriate option is line charts.

In [19], a comparative study was realized for parallel coordinates, scatterplot matrices, and tabular visualization. The evaluation demonstrated that tabular visualization was familiar, accurate, and time-efficient for the retrieve value task. However, we recommend using the bar charts for the retrieve value task because it is very close to the clustering task in the software engineering context.

6 CONCLUSIONS

Decision-making in software engineering has become increasingly complex, spurring the need for effective decision-support tools. Numerical data visualization is a simple, fast, and effective way to enhance decision-making. In this paper, we conducted a literature analysis on software metrics visualization. As a result of the study, we identified the most frequent metrics used, the trend of simultaneous visualization of multiple metrics, and the criteria for choosing chart types.

Furthermore, the development of specialized software systems—such as embedded software and AI-based software—has created a demand for specialized metrics and, consequently, for visualization tools capable of making sense of these metrics. As such, it is of utmost importance to formalize and study the properties of various visualization tools for software metrics.

ACKNOWLEDGMENTS

This work was sponsored by the Federal Foreign Office through the Stibet I program of the German Academic Exchange Service (DAAD).

REFERENCES

[1] M. Balzer, O. Deussen, and C. Lewerentz, “Voronoi treemaps for the visualization of software metrics,” in Proceedings of the ACM symposium on Software visualization (SoftVis '05), pp. 165-172, 2005.

[2] H. Byelas and A. Telea, “The metric lens: visualizing metrics and structure on software diagrams,” in 15th Working Conference on Reverse Engineering, pp. 339-340, 2008.

[3] A. Gonzalez, R. Theron, A. Telea, and F. J. Garcia, “Combined visualization of structural and metric information for software evolution analysis,” in Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops (IWPSE-Evol '09), pp. 25-30, 2009.

[4] R. Francese, M. Risi, and G. Scanniello, “Enhancing software visualization with information retrieval,” in 19th International Conference on Information Visualisation, pp. 189-194, 2015.

[5] B. Popović, A. Balota, and D. Strujić, “Visual representation of predictions in software development based on software metrics history data,” in 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 352-357, 2016.

[6] T. Brunner and Z. Porkoláb, “Two-dimensional visualization of software metrics,” in Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, pp. 2:1-2:6, 2017.

[7] M. Alnabhan, A. Hammouri, M. Hammad, M. Atoum, and O. Al-Thnebat, “2D visualization for object-oriented software systems,” in International Conference on Intelligent Systems and Computer Vision (ISCV), pp. 1-6, 2018.

[8] J. Slater, C. Anslow, J. Dietrich, and L. Merino, “CorpusVis - visualizing software metrics at scale,” in Working Conference on Software Visualization (VISSOFT), pp. 99-109, 2019.

[9] G. Lacerda, F. Petrillo, and M. S. Pimenta, “DR-Tools: a suite of lightweight open-source tools to measure and visualize Java source code,” in IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 802-805, 2020.

[10] H. Mumtaz, F. Beck, and D. Weiskopf, “Detecting bad smells in software systems with linked multivariate visualizations,” in IEEE Working Conference on Software Visualization (VISSOFT), pp. 12-20, 2018.

[11] A. Yusuf and M. Hammad, “An approach to automatically measure and visualize class cohesion in object-oriented systems,” in International Conference on Decision Aid Sciences and Application (DASA), pp. 1174-1179, 2020.

[12] A. Yusuf and M. Hammad, “An automatic approach to measure and visualize coupling in object-oriented programs,” in International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT), pp. 1-6, 2020.

[13] Y. Muto, K. Okano, and S. Kusumoto, “A visualization technique for unit testing and static checking with caller–callee relationships,” Journal of Convergence, vol. 2(2), pp. 1-8, 2011.

[14] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, “Visualizing multiple evolution metrics,” in Proceedings of the ACM symposium on Software visualization (SoftVis '05), pp. 67-75, 2005.

[15] A. Kerren and I. Jusufi, “Novel visual representations for software metrics using 3D and animation,” in Software Engineering, J. Münch and P. Liggesmeyer, Hrsg. Bonn: Gesellschaft für Informatik e.V., 2009, pp. 147-154.

[16] S. R. Humayoun, S. M. Hasan, R. AlTarawneh, and A. Ebert, “Visualizing software hierarchy and metrics over releases,” in Proceedings of the International Conference on Advanced Visual Interfaces (AVI '18),” Article 40, pp. 1-5, 2018.

[17] X. Bai, D. White, and D. Sundaram, “Context adaptive visualization for effective business intelligence,” in 15th IEEE International Conference on Communication Technology, pp. 786-790, 2013.

[18] B. Saket, A. Endert, and Ç. Demiralp, “Task-Based Effectiveness of Basic Visualizations,” in IEEE Transactions on Visualization and Computer Graphics, vol. 25(7), pp. 2505-2512, 2019.

[19] G. J. Quadri and P. Rosen, “A survey of perception-based visualization studies by task,” in IEEE Transactions on Visualization and Computer Graphics, vol. 28(12), pp. 5026-5048, 2022.