Assessment of Geant4 Maintainability with respect to software engineering references

Assessment of Geant4 Maintainability with respect to software engineering references

Elisabetta Ronchieri[1], **Maria Grazia Pia**[2], **Marco Canaparo**[1]

[1] INFN CNAF, Bologna 40126, Italy
[2] INFN, Sezione di Genova, Genoa 16146, Italy

E-mail: elisabetta.ronchieri@cnaf.infn.it, mariagrazia.pia@ge.infn.it, marco.canaparo@cnaf.infn.it

Abstract. Over time computer scientists have been provided metrics to measure software maintainability. In existing literature, a large number of references can be found about this topic; nevertheless, a lack of quantitative assessment of maintainability metrics has been observed. In this paper, we summarize the challenges of adopting code measurements in the context of physics software system. In this pilot study, we have used Geant4 - a twenty-year-old software system - to conduct this research and set the grounds for further discussion.

1. Introduction

Maintaining software is a widely acknowledged issue according to software manufactures. Existing literature provides information about tools and processes that can facilitate the production of maintainable code (see e.g. the IEEE International Conference on Software Maintenance, ICSM). In our study, we have observed that there are many proposals that contribute to the challenge of measuring maintainability; however, to the best of our understanding, a universal agreement of its measurement is ongoing.

In literature, computer scientists use the term maintainability in two ways: the former is descriptive, the latter derives from source code measurements. Amongst the various definitions we choose the one of IEEE Standard Glossary [1] according to which: "maintainability is the ease with which a software system or component can be modified to correct false, improve performance or rather attributes, or adapt to a changed environment." Such definition lacks details about its estimation and measurement. Many researchers have tried to define maintainability through different types of metrics [2] (such as Maintainability Index [3] and Halstead source code [4]), whose measures are often validated by using expert judgements and empirical considerations. Therefore, it is a bit of a challenge to use these metrics in order to make suitable decisions on the base of their measurements.

Maintainability is a software characteristic that is going to be assessed in various contexts including scientific applications. In this pilot study, we use Geant4 [5, 6] - a twenty-year-old software system - to conduct this research and set the grounds for further discussion. Geant4 is an ideal playground to assess maintainability of large scale high energy physics software systems over the range of decades. Due to the amount of data, this paper provides a sample of results whose discussion is going to be published in a journal.

The remainder of this paper is structured as follows. Section 2 details the challenges that we have identified to fulfil this research. Section 3 explains the methodology we have used to realize this study, whilst Section 4 shows a sample of results. Finally, Section 5 concludes.

2. Challenges

In the software development life cycle, maintainability is the most expensive factor that consumes more than 40 to 70 percent of time and resources [7, 8]. Existing literature suggests that the control of software maintenance begins at the earliest point of the life cycle [9,10]; however, it is uneasy to perform this activity over the development phase [11]. For example, as the complexity level of a piece of software increases over its life cycle, the code becomes difficult to understand and, therefore, uneasy to comprehend and more likely to contain errors, making its maintenance non-trivial [12].

Software metrics, when correctly defined, implemented and used, have been shown to be valuable indicators of quantitative evaluation of software complexity [13–15]: on one hand, they can identify improper integration of functional enhancements; on the other hand, they can predict the error-prone parta of software. In order to use metrics successfully, we have to face challenges that are explained in the following sections.

2.1. Employment of metric analysis techniques

By exploiting statistics we can identify different analysis techniques that employ metric values to highlight error-prone code. In the following we list a subset of these techniques.

Basic statistics concepts, such as means and standard deviation, contribute to getting a global picture of the code status.

Threshold analysis classifies metric values setting ranges on them according to the level of worrying caused by the metric. This activity entails a deep metric knowledge and adequate range interpretation.

Prediction equation determines which parts of the software may need attention by using a substantial amount of error data and code.

Trend analysis identifies patterns in a series of data to predict a trend analysing past events in the code.

Inequality analysis, such as Gini index [16], may determine the gap between good and bad code.

Their employment in the development life cycle requires: learning and choosing the proper technique; having available a large amount of measures; having a deep knowledge of the language and code context.

2.2. Usage of automated tools for metric collection and analysis

During the development life cycle, it is recommended using a tool that supports the following three activities: parsing the source code, generation of the metric values and assistance of the interpretation of results.

Nowadays, several commercial and free tools (such as Imagix 4D [17] and CLOC [18]) are able to fulfil the first two activities. However, existing tools often implement a different subset of metrics, leading scientists to use more then one tool or to choose a commercial version. Furthermore, metrics can be not unequivocally defined, favouring different interpretations and consequently different implementations, which lead to non-comparable values from distinct tools.

The last activity is the most difficult to include in the development life cycle, since existing tools (such as WebMetric [19]) - to the best of our knowledge - may help in the interpretation of the metric values, but they may not support the chosen analysis technique.

2.3. Selection of the right set of metrics

In order to properly assess code, it is important to select the right set of metrics. Typically, they include the most popular metric known as Line Of Code (LOC) that measures the size of software products [20,21]. In addition to that, the chosen set of metrics envelops object oriented metrics to obtain size, coupling, cohesion and inheritance information [21–23]; moreover, there is also the Cyclomatic Complexity metric that evaluates the difficulties in maintaining a given code and the likelihood in producing errors [24]. Table 1 shows the most popular metrics to measure maintainability.

Table 1. The most popular metrics to measure software maintainability

Metrics	Source
Chidamber & Kemerer's Metrics: Coupling Between Object (CBO), Depth of Inheritance Tree (DIT), Lack of Cohesion in Method (LCOM), Number of Children (NOC), Response for a Class (RFC), Weighted Method Count (WMC)	[22,25,26]
Data Abstraction Coupling (DAC), Message Passing Coupling (MPC), Number of Local Methods (NOM), Number of Attributes and Methods (NAM), Length of Class Names (LCN)	[21,26,27]
Locality of Data (LD), Improvement of LCOM (ILCOM)	[28]
Halstead's Metrics: Program Vocabulary, Program Length and Program Volume; Tight Class Cohesion (TCC); Number of errors detected by code inspection; Number of code changes required; Information Flow Metric; Cyclomatic Complexity; Number of Catch block per Class (NCBC); Exception Handling Factor (EHF); Lines of Code (LOC)	[29–31]
Metric MOOD: Method Hiding Factor (MHF), Active Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF) and Coupling Factor (CF); Martin Agile Suite metrics: Number of Classes (NC), Afferent coupling (Ca), Efferent Coupling (Ce), Instability (I) and Distance (D)	[22]

Selecting the right set of metrics is a challenging task due to the high number of possibilities that can address a given issue. Furthermore, the employed tool may not include or interpret them properly. On the other hand, the analysis technique has to consider possible correlation between each pair of metrics in order to avoid bias.

2.4. Usage of historical data

Metric analysis technique have to be developed and refined by using historical data code. For example, prediction equation may rely on error data, whose collection may be particularly challenging. Generally speaking, recording this information during the development process is relatively cheap and produces large benefits: this activity is part of guidelines for good programming that can be recommended at managerial level but they cannot be imposed to developers.

The quality of data is essential. They can make the metric analysis techniques robust as much as data are accurate and detailed. For this reason, it is extremely important to validate both data provided by the selected tool and by the developers over time.

2.5. Interpretation of metric values

As shown in Table 1, metrics deal with software complexity. This may help developers in ranking and addressing error-prone software, while it is developed.

Accepting a metric value may be dangerous if done without a rationale. In fact, to judge a certain metric value, we have to understand what the metric attempts to measure at least in general terms; otherwise, we risk to make a wrong assessment.

Drawing conclusions from the value of a single metric maybe considered a mistake. In fact, a metric may exceed its limits without being regarded as a problem, and to assess software maintenance we should use multiple merics in order to determine where to put effort.

3. Methodology

To perform this maintainability assessment, we have defined a methodology, which is designed to integrate maintainability into software as it is being developed. The methodology aims at developing a maintainable system by highlighting the most complex code that needs deeper attention. In the following, we describe the rectangles in Figure 1 that summarize our methodology:

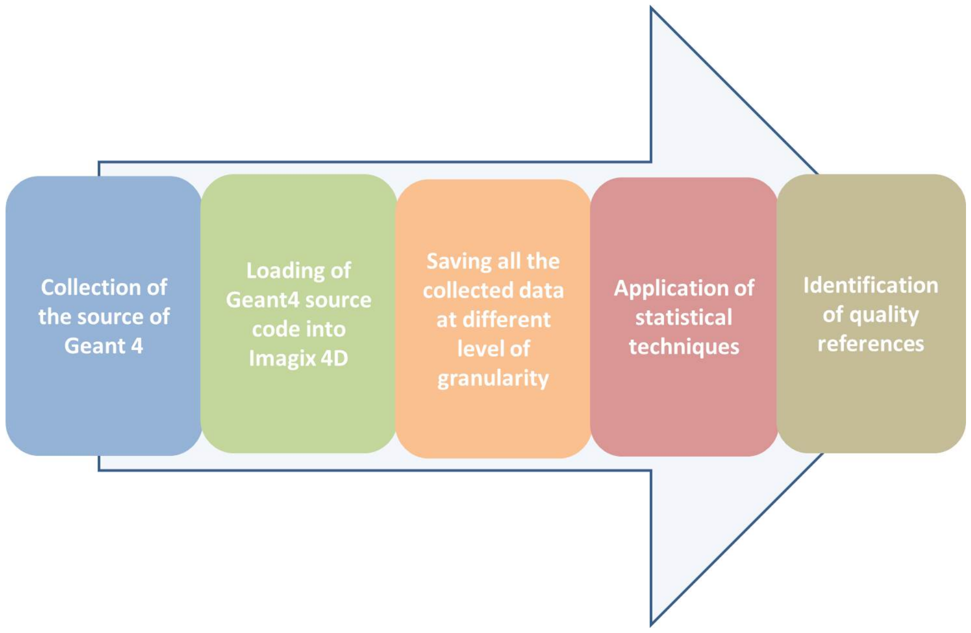
Collection of the source of Geant 4
Loading of Geant4 source code into Imagix 4D
Saving all the collected data at different level of granularity
Application of statistical techniques
Identification of quality references

Figure 1. Research methodology right arrow

- collection of the source code of all Geant4 [5, 6] versions from 0 to the current one 10.2;
- loading of the Geant4 source code into Imagix 4D version 8.0.4 to measure a large number of metrics in order to obtain code information with respect to size, complexity, coupling, inheritance, cohesion, complexity, control-flow structuredness (see Tables 2, 3 and 4);
- saving all the collected data at different levels of software granularity, such as file, function, class, directory, variable and namespace;
- application of statistical techniques for the analysis of metric values [32];
- identification of quality (goodness ranges of) references with respect to size, complexity, coupling, inheritance, cohesion, complexity, control-flow structuredness that derive from relevant peer-reviewed papers, conference proceedings and technical reports [33] (see Table 5).

During this study, we have considered 32 Geant4 versions: this means looking at ∼150 packages, ∼3000 classes, ∼6000 files and ∼30000 functions.

Tables 2, 3 and 4 show some metrics that we have measured by using Imagix 4D tool that is able to provide various metrics at different levels of software granularity: 24 metrics at class level, 22 metrics at file level, 21 metrics at directory level, 21 metrics at function level, 4 metrics at variable level. Due to the amount of metrics, in these Tables, we have summarized a subset of these metrics with respect to size, complexity and object orientation categories.

Size metrics [34] are typically a direct count of selected characteristics to describe the volume of a software, estimating software productivity and quality.

Complexity metrics, such as McCabe [35], Halstead [4] and Welker [36] metrics, measure the simplicity of the system design.

Object-oriented metrics [22] measure complexity, maintenance and clarity; they estimate to which extent the system adheres to the object orientation.

Table 2. Size Category

Level of granularity	Metric	Source
File	Comment Ratio	[34]
	Declarations in File	[34]
	File Size	[34]
	Functions in File	[34]
	Lines in File	[34]
	Lines of Source Code	[34]
	Lines of Comments	[34]
	Number of Statements	[34]
	Variables in File	[34]
Function	Lines in Function	[34]
	Lines of Source Code	[34]
	Variables in Function	[34]

Table 3. Complexity Category

Level of granularity	Metric	Source
File, Function, Class	Halstead Intelligent Content	[4]
	Halstead Mental Effort	[4]
	Halstead Program Volume	[4]
	Halstead Program Difficulty	[4]
File, Class	McCabe Average Complexity	[35]
	McCabe Maximum Complexity	[35]
	McCabe Total Complexity	[35]
File	Maintainability Index	[36]
Function	McCabe Cyclomatic Complexity	[35]
	McCabe Decision Density	[35]
	McCabe Essential Complexity	[35]
	McCabe Essential Density	[35]

Table 4. Object-Oriented Category

Level of granularity	Metric	Source
Class	Class Cohesion (LCOM)	[22]
	Class Coupling (CBO)	[22]
	Depth of Inheritance (DIT)	[22]
	Number of Children (NOC)	[22]
	Response for Class (RFC)	[22]
	Weighted Methods (WMC)	[22]

Table 5. A Sample of Quality References

Acronym	Reference	Source
Comment Ratio	0.08	[37]
SLOC (Source Lines Of Code)	60 at file level	[37]
HPV (Halstead Programme Volume)	1500 at function level	[37]
	[100,8000] at file level	[38]
	> 800 too many things at file level	[38]
	[20, 1000] at function level	[38]
	> 1000 too many things at function level	[38]
MI (Maintainability Index)	<65 poor maintainability	[39]
	[65, 84] fair maintainability	[39]
	≥85 excellent maintainability	[39]
MCMCC (McCabes Maximum Cyclomatic Complexity)	[1, 10] low cyclomatic complexity	[40]
	[11, 15] medium cyclomatic complexity	[40]
	[16,30] high cyclomatic complexity	[40]
	>31 very high cyclomatic complexity	[40]
	[1, 10] low cyclomatic complexity	[37]
	[11, 20] medium cyclomatic complexity	[37]
	[21. 50] high cyclomatic complexity	[37]
	>51 very high cyclomatic complexity	[37]

Once validated all measurements, we have passed them to statistical tools in order to perform a correct interpretation. Concerning the analysis techniques, we are employing those described in Section 2. We are also started considering economic and ecology indexes to obtain information on the concentration of errors in the code [32]. Table 5 shows a sample of quality references whose selection is detailed in [33].

4. A sample of plots and results

In the course of this work, we have sifted through ∼6000 files of Geant4 in order to get potential issues in the code. Due to the complicated problems implemented in Geant4, it is not a trivial task to determine code that is really complex to maintain. Furthermore, according to our understanding, an approach based only on one metric may be considered not completely correct. It will be necessary to analyse the results of a combination of more than one metrics in order to get a more precise figure of the code's status. For this reason, we currently apply trend and inequality techniques to assess software quality metrics. As regards inequality measures, we invite readers to check a dedicated paper [32].

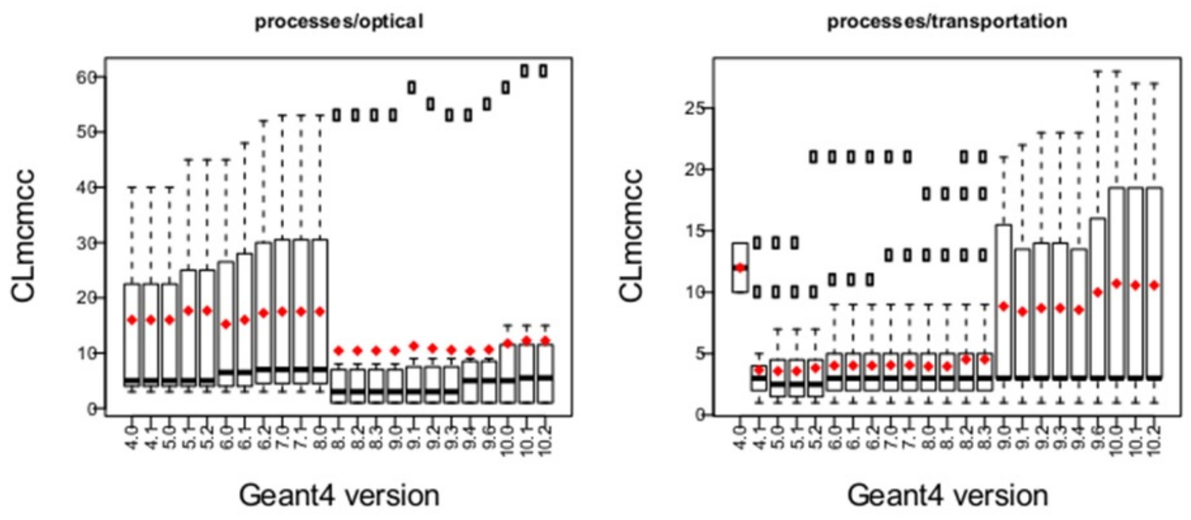

Figure 2. Maximum McCabe Cyclomatic Complexity at class level.

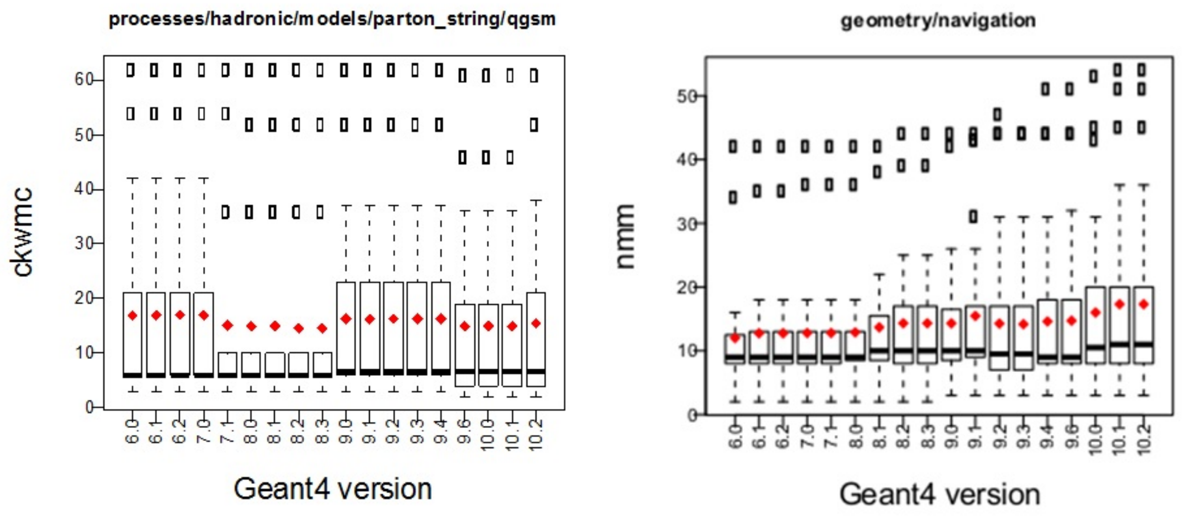

Figure 3. WMC and NMM at class level.

In this Section, we present the trend results for a small set of Geant4 packages and metrics. Such technique contributes to identifying how software characteristics have evolved over time.

Figure 2 shows the trend of McCabe Maximum Cyclomatic Complexity at class level for the processes optical package and the processes transportation package. Figure 3 shows the trend of WMC and NMM metrics at class level for one of the processes package and the geometry package.

5. Conclusion

The use of metrics can contribute to monitoring the internal quality of software. Further investigation is in progress to identify appropriate ranges of metric values for the Geant4 packages by using statistical methods. More extensive results will be discussed in a forthcoming full paper.

Acknowledgment

The authors thank Francesco Giacomini for interesting discussions and INFN CCR for partly funding this work. We also thank the Imagix Corporation that provided an extended free full license of Imagix 4D for performing this work and CERN library for providing papers and books.

References

[1] IEEE, *report IEEE Std 610.12-1990, IEEE*, 1990.
[2] Coleman D, Ash D, Lowther B and Oman P, *IEEE Computer*, vol 27, issue 8, 1994.
[3] SEI Software Technology Review, `http://www.sei.cmu.edu/`
[4] Halstead M H, *Elsevier Science*, 1977.
[5] S. Agostinelli et al., *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol 506, n 3, pp 250–303, 2003.
[6] Allison J et al., *IEEE Transactions on Nuclear Science*, vol 53, n 1, pp 270–278, 2006.
[7] Soi I, *Microelectronic Reliability*, vol 51, n 2, pp 223–228, 1985.
[8] Jabri M, Jerbi H and Braiek N B, *International Review on Modeling and Simulations*, vol 3, n 1, pp 38–47, 2010.
[9] Yau S S and Chang P S, *IEEE Conference on Software Maintenance*, pp 374–381, 1988.
[10] Dubey S and Rana A, *ACM SIGSOFT Software Engineering Notes*, vol 36, n 5, pp 1–7, 2011.
[11] Munson J B, *IEEE Computer Software and Applications Conference*, vol 51, n 2, pp 223–228, 1985.
[12] Belady L A and Lehman M, *IBM System's Journal*, n 3, pp 225–252, 1976.
[13] Conte S D, Dunsmore H E and Shen V Y, *The Benjamin/Cummings Publishing Company, Inc.*, 1986.
[14] Kafura D and Reddy R R, *IEEE Transactions on Software Engineering*, vol SE-13, n 3, pp 335–343, 1987.
[15] Wake S and Henry S, *IEEE Conference on Software Maintenance*, pp 382–387, 1988.
[16] Gini C, *Tipografia di Paolo Cuppini*, Bologna, 1912.
[17] Imagix, Online. Available: `https://www.imagix.com/products/source-code-analysis.html`
[18] CLOC, Online. Available: `https://github.com/AlDanial/cloc`
[19] Scotto M, Sillitti A, Succi G and Vernazza T, *Studia Informatica Universalis*, pp 90–99, 1996.
[20] Melo W and Abreu B, *Proceedings of the 3rd International Symposium on Software Metrics*, pp 90–99, 1996.
[21] Elish M, Yafei A A and Al-Mulhem M, *Advances in Engineering Software*, vol 42, pp 852–859, 2011.
[22] Chidamber S R and Kemerer C F, *IEEE Transaction on Software Engineering*, vol 20, pp 476–493, 1994.
[23] Barkmann H, Lincke R and Lowe W, *International Conference on Advanced Information Networking and Applications Workshops*, pp 1067-1072, 2009.
[24] Sastry B and Saradhi M, *International Journal of Engineering Science and Technology*, vol 2, n 2, pp 67–76, 2010.
[25] Stol K, Babar M, Avgeriou P and Fitzgerald B, *Information and Software Technology*, vol 53, pp 1319–1336, 2011.
[26] Dalal J and Briand L, *Simula Technical Report (2009-1)*, version 2, n 1, 2009.
[27] Chen Z, Zhou Y and Xu B, *Proceedings of the Internal Conference on Software Maintenance*, pp 377–384, 2002.
[28] Li W and Henry S, *Journal of System Software*, vol 23, pp 111–112, 1993.
[29] Weyuker E, *IEEE Transactions on Software Engineering*, vol 14, n 9, pp 1357–1365, 1988.
[30] Sharma A, Kumar R and Grover P, *Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, Corfu Island, Greece*, pp 24–29, 2007.
[31] Hitz M and Montazeri B, *Proceedings of International Applied Cooperate Computing (ISAAC'95)*, 1995.
[32] Pia M G and Ronchieri E, *Proceedings of IEEE NSS 2016*, 2016.
[33] Ronchieri E and Canaparo M, *Proceedings of ICSOFT-EA 2016*, pp 232–240, 2016.

[34] Lorenz M and Kidd J, *Prentice-Hall*, 1994.
[35] McCabe T J, *IEEE Transaction on Software Engineering*, vol SE-2, n 4, pp 308–320, 1976.
[36] Welker K D, *The Journal of Defence Software Engineering*, CrossTalk, pp 18–21, 2001.
[37] McCabe Software, Online. `http://www.mccabe.com/pdf/McCabe\%20IQ\%20Metrics.pdf`.
[38] Verifysoft Technology, Online. Available: `http://www.verifysoft.com/en_halstead_metrics.html`
[39] Coleman D, Lowther B and Oman P, *Journal of Systems and Software*, vol 29, n 1, pp 3–16, 1995.
[40] CppDepend, Online. Available: `http://www.cppdepend.com/`.