Integrating Software Metrics with UML Class Diagrams

Metin Turan

Abstract—Software measurement tools have been used to support quality assurance through the development cycle with an increasing popularity recently. However, understanding the results of these tools still is a big problem. They generally present polymetric views which are too complex and hard to comment on.

In this study, a software tool was developed to present metrics in UML class diagrams. Colors of traffic lights are used to colorize the status of units in classes. Status is determined by techniques called limit and point approaches. Tool includes common structured and class based complexity metrics and only parses C# grammar presently.

Finally a survey is applied to computer engineering students taken software engineering course. The obtained feedback is affirmative and useful for further steps.

Index Terms—Software metrics, software quality, software tool, UML.

I. INTRODUCTION

Software tools facilitate to meet software quality targets. If software developers had a reliable tool, then they would determine the deviation and take care as soon as possible. Generally, early realization saves your money and time. But the most importantly, you have the chance to understand your product quality.

Software tools could be categorized by different ways. Most of them are used as development IDE. Some of them produce source code from your design. Few of them translate your existing code to a higher version or another programming language. Rest of them helps to estimate cost or schedule of a project. They all guide us make our work easily. But, if you want to measure software artifact quality, then you need software metric tools through the software cycle.

Software metrics research has been started at early of 1970's. Computers had limited resources although the size of codes increased rapidly. Programmers tried to find an efficient way to executing bigger code on the same memory size. What could affect the size of an executable? Size and productivity metrics have been discovered. Then using the other physical characteristics of software, taking into consideration of the software modeling, lots of metrics have been emerged. Some of these metrics were painful to measure. As a result, they have not been commonly used and accepted by developers.

When big software projects have been emerged by 1990's, software metrics have gained importance again. Object-oriented programming, aspect oriented programming

and service-oriented programming models reveal the importance of software metrics. On the other hand, it has been a research area to adopt existing structured programming metrics to new programming models and finding new ones. However, the proposed software metrics up to date have been practically discovered and theoretically proof remains weak. Although there are information [1]-[3], graph [4]-[6], lattice [7], communication [8]-[10] and other [11]-[13] theoretic researches, this stands still a handicap for commonly usage of software metrics.

There are many metric tools available in the market (PMD with Eclipse) and most of them are available on the Internet (Squale, Sonar or Moose). They present metrics in polymetric views [14], [15]. The types of the metrics they support are differentiated [16]. Tools generally support a few programming languages, not all.

Software products generally suffer from complexity. This leads to failure of software product in a short-term life. Sometimes, software may need refactoring in the case of performance lack or requirement changes [17], [18].

A metric tool which shows the complex code segments of a project developed by using C# programming language on the UML class diagram has been developed. The research has been divided into two sub-projects (Fig. 1). First sub-project aim is to calculate and save metrics in an XML file. Second sub-project uses this XML file to represent selected metrics on an UML class diagram.

The parsers used in project described at Section II. Quality metrics used are explained at Section III. XML design strategy and UML presentation is given at Section IV and Section V respectively. Application user interface is picturized at Section VI. Survey result is given at Section VII. Finally, the conclusion and future works are discussed at Section IX and Section X respectively.

II. WORK

A. Parsers

Project has been developed for C# projects initially. Gold and CM parsers have been selected between lots of free parsers after a deeply research. Gold parser gives only tokens and types of these tokens for any programming language grammar. Gold parser has been used for calculating line counts, comment counts and other countable metrics of software for this reason. However, CS parser is a utility which parses the C# source code and creates a tree of the code nested within the other. CS Parser has been preferred for calculating object oriented metrics.

B. Metrics

Metrics are measure of some property of a piece of software or its specifications. Since quantitative methods

have proved so powerful in the other sciences, computer science practitioners and theoreticians have worked hard to bring similar approaches to software development.

Software metrics provides a means of measuring software both under development and after a system is deployed. However, software is an abstract construct and measuring is really hard.

Fig. 1. Schema of metric evaluation.

The approved software complexity metrics up to now have been generally related to the product metrics. The rest have been related to the other programming models like OOP (Object-Oriented Programming).

The metrics listed at Table I have been selected for the first version of the software from the recommended metric suits [19]-[21].

New metrics at Table II has been added to the project finally [22]-[25].

TABLE I: FIRST VERSION METRICS

Metric
RFC (Response for a class)
MPC(Message passing coupling)
Fan-In
Fan-Out
McCabe Cyclomatic Complexity
Halstead Metrics
Lines of Code
Comment Percentage
Total Function Call(Fan out)

TABLE II: FINAL VERSION ADDED METRICS

Metric
Lack of Cohesion of Methods
Cohesion Ratio
R0,R1,R4 Coupling [16]
Depth of Inheritance Tree
Number of Children
Weighted Methods per Class
Average Class Complexity
Number of Methods per Class
Number of Attributes per Class
Information Flow Complexity
Class of Lines

C. XML Design

XML structure is designed to support data representation in UML class diagram. Tag names are selected fastidiously. Final XML structure is generic and free to extend.

```

<?xml version="1.0" encoding="utf-8" ?>
- <exp>
- <Class>
  - <Properties>
    <Name>mainFrm</Name>
    <Path>C:\Sources\Form1.cs</Path>
    <Scope>Public</Scope>
    <Inherits>Form</Inherits>
    <IsPartial>True</IsPartial>
    <IsStatic>False</IsStatic>
  </Properties>
  - <Metrics>
    <RFC>0</RFC>
    <MPC>0</MPC>
  - <FanInReferencedMethods>
    <ClassName />
    <MethodName />
    <Count />
  </FanInReferencedMethods>
  - <FanOutReferencedMethods>
    <ClassName />
    <MethodName />
    <Count />
  </FanOutReferencedMethods>
    <DepthOfInheritanceTree>0</DepthOfInheritanceTree>
    <NumberOfChildren>0</NumberOfChildren>
    <WeightedMethodPerClass>6</WeightedMethodPerClass>
    <AverageClassComplexity>2</AverageClassComplexity>
    <InformationFlowComplexity>0</InformationFlowComplexity>
  - <NumberOfMethodsPerClass>
    <Total>3</Total>
    <External>0</External>
    <Internal>3</Internal>
    <Static>0</Static>
  </NumberOfMethodsPerClass>
  - <NumberOfAttributesPerClass>
    <Total>0</Total>
  </NumberOfAttributesPerClass>
    <CohesionRatio>0</CohesionRatio>
    <LackOfCohesion>0</LackOfCohesion>
  - <Coupling>
    <R0>0</R0>
    <R1>0</R1>
    <R4>0</R4>
  </Coupling>
    <Coupling />
    <ClassLineCount>20</ClassLineCount>
  </Metrics>
- <Method>

```

Fig. 2. XML class design.

The Fig. 2 shows the class tag structure of an example XML data. Class is tagged with *Class* tag and includes properties, metrics and methods. *Properties* tag contains name, path, scope, inherited classes and C# specific information partial or static. *Metrics* tag includes class based metrics given at Section III.

The Fig. 3 shows the *Method* tag structure of an example XML data. A method is tagged with *Method* tag and includes properties and metrics. *Properties* tag includes name, scope, return type and parameters of the method. Each parameter is also defined in *Parameters* tag individually with parameter properties. Metrics for methods includes structure based

metrics like size, Cyclomatic Complexity and Halstead.

D. UML Presentation

The UML class diagram has ability to show the dependency and inheritance relations between the classes referred in the given sources. Referenced fan-in and fan-out metrics are also included in the UML diagram. The UML class diagram drawn in Fig. 4 is evaluation of an example class.

```
<?xml version="1.0" encoding="utf-8" ?>
- <exp>
- <Class>
+ <Properties>
+ <Metrics>
- <Method>
- <Properties>
  <Name>openToolStripMenuItem_Click</Name>
  <Scope>Private</Scope>
  <ReturnType>Void</ReturnType>
- <Parameters>
- <Parameter>
  <Name>sender</Name>
  <Type>Object</Type>
</Parameter>
- <Parameter>
  <Name>e</Name>
  <Type>Unknown</Type>
</Parameter>
</Parameters>
</Properties>
- <Metrics>
  <LineCount>4</LineCount>
  <CommentPercentage>0</CommentPercentage>
  <TotalFunctionCall>3</TotalFunctionCall>
  <CyclomaticComplexity>3</CyclomaticComplexity>
  <ForLoopCount>0</ForLoopCount>
  <ForeachLoopCount>1</ForeachLoopCount>
  <WhileLoopCount>0</WhileLoopCount>
  <IfStatementCount>1</IfStatementCount>
  <SwitchStatementCountNode>0</SwitchStatementCountNode>
  <HalsteadLength>0</HalsteadLength>
  <HalsteadProgramVolume>0</HalsteadProgramVolume>
  <HalsteadProgramLevel>NaN</HalsteadProgramLevel>
  <HalsteadProgramLevelEquation>0</HalsteadProgramLevelEquation>
  <HalsteadIntelligentContent>0</HalsteadIntelligentContent>
  <HalsteadPotentialVolume>0</HalsteadPotentialVolume>
  <HalsteadEffortEquation>NaN</HalsteadEffortEquation>
  <HalsteadTimeEquation>0</HalsteadTimeEquation>
</Metrics>
</Method>
+ <Method>
+ <Method>
</Class>
</exp>
```

Fig. 3. XML method design.

TABLE III: SURVEY OF EXPERIENCE

Question	Yes	No	Percentage
Is the developed tool easy and comprehensible?	14	3	%82
Would the developed tool be useful in a software course?	12	5	%70
Is it useful to detect deficiency in a software system using class diagram integrated with software metrics?	11	6	%64

In the literature threshold values for software metrics has not been determined clearly yet. This leads to misunderstanding or misuse in the case of evaluation

software metrics. The web site of NASA is a well-known resource to apply for that kind of reference [26]. But, only the threshold values for Line Count and Cyclomatic Complexity are defined to be 200 and 10 respectively. This is a big drawback for evaluation.

On this point we suggest a new approach and applied for evaluation. First of all, all software systems have different complexity (functional analysis supports that) due to the subject or application area. Moreover although the problem was same the analysis and design strategy would cause different complexity. As a result, each system is different theoretically and this leads to evaluate software systems independently. Each has own characteristics and deviations these must be evaluated separately. Deviations must be handled in the evaluated system comparing with other units not by another system or general values.

Based on the previous idea, software metrics are applied to the software units. The average metric value of all units is calculated to get a threshold value which shows above or under success. This is a point approach. On the other hand, the point may oscillate under or above by standard deviation. This is a limit approach. Former one is more restricted. In an Object-Oriented Software System, we suggest using point approach for class based metrics and limit approach for method based metrics.

Traffic lights are selected to colorize status of software units after measurement. Green, yellow and red colors show that metric value is very good, acceptable but need attention, and unacceptable respectively. Point approach contains only red and green values. However, limit approach has yellow values which show the values that are acceptable. For the current version, the metrics used for colorization are limited to some class based (*RFC*, *MPC*, *Fan in Referenced Methods*, *Fan out Referenced Methods*) and method based (*Line Count*, *Comment Percentage*, *Total Function Call*, *Halstead Length*), metrics.

It should be realized that proposed technique is separated from the metric lens and other approaches which produces information from the UML diagrams [27]-[29].

E. Application

The captured application main screen (Fig. 5) points out that usage is simple. It is composed of three parts. The left side shows the previously created projects. The middle part shows the list of the class files included in the project. The right side shows the metrics exist in the project. The metrics are grouped meaningfully and user may select metrics checking a metric group or individually one by one. On the above of the middle part an operational menu resides which includes opening an existing project, creating new project and including a new class file into the project options.

Calculate Metrics button is used to calculate selected metrics on the source files. When metrics calculated they are stored at a local database table and an explorer window opens automatically to show the data in XML format. User can save results pressing *Export to XML File* button. User may calculate different metrics for each experiment and give them a meaningful file name to evaluate later.

Every project is stored at the file system with a special subfolder using the project name. All resources (project class

files and XML output) are saved there. When application starts then all previous projects are reloaded. User may select one of them or open another one. User can add new class files to the previous projects and recalculate metrics.

III. EVALUATION

The tool is experienced by seventeen students (they were

registered to software engineering course). Students set free to select a couple of source code written by them in C#. They applied the tool and observed the UML diagram presented.

The following survey questions were asked and the answers were registered. The statistical results of survey are presented at Table III.

Fig. 4. UML presentation.

Fig. 5. User interface.

IV. CONCLUSION

In this project a method for detecting defective units of a software product and presentation technique has been

proposed and developed by a metric tool.

Polymetric views are graphical visualization of software enriched by metrics(system complexity, class blueprint). However, it is too complex and condensed representation to

understand even for an experienced software specialist. We suggested a simple UML presentation enriched with metrics and colorized defective units to be noticed easily. It is experienced by students and the results are acceptable.

It is realized that metrics chosen for a tool is important to describe and report software product. Metrics should be grouped into some templates, such as quality, complexity and maintenance. This is a drawback for researches because standards have not been accepted yet.

On the other hand, decision about which metric value is under the limit is nearly impossible. There is some values exist in the literature, but they are generally empirical and not accepted by community. The suggested method in this article is a solution due to the nature of a software product. Every software product could be evaluated by its internal measures.

V. FURTHER WORK

This project handles only C# grammar presently. The well-known other language grammars (C typed languages Java and C++ especially) could be integrated by the time.

The metrics selected in this project calculates the complexity of the software units. Project needs to be extended include some templates, such as quality, complexity and maintenance.

UML representation should be extended to cover more graphical abilities.

REFERENCES

- [1] M. H. Halstead, *Elements of Software Science*, Elsevier Science Inc., 1977.
- [2] B. Ramamurthy and A. Melton, "A synthesis of software science measures and the cyclomatic number," *IEEE Trans. Soft. Eng.*, vol. 14, pp. 1116-1121.
- [3] W. J. Hansen, "Measurement of program complexity by the pair," *ACM SIGPLAN Notices*, 1978, pp. 29-33.
- [4] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, pp. 308-320, 1976.
- [5] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE Trans. Software Eng.*, pp. 1284-1288, 1991.
- [6] T. J. Emerson, "A discriminant matrix for module cohesion," *Int. Conf. Software Eng.*, IEEE Computer Society Press, 1984, pp. 294-303.
- [7] N. E. Fenton and R. W. Whitty, "Axiomatic approach to software metrication through program decomposition," *Computer J.*, pp. 330-339, 1986.
- [8] B. Henderson-Sellers and D. Tegarden, "Clarification concerning modularization and McCabe's cyclomatic complexity," *Comm. ACM*, 1994, pp. 92-94.
- [9] W. Harrison and K. Magel, "Atopological analysis of computer programs with less than three binary branches," *ACM SIGPLAN Notices*, 1981, pp. 51-63.
- [10] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Trans. Software Eng.*, pp. 510-518, 1981.
- [11] M. Shepperd, "Early life-cycle metrics and software quality modules," *Inf. Software Technol.*, pp. 311-316, 1990.
- [12] J. M. Bieman and N. C. Debnath, "An analysis of software structure using a generalized program graph," *Procs. COMPSAC 85*, pp. 254-259, IEEE Computer Society Press, 1985.
- [13] J. C. Munson and T. M. Khoshgoftaar, "Measurement of data structure complexity," *J. Systems and Software*, pp. 217-225, 1993.
- [14] O. Nierstrasz, S. Ducasse, and T. Girba, "The story of Moose: An agile reengineering environment," *Procs. ESECFSE*, 2005, pp. 5-9.
- [15] M. Lanza and S. Ducasse, "Polymetric views- A lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, pp. 782-795, 1993.
- [16] J. C. Munson and T. M. Khoshgoftaar, "Applications of relative complexity metric for software project management," *J. Systems and Software*, pp. 283-291, 1990.
- [17] E. I. Oviedo, "Control flow, data flow and program complexity," in *Proc. COMPSAC 80*, pp. 146-152, 1980.
- [18] S. Ducasse and M. Lanza, "The class blueprint: Visually supporting the understanding of classes," *IEEE Transactions on Software Engineering*, pp. 75-90, 2005.
- [19] E. F. Norman and L. P. Shari, *Software Metrics*, PWS Publishing, 1997.
- [20] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Trans. Software Eng.*, vol. 7, pp. 510-518, 1991.
- [21] S. Chidamber and C. Kemerer, "Towards a metric suite for object-oriented design," in *Proc. OOPSLA '91*, pp. 197-211, Sigplan Notices, 1991.
- [22] S. Henry and W. Li, "Metrics for object-oriented systems," Workshop on Metrics for Object-Oriented Software Development, 1992.
- [23] D. P. Tegarden and S. D. Sheetz, "Effectiveness of traditional software metrics for object-oriented systems," *HICSS-92*, pp. 359-368, 1992.
- [24] B. Henderson-Sellers and J. M. Edwards, *Booktawo of Object-Oriented Knowledge: The Working Object*, Prentice Hall, 1994.
- [25] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice Hall, 1994.
- [26] Metrics Data Program. (December 16, 2004). NASA IV&V Facility. [Online]. Available: <http://mdp.ivv.nasa.gov/>
- [27] H. Byelas and A. Telea, "Visualization of areas of interest in software architecture diagrams," in *Proc. ACM SoftVis*, 2006, pp. 105-114.
- [28] M. Termeer, C. Lange, A. Telea, and M. Chaudron, "Visual exploration of combined architectural and metric information," in *Proc. VISSOFT*, 2005, pp. 21-26.
- [29] H. Byelas and A. Telea, "The metric lens: Visualizing metrics and structure on software diagrams," in *Proc. 15th Working Conference on Reverse Engineering*, 2008, pp. 339-340.

Metin Turan was born in İstanbul, in 1967. He received B.Sc and M.Sc degrees from the Department of Computer Sciences of Hacettepe University in Ankara in 1990 and 1993 respectively. He is currently a doctorate student at the Department of Computer Engineering of Yıldız Technical University.

He worked as a system analyst, software specialist, project manager and software manager positions more than 15 years in industry. His last job was at the Department of Computer Engineering of İstanbul Kültür University. He worked as a lecturer and taught Int. to programming, object-oriented programming, data structures, visual programming, web programming, software engineering and programming languages courses more than 8 years. His major research areas are software metrics, software engineering, document summarization, text mining and algorithms. He is currently working as a consultant for software houses.

He is a member of IRED and IACSIT societies. He has published 8 technical papers in national and international journals.