Calculation and Visualization of Software Product Metrics

Matthias Meyer*
Software Engineering Group
Department of Computer Science
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
mm@uni-paderborn.de

Jörg Niere
Software Engineering Group
Department of Computer Science
University of Siegen
Hölderlinstr. 3
57068 Siegen, Germany
joerg.niere@uni-siegen.de

ABSTRACT

The paper presents a further step of the Fujaba Tool Suite RE to support coarse-grained analyses based on metrics and especially polymetric views. Polymetric views are graphical representations of certain metric combinations. Following an interactive reverse engineering approach, polymetric views can be created on demand. The reverse engineer is able to define new polymetric view descriptions and create new views afterwards.

1. INTRODUCTION

Software product metrics are one opportunity to perform coarse-grained analyses. Metrics such as lines of code, number of attributes or methods of a class, lack of cohesion or depth of inheritance hierarchies ([3, 4, 5, 7]) allow for producing quantitative analysis results of a software system. A combination of different metrics allows to draw conclusions such as problematic or high influencing system parts. To overcome the flood of numbers produced by the metrics, Lanza proposes in [6] a graphical representation of the metric combinations. So-called polymetric views are an ideal means to get a first impression of a system.

The Fujaba Tool Suite RE is a collection of reverse engineering tools based on the Fujaba Tool Suite [10] and several Fujaba plug-ins. The Fujaba Tool Suite RE allows for parsing Java source code into an Abstract Syntax Graph (ASG) representation, which serves as central repository to all further analyses. Currently the Tool Suite RE consists of static and dynamic analysis techniques to recognize implementations of patterns [8], such as design patterns or antipatterns [1]. The techniques allow for performing a fine-grained analysis of a system. Coarse-grained analyses are also possible, but produce too many uncertain results.

In order to support also fast and reliable coarse-grained analyses, we extended the Fujaba Tool Suite RE with two plug-ins. The first plug-in, called MetricsCalculation and described in Section 2, offers the calculation of several object-oriented software product metrics. The second plug-in PolymetricViews, described in Section 3, allows for viewing the metric results calculated by the first plug-in in polymetric views as introduced by Lanza. The paper closes with some future work issues.

*This work is part of the FINITE project funded by the German Research Foundation (DFG), prj-no. SCHA 745/2-2.

2. METRICS CALCULATION

Before the MetricsCalculation plug-in is able to calculate software product metrics for certain model elements, the system to be analyzed has to be parsed into the Abstract Syntax Graph (ASG) representation. Therefore the MetricsCalculation plug-in uses the JavaAST plug-in and the JavaParser plug-in. The ASG comprises UML elements such as classes, attributes and methods as well as elements corresponding to classical syntax trees such as literals or assignments. Whereas the UML elements are used to represent declaration parts, the other elements are used to represent method bodies. In the following we call the whole representation the model of the source code.

Each metric has a unique acronym, e.g. LOC which stands for Lines Of Code or NOC, Number Of Children, which is the number of direct sub classes of a class. The user may select the metrics to be calculated from the list of all supported metrics (cf. Table 1). Each metric value together with its acronym is stored in a separate result object that is linked to the corresponding model element via Meta-Model Integration (MMI) pattern [2]. The plug-in offers to present all results in a table.

2.1 Contributing a new metric

Far more metrics exist than currently are supported by the plug-in. Therefore, the plug-in was designed to be easily extended by new metrics. Each metric can be calculated for a particular type of model element. LOC, for example, is calculated for a method, the metric WLOC computes the lines of code for a whole class, and the number of children (NOC) is calculated on classes as well. For each metric, a calculator class exists which takes a model element of the appropriate type as input and calculates its metric value. All calculator classes implement a common interface. The data about available metrics is stored in an XML file. The file contains an XML element for each metric, which besides the unique acronym contains a name, a description, and the fully qualified name of the calculator class. Thus, in order to contribute a new metric, a calculator class that implements the common interface must be developed and an element describing the metric has to be added to the XML file.

2.2 Metric thresholds

For each metric, the user may additionally configure a threshold. If the metric value of a model element exceeds this threshold, an annotation is created which connects the model element and the result object (again via MMI pattern). Currently, the annotations are only visible in class

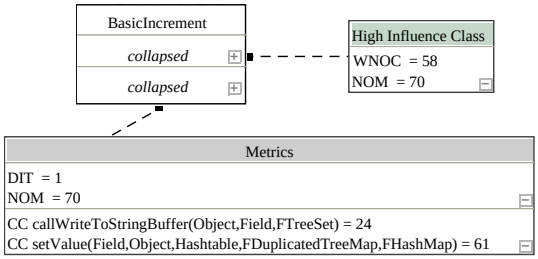

Figure 1: A class diagram with metric annotations.

diagrams. The shape of an annotation is a rectangle labeled with "Metrics". The rectangle is connected to the shape of the class containing the annotated model element (cf. Figure 1). In addition, if a class contains several model elements with metric annotations, only one annotation which is a union of all metric annotations is shown.

The results shown in Figure 1 were produced by first parsing the source code of the *basic*, *asg*, and *uml* packages of Fujaba (including sub packages). Afterwards, the metrics DIT, NOM and CC were selected to be calculated and annotated with thresholds 0, 20, and 20, respectively. The results show that the class BasicIncrement is at the first level in its inheritance hierarchy (DIT = 1) and defines 70 methods (NOM = 70). It has two methods with cyclomatic complexity (CC) 24 and 61, respectively. All other methods have CC values lower than 20. According to [9], CC values from 11 to 20 are still acceptable whereas higher values should be avoided.

2.3 Metric combinations

Single metric values are sometimes not very expressive. The fact, for example, that BasicIncrement has 70 methods, is not too edifying but not too interesting either. However, BasicIncrement has also a rather high number of all descendant classes (WNOC), i.e. 58. The 58 subclasses of BasicIncrement inherit its 70 methods which means that BasicIncrement has a high influence in its inheritance hierarchy. Note that the WNOC value was calculated when only the source files in the *basic*, *asg*, and *uml* packages of Fujaba (including sub packages) were parsed. The WNOC value of BasicIncrement would be much higher if Fujaba had been parsed completely.

To detect combinations of certain metric values, we offer the specification of boolean expressions over the values calculated by (possibly different) metrics for the same model element. A metric combination for high influence classes could be specified as e.g. (WNOC $\geq$ 10) & (NOM $\geq$ 20). In metric combination expressions the logical operators AND (&), OR (|), and NOT (!) may be used to join an arbitrary number of terms, possibly nested with parentheses. Each term may compare a metric value, represented by its acronym, with a number. As comparison operators ==, $<$, $\leq$, $>$, and $\geq$ are supported.

If a metric expression evaluates to true, an annotation is created and linked to the respective model element. The annotations created by metric combinations are displayed on class diagrams as well. They are also rendered as rectangles labeled with a name for the combination, e.g. High Influencing Class (cf. Figure 1), and linked to the class containing the annotated model element. In addition, the annotation shows the values of all metrics involved in the combination.

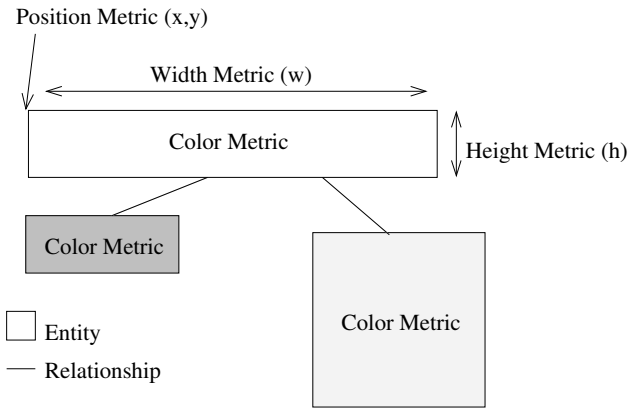

Figure 2: Up to 5 metrics can be visualized for one entity. In addition, entities may have relationships that do not carry metric values.

3. METRICS VISUALISATION

Metrics are numbers representing facts. We use class metrics such as number of methods (NOM) or number of attributes (NOA) or method metrics such as number of parameters (NOP). At the end of the previous section we argue that combinations of metric values are more expressive than a single value. We used a certain combination to give some hints to interesting parts in a software system, i.e. high influence classes. This approach has some drawbacks. We firstly have used absolute values and secondly the values are based on experience. Whereas normalization is a solution for the first problem, the second still remains, namely the definition of what are runaways in the context of the actual software system to be analyzed. In [6] Lanza presents Polymetric Views, which provide a visual representation of metric combinations. Detecting runaways is done manually by a developer looking at the produced pictures. Starting from the work presented in [6], we developed a Fujaba plug-in to visualize metric values as polymetric views.

3.1 Polymetic Views

Polymetric views are two-dimensional graphs containing nodes and sometimes edges, which are arranged in a certain layout. Nodes represent entities of the analyzed software system and edges represent relationships between the entities. Each node is a rectangle and can carry up to 5 metric values depending on the certain layout, whereas edges do not carry any metric information. Figure 2 illustrates the 5 potential metrics of an entity:

- **Size:** Either the width and height of a node (w,h) can represent a metric value. Both values have to be greater than 0.
- **Color:** The color of a node may also represent a metric. Valid values are one of 256 gray-scale values from black to white.
- **Position:** The position of a node (x,y) are the forth and fifth possible metric values of an entity. This assumes that the used layout allows for positioning nodes freely.
- **Layout:** Lanza proposes five major layout algorithms: Tree, Scatterplot, Histogram, Checker, and Stapled. Our prototype currently provides Tree and Checker layout.

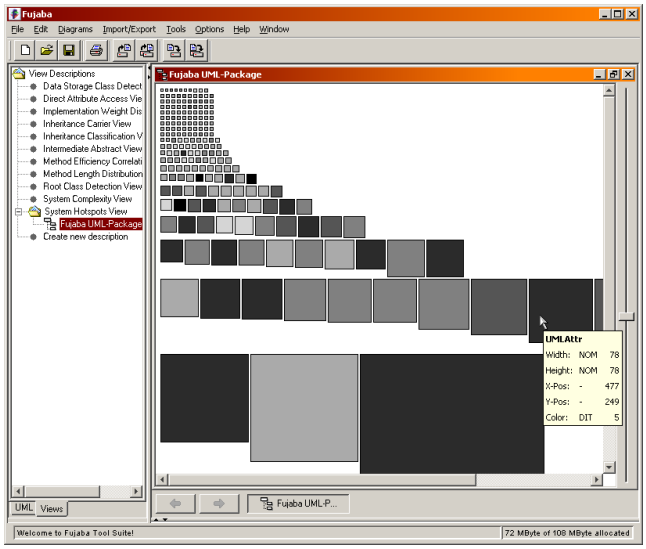

Figure 3: System-Hot-Spot view of the de.uni-paderborn.fujaba.uml package.

> The first considers relationships as hierarchical order to arrange nodes and the second places nodes with the same metric value in the same row or column.

Absolute metric values may be problematic in polymetric views. High metric values used for the size or the position of a node may result in views where only a limited part is visible on the screen. For example a white colored node with a width and height larger than 3-5 times of the screen size may result in a white screen showing the inner part of the node only. Thus, a developer is not able to get an overview of all nodes in the view. To solve this problem, we use a mapping function that maps metric values to values better suited for the screen-size. Mainly the mapping function scales absolute metric values. To let the developer see all entities even the ones with metric values of 0, we add a constant value (minimal node size) to the scaled metric values. Both values, the zoom factor as well as the minimal node size value are interactively changeable for a certain view.

3.2 Prototype

Entities of a polymetric view correspond to parts of the software system currently under investigation. A certain polymetric view is an instance of a polymetric view description on a certain part of the software, called polymetric view context. Hence we currently focus on coarse-grained analysis to get a first impression of the software, the context is a set of classes. Our prototype consists of a dialog to select the context based on the current package structure of the software.

Figure 3 shows the prototype of the PolymetricViews plug-in. The project tree on the left hand side shows all currently available polymetric view descriptions. The current view is the System-Hot-Spot view of the core meta-model classes of Fujaba located in the *uml* and all sub packages.

Each of the 203 entities in the polymetric view corresponds to one class. The width and height of a node carry the number of methods of the class (NOM) and the color carries the class' depth in the inheritance tree (DIT). The checker layout arranges the nodes according to their size in ascending order. Due to the 2-column layout of this paper, each line has a fixed number of 10 nodes, except the last line.

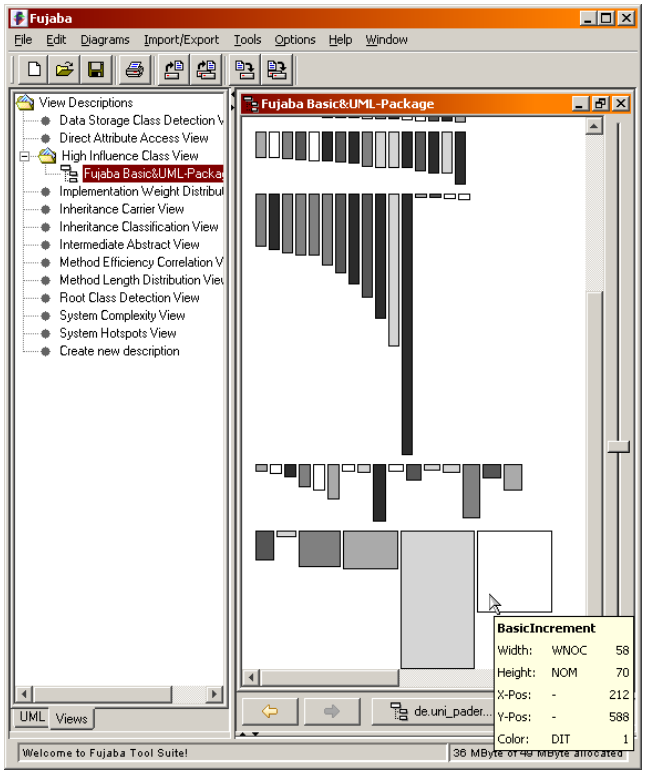

Figure 4: High-Influence-Class view example.

The zoom value can be modified with the slider on the right hand side. Going over a node, the appearing tool tip note shows the absolute metric values of the entity. For example, the UMLAttr class has 78 methods and 4 super classes in the inheritance tree.

What is the interpretation of this polymetric view? The meta-model classes of Fujaba have usually small interfaces, i.e. number of methods. The UMLAttr entity is at a depth of 5 in the inheritance tree. The black color of the UMLAttr entity indicates that this is also the maximum depth of the tree. Furthermore, there is no really dominating color, which would indicate an inheritance level with many classes. More problematic is the rightmost node in the last line. The node that corresponds to the UMLClass class is nearly double sized compared to the next smaller one, which means the entity has nearly twice the number of methods. Such classes in general need further analysis. In this case we observed that UMLClass is the central part of the meta-model of Fujaba and has many associations to other classes in the meta-model. Hence we originally generated the class and map associations to access methods, it has a huge number of association access methods. Thus it would be better to count only non-access methods. Unfortunately software product metrics are inappropriate to classify methods in that way. For this purpose, the already existing pattern instance detection is better suited.

3.3 Defining Polymetric Views

Lanza proposes 12 polymetric views organized in 3 categories for a coarse-grained analysis of a system. The first category is titled *First Contact Views* such as the *System Hot Spot* view shown in Figure 3. The second category *Inheritance Assessment Views* contains views to analyze the

inheritance structure. Views in the third category *Candidate Detection Views* detect entities that need further analysis.

In Section 2 we have proposed a metric combination of the number of methods (NOM) metric and the number of all descendant classes (WNOC) metric. Classes with metric values (WNOC $\geq$ 10) & (NOM $\geq$ 20) got a *High Influencing Class* annotation. Non of Lanza's polymetric views offers this metric combination, thus we have to define a new one.

The PolymetricViews plug-in allows the developer to dynamically define new view descriptions. A new polymetric view description consists of the following parts:

- the assignment of metrics to the size, position and color of an entity.
- a layout that arranges the entities and relationships.
- a factory that provides entities and relationships.

After the developer has defined the new polymetric view description, new views can be created. For example, Figure 4 shows the above described combination of the NOM and WNOC metrics of classes. The largest sized node is the BasicIncrement class. To our surprise the second largest sized node corresponds to class UMLIncrement that is located 2 inheritance levels below the BasicIncrement class. The UMLIncrement entity has a WNOC value of 56 that is 2 less than the BasicIncrement entity but much more methods, i.e. 195. We can not make the statement that this is ugly design, because the methods in UMLIncrement may override the ones in BasicIncrement. To strengthen an ugly design statement we have to make further investigation, perhaps enhance the new polymetric view description with the number of overridden methods (NMO) metric.

4. FUTURE WORK

Primary future work is the seamless integration of the metric analysis techniques into the overall reverse engineering process supported by the Fujaba Tool Suite RE. In particular, the existing pattern instance recognition and the metrics calculation will be integrated in such a way, that certain metric values or combinations may be used as triggers for fine-grained analyses with the pattern recognition. Furthermore, pattern specifications will be enabled to require that the metric values calculated for certain model elements do (not) exceed specific threshold values. The polymetric views will be used to determine those threshold values for the actual system to be analyzed.

Acknowledgments

We thank Lukas Roth and Jens Falk who implemented the MetricsCalculation and the PolymetricViews plug-ins, respectively, as part of their bachelor theses.

5. REFERENCES

[1] W. Brown, R. Malveau, H. McCormick, and T. Mombray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, Inc., 1998.

[2] S. Burmester, H. Giese, J. Niere, M. Tichy, J. P. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool integration at the meta-model level within the fujaba tool suite. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(3):203–218, Aug. 2004.

Acronym	Short description	Scope
ADIT	Attribute Depth of Inheritance Tree	Attribute
AvgCC	Average cyclomatic complexity	Class
AvgNLA	Average number of local accesses	Class
CC	McCabes cyclomatic complexity	Method
DIT	Depth of inheritance tree	Class
LCOM	Lack of cohesion in methods	Class
LOC	Lines of code in method	Method
MDIT	Method depth of inheritance tree	Method
NAM	Number of abstract methods	Class
NBLD	Nested block depth	Method
NCV	Number of class variables	Class
NI	Number of invocations	Method
NIA	Number of inherited attributes	Class
NIV	Number of instance variables	Class
NLA	Number of local accesses	Attribute
NMA	Number of methods added	Class
NMAA	Number of accesses on attributes	Method
NME	Number of methods extended	Class
NMI	Number of methods inherited	Class
NMO	Number of methods overridden	Class
NOA	Number of attributes	Class
NOC	Number of children	Class
NOCL	Number of classes	Project
NOINT	Number of interfaces	Project
NOM	Number of methods	Class
NOP	Number of parameters	Method
NOS	Number of statements	Method
NPA	Number of public attributes	Class
PLOC	Lines of code in project	Project
SIX	Specialization index	Class
WLOC	Lines of code in class	Class
WMC	Weighted methods per class	Class
WNI	Number of all method invocations	Class
WNLA	Sum over NLA	Class
WNMAA	Sum over NMAA	Class
WNOC	Number of all descendant classes	Class
WNOS	Number of statements in class	Class

Table 1: Currently supported metrics. Each metric has a unique acronym, a name, and a scope, which indicates the type of ASG element the metric can be calculated for.

[3] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[4] N. E. Fenton and S. L. Pfleeger. *Software Metrics - A Rigorous & Practical Approach*. International Thompson Computer Press, second edition edition, 1996.

[5] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice-Hall, 1996.

[6] M. Lanza. *Object-Oriented Reverse Engieering*. PhD thesis, University of Berne, Switzerland, 2003.

[7] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, 1994.

[8] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *Proc. of the* 24^{th} *International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, pages 338–348. ACM Press, May 2002.

[9] Software Engineering Institute, Carnegie Mellon University, USA. *Cyclomatic Complexity: Software Engineering Roadmap. Online at http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html*.

[10] University of Paderborn, Germany. *Fujaba Tool Suite. Online at http://www.fujaba.de/*.