6th International Conference on Applied Informatics

Eger, Hungary, January 27–31, 2004.

Development trends in refactoring and measurement tools

István Juhász[a], Gábor Guta[b]

Department of Information Technology, Institute of Informatics,
The University of Debrecen

[a]pici@inf.unideb.hu
[b]gutag@delfin.unideb.hu

Abstract

In recent years refactoring and measurement tools have typically been available for every popular development environment or composed an integral part of them. Refactoring tools seem to be widely accepted, while developer communities are just getting into the habit of using metrics, and the connection between these two has hardly been explored.

This paper tries to show the development trends of these tools and to identify areas where future research is required. First, we summarise the capabilities and limitations of the recent tools. We also have done some investigation in default metrics values for Java. Then we will show some known connection between refactoring and metrics anomalies. We then point out how design metric values are affected by refactorings. Some other existing factors which behave as indicators besides metrics will also be shown. Finally, some further ideas will be presented about basic requirements for the new generations of IDEs.

Categories and Subject Descriptors: D.1.5 [Programming Techniques]: Object-oriented Programming; D.2.3 [Software Engineering]: Coding Tools and Techniques - Object-oriented programming; D.2.6 [Software Engineering]: Programming Environments - Integrated environments; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement - Restructuring; D.2.8 [Software Engineering]: Metrics - Complexity measures, Product metrics

Key Words and Phrases: Measurement tools, software metrics, refactoring tools, Java, Eclipse, Agile

1. Introduction

Several changes have been done in the last 10 years that have set up new requirements for our tools. Typical buzz-words related to these changes are: open-source, agile methodologies, software components, but the exact relationship of these new technologies and the requirements fall beyond the scope of the present article. Furthermore, new languages have also appeared like Java and C#. The growing number and size of class libraries and frameworks are also an important factor. The standard class library of Java 1.4 is a good example, which was introduced in early 2002, including 3395 classes and interfaces [14]. Fortunately, development tools for object-oriented languages have evolved dynamically to fulfil these new requirements. New IDEs serve also as integration platforms for development tools. They have an advanced search and source-code navigation. Recently refactoring and measurement tools are available or integral part of the Java IDEs and we think that these tools will gain greater importance in the future.

In Agile methodologies the good quality of the internal structure of code is an important issue, which is usually achieved by continuous refactorings. Refactoring tools help to carry them out in a more error free manner and measurement tools help to identify when and where refactorings are necessary.

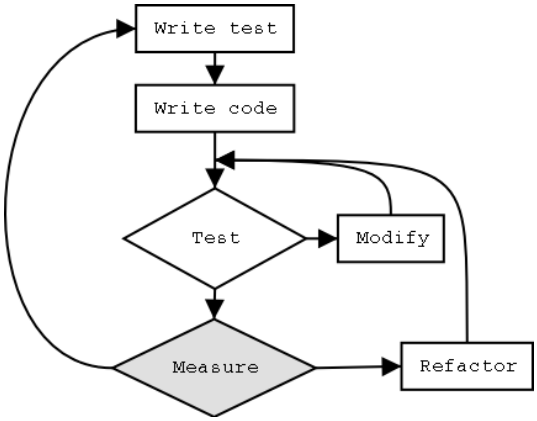

Figure 1: Possible role of measurement in Agile methodologies

Our aim with this article is to summarise the existing results and trends as a starting point for our future research activity.

2. Capabilities of recent tools

Naturally, we cannot compare the full scale of tools, so we have selected 4 tools for both groups. We selected those tools for comparison which seem to provide the most complete support for the concepts mentioned above. Groups consist of 1 open-source tool and 3 commercial ones. We have excluded those tools which

can not be available for evaluation purposes. In the case of refactoring tools our selection was based on computer journals' reader and editor choices. (for example, the Editors' Choice Award 2003 of the 1st April issue of *JavaWorld* in the category of Best IDE was given to IntelliJ IDEA 3.0 with Borland JBuilder 8.0, eclipse 2.1 being the other finalists.) In the case of metrics plug-ins we used IDEs plug-in listings. Some of the selected tools have already had betas and preliminary product information, but we have not found any conceptually new features. They are perfections and completions of existing features or extensions of functionality that have already been implemented by other tools on the market.

2.1. Refactoring tools

We have selected 3 IDEs and 1 stand-alone tool for comparison of refactoring capabilities. The IDEs are the following: (i) Eclipse 2.1.2 the last stable release of Eclipse platform; (ii) IntelliJ IDEA 3.0.5, which is just simply called "the refactoring IDE" by its creators and (iii) Borland JBuilder X is also an impressively developing IDE. (iv) RefactorIT 1.4.1 is tested in stand-alone mode. We have to mention that this tool can be used as plug-in in NetBeans, JDeveloper, JBuilder and in other environments. Surprisingly, NetBeans 3.5.1 has no built-in refactoring capabilities. The tools which are not included in this article may be subject of future comparison.

A reference list can be found at www.refactoring.com. There are also some links to refactoring-related sites, information about tools and much more interesting material. According to the above-mentioned website most of the refactoring tools are available for the Java language. The support of C# also seems promising.

During the evaluation of tools we have found that a more sophisticated classification of the refectoring tools could be useful. We propose to introduce complexity and concreteness dimensions for classification. Some refactoring can be decomposable into mini-refactoring steps. So under complexity we mean the number which indicates the required number of mini-refactorings necessary for completing the refactoring. By the concreteness dimension we consider how exactly refactoring is specified. For example, 'substitute algorithm' is a refactoring with a low concreteness value, because it is not clear what should be replaced by what, while the 'extract method' can be considered to have a high value of concreteness. Tools mainly support the same subset of refactorings. The supported refactorings have high concreteness and low complexity values according to our classification.

Category	**Specified**	**Supported**
Composing Methods	9	50%
Moving Features Between Objects	8	1-2
Organizing Data	16	1-2
Simplifying Conditional Expressions	8	0
Making Method Calls Simpler	15	33%
Dealing with Generalisation	12	66%

Figure 2: Supported refactorings (approximate values)

Code fragments, where the manipulations are done through the reflection API, can not be modified automatically. Some tools warn the user if a string which contains the name of affected functions or classes is found. So handling dynamic constructions seems problematic.
Tools with refactoring capabilities usually have other automatic code generation features, like creating skeletons for overriding class methods, extending abstract classes, etc. Extended code navigation functions are also common functions of these tools.

2.2. Measurement tools

On the one hand, static design metrics seem to be the most popular theme of research papers recently, so we try to explore that from the practical side. On the other, these kinds of metrics can provide immediate help for developers in coding. Purao Sandeep and Vijay Vaishnavi mention about 375 product metrics in their article [11]. Unfortunately, there was only a small subset of these metrics implemented by tools in a not comparable manner. (While in articles C&K metrics [1] are considered a common reference, just 3 of the proposed 6 metrics are usually collected by tools.) We think that another problem is that scientific papers do not always pay enough attention to implementation questions. It is also commonly held that there is a lack of well grounded mathematical models [3, 4]. Measurement tools usually do not count on inherited complexity and size, because they just calculate the class values, without adding the complexity of the parents class.

We have selected the following Eclipse plug-ins for comparison:
net.sourceforge.metrics 1.3.4, which is an open-source one; Team in a Box's eclipse metrics 1.3.0, which is a smaller one and Instantiation's CodePro Studio 2.3.1, which has the most of functionality. We also tried out Aqris RefactorIT 1.4.1, only focusing on its measurement capabilities. There are several other commercial and open-source tools.

Code audit capability is an interesting feature of some tools, it make some syntactical check, which helps to identify dangerous coding constructions and code fragments which do not satisfy the pre-defined coding conventions. Usually, they do not just warn but also propose corrective actions (some of the corrective actions are refactorings!). We think that in the near future metrics could be used in a similar way.

Measurement tools have different approaches: RefactorIT just measure and export the measured values, other tools just warn if some values are not within safe-ranges. Collected metrics vary from tools to tools, even if those collect the same subset of metrics. For instance, if we examine the most simple metrics like lines of code, we can see that some tools count all the lines which end with a new line character, some only count the not blank lines, while some others just count the not comment lines.

Category	Metrics
Basic structured	Lines of Code, Commented LOC, Cyclomatic Complexity, Number of Parameters, Block Depth
Basic OO	Number of Methods, Number of Fields, WMC, Number of Types, Number of Children
Inheritance	Number of Children, Depth of Inheritance tree
Dependency (Martin metrics)	Afferent Couplings, Efferent C., Instability, Abstractness, Distance from the Main Sequence
Other	Lack of Cohesion in Methods, Response for Class, Number of Constructors, Number of Characters, Number of Semicolons, Executable Statements, Number of Package, Cyclic dependency, etc.

Figure 3: Superset of supported metrics by the examined tools

Category	Instantiation CodePro Studio	Team in a Box eclipse metrics	Aqris RefactorIT	sourceforge.net metrics
Basic structured	all	except CLOC	except Block Depth	except CLOC
Basic OO	all	partially	all	all
Inheritance	all	none	all	all
Dependency	all	Efferent Coupling	all	all
Other	partially	partially	partially	partially
Range check	yes	yes	no	yes
HTML	yes	yes	yes	no
XML	yes	no	no	yes

Figure 4: Properties of tools

3. Common Java values

A typical practical question is: what are the ideal common or threshold values for a specified environment. Articles usually leave this question open, mentioning that these values depend on several parameters and suggest doing a calibration on similar projects. Although the above statements are valid, there are common average values which can be used as a starting point for a given language. Lorenz and Kidd present common values for C++ and Smalltalk in their book [9]. In our experience Java values should be somewhere between Smalltalk and C++ values. Unfortunately, we have not found any well usable reference measurement, which can be reproduced. Naturally, there are some articles similar to Xiao's [15] work, but they are based on a limited number of projects and metrics. Some others, however, measured propertial source code with propertial tools.

So we have two options: (i) There are default threshold values found in the measurement tools, which can be compared and evaluated; or we can (ii) measure some open-source project with open-source tools. We use eclipse 2.1.2 and

net.sourceforge.metrics 1.3.4 for our measurement. As it is an open source tool, any question in connection with the implementation can easily be examined.

3.1. Default threshold values in the tools

The first idea mentioned above does not really work because of the divergence of default values and types of metrics. So we only have the other alternative left: we have to measure the average values. Below we include a short comparison of some default values of two tools, which we have found interesting. Of the tested tools, three do not include many more comparable values, while the fourth tool does not have default value settings.

	net.sourceforge.metrics	**com.teaminabox.metics**
Lines of Code	50	15
McCabe's Cyclomatic Complexity	10	4
Nested Block Depth	5	4
Number of Parameters	5	4

Figure 5: Default threshold values

3.2. Measured values

We also found unexpected difficulties during the measurement. There is a lack of standard metrics sets or guidelines, i.e. how we can set up a reference measurement for an unknown goal. We selected some applications by the following criteria: the selected applications should be an open-source software in stable or mature release with a large user community.

It seems obvious that different types of applications have different "good" values. For the first step let us take the following categorisation: *desktop application, server application* and *library.* A more sophisticated categorisation is also necessary in this question. Theoretically, appli cations and libraries have different coupling and cohesion parameters, so this seems to be a good dimension. The basic problem is that some applications have plug-in interfaces which can be considered a kind of class library. In the case of applications we have found in the literature that GUI based applications significantly differ from no-GUI applications [8]. Another influencing factor is that the measured values can be heavily affected by the number and type of the used libraries and framework. We suggest the exclusion of auto-generated code fragments.

	jgraph 3.0	**jgraph 3.1**	**jgraphpad 3.0**	**jedit core 4.1**	**jedit 4.1**	**metrics 1.3.4**
Total Lines of Code	**5929**	**5950**	**16665**	**33581**	**52142**	**7307**
Average of Lines of Code (per method)	6.27	6.30	8.91	12.54	12.11	5.26
Average of Lines of Code (per class)	98.82	99.17	54.11	79.01	92.78	37.66
Total Number of Classes	**60**	**60**	**308**	**425**	**562**	**194**
Average of Number of Classes (per package fragment)	12	12	28.00	28.33	28.10	12.93
Average of Number of Methods (per type)	13.95	13.93	5.61	5.30	6.68	6.83
Average of Number of Attributes (per type)	4.83	4.83	1.98	3.14	3.21	2.21
Average of Weighted methods (per type)	40.15	40.25	13.94	19.88	29.43	13.60
Average of Number of Overridden Methods (per type)	1.43	1.43	0.37	0.72	0.72	0.54
Average of Number of Children (per type)	0.17	0.17	0.96	0.22	0.29	0.49
Average of Depth of Inheritance Tree (per type)	1.97	1.97	3.20	2.54	2.39	2.11
Maximum of Depth of Inheritance Tree (per type)	6	6	6	8	8	6
Average of Nested Block Depth (per method)	1.48	1.48	1.50	1.66	1.64	1.51
Average of Number of Parameters (per method)	1.08	1.08	1.04	1.11	1.06	0.79
Average of McCabe Cyclomatic Complexity (per method)	2.55	2.56	2.30	3.16	3.84	1.90

Figure 6: Averages of some projects

3.3. Affected metrics

Refactorings usually just slightly change the averages values, and in an optimal case they cancel the exceptionally high values. The extent of change of average values also depends on what kind of refactorings are applied. We have found an experimental report which confirmed our experiences, even though the writers declared their results preliminary [13].

4. The connection between the metrics and the refactorings

Quality issues are classified in two main categories: external and internal [9]. The direct connection between the external quality and metrics is rather fuzzy [3]. We think it is more advisable to handle product metrics as properties of the internal quality. Internal quality mainly affects maintainability and readability of code. Much of the literature agree that maintenance can be quite expensive or even

impossible in a not well designed application, so internal quality can be considered to be an important factor [10]. That is why we believe that metrics will play a more and more important role in internal quality improvements.

The basic question of refactoring is when we have to apply refactorings. What is the economical balance of clarity of internal structure and the effort paid for it? We think that it would be more economical only to select the code fragments we want to modify or attach a new module for refactoring. In this way we can save development effort.

A researcher's dream is a fully automated metric based refactoring process. This means that refactoring tools automatically identify source code anomalies and apply the refactoring. Unfortunately, this seems impossible. It always needs human interaction because of irregular situations. That is why the ultimate aim is to create a system that helps the developer as much as possible.

4.1. Results of recent research

Martin Fowler defined 22 "bad smells" in his book and he declared that identifying of refactorings needs "human intuition" [5]. The examples of refactorings help to understand the process of refactoring, but they are too small to give an impression of which real situation refactorings need to be applied in (and often he uses very similar examples both for forward and reverse transformations). In recent research, results shows that metrics can help to identify these situations.

We have found 3 main approaches. One of them is hierarchy restructuring, based on similarities of methods. It simply works the following way: It flattens the classes, then removes the hierarchical relations. Finally, it rebuilds the hole hierarchy based on similarities [6]. The second idea is that we can identify which methods and variables need to move to other classes, based on coupling [12]. As the third approach, we found the theory of semantic metrics to be the most interesting concept. The idea is that we can make an ontology from the design documentation and we can measure the connection between our semantic model and the class structure [2].

We have to mention that metrics is not the only way to help to identify the necessary refactorings, there are some others, for example, 'finding invariants' [7].

5. Summary

We have presented a brief overview of refactoring and measurement tools for Java and shown some connection between these tools and Java specific information that can help in further application and research. We found that the tendency of recent years shows that more and more refactorings are covered. Unfortunately, the situation with metrics is rather fuzzy, there are several different tools, with divergent properties. We think reverse engineering tools will be included in the IDEs of the future to help a better understanding and an ability for large scale restructuring over source code through visualisation.

For further development in software measurement, we need a much larger amount of comparable data and more interoperable tools. In order to achieve this, measurement tools need some standardisation.

We will make available our measurements in XML format of net.sourceforge.metrics plug-in and the links to web pages of related tools on the following site: http://inf.unideb.hu/metrics

6. Acknowledgment

One of the authors, Gábor Guta, would like to thank 'AKTION Österreich-Ungarn' foundation for providing a scholarship at the Technical University of Vienna that made it possible to complete and enhance this article.

References

[1] Chidamber, S., Kemerer, C.: A metrics suite for object oriented design, in *IEEE Transactions on Software Engineering*, Vol. 20, Num. 6 (1994), 476-493

[2] Etzkorn, L., Delugach, H.: Towards a Semantic Metrics Suite for Object-orinted Design, in Proc. International Software Metrics Symposium 2000, 71-80

[3] Fenton, E. N., Shari Lawrence Pfleenger: *Software Metrics: A rigorous approach,* Internation Thomson Publishing, 1996

[4] Fenton, N.: Software Measurement: A Necessary Scientific Basis, in *IEEE Transactions on Software Engineering,* Vol. 20, Num. 3, (1994), 199-206

[5] Fowler, M.: *Refactoring: Improving the design of existing code*, Addison-Wesley, 1999

[6] Ivan M.: Automatic Inheritance Hierarchy Restructuring and Method Refactoring, in Proc. *Conference on Object Oriented Programming Systems Languages and Applications 1996*, 235-250

[7] Kataoka, Y., Ernst, D. M., Griswold G. W., Notkin, D.: Automated Support for Program Refactoring using Invariants, in Proc. *IEEE International Conference on Software Maintenance 2001,* 736-743

[8] Lorenz, M., Kidd, J.: *Object-oriented Software Metrics: A practical Guide,* Prentice-Hall, 1994

[9] Meyer, B.: *Object-oriented software construction, 2^{nd} Edition*, Prentice-Hall, 1997

[10] Pigoski, Thomas M.: *Practical software maintenance: best practices for managing your software investment*, John Wiley Computer Publishing, 1997

[11] Purao, S., Vaishnavi, V.: Product Metrics for Object-Oriented Systems, in *ACM Computing Survey,* Vol. 35, Num. 2,(2003), 191-221

[12] Simon, F., Steinbrückner, F., Lewerentz, C.: Metrics based refactoring, in Proc. *European Confonference on Software Maintenance and Reengineering 2001*, 30-38

[13] Stroulia, E., Kapoor, R.: Metrics of Refactoring-based Development: An Experience Report *http:/www.cs.ualberta.ca/∼stroulia/Papers/ooispdf,* last visited: 2004.1.12.

[14] Szyperski, C.: *Component Software, Beyond Object-Oriented Programming, 2nd Edition*, Addison-Wesley, 2002

[15] Xiao, Q.: Object-oriented Metrics Analysis on Java Software Projects *http://ww.isse.gmu.edu/faculty/ofut/classes/763/studpapers/xiao_ q.pdf*, last visited: 2004.1.12.

Postal address

István Juhász
Gábor Guta
Department of Information Technology
Institute of Informatics
The University of Debrecen
1, Egyetem tér, 4032 Debrecen
Hungary