COMPARISON OF SOFTWARE METRICS TOOLS FOR .NET

Jernej Novak
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova ulica 17, 2000 Maribor, Slovenia
Tel: +386 22207187; fax: +386 22207272
e-mail: jernej.novak1@uni-mb.si

Gordana Rakić
University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
e-mail: goca@DMI.uns.ac.rs

ABSTRACT
In this paper we will compare tools for calculating software metrics. We will examine tools that are available free on the internet and capable of calculating software metrics for C# language. The purpose of this paper is to compare the calculations of software metrics and determine if the tools are providing the same results. We would like to discover if choosing a different tool can provide different results and consequently bring different decisions.

1 INTRODUCTION
To make software that is good and reliable we need a suitable quality model. One of the quality models is the ISO 9126, which among other things, requires software metrics [1][2]. To make a reliable quality model we need tools that can provide us with software metrics calculations. To make a solid and reliable quality model we need tools that can provide us with accurate data.

Software metrics are a mathematical definition for mapping the entities of a software system to numeric metric values. We also assume that software metric tools implement software metrics according to their definitions.

There are quite a few available tools that can provide us with software metric values for C# language or .NET assemblies. In this paper, we will not deal with whether this information is accurate or not whether different tools give us different results.

The goal of this paper is to answer if metric tools for C# language give us different results. After a review of existing literature, we found a similar paper but this paper was testing tools for the Java language and was using a different comparison technique [3].

In the next section, we will describe what tools we used, what tools we chose for metrics calculations and we will offer a description of the analysis. The third section presents the results and an analysis of them. We conclude with a summary of the results and some questions for further work.

2 DESCRIPTION OF THE STUDY
In this chapter we will describe how we performed the experiment.
All experiments were performed on a standard PC which always met the minimum requirements expected for all software metrics tools.

The data collected from software metric tools was stored for further processing. All results were manually migrated to an Excel file for easier further processing.

First we chose tools for calculating software metrics, afterwards we selected software metric tools for testing. Then we chose software on which we will performing metric calculations. At the end of this chapter, we will describe in detail how the analysis of results was carried out.

2.1 Software Metrics Tool Selection
First, we performed a free search on the internet to find available tools for the computation of software metrics. The first criteria was that the tool can calculate any software metrics for C# language or .NET assemblies. We found ten tools, of which five met all the required criteria. The criteria was that the tool is available for free, or that at least a trial version is available for a specific period of time, and that the tool has a graphical user interface from which we could manage the tool. In the following paragraphs we will briefly describe the five chosen tools for our experiment.

**CodeMetrics (Add-in for Reflector)** analyses and computes several code quality metrics on .NET assemblies. This tool and its source code are freely available on the [4].

**NDepend** is a Visual Studio tool to manage complex .NET code and achieve high Code Quality. With NDepend,
software quality can be measured using Code Metrics, visualized using Graphs and Treemaps, and enforced using standard and custom Rules. Software metrics calculation is also done on .NET assemblies. The NDepend is shareware but is freely available for research projects.[5]  

**SourceMonitor** is a freeware program that lets you look inside your software source code to find out how much code you have and to identify the relative complexity of your modules. This tool runs on C# source code and it is freely available to download over the internet.[6]  

**Visual Studio Code Metrics** is a set of software measures that provide developers with a better insight into the code they are developing. This tool comes free with Visual Studio in a premium and ultimate edition.[7]  

**Borland Together (add-in for Visual Studio)** enables you to analyze, design and implement flexible and maintainable software architectures that can be easily modified as requirements change. It runs on C# source code. It is available for shareware usage for 15 days.[8]  

### 2.2 Software Metrics Selection  

First, we made a list of all the available software metrics that the tools provided. Tools provide different set metrics. But after analyzing them, we recognized metrics with different names but with the same definition and calculating algorithms. We decided to choose seven software metrics, of which we could say that they represent the same software metrics definitions. After considering all the parameters we decided to include the following seven metrics in our research:

- **Maintainability index** includes composed metrics. It is calculated with a certain formula from lines-of-code measures, McCabe measures and Halstead complexity measures.[9]  
- **Average Cyclomatic Complexity** is used to indicate the complexity of the program. It measures the number of linearly independent paths through the program source code [10].  
- **Average Depth of Inheritance Tree (DIT)** is the maximum length of a path from class to root class in the inheritance structure of a system [11].  
- **Average Coupling Between Objects (CBO)** is the number of classes that a class is coupled to [11].  
- **Lines of Code (LOC)** is used to calculate the size of a software program by counting the number of lines in the text of the program or intermediate language [12].  
- **Average Lines of Code** represents the average LOC metrics per module.  
- **Average Comment Ratio** is calculated from the average lines of code divided by the average comments count per module.  

For an unambiguous definition for software metrics please refer to the original sources defined in the references.

<table>
<thead>
<tr>
<th>Tool</th>
<th>SoftwareMetrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
</tr>
<tr>
<td>Program name</td>
<td></td>
</tr>
<tr>
<td>Visual Studio</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✗</td>
</tr>
<tr>
<td>Borland</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✗</td>
</tr>
<tr>
<td>SourceMonitor</td>
<td>✗ ✔ ✔ ✔ ✔ ✔ ✔</td>
</tr>
<tr>
<td>ReflectorAddin</td>
<td>✗ ✔ ✔ ✔ ✔ ✔ ✔</td>
</tr>
<tr>
<td>NDepend</td>
<td>✗ ✔ ✔ ✔ ✔ ✔ ✔</td>
</tr>
</tbody>
</table>

Table 1: Tools and software metrics.

In Table 1, we can see the used tools and metrics for our experiment. The fields selected with a check show that the tool supports selected metrics and the spaces marked with an X show that the tool does not support selected metrics. All tools have a solid coverage of software metrics except the Reflector addin.

### 2.3 Choosing Software  

To choose testing tools, we limited ourselves to testing systems written in C# language. We performed a free search on the SourceForge.net page and randomly chose five different projects. We chose only Windows projects for easier compilation and calculation of software metrics. We were also trying to choose different sizes and projects of varying origins. In the next paragraph we will present the five programs that we chose for our experiment:

- **CompressFolders** is a windows application that allows us to compress folders and files.  
- **FileManager** is a Windows application that enables us to manage files, upload and download files from a server and to see the history of uploaded and downloaded files.  
- **Gammit** is a Windows application that allows us to set the gamma level for an LCD monitor. It also corrects the lighting of an LCD monitor.  
- **Palantir** is a Windows application that allows us to review all remote computer connections.  
- **ZedGraphWin** is a Windows controller for drawing graphs for Windows applications.

### 2.4 Analysis description  

To collect data, we installed all the chosen tools. Between tools there were no dependencies or side effects that we should consider. The tested software was saved in a dedicated space so that all tools were performing an analysis on the same source code. To avoid unwanted changes or mistakes on the source code, we stored the analyzed source code prior to testing in a read-only drive.
When we installed all the tools and when all the tested software was ready, we ran an analysis with each tool for each tested software. Then we exported all data to MS Excel 2007 and there conducted some further analysis, as described later on in this paper.

Metric $k$  

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Project 1</th>
<th>Project 2</th>
<th>...</th>
<th>Project $i$</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool 1</td>
<td>$M_k V_{1,1}$</td>
<td>$M_k V_{1,2}$</td>
<td>...</td>
<td>$M_k V_{1,i}$</td>
<td>$M_k V_i$</td>
</tr>
<tr>
<td>Tool 2</td>
<td>$M_k V_{2,1}$</td>
<td>$M_k V_{2,2}$</td>
<td>...</td>
<td>$M_k V_{2,i}$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Tool $j$</td>
<td>$M_k V_{j,1}$</td>
<td>$M_k V_{j,2}$</td>
<td>...</td>
<td>$M_k V_{j,i}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Calculation of average values by tool for multiple projects.

All stored data was saved to a dedicated excel file. As shown in Table 2, we calculated the average metric values for every metric. So we had seven tables with data, each for every software metric. Most data collection and data organization was manual. We calculated the average values for each tool from 1 to $j$. For each tool and for each software metric we got average values from $M_k V_{1}$ to $M_k V_{j}$, where $k$ is the number of software metrics and $j$ is the number of tools. From these values we then made the vectors $\overrightarrow{T_1}$ until $\overrightarrow{T_j}$ and with these values we made T-Tests.

In Table 4, we can see a comparison of each tool with each other with a T-test. Each vector $\overrightarrow{T}$ from one tool we compared to another tool. After conducting every comparison, we got metrics as seen in Table 4. For significantly different results, we will take those that are in an 80% of interval of trust. We decided on the value 80% after reviewing the existing literature review [13]. This means that two tools will be different if their values of T-Test are under 0.05.

### 3 RESULTS AND ANALYSIS

A quick glance at the results confirmed that results can differ significantly between tools. We also calculated standard deviations for each metric. In Table 6, we can see values differing especially for LOC software metrics. The most consistent results were seen with DIT software metrics. A very significant deviation in LOC software metrics can be assigned to many different poorly defined LOC software metrics.

#### 3.1 T-Test results

We can look at the obtained results of the T-Test in Table 6 and if we accept an 80% trust interval, we can see that most of the tools give us significantly different results.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Tool 1</th>
<th>Tool 2</th>
<th>...</th>
<th>Tool $j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool 1</td>
<td>T-Test($\overrightarrow{T_1}, \overrightarrow{T_2}$)</td>
<td>...</td>
<td>T-Test($\overrightarrow{T_{j-1}}, \overrightarrow{T_j}$)</td>
<td></td>
</tr>
<tr>
<td>Tool 2</td>
<td>...</td>
<td>T-Test($\overrightarrow{T_2}$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool $j$-1</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool $j$</td>
<td>T-Test($\overrightarrow{T_{j-1}}, \overrightarrow{T_j}$)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Tool comparison table.

The T-Test fails only on results that give us the two pairs of tools. These are Reflector.CodeMetrics and SourceMonitor pair and Reflector.CodeMetrics and NDepend pair. And even this test failure can be assigned to the small vector $\overrightarrow{T}$. Reflector.CodeMetrics had the small vector $\overrightarrow{T}$ because it only supported three of the seven tested software metrics. All other tool results were confirmed, so that we can claim that tool results differ significantly.

Table 5: Average standard deviations for specific metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>M1</th>
<th>avg CC</th>
<th>avg DIT</th>
<th>CBO</th>
<th>avg LOC</th>
<th>LOC avg com</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>7.5</td>
<td>6.9</td>
<td>0.8</td>
<td>7.6</td>
<td>49</td>
<td>3104</td>
</tr>
<tr>
<td>Average standard deviation</td>
<td>7.5</td>
<td>6.9</td>
<td>0.8</td>
<td>7.6</td>
<td>49</td>
<td>3104</td>
</tr>
<tr>
<td>Table 6: Values of the T-Test for each tool.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In Figure 1 we can see that the only values of the Reflector and NDepend did not pass the T-Test so we cannot claim that these two tools gave us different results.

We showed that C# metrics tools differ from each other. So we only confirmed what was shown in the article “Comparing Software Metrics Tools” [3] with the difference of using different tools (C# instead of Java) and by applying a different method.

4 CONCLUSION AND FUTURE WORK

Developers and engineers of software must rely on scientific data and this data should be accurate. Especially when these data are the foundation for making important decisions. If we make decisions based on inaccurate data, we can change the development lifecycle and with this we can affect the quality of a software product.

In this experiment we showed that the metrics tool gave us different results for the same inputs. Results can vary so much that engineers can adopt different decisions just because they chose another tool. So choosing the right tool is important. We also must know that we cannot compare metrics data given from different tools. We hope that in the future tools will get more consistent and will give us comparable results.

In this paper we did not answer which tool is better and give us more accurate results. In the future maybe we should mathematically review the metrics and write test cases from which we could then determine if the metrics tool calculated the correct metrics value.

Also, if we keep in mind the conclusions in this paper and the results from [14] we have to consider problems in large projects developed in several languages. In these projects, several tools have to be used to measure all the components of the project. In that case, we have to be prepared to gain inconsistent and incomparable metric results. These conclusions confirm the need for new “universal” software metric tools that would provide comparable results for the whole project independent of the languages used for the development of the project.

References