Evaluating Software Metrics for detecting code clones from potential clones

Sandeep Bal[#1], Sumesh Sood[*2]
[#]*Research Scholar, Dept of RIC, IKG-PTU, Kapurthala, Punjab, India*
[1]sandeep.bal84@gmail.com

**Asst. Prof., Department of Computer Applications,*
IKG-PTU Campus, Dinanagar, Punjab, India
[2]Sumesh64@gmail.com

Abstract: Software metrics give the numerical values of the characteristics of software or units of software. The clone detection literature contains a large number of metrics which helps in measuring the functions as software units. It is the choice of the researcher to finalize any set of metrics in order to conclude whether potential clones are actual clones or not. This paper contains one such metrics based clone detection analysis which takes a set of metrics for the given software. The metrics are calculated using a tool named Understand™ and the results are passed to another web based application specifically designed to find the percentage of similarities between two sets of coding between different releases/versions of software.
Keywords: metrics, potential clones, open source software, versions of software.

I. Introduction

Software evolution is the sequence of changes to a software system over its lifetime; it encompasses both development and maintenance [1].

Software metrics capture different aspects of software complexity. This paper evaluates software metrics and presents their comparison over different releases of the same software. The impact of their change on the health of the software over its evolution is studied. Information gained from metrics can be used in the management and control of the development process in order to improve results.

In this paper an Open Source Software namely NALCG (Not another Lousy Chess Game) has been selected. Its 8 releases are hosted on GitHub [2]. It is tested using **Test-driven development** (**TDD**) process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, and then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards [3].

Understand™ [4] is a Static Code Analysis tool aiming to achieve complete code navigation, control flow graph generation, Metrics generation, code comparison, checking on the adherence of a code to some specific coding standards like MISRA and code reengineering for an array of programming languages like C, C++, Java, Jovial, Pascal, ADA, .NET and more. It is a tool from "Scientific Tools Inc." and based out of St. George, Utah, US. This tool has been used since long in leading automotive, aerospace, defence and loads of other critical industries for understanding large legacy codes and take up Static Code Analysis on them.

Understand™ as a tool has also a well developed command line interface which allows this tool to be integrated with any of the existing tool chain that the software companies normally use. Since the commencement of the company (Scientific Tools Inc.) in 1996, it has been used by more than 200 companies worldwide to

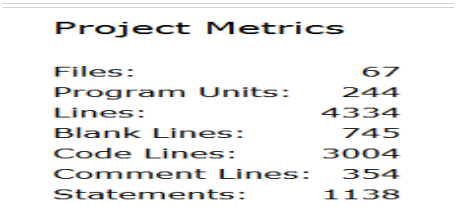
understand their codes and development further on them.

Section II of the paper shows the study of the versions of the software used in the study. Section III contains the names of selected metrics in order to perform the comparative study across versions. The paper concludes in section IV with the conclusion.

II. Versions of NALCG

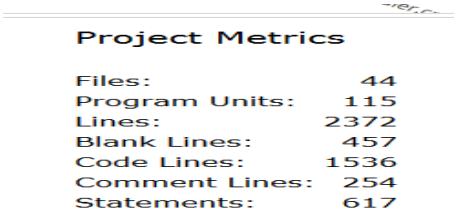
An Open Source Software [5] namely NALCG (Not another Lousy Chess Game) has been selected and Total 8 versions of this [6] are available till now:

1) Test release: Initially released version of the software:



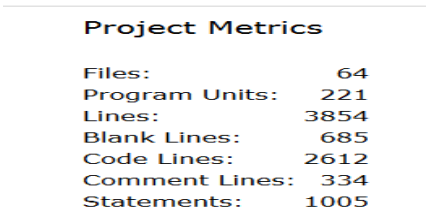
Project Metrics	
Files:	67
Program Units:	244
Lines:	4334
Blank Lines:	745
Code Lines:	3004
Comment Lines:	354
Statements:	1138

2) Version 0.5



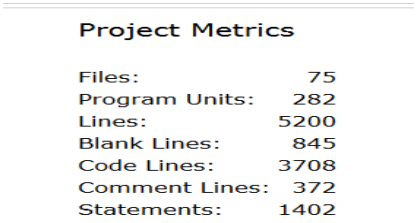
Project Metrics	
Files:	44
Program Units:	115
Lines:	2372
Blank Lines:	457
Code Lines:	1536
Comment Lines:	254
Statements:	617

3) Version 0.7



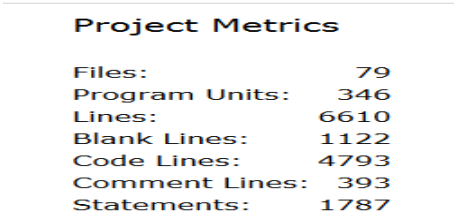
Project Metrics	
Files:	64
Program Units:	221
Lines:	3854
Blank Lines:	685
Code Lines:	2612
Comment Lines:	334
Statements:	1005

4) Version 0.9



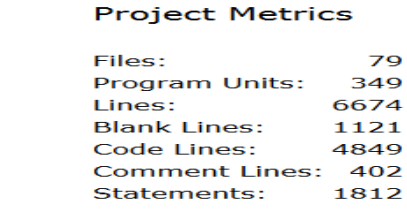
Project Metrics	
Files:	75
Program Units:	282
Lines:	5200
Blank Lines:	845
Code Lines:	3708
Comment Lines:	372
Statements:	1402

5) Version 0.9.7 : Pre release before demo



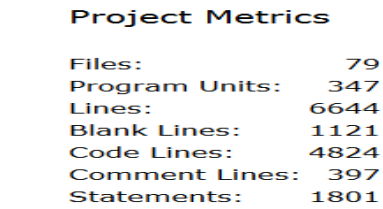
Project Metrics	
Files:	79
Program Units:	346
Lines:	6610
Blank Lines:	1122
Code Lines:	4793
Comment Lines:	393
Statements:	1787

6) Version 1.0: First full release version



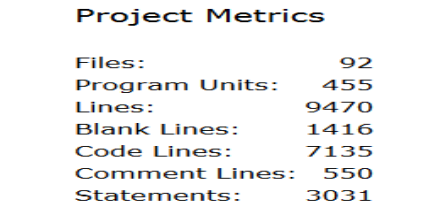
Project Metrics	
Files:	79
Program Units:	349
Lines:	6674
Blank Lines:	1121
Code Lines:	4849
Comment Lines:	402
Statements:	1812

7) Version 1.0.0rc1: Offline version of the gaming software



Project Metrics	
Files:	79
Program Units:	347
Lines:	6644
Blank Lines:	1121
Code Lines:	4824
Comment Lines:	397
Statements:	1801

8) Version 1.5rc1: Final version



Project Metrics	
Files:	92
Program Units:	455
Lines:	9470
Blank Lines:	1416
Code Lines:	7135
Comment Lines:	550
Statements:	3031

Software development is a dynamic process where we need to keep all the associated metrics in check otherwise, the software shape and complexity might spiral out of control [7]. As the study of dynamical systems have demonstrated, and as reflected in this study, where one parameter left unchecked led to an increase in overall complexity, we cannot completely control complexity. What we would want to do is ensure that as it scales up in size (and complexity), it stays manageable, accessible and clean. And keeping gauge of metrics will play a major role in helping us ensure that.

III. Metrics & their comparison

The following table gives the name, detail of each metric and number assigned to each metric which is used in the approach to refer to these metrics.

1. CountLine
2. Cyclomatic
3. CountDeclClass
4. CountDeclFunction
5. CountDeclMethod
6. CountInput
7. CountOutput
8. CountPath
9. CountStmtDecl
10. CountStmtExe
11. PercentLackOfCohesion

Thus a total of eleven metrics are evaluated in this metrics based technique of code clone detection.

Understand™ tool is used to compute the values of required metrics for implementing the approach. This tool is able to export the metrics values as Comma separated values (CSV) file. These CSV files are input to another tool which performs the comparisons of the files. Each CSV files belongs to a different version of the software. Since each version has its own code and data so the corresponding metrics values of each file varies accordingly.

The comparison helps to identify the number of similarities and compare the metrics value. This comparison suggests about the overall change in software architecture and the dynamic changes occurring (if any) during its evolution.

Table 1 presents the brief description of each of the metrics along with their name.

No.	Metrics	Brief Description
1	CountLine	Number of all lines
2	Cyclomatic	Number of linearly independent paths through a program's source code
3	CountDeclClass	Number of classes.
4	CountDeclFunction	Number of functions
5	CountDeclMethod	Number of local (not inherited) methods
6	CountInput	The no. of inputs a function uses & the no. of unique sub programs calling the function
7	CountOutput	The number of outputs that are set
8	CountPath	Number of unique paths through a function
9	CountStmtDecl	Number of declarative statements
10	CountStmtExe	Number of executable statements
11	PercentLack OfCohesion	Calculates what percentage of class methods use a given class instance variable. A lower percentage means higher cohesion between class data and methods.

Table 1

Sr. No.	Versions	Metrics				
		Classes	Files	Program Units	LOC	Executable Statements
1	Test Release	32	67	244	3004	1138
2	Version 0.5	21	44	115	1536	617
3	Version 0.7	31	64	221	2612	1005
4	Version 0.9	35	75	282	3708	1402
5	Version 0.9.7	37	79	346	4793	1787
6	Version 1.0	37	79	349	4849	1812
7	Version 1.0.0rc1	37	79	347	4824	1801
8	Version 1.5rc1	44	92	455	7135	3031

Table 2

Sr. No.	Metrics	COMPARISON OF METRICS FOR TEST RELEASE OF SOFTWARE WITH EACH OF ITS SUCCESSOR VERSION						
		Test Release *VS* Version 0.5	Test Release *VS* Version 0.7	Test Release *VS* Version 0.9	Test Release *VS* Version 0.9.7	Test Release *VS* Version 1.0	Test Release *VS* Version 1.0.0rc1	Test Release *VS* Version 1.5rc1
1	CountLine	28 Times	49 Times	46 Times	45 Times	45 Times	45 Times	26 Times
2	Cyclomatic	192 Times	205 Times	206 Times	204 Times	204 Times	204 Times	195 Times
3	CountDeclClass	219 Times	242 Times	224 Times	224 Times	224 Times	224 Times	201 Times
4	CountDeclFunction	245 Times	250 Times	248 Times	248 Times	248 Times	248 Times	242 Times
5	CountDeclMethod	284 Times	285 Times	285 Times	285 Times	285 Times	285 Times	282 Times
6	CountInput	154 Times	142 Times	143 Times	142 Times	142 Times	142 Times	152 Times
7	CountOutput	150 Times	168 Times	164 Times	162 Times	162 Times	162 Times	162 Times
8	CountPath	198 Times	210 Times	212 Times	210 Times	210 Times	210 Times	205 Times
9	CountStmtDecl	116 Times	129 Times	128 Times	125 Times	125 Times	125 Times	104 Times
10	CountStmtExe	49 Times	105 Times	107 Times	105 Times	105 Times	105 Times	100 Times
11	PercentLackOfCohesion	225 Times	248 Times	228 Times	228 Times	228 Times	228 Times	200 Times
		1st with 2nd	**1st with 3rd**	**1st with 4th**	**1st with 5th**	**1st with 6th**	**1st with 7th**	**1st with 8th**

Table 3

IV. Conclusion

The CSV files (generated by Understand™ tool) of all the versions are input to the newly made tool one by one. It starts comparing the values of the metrics already calculated by Understand™. The comparison is performed on the basis of all the different source code files in Test Release version and each of its successor versions one by one. The initially designed Test version is compared with all other versions where it reports for the number of matches occurred in two different versions. E.g. CountLine matches 28 Times amongst the two versions i.e. Test Release and Version 0.5.

When the source code of 1st version (Test Release) is compared with the rest of the versions of the software, 8th version (1.5rc1) of the software shows the least number of matches of CountLine i.e. 26 Times.

Therefore, it is believed that the overall number of lines remains quite stable during these two versions as the number of matches detected is minimal although CountLine might also change continuously with every version due to the editing (addition/ deletion of source code) of the files. The fact is quite visible during the comparisons of other versions as the number keeps changing a little.

The value of Cyclomatic complexity remains almost stable while comparing all versions except the 2nd version (0.5) and 8th version (1.5rc1) where it decreases a little.

CountDeclClass shows the number of declared classes in a version. The declared number of classes remains same while comparing 1st version with 4th, 5th, 6th and 7th version consecutively while the values in 1st, 2nd and 8th are found to be changing a little.

CountDeclFunction and CountDeclMethod metrics remains same during 1st, 4th, 5th, 6th and 7th versions. In fact, very negligible changes are observed amongst all the versions.

Similar results are observed for the rest of the metrics i.e. CountInput, CountOutput, CountPath, CountStmtDecl, CountStmtExe and PercentLackOfCohesion. The comparisons of the metrics values of 1st version with versions 5th, 6th and 7th are found to be most stable and 1st version with 2nd and 8th version also follow the similar study.

Hence, it is concluded that the software architecture remains stable when the number of matches occurring in metrics values of different versions of any software code are more whereas the code cloning aspect states that the two compared versions appears to be replica of each other when the metrics are same.

REFERENCES

[1] Cook, S., Ji, H., & Harrison, R. (2000). Software evolution and software evolvability. *University of Reading, UK*, 1-12.
[2] https://github.com/Meelo/NALCG?source=cc
[3] http://en.wikipedia.org/wiki/Testdriven_development
[4] http://www.meteonic.com/solution/static-codeanalysis/understand.html
[5] https://github.com/Meelo/NALCG?source=cc
[6] https://github.com/Meelo/NALCG/releases
[7] Bal, S., & Sood, S. (2014). Impact of Software Metrics on Complexity during Evolution of a Software. *National Conference on Emerging Trends in Computer Science and its Applications in 21st Century*