

Reuse Ratio Metrics RL and RF

William B. Frakes, Reghu Anguswamy, and Suvelee Sarpotdar
Computer Science Department, Virginia Tech, Falls Church, VA USA

Abstract – In this demo we will show various implementations of RL (reuse level) and RF (reuse frequency) tools for measuring the amount of reuse in software. We will also discuss how these metrics have been used in several studies.

1 INTRODUCTION

These demos are of reuse level and reuse frequency ratio metrics. Reuse level (RL) was defined in 1990 as having two aspects, internal reuse level (IRL) and external reuse level (ERL) [6]. A tool was developed on Unix to measure the reuse level of functions in C code. Measures of reuse level of functions in C programs found external functions reuse was around 50% an order of magnitude higher than some estimates at the time. Reuse frequency (RF) was defined in 1996 and a tool was developed to measure it [2][3]. As part of the research, the RL and RF tools were modified to make the tool compatible with the newer versions of tools such as cflow. RL and RF have since been used in various studies including reuse in open source C, Java, and C++ software [4], to study reuse in C programs [1], and to study productivity gains from reuse [5]. More recent work has focused on the use of more powerful tool environments for creating RL and RF metrics [7].

2 RL AND RF

2.1 Reuse Level and Reuse Frequency

Reuse level (RL) is defined as the ratio of reused items to the total number of items in a product [6]. The measurement is applicable to hierarchical systems. A C program, for example, can be viewed as a collection of functions that are in turn composed of C statements. A higher-level element can be decomposed into its lower level components. A lower level component can be classified as internal or external. A lower level component created for use in a higher level component is called an internal component. A lower level component that is created for some other purpose, such as for a component library, but is used in a higher level component is called an external component. A common example of such a library is `stdio.h` for C.

Two other concepts needed to define RL and RF are:

- Internal Threshold Level (ITL) – The minimum number of times an internal lower level item must be used in a higher level item before it is considered a reused component. For example if $ITL = 1$, then a function is considered reused if it is used more than once in a system.
- External Threshold Level (ETL) – The minimum number of times an external lower level item must be used in higher level item before it can be considered for reuse. For example if $ETL = 0$, then any external function called even once in the system is considered reused.

Given the following:

- Internal Reuse (IU) – number of internal lower level items which are used more than ITL.
- External Reuse (EU) – number of external lower level items which are used more than ETL.
- Total items (T) – total number of lower order items in the higher order item including both internal and external items.

We can define:

- $RL_{internal} = IU/T$
- $RL_{external} = EU/T$

2.2 Reuse Frequency

Often a lower level item is used more than once in a higher level item. Counting the multiple uses of a lower level item is not done for Reuse Level. Hence a new term, Reuse Frequency, was defined which is a count of the number of references to a lower level item from a higher level item [6]. In compiler terms, RL counts types while RF counts tokens. Using the same threshold concepts above,

- $RF_{internal} = IUF/T$ $RF_{external} = EUF/T$

2.3 Ramel Study

Ramel measured ERL for C/C++ and Java programs found in Free and Open Source Software (FOSS) projects [4]. Ramel hypothesized that Free and Open Source Software (FOSS) encourages component-based development and reuse given the way FOSS software is developed. The (FOSS) development environment consists of multiple dispersed teams of developers collaborating to integrate reusable components and libraries into sophisticated software systems.

To test her hypothesis Ramel analyzed the top layer hierarchies of C++ and Java software. The layers analyzed include the subsystem and internal and external library reuse information. The system level of C++ measured was the redhat package manager (rpm). For Java, packages, specifically jar files, were considered part of the software subsystem. Across both Java and C++ the internal and external libraries were measured for the amount of reuse.

Two tools were used to measure Java and C++ code. JDepend was used to measure the package level and internal and external class reuse in Java code. JDepend analyzes Java jar files and provides metrics on the jar file internal and external dependencies. JDepend also counts the number and type of classes in the Java jar file.

The C++ evaluation included the rpm files of the Suse kernel. A perl script was created to parse C++ rpm files. The perl script counted the library dependencies with “.so” that were not included in standard libraries. The script then compared this number of external libraries with the number of library files in the rpm file. Figure 1 shows the distributions of ERL_0 values for three datasets. ERL 1 is for C/C++ shared libraries, ERL 2 is for Java packages from sourceforge.net and ERL 3 is for Java packages from java.net.

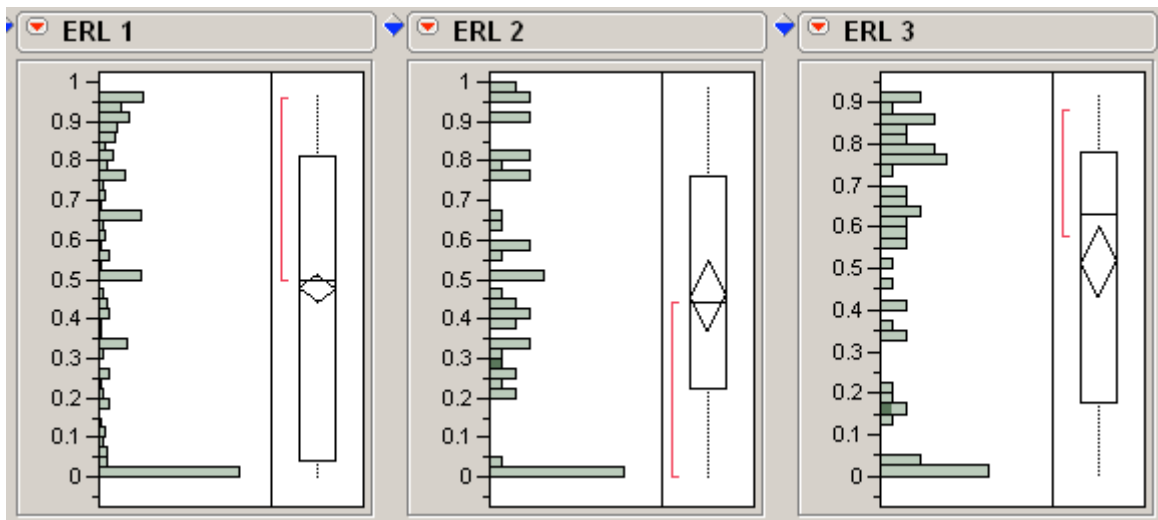


Figure 1: Summary of ERL_0 for Ramel

2.4 SAM

SAM [7], the Simple API for Object-Oriented Code Metrics, is a framework for creating metrics programs using an event driven architecture as can be seen in figure 2. The two main components are: CodeReader and CodeHandler. Code reader interacts with a parser for a given language and has two methods: `parse(InputSource)` and `setCodeHandler(CodeHandler)`. The CodeHandler is a class that contains methods which interact with the code reader by registering for event notifications. The code handler is where metrics analysis occurs. SAM can collect a larger set of metrics than current tools since it maintains context during analysis. SAM is also language independent since it specifies the types of events a parser must generate, but not how to generate those events.

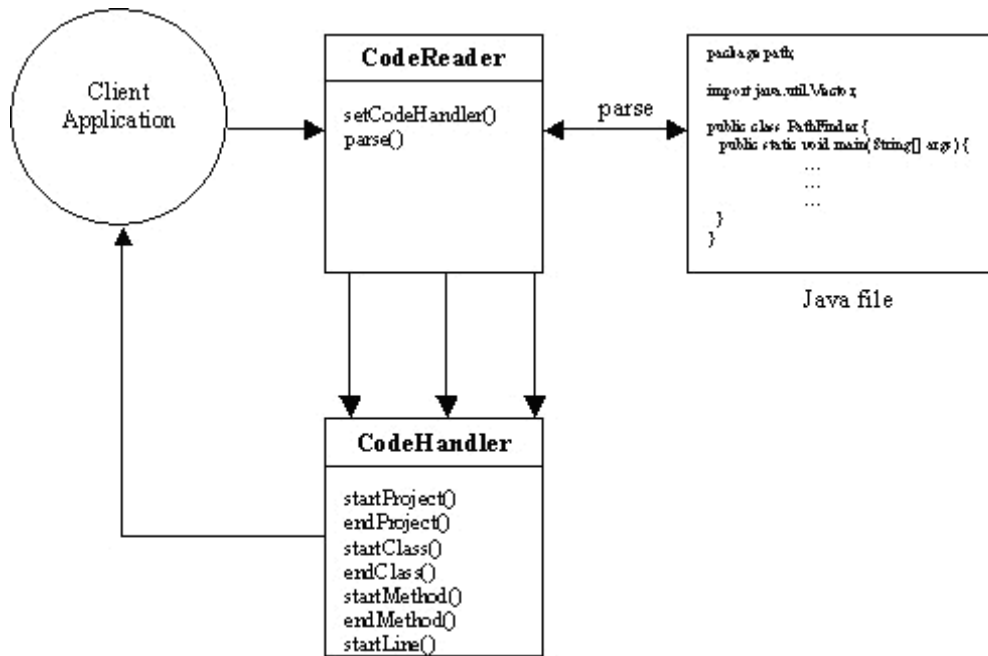


Figure 2: High level architecture of SAM

SAM has been used to create RL and RF measurement tools for C, as follows. Sam for RL and RF measurement of C is invoked with the command line:

```
$java -jar ./reusemetrics.jar [-ITL:<Internal Threshold Level>] [-ETL:<External Threshold Level>] [inputfile.c].
```

For example, given the following code (rlTest.c):

```
#include <stdio.h>
fl(int n)
{
    static int st=0; st += n;
    printf("value: %d",st);
}
main()
{
    int i; i = 1;
    fl(i);
}
```

The ERL analysis using sam is run with the following command line.

```
$java -jar reusemetrics.jar -ITL:1 -ETL:0 rlTest.c
```

Which produces the following result.

```
File, Internal Reuse Level, External Reuse Level, Internal Reuse Frequency, External Reuse Frequency
rlTest.c, 0.0, 0.33333334, 0.0, 0.33333334
```

This is because there are two internal functions: `main()` and `fl()`, and one external function: `printf()`. The gives ratios of 1/3 for external reuse level and 0/3 for internal and external reuse frequency.

3 CURRENT AND FUTURE WORK

We are working on extending the SAM framework to calculate RL and RF for C++ and Java programs. Future work may include extending it to other languages such as C#.

REFERENCES

- [1] W. Curry, G. Succi, M. Smith, E. Liu and R. Wong, "Empirical Analysis of the Correlation between Amount-of-Reuse metrics in the C programming language", *ACM SIGSOFT Symposium on Software Reusability*, 1999, pp. 135-140, .
- [2] W. Frakes and C. Terry, "Software Reuse Metrics and Models: A Survey", *ACM Computing Surveys*, Volume 28, June 1996, pp. 415-435.
- [3] W. Frakes, C. Terry, "Reuse Level Metrics", *Software Reuse: Advances in Software Reusability, 1994. Proceedings, Third International Conference on Software Reuse*, Nov 1994, 139-148.
- [4] S. Ramel, "Metrics of software reuse for free and open source software", (<http://libre.tudor.lu/results/FOSSSoftwareReuse-Metrics-v1.0.pdf>).
- [5] W. Frakes and G. Succi, "An industrial study of reuse, quality and productivity", *Journal of Systems and Software*, 2001
- [6] Frakes, Bill, "An Empirical Framework for Software Reuse Research", *Proceedings of the Third Workshop on Methods and Tools for Reuse, Syracuse University CASE Center Technical Report*, no. 9014, 5 pgs., 1990.
- [7] Adam Edelman, William B. Frakes and Charles Lillie, "SAM: Simple API for Object-Oriented Code Metrics," *ICSR 2008*, pp. 347-359.