A NEW FRAMEWORK FOR SOFTWARE LIBRARY INVESTMENT METRICS

Mohammed Q. Shatnawi[1], Ismaeil Hmeidi[2], Anas Shatnawi[3]
Jordan University of Science and Technology, Jordan
[1] mshatnawi@just.edu.jo, [2] hmeidi@just.edu.jo, [3] anasshatnawi@gmail.com

ABSTRACT

Software quality is considered as one of the most important challenges in software engineering. It has many dimensions which differ from users' point of view that depend on their requirements. Therefore, those dimensions lead to difficulty in measuring and defining the software quality properly. Software quality measurement is the main core of the software quality. Thus, it is necessary to study and develop the software measurements to meet the better quality.
The use of libraries increases software quality more than that of using generic programming because these libraries are prepared and tested in advance. In addition, these libraries reduce the effort that is spent in designing, testing, and maintenance processes.
In this research, we presented a new model to calculate the saved effort that results from using libraries instead of generic programming in the coding, testing, and productivity processes.
The proposed model consists of three metrics that are Library Investment Ratio, Library Investment Level, and Program Simplicity. An empirical analyzes has been applied into ten projects to compare the results of the model with Reuse Percent. The results show that the model has better indication of the improvement of software quality and productivity rather than
Reuse Percent.

KEYWORDS

Software quality; software engineering; software metrics; library; framework

1 INTRODUCTION

Nowadays, almost every aspect and tale of human lives depend on information and communication technology in vast manner, so the quality of software must be improved to get a better life [1] [2]. Software engineering is a science that
manages the software development process to meet high quality with lowest cost [1] [3]. Therefore, the software quality becomes a critical challenge in the software engineering. The researchers have to study and develop new software metrics to meet better software quality [4].

2 RELATED WORK

2.1 Software Quality Measurements

Software quality is one of the most important topics in software engineering; it is considered as the main core of competition in the software market [1]. It has many dimensions which differ from user requirements point of view; this leads to many definitions. These definitions can be simplified as Garvin and Juran to Fitness to use or Conformance on Requirements [4]. It is difficult to measure how much such a system is fitness to use. Therefore the researchers used software quality attributes or aspects to measure how much a system fitness to use. For example understandability, traceability, complexity, testability, and so on. Software quality is very important for both costumers and manufactures. Costumers need reliable system that easy to learn and use, and manufactures need a reusable system that is easy to maintenance and test [1] [4].
Software quality measurements and metrics are the main core of the software quality; it is autism views and reduces the difference [1]. It was necessary to study and develop the software metrics to meet the good quality. Software measurement is defined as a process of driving the quantities
from features of software entity. Software metric is a quantitative indicator of the software and the software production process attributes that is made us able to sense these attributes [4] [5]. Software metrics can be used to measure product such as source code, development process such as design process, and resources such as production cost [4]. There are two problems that are related to any measurement system, which are representation problem and uniqueness problem. The representation problem is a problem that occurs during the process of formatting particular empirical system to numerical system that made us able to sense the attribute of this system. The uniqueness problem is a problem of finding a good scale system that is used to convert the result of such a metric to another representation [1]. Many software

metrics that have correlation between each others can be integrated to build a software model [6]. Several kinds of software metrics and models were proposed to measure different types of software quality attributes. For example, Halstead Complexity Model was presented as a complexity measurement [7], and Reuse Level to measure the amount of reuse [8].

2.2 Software Reuse

Software reuse is a process of reusing existing software artifacts during software life cycle. When a systematic reuse is applied the software quality and productivity are improved by reducing the development time, and the cost [9] [10] [11] [12][13]. Software reuse does not mean that reusing the source code only, but it can be applied to any development phase such as requirement phase by reusing the experiences and documents [6] [11]. This research is interested in software source code reuse. Reusability of software artifacts is a degree of how much such an artifact is suitable to reuse to get the expected benefits [14] [15] [6] [12]. Organizations have to measure the process of software reuse to find the benefits of reusing, which can be done through software reuse metrics.

Thus, software reuse metrics became very important research field in the software engineering science.

Software reuse and reusability metrics are very important topics in software quality science, which is used to assess the reuse process [1]. Software reuse metric is a quantitative indicator that is used to measure the amount of reuse in the software [16]. Software reusability metric is an indicator that finds the ability of software component or software artifacts for reusing in other system [13].

There are several proposed software reuse and reusability metrics and models. These models can be classified as [6] to amount of reuse metrics, reusability assessment, cost benefit analysis, maturity assessment, and reuse library metrics. Amount of reuse metrics are used to measure the reuse percentage in the software as authors described in [8]. Reusability assessment is used to measure the reusability of software artifacts as authors presented in [12]. Cost benefit analysis is interested in finding the quality and productivity revenue of reuse as appears in [16]. Maturity assessment used to evaluate the reuse process to improve its weaknesses [17]. Reuse library metrics are used to find the investment of reusing library [6].

In this research the focus will be on Halstead Complexity Model HCM [7] to measure the saved effort that is resulted from library reuse. Halstead Complexity Model divides the text of program code using lexical analyzer into tokens. These tokens are classified into two factors operands and operators. After that, statistical analysis tools are applied on these factors to compute some metrics. Consequently, number of vocabulary, program length, program volume, program level, program difficulty and program effort can be calculated. All of these are related to program complexity [7].

2.3 Library Investment Model

The developed and implemented model contains three library investment metrics, which were derived based on Halstead Complexity Model [7]. Halstead presented the concept of potential program volume as a perfect program volume that is implemented using a typical programming language that can represent the needed operators by predefined functions without any need to implement the algorithms. It needs only to identify the operands. Therefore, the software quality is better whenever the program volume is closely to potential program volume, which can be done through library reuse.

The model presents three metrics, which are Library Investment Ratio (LIR), Library Investment Level (LIL), and Program Simplicity (PS), that are calculated based on Program Volume (V). The following formula is used to find the program volume:

$V = N \log (n)$.

Where:

V: Program Volume.

n: is the number of unique operands and operators.

N: is the total number of operands and operators with frequent.

The model depends on three parameters, which are original program volume (Vorg) that comes from library reuse, program volume without library reuses (Vnr), and the reduction volume (Vr) that is resulted from library reuse.

These volumes are calculated using the following formulas:

$Vr = \Sigma (fci * Vci)$.

$Vci = Nci \log (nci)$.

$Vorg = N \log (n)$.

$Vnr = Vorg + Vr$.

Where:

Vr: the reduction volume that resulted from library reuse.

fci: the frequent number of used of library component c.

Vci: the volume of library component c.

i: refers to a series of library components.

Nci: is the total number of operands and operators with repetition in a component c.

nci: is the number of unique operands and operators

in a component c.
Vorg: program volume for original program.
n: is the number of unique operands and operators.
N: is the total number of operands and operators with repetition.
Vnr: program volume without reuse.
Thus after, the following sections presented the developed metrics, starting from the Library Investment Ratio Metrics.

2.3.1 Library Investment Ratio Metric (LIR)

The LIR metric is developed to measure the reduction volume ratio that is resulted from using library instead of generic programming. Generic programming is the programming pattern that has not used any library reuse.
LIR represents the ratio between Vr and Vnr. Vr is the program volume that is resulted from library reuse. Vnr is the expected program volume that is resulted without library reuse
(Vnr). The formula of LIR is:
LIR = Vr / Vnr.
Where:
LIR: Library Investment Ratio.
Vnr: program volume without reuse.
Vr: the reduction volume that resulted from library reuse. LIR metric is used to measure the reduction in software complexity, software design cost, and software testing cost that are resulted from library reuse. The LIR result range should be between zero and one. The worst case of LLR is 0, where the library reuse has never been used (Vr = 0). The value one is unachievable because it means that there is no a new software and it is used an existing software (Vr = Vnr). Therefore, the LIR value is better whenever it is increased as much as possible.

2.3.2 Library Investment Level Metric (LIL)

Library Investment Level is used as indicator to the investment level that is resulted from library reuse. Investment level refers to the reduction level that resulted from reusing library. LIL is used to measure the improvement in software productivity. LIL is the percentage between Vr and Vorg. Vr is the reduction program volume that is resulted from library reuse. Vorg is the program volume of current program lonely. LIL is computed using the following equation:
LIL = Vr / Vorg.
Where:
LIL: Library Investment Level.
Vorg: program volume for original program.
Vr: the reduction volume that is resulted from library reuse.
The minimum value of LIL is zero (i.e. when Vr = 0). The zero value means that the library has not been invested. LIL is increased whenever library reusing increases. This metric can be used as a factor that helps the decision maker to manage the available resources to improve its productivity.

2.3.3 **Program Simplicity Metric (PS)**

Program Simplicity metric is used to measure the simplicity ratio that is resulted from library reuse.
The formula of PS is:
PS = 1 – (Vorg / Vnr).
Where:
PS: Program Simplicity.
Vnr: program volume without reuse.
Vorg: program volume for original program.
PS value should be between zero and one (i.e. 0 <= PS < 1), the value zero means that there is no any simplicity (i.e. where Vorg = Vnr), the higher PS means more simplicity ratio.

3 THE MODEL STRUCTURE

The model structure appears in figure 1.

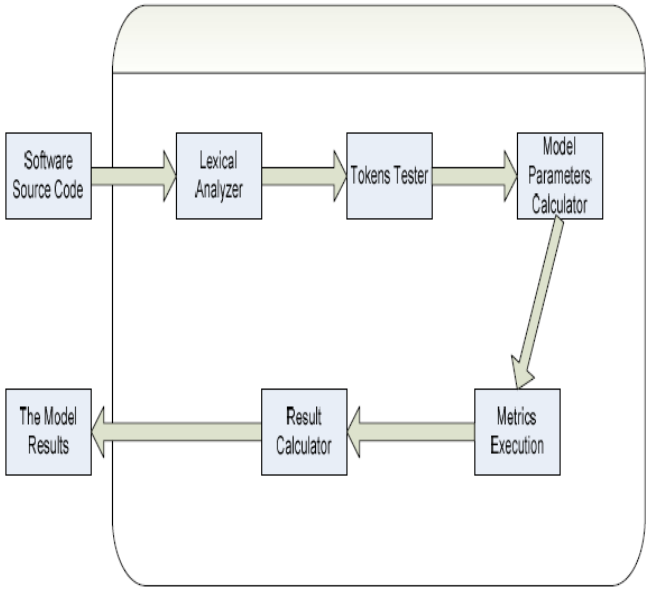

Figure 1: the Model Structure.

The model works as follow. The software source code files are sent to the lexical analyzer that divides the source code into tokens. The Token tester testes the program tokens to classify them into operands, and operators. Model Parameters Calculator calculates Vr, Vorg, and Vnr. After that, the Metrics Execution applies the model equation to find the result of LIR,

LIL, and PS. In the last step, the model results are output using Result Calculator.

3.1 The Model Main Algorithm

The main algorithm that is used to implement the model is presented in this section. It is used to manage the model components. The algorithm is:

```
Library Investment Model ()
{   Tokens = Lexical Analyzer (source code
          files)
  Oprnds_Oprtrs_List = Token Tester
                (Tokens)
Vorg_Vr_Vnr_Vector = Model Parameters
Calculator (Oprnds_Oprtrs_List)
 Result = Model Execution
(Vorg_Vr_Vnr_Vector)
Print (Result)
}
```

3.2 Lexical Analyzer

Lexical Analyzer is used to scan the target software source code to divide the program source code into tokens. These tokens can be operators, identifiers, keywords, digit, and so on. After that, the comment statements are removed by the lexical analyzer. The algorithm of lexical analyzer is:

```
Lexical Analyzer (source code files) {
        Ch = getCh ()
        While (Ch != end of file)
        {       If (is part of (operator)) {
                        OperatorToken
                =OperatorToken + Ch}
                Else If (is part of (operand))    {
                        OperandToken =
                        OperandToken + Ch     }

                Else If (is separator (Ch){
                        Save current token and
                establish a new search for a
                new token}
                Ch = get character from input file}
}
```

3.3 Tokens Tester

Token Tester is used to test the program tokens that are resulted from the lexical analyzer to find the needed factors. These factors are operands, operands frequency, operators, and operators' frequency. The algorithm of Token Tester is:

```
Token Tester () {
        Cur = Get token ()
        While (Cur != NULL) Do
        {       If (isOperator (Cur)){
                        1-OperatorsList.Add (Cur)
                        2-OperatorsCounter=
                OperatorsCounter + 1}
                Else If (isOperand (Cur)){
                        1-OperandsList.Add (Cur)
                        2-OperandCounter =
                        OperandCounter + 1     }
                Cur = next token}}
```

3.4 Model Parameter's Calculator

Model Parameter's Calculator is used to find the parameters that are used in the introduced metrics, which are, Vr, Vorg, and Vnr. These parameters are calculated based on the results of Token Tester, which are operands, operands frequency, operators, and operators' frequency. It uses the following equations:

$$Vr = \Sigma (fci * Vci).$$

$$Vorg = N \log (n).$$

$$Vnr = Vorg + Vr.$$

Where:

Vr: the reduction volume that is resulted from library reuse.

fci: the frequent number of references of the library component c.

Vci: the volume of library component c.

i: refers to series of library components.

Vorg: program volume for original program.

n: is the number of unique operands and operators.

N: is the total number of frequent of the operands and operators.

Vnr: program volume without reuse.

3.5 Metrics Execution

Metrics Execution applies the equations of the metrics to find the model results. It takes Vr, Vorg, and Vnr, which are resulted from the Model Parameter Calculator. The results of the Metrics Execution are the LIR, LIL, and PS. The following formulas are used to find these metrics:

$$LIR = Vr / Vnr.$$

$$LIL = Vr / Vorg.$$

$$PS = 1 – (Vorg / Vnr).$$

Where:

LIR: Library Investment Ratio.

LIL: Library Investment Level.

PS: Program Simplicity.

Vnr: Program Volume without reuse.

Vorg: Program Volume for original program.

Vr: the reduction volume that is resulted from library reuse.

4 EXAMPLE OF THE INTRODUCED METRICS

In this section, an example that describes the process of the model is introduced and discussed. A sample of source code that is written in C++ programming language is used to test the model metrics. In addition to, a comparison benchmark is used to compare the results of the model with Reuse Percent (RP).

Reuse Percent is proposed by [9]. It is used as indicator for the reuse amount in the software source code. RP is the ratio between the total number of lines of code in the software and number of reused line of code. The formula of RP is: RP = RSI / (RSI + SSI).

Where:

RP: Reuse Percent.
RSI: Reused Source Instructions.
SSI: Shipped Source Instructions.

The sample is presented in table 1. It uses stack file as a library file by calling Stack.h. The lower case of alphabetic "h" refers to the header file in the C++. The sample code creates an object that is belonged to the stack type. After that, the numbers (between 0 and 1) are pushed to the stack object. Then, the program retrieves the stack value using pop method to print the popped results.

Table 1: C++ Sample Program

Original program	**Stack file that is used from the library**

#include <iostream.h>
#include "Stack.h";
int main( )
{
Stack<int> s;
int stackSize;
cout<<"Enter Stack Size:";
cin>>stackSize;
for(int i = 0; i < stackSize; i++)
s.push(i);
while(!s.isEmpty( ))
cout << s.Pop( ) << endl;
return 0;
}
	Stack:: Stack() {
	topOfStack = -1;
	}
	bool Stack::isEmpty( )
	{ return topOfStack == -1; }
	bool Stack:: isFull( )
	{ return topOfStack == SIZE - 1;
	}
	void Stack::makeEmpty( )
	{topOfStack = -1;
	}
	int Stack::pop( ) {
	return theArray[topOfStack--];
	}
	void Stack::push(int & x) {
	theArray[++topOfStack] = x; }

Lexical Analyzer parses the source code into tokens; these tokens are the input to the Token Tester. Token Tester testes the tokens and divides them into operands and operators.

Table 2 shows the results of Token Tester for both original program source code and the used methods from the library. For the above example the used methods from the stack file are the stack constrictor, pop, push, and isEmpty methods.

Therfore, these methods are only considered when calculating the Vr.

Table 2: Token Tester Results.

	Original Program	The Used Functions
# of Unique Operands	7	8
Total # of Operands with Frequent	12	15
# of Unique Operators	17	13
Total # of Operators with Frequent	37	18

Model Parameter's Calculator uses the results of Token Tester to find the model parameters. The model parameters are Vorg, Vr, and Vnr. Vorg is the program volume of original program source code only. Vr is the reduction volume that is resulted from the library reuse. Vnr is the expected program volume of the software without reusing any library, in which the Vnr = Vorg + Vr. The results of the Model Parameters Calculator are:

Vorg = 49 log (24) = 67.63.
Vr = 33 log (21) = 43.63.
Vnr = Vorg + Vr = 111.26.

Metrics Execution is used to find the values of the model metrics, which are:

LIR = Vr / Vnr = 43.63 / 111.26 = 0.39.
LIL = Vr / Vorg = 43.63 / 67.63 = 0.64.
PS = 1 – (Vorg / Vnr) = 1 – (67.63 / 111.26) = 0.39.

From the above results; the LIR indicates the reduction ratio in software complexity, software design cost, and software testing cost. In this case, 39% is the reduction ratio that is resulted from reusing library. LIL is 0.64, which indicates the improvement level in software productivity. The simplicity that comes from library reuse is 0.39 based on the PS result.

Reuse Percent is calculated by finding two factors. The factors are number of reused source instructions (RSI), and shipped source code (SSI). RSI is the numbers of line of code for methods that are used from the library. SSI is the numbers of line of code for the original program only. In this case, RSI = 4, and SSI = 14. Thus, the RP is:

RP = 4 / 14 = 0.28.

By comparing the results of Library Investment Ratio and Reuse Percent, the conclusion can be drown, in which the LIR indicates that 39% of reuse percentage and 28% for RP. The large gab between them is generated from the differences in the calculation methods. RP finds the reuse ratio based on the numbers of line code, but LIR deepens in the line of code by taking the content of the line of code in its consideration. Therefore, the results of the model metrics are better than
Reuse Percent.

5 EXPERIMENTS AND RESULTS

Ten projects that developed by Maysalward are collected. MRD is a small size Jordanian company that develops mobile and online games. MRD has about 20 employees. The gathered projects are belonged to several game categories. These categories are card, puzzle, arcade, sport, and educational one. Each project has been developed by different teams. These teams have different technical and programming skills. Table 3 shows some descriptive statistical information about the gathered projects.

Table 3: Descriptive Statistical Information about Projects.

Project Name	# of Line of Code	Category
Arcanoid	6091	Arcade
Balot	18458	Card
Carrom	6973	Board
Fruity	3906	Educational
Goal Englizi	7942	Sport
Loteria	6255	Card
Minesweeper	2752	Puzzle
Tarneeb	8546	Card
Taxi Escape	3637	Arcade
Trix	14327	Card

The results of applying the model metrics into the target projects are presented and discussed in this section. Figure 2 shows the program volumes for the experimental projects. These volumes are Vorg, Vr, and Vnr. Vorg is the program volume of the original source code of the project only. Vr is the program volume of the source code of the methods that are used from the library. Vnr is the expected program volume of the project without reusing any library. The results show that Trix has the highest reduction volume Vr, and then Goal Englizi, Arcanoid, Tarneeb, Loteria, Carrom, Balot, Fruity, Taxi Escape, and Minesweeper in decreasing order.

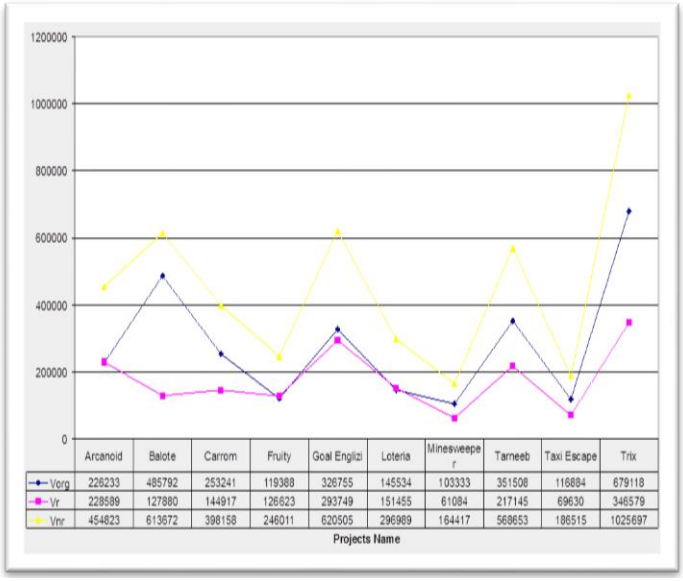

Figure 2: The Vorg, Vr, and Vnr of Experimental Projects.

6 CONCLUSION

In this research, three library reuse metrics developed that measure library investment based on Halstead Program Volume. These metrics are Library Investment Ratio (LIR), Library Investment Level (LIL), and Program Simplicity (PS). LIR is used to measure the reduction in software complexity, software design and testing cost that are resulted from library reuse instead of using generic programming. The formula of LIR is:

LIR = Vr / Vnr.

Where:

Vnr: program volume without reuse.
Vr: the reduction volume that is resulted from library reuse.

LIL measures the investment level, which is related to the software productivity. LIL is used as a factor that helps the decision maker to manage the available resources to improve its productivity. LIL is calculated as follow:

LIL = Vr / Vorg.

Where:

Vorg: program volume for original program.
Vr: the reduction volume that is resulted from library reuse.

Program Simplicity is used as indicator to the simplicity ratio that is resulted. PS is:

PS = 1 – (Vorg / Vnr).

Where:

Vnr: program volume without reuse.
Vorg: program volume for original program.

The model is implemented using Java programming language. The model is applied into several projects that collected from Maysalward Inc Company to find the results of our model. The results show that library reuse is improved software quality, and productivity. It is reduced production time, and development cost. The model is compared with Reuse Percent (RP). The results show that the model has introduced better results rather than RP.

7 REFERENCES

[1] S Abd Alazeez, A Ali. Object Oriented Software Quality. Dar Majdalawi Pub. & Dis, 2006.

[2] M Usrey, K Dooley. The Dimensions of Software Quality. Quality Management Journal, 3(3): 67-86, 1996.

[3] S. Kan. Metrics and Models in Software Quality Engineering 2nd. Addison-Wesley Professional, 0-201-72915-6, 2002.

[4] T Honglei, S Wei, Z Yanan. The Research on Software Metrics and Software Complexity Metrics. International Forum on Computer Science-Technology and Applications, 2006.

[5] S Yu, S Zhou. A Survey on Metric of Software Complexity. Information Management and Engineering (ICIME), the 2nd IEEE International
Conference, pp.352-356, 2010.

[6] W FRAKES, C TERRY. Software Reuse: Metrics and Models. ACM Computing Surveys, Vol. 28, No. 2, 1996.

[7] M Halstead. Elements of software science. Elsevier North Holland Inc,
1977.

[8] W Frakes, C Terry, Reuse Level Metrics, IEEE Proceedings of the 3rd International Conference on Software Reuse: Advances in Software Reusability, 1994.

[9] W Frakes, K Kang. Software Reuse Research: Status and Future. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2005.

[10] S Singh, M Thapa, S Singh, G Singh. Software Engineering - Survey of Reusability Based on Software Component. International Journal of Computer Applications (0975 – 8887) Volume 8 – No.12, October 2010.

[11] E Almeida, A Alvaro, D Lucredio, V Garcia, S Meira. RiSE Project: Towards a Robust Framework for Software Reuse. Information Reuse and Integration, Proceedings of the IEEE International Conference, pp. 48- 53, 8-10, 2004.

[12] P Mohagheghi, R Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. Empir Software Eng 12:471–516 DOI 10.1007/s10664-007-9040, 2007.

[13] G Singaravel, V Palanisamy, A Krishnan. OVERVIEW ANALYSIS OF REUSABILITY METRICS IN SOFTWARE DEVELOPMENT FOR RISK REDUCTION. Innovative Computing Technologies (ICICT), pp.1-5, 12-13, 2010.

[14] K Aggarwal, Y Singh, A Kaur, R Malhotra. Software Reuse Metrics for Object-Oriented Systems. Conference on Software Engineering
Research, Management and Applications, 2005.

[15] Huan Li, A Novel Coupling Metric for Object-Oriented Software Systems, State Key Laboratory of Software Engineering, 2008.

[16] J Poulin, J Caruso, A Reuse Metrics and Return on Investment Model, IEEE Proceedings of the 2nd Workshop on Software Reuse: Advances
in Software Reusability, I 1993.

[17] S Shiva, L Abou Shala. Software Reuse: Research and Practice.International Conference on Information Technology (ITNG'07) 0-7695-
2776-0/07, IEEE, 2007.

[18] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil.
Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[19] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol2. Oxford: Clarendon, 1892, pp.68–73.

[20] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[21] K. Elissa, "Title of paper if known," unpublished.

[22] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[23] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982.

[24] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.