55

A Multi-layer Metrics Combination Model for Reusable Software Components

Sen-Tarng Lai[ab] and Chien-Chiao Yang[a]

[a]Dept. of Electronic Engineering, National Taiwan Institute of Technology, 43 Keelung Road, Section 4 Taipei, TAIWAN

[b]Telecommunication Laboratories, Ministry of Transportation and Communications, 9 Lane 74, Hsin-Yi Road, Sec. 4 Taipei, TAIWAN

Abstract

Software reuse is an important approach to increase software quality and productivity. There are many factors may affect the result of software reuse, however, software component extraction is one of the most important and influential factors. Defining a perfect software reuse metric is a necessary condition for identify high reusable software components and retrieve the more suitable candidate components. In this paper, we propose a multi-layer metrics combination model for reusable component extraction. In this model, each layer combination can apply different linear combination models for specific purpose. This feature provides high flexibility to adjust the weighting value of combination model and high capability to improve measurement of reusable software component. Based on the multi-layer metrics combination model, we also can create a qualification threshold for extracting the reusable software component and defining a ranking schema for candidate components in component retrieval.

Keyword Codes: D.2.m; D.2.8
Keywords: Software Reuse; Software Metrics; Combination Model; Primitive Metrics.

1. INTRODUCTION

Software reuse has the potential to *improve software quality, reduce development costs*, and *increase productivity*. Improving software quality and productivity is the primary objective of software reuse. There are many problems must be resolved for attaching this objective. For example, five major steps of software reuse are: extracting reusable software components, packaging software components, classifying and retrieving software components, modifying software components, and adapting software components to the new software system. In first step, software component extraction is one of the most important and influential factors. Each phase's products of software life cycle has the potential for reuse. Code reuse is better understand and more prevalent by far than other software development phase of reuse [1]. Since code components have a high degree of specificity, the most highly reusable components tend to be small. A code component in a reuse library is likely to be of little value and the detailed design documents should be very valuable in understanding code component. Thus it is extremely important that detailed design documents associated with code modules to be a reusable component [2].

M. Lee et al. (eds.), *Software Quality and Productivity*

Similarly, each module of the detailed design and coding phase has the opportunity to be a reusable software component in existing system. In software reuse, to determine the qualification of reusable software components is one of the most important job [3]. Reusing high quality software components is a necessary condition for improving software quality and productivity. Quality of software component is a key point for judge the qualification of reusable software components. In an important paper by Boehm [4], an attempt is made to define software quality in terms of some high-level software characteristics. These characteristics are: reliability, portability, efficiency, human engineering, testability, understandability, modifiability.

Some software characteristics can help us extract and identify reusable software components. These software characteristics are high-level software characteristics which can be decomposed into several primitive characteristics. On the other side, it is necessary to combine the primitive characteristics for measuring a particular characteristic. In this paper, we propose *a multi-layer metric combination model* which is based on the linear combination models. In this model, each layer combination can apply different linear combination models for increasing flexibility. Separate combination for different purposes can improve the efficiency and capability of measurement. In Section two, we describe the existing software metrics for design and coding phase. Then we describe the metrics' data collection and normalization in Section three. In Section four, we discuss the metric combination model and define the multi-layer metric combination model. Finally, we make a summary and discuss our future work in Section 5.

2. SOFTWARE METRICS FOR DESIGN AND CODING PHASES

Although, some metrics are impossible to measure or predict directly, there are still exist many software metrics for measuring the characteristic of software. According to the *meta-metrics* defined by the conte [5], we are interested in metrics that are *simple, robust, useful for design and coding phase,* and that *can be analyzed properly.* These metrics include:

(a) *Size metric (Lines of code):* A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. It is one of the most familiar software measure.

(b) *Data structure metric:* An intra-modular measure of the information flow complexity of each module *M* in a system is defined by the Henry-Kafura [6], as follow:

*Complexity of module M = length (M) * (fan-in (M) * fan-out (M)) ** 2*

The Shepperd [4] refines the above formation like the following:

*Complexity of module M = (fan-in (M) * fan-out (M)) ** 2*

(c) *Logic structure metric (McCabe's Cyclomatic complexity):* McCabe has observed that the difficulty of understanding a program is largely determined by the complexity of the control flow graph for that program [7]. For each module of a system, we can draw the control flow graph G for that module. The cyclomatic complexity metric for the module, denoted V(G), is the number of regions into which G divided the page. McCabe has shown that V(G) is equivalent to one more than the number of decisions in the flow diagram.

(d) *Nesting level of program construct:* As a general guideline, nesting of program constructs to deprhs greater than three or four levels should be avoided.

(e) *Ratio of statement and branch coverage:* There are many coverage measures for unit programs testability: statement coverage, branch coverage, decision coverage, and path coverage. Statement and branch coverage are more useful in unit testing.

(f) *Coupling and cohesion:* Coupling is a measure of the degree of interdependence between modules. Cohesion is an attribute of individual modules, describing their functional strength.

3. DATA COLLECTION AND SCALE NORMALIZATION

In this Section, we describe how to collect the metric data and normalize the metric data for combination.

3.1. Data collection

Using *static program analyzer* and each *metric definition* or *formula*, we can compute the values of Halstead's Measure, Information Flow Structure Metric, McCabe's Cyclomatic Complexity and nesting level of programming structure. Based on *the test coverage analyzer*, we can get the values of statement coverage and branch coverage of unit testing. About the coupling and cohesion, there are a number of proposed classes of coupling and cohesion that are believed to provide an ordinal scale of measurement. It is difficult to determine the scale of coupling and cohesion by traditional software tools. According to the clearly definitions of scale of coupling and cohesion, we apply them into a rule-based system which can help us determine the scale of coupling and cohesion, from highest to lowest.

3.2. Scale normalization

In general case, a potential software characteristic is combined by several primitive characteristics. Some software primitive metrics which are concerned with the quality of software component, have different scale values in their representation. For combining these primitive metrics, we recommend all scale values of the primitive metrics shall be normalized among 0 and 1. Near to 1 represent the most desirable value, and near to 0 represent the least desirable value. The following two tables are the examples to represent the normalized value of primitive metrics: Table 1 is the normalized value of lines of code and Table 2 is the normalized value of nesting level of program construct.

Table 1.
Normalized value of Lines of Code

LOC	Normalized Value
1 - 50	1.0
51- 100	0.8
100 - 150	0.6
150 - 200	0.4
200 - 250	0.2
> 250	0.0

Table 2.
Normalized value of nesting level of program construct

Level of nesting of program construct	Normalized Value
0 - 1	1.0
2 - 3	0.8
4 - 5	0.5
6 - 7	0.3
8 - 9	0.1
> 9	0.0

4. METRICS COMBINATION

In [4], each high-level software characteristic can be decomposed into several primitive characteristics. On the other side, it is necessary to combine the primitive metrics for measuring a particular characteristic. In this Section, we will discuss the metric combination models.

4.1. The linear combination

The goal of metrics combination is to improve the measurement capability and flexibility. In this Section, we consider three linear combination models:

(a) *Equally Weighted Linear Combination:*

This model is the simplest combination to form. Each primitive metric has an equal weight constant.

$$HLM = \frac{1}{n}\sum_{i=1}^{n} PM_i$$

where,

- *PM:* Normalized value of Primitive Metric
- *HLM:* High-level Metric

(b) *Unequally Weighted Linear Combination:*

In this model, according to the optimistic and pessimistic predications, different weights are assigned to different primitive metrics. Such that,

$$HLM = \sum_{i=1}^{n} W_i PM_i$$

, and

$$1 = \sum_{i=1}^{n} W_i$$

where W_i is the weight constant of ith primitive metricPM_i.

(c) *Dynamically Weighted Linear Combination:*

In this model, we can adjust the weights of any primitive metric for adapting different applications.

4.2. Multi-layer metrics combination model

In this paper, we select six primitive metrics to measure the reusable characteristic of software component. These metrics are simple, valid, robust, useful for development, and that can be analyzed properly. According to the different characteristics of primitive metrics, we divide them into three groups:

(a) *Complexity metrics:* include control flow, information flow, line of code, and nesting level of program construct.

(b) *Modularity metrics:* include coupling and cohesion.

(c) *Testability metrics:* include statement coverage and branch coverage.

For combining these primitive metrics, we take three steps. In first step, we consider the characteristics which are potentially contradictory. For example, reduces in logic structure complexity usually result in increased data structure complexity. Then, we combine the primitive metrics which belong to same class in step two. Finally, we combine these three classes' metrics into a software reuse metric. We call this combination model *a multi-layer metrics combination (MLMC) model* (see Figure 1). In this model, each layer combination can apply different linear combination models for increasing flexibility. Separate combination for different purposes can improve the efficiency and capability of measurement.

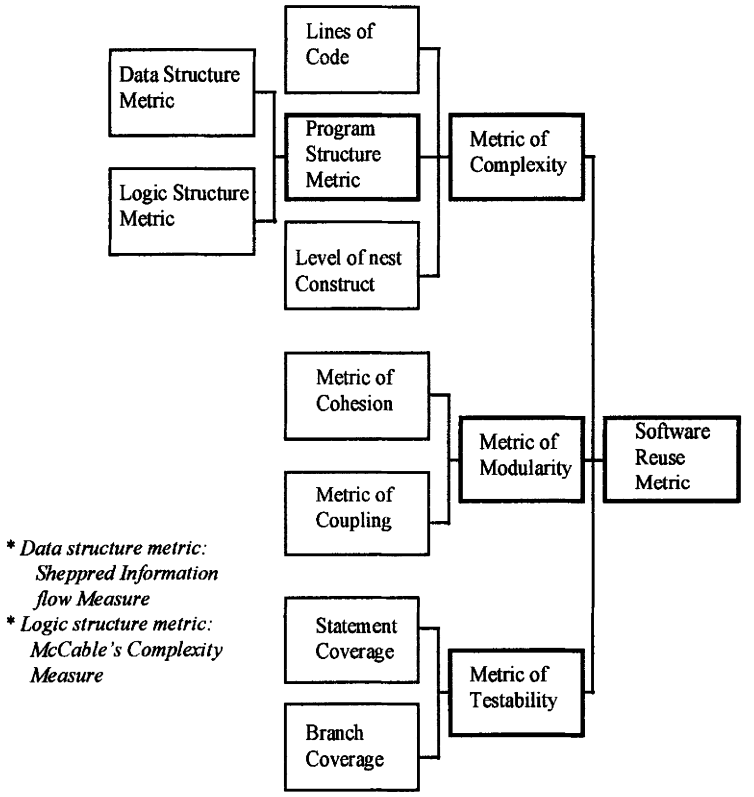

Figure 1. The Mutli-layer metrics combination model

5. CONCLUSIONS

A potential software characteristic is composed of several primitive characteristics. Combination these primitive characteristics is a major approach to measure the potential characteristic. In software reuse, it is necessary to consider more than one primitive characteristic for extracting suitable and reusable software components. In this paper, we propose the multi-layer metric combination model which based on the linear combination models for reusable components. The goal of this combination model is to improve *the measurement capability and flexibility* for reusable software components. In *MLMC* model, each layer combination can apply different linear combination models for specific purpose. This feature provides high flexibility to adjust the weighting value of combination model and high capability to improve measurement of reusable software component.

Based on the *MLMC* model, we also can create an automatic extraction tool for extracting the reusable components from existing software and define a ranking schema for candidate components in component retrieval. We propose a component extraction approach which based on the primitive metric's combination model. Using this approach, we can clear define the extraction qualification for reusable components.

Our future work is to define the qualification threshold for extracting reusable software components from existing software with the *MLMC* model. For improving the practically of this model, we will use mass experiment data help us adjust the weight constants and normalization value of primitive metrics. Further, to develop an automatic reusable software component extraction tool is helpful in software reuse.

REFERENCES

1. LANERGAN, R.G. and GRASSO, C.A.: **'SOFTWARE ENGINEERING WITH REUSABLE DESIGNS AND CODE'**, *IEEE Trans. Software Eng.*, 1984, Vol.10, (5), pp.498-501
2. TRACZ, W.: **'SOFTWARE REUSE MYTHS'**, *ACM SIGSOFT Software Emgineering Notes*, 1988, Vol.13, (1), pp.17-21
3. CALDIERA, G. and BASILI, V.R.: **'IDENTIFYING AND QUALIFYING REUSABLE SOFTWARE COMPONENTS'**, *IEEE Computer*, 1991, Vol.24, (2), pp. 61-70
4. BOEHM, B.W., BROWN, J.R. and LIPOW, M.: **'QUANTITATIVE EVALUATION OF SOFTWARE QUALITY'**, *Proceedings of the Secondard International Conference on Software Engineering*, 1976, pp. 592-605
5. CONTE, S.D., DUNSMORE, H.E. and SHEN, V.Y.: **'SOFTWARE ENGINEERING METRICS AND MODELS'**, Benjamin/Cummings, (Menlo Park, 1986)
6. FENTON, N.E.: **'SOFTWARE METRICS - A REIGOROUS APPROACH'**, (Chapman & Hall, 1991)
7. MCCABE, T.: **'A COMPLEXITY MEASURE'**, *IEEE Trans. Software Eng.*, 1976, Vol.2, (4), pp.308-320