International Journal of Information Technology and Knowledge Management
January June 2009, Volume 2, No. 1, pp. 33-35

VALIDATION OF OVERALL PROGRAM WEAKNESS MEASURE

Jitender Kumar Chhabra, Ravinder Singh Bhatia & V. P. Singh

The measurement of complexity of software has always been a demanding area for software industry and key research area for the researchers of software engineering. But complexity measurement alone is never sufficient and acceptable until it has been validated on basis of some acceptable frameworks. One of the most widely discussed and accepted framework was proposed by Briand et al and is being used very frequently to validate various software metrics. The overall program weakness is one structural complexity measure which has been quite useful indicator of the complexity. This paper validates this measure using the Briand's framework. The results of this study show that the overall program weakness metrics satisfies all properties and parameters required by the formal evaluation framework. Thus value of overall program weakness measure can be used by software managers as a tool to control the complexity.

1. Weakness of a Module

The module weakness has been defined in [1] using the average number of live variables (LV) and average life of variables (β), as:

$$WM = LV^* \beta$$

Average number of live variables and average life of variables can be found using some automated tools [2, 3]. The concept of module weakness was extended to measure the program weakness by finding average of weaknesses of all modules. But there is a significant difference between computing module weakness and program weakness. The program weakness cannot be computed by just averaging the module weaknesses of all modules. An initial definition of program weakness was proposed as [1]

$$WP = \left(\sum_{i=1}^{m} WM_i \right) \Big/ m \qquad (1)$$

where WM_i: weakness of ith module.

WP : weakness of the program

m : Total number of modules in the program.

But this definition lacked in considering the module coupling while shifting from module to program weakness. As per this definition, two different programs had same weakness if both have same values of module weaknesses although one program had lot of coupling while other program had no coupling at all, which was not acceptable. Hence this averaging of module weaknesses is unacceptable to compute weakness of a program.

2. Overall Weakness of a Program

The definition of the weakness of the program is redefined by including the effect of the coupling among the modules [4]. So now average weakness of a program is defined as:

$$AWP = \frac{\sum_{i=1}^{m} WM_i + \sum_{i=1}^{m} \sum_{j=1}^{n} CM_{ij}}{m} \qquad (2)$$

where WM_i : Weakness of *i*th module.

CMij : *j*th type of coupling of ith module.

m : Total number of modules.

n : Total number of types of coupling

Here $j = 1$ represents data coupling, $j = 2$ represents control coupling, $j = 3$ refers to global coupling, $j = 4$ corresponds to environment coupling etc.

In order to reflect overall weakness of the program, the averaging does not help. Most of the researchers have recommended that the complexity of the bigger system is always aggregation of their individual complexities, and thus the overall weakness of the program will be defined as summation of module weaknesses along with their coupling summation. i.e.

$$OWP = \sum_{i=1}^{m} WM_i + \sum_{i=1}^{m} \sum_{j=1}^{n} CM_{ij} \qquad (3)$$

3. Validation of Overall Program Weakness Measure

Briand *et al*. developed a formal list of five properties for evaluating software complexity metrics [5]. These properties are used to determine the usefulness of various software complexity measures and a good complexity measure should satisfy most of the Briand's properties. These properties have

* National Institute of Technology, Kurukshetra-136119 INDIA, *E-mail: jitenderchhabra@rediffmail.com, rsibhatia@yahoo.co.in, vpsingh72@yahoo.com*

been recommended by many eminent researchers for evaluating their measures [6-10] and are well accepted in the literature to evaluate the usefulness of the metrics. The above described program weakness measure is validated here using these five properties proposed by Briand et al. Before applying the program weakness metric against this framework, let us define the basic terms and properties for complexity measures given in the framework.

System: *A* system *S* is represented as a <*E*, *R*>, where *E* represents the set of elements of *S*, and *R* is a binary relation on *E* ($R \subseteq E \times E$) representing the relationships between elements of *S*.

Module: For a given a system $S = <E, R>$, a system $m = <E_m, R_m>$ is a module of *S* if and only if $E_m \subseteq E$, $R \subseteq E_m \times E_m$ and $R_m \subseteq R$. A module *m* may be a code segment or a subprogram.

For the purpose of evaluation of program weakness complexity measures, *E* is defined as the set of modules and *R* as the set of usages of modules.

Complexity: The complexity of a system S is a function Complexity(S) that is described by Property 1 to Property 5.

Property 1 (Nonnegative): The complexity of a system $S = < E, R >$ is nonnegative if Complexity (S) ≥ 0.

Proof: Since program weakness is defined as average of summation of module weakness and coupling of the module with other modules, which are always non-negative numbers. The coupling of a module with other modules can be either zero or a positive number. Similarly module weakness is computed through average life of variables and average number of live variables and both are always positive or zero. So, this property is well satisfied by program weakness metric.

Property 2 (Null Value): The complexity of a system $S = < E, R>$ is null if *R* is empty i.e.

$$R = \varnothing \Rightarrow Complexity\ (S) = 0.$$

Proof: As defined above, *R* is a set of usages of modules in context of program weakness. If there is no usage of any module in the program, value of the program weakness would be obviously zero, because in absence of modules, the module weakness will be zero, and the coupling amongst modules will also be zero obviously. Hence, Property 2 is also satisfied by this measure.

Property 3 (Symmetry): The complexity of a system $S = <E, R>$ does not depend on the convention chosen to represent the relationships between its elements i.e. ($S = <E, R>$ and $S^{-1} = <E, R^{-1}>$) ⇒ *Complexity*(*S*) = *Complexity*(S^{-1}).

Proof: As per the definition of the program weakness metric, its value for a program depends on the variables' life and inter-connection among modules, but it is never dependent on the conventions chosen to represent the usage of the modules. In fact the representation of the usage of the module may be different in different programming languages, but program weakness does not get affected by the syntax. Thus, the program weakness satisfies this property.

Property 4 (Module Monotonicity): The complexity of a system $S = <E, R>$ is no less than the sum of the complexities of any two of its modules with no relationships in common i.e.

$S = <E, R>$ and $m_1 = <E_{m1}, R_{m1}>$ and $m_2 = <E_{m2}, R_{m2}>$ and $m_1 \cup m_2 \subseteq S$ and $R_{m1} \cap R_{m2} = \varnothing$) ⇒ *Complexity(S)* ≥ *Complexity* (m_1) + *Complexity* (m_2).

Proof: This property is very important for validation of any measure. Let us consider a program consisting of 2 modules, each having module weakness as WM_1 and WM_2. If there is no inter relationship between these two modules, then there does not exist any type of coupling between these two modules. So as per equation (3), the complexity of this program will be (WM1 + WM2). However if the modules have some inter-connection (which is quite common) then the value of coupling will never be zero. Let us assume C_{11}, C_{12}, C_{13}, … be various types of coupling of module 1 and C_{21}, C_{22}, C_{23}, … are various types of coupling of module 2. So now overall program weakness is computed using equation (3) as

$$OWP = (WM_1 + \sum_{j=1}^{n} C_{1j}) + (WM_2 + \sum_{j=1}^{n} C_{2j})$$

$$= WM_1 + WM_2 + \sum_{j=1}^{n} C_{1j} + \sum_{j=1}^{n} C_{2j}$$

Thus $OWP \geq WM_1 + WM_2$

So the program weakness of the program having some non-zero value of coupling will be definitely higher than a program having no coupling. Thus Property 4 is well-satisfied by the measure under consideration.

Property 5 (Disjoint Module Additivity): The complexity of a system $S = <E, R>$ composed of two disjoint modules m_1, m_2, is equal to the sum of the complexities of the two modules i.e.

($S = <E, R>$ and $S = m_1 \cup m_2$, and $m_1 \cap m_2 = \varnothing$) ⇒ *Complexity* (*S*) = *Complexity* (m_1) + *Complexity* (m_2).

Proof: If two disjoint modules are combined together, then the coupling amongst these modules will be zero due to their disjointness. Thus as per equation (3), the program weakness of the composed program will be

$$OWP = (WM_1 + 0) + (WM_2 + 0) = WM_1 + WM_2$$

Thus WP is equal to summation of module weakness of module 1 and module 2, and hence the property 5 is also satisfied.

CONCLUSION

The overall program weakness is computed by aggregating the module weaknesses and coupling values of all modules. The program weakness measure is validated here using the well known framework of Briand *et al*. The overall program weakness metric is found to satisfy all five properties of this framework. The theoretical and formal evaluation carried out in this paper indicated that overall program weakness is a robust and useful measure of complexity.

References

[1] Yogesh Singh, Pradeep Bhatia, "Module Weakness : A New Measure", *ACM SIGSOFT*, (July,1998), 82–82.

[2] Conte Dunsmore, Shen, "Software Engineering Metrics and Models", Cummings Pub. Coi. Inc. USA, (1986).

[3] Bache, Monica, "Measures of Testability as a Basis for Quality Assurance, *Software Engineering Journal*, (March, 1990), 86–92.

[4] K. K. Aggarwal, Yogesh Singh, Jitender Kumar Chhabra, "Computing Program Weakness using Module Coupling", *ACM SIGSOFT*, **27**, (1), (Jan 2002), 63–66.

[5] L. C. Briand, S. Morasca, V. R. Basili, "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, **22** (1), (Jan 1996), 68–86.

[6] S. Misra, A. K. Misra, "Evaluating Cognitive Complexity Measure with Weyuker Properties" *Proceedings of Third IEEE International Conference on Cognitive Informatics* (ICCI2004), 103–108.

[7] S. Misra, A. K. Misra, "Evaluation and Comparison of Cognitive Complexity Measure", *ACM SIGSOFT Software Engineering Notes*. **32**, (2), (Mar 2007), 1–5.

[8] S. Misra, 'Validating Modified Cognitive Complexity Measure' *ACM SIGSOFT Software Engineering Notes*. **32**, (3), (2007), 1–5.

[9] D. S. Kushwaha, A. K. Misra, "Robustness Analysis of Cognitive Information Complexity Measure using Weyuker's Properties". *ACM SIGSOFT Software Engineering Notes*, **31**, (1), (2006), 1–6.

[10] Y. Wang, "On the Informatics Laws and Deductive Semantics of Software", *IEEE Transactions on Systems, Man and Cybernetics.* **36**, (2), (2006), 161–171.