World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:8, 2014

Further the Effectiveness of Software Testability Measure

Liang Zhao, Feng Wang, Bo Deng, Bo Yang

***Abstract*—**Software testability is proposed to address the problem of increasing cost of test and the quality of software. Testability measure provides a quantified way to denote the testability of software. Since 1990s, many testability measure models are proposed to address the problem. By discussing the contradiction between domain testability and domain range ratio (DRR), a new testability measure, semantic fault distance, is proposed. Its validity is discussed.

***Keywords*—**Software testability, DRR, Domain testability.

Open Science Index, Computer and Information Engineering Vol:8, No:8, 2014 publications.waset.org/9999083.pdf

I. Introduction

WITH the invasive application of information system in society, software begins to take an important role in everyday life. Software quality becomes more and more important. Software testing is a main method for software quality assurance. With the increment of software scale and complexity, its testing becomes more difficult. This suggests that software should be designed to be tested easily and testability should be adopted as a design parameter.

Software testability evolves from hardware. For very large circuits the problem of test generation is in general NP-complete and thus very intractable [1], and so people introduce testability analysis to indicate how easy or difficult to generate tests for a circuit and identifying the areas of poor testability. Software is complex temporal logic; its test problem is more complex than hardware. So the analysis of software testability is more complex and difficult than hardware.

IEEE definition of software testability is “the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met” [8]. It has been admitted as a quality factor in ISO 9126[9].To accurately indict testability, there has been a great deal of work that deal with testability measure since 1990s.

In this paper, we focus on analyses of the contradiction between two testability measure, which are DRR (Domain to Range Ratio) and domain testability.

The paper is organized as follows: Section III introduces related works on software testability measure. Section III introduces domain testability, DRR and the contradiction of between them. Section IV comprehensively discusses the problem and proposes a new measure, semantic fault distance. Section V gives our conclusion and future work on testability measure.

Liang Zhao, Dr., is with the Beijing Institute of System Engineering, CO 100101China (phone: 8610-64836117; fax: 8610-64836117; e-mail: liangzhao@tsinghua.org.cn).

F. Wang, Dr., B. Deng, and B. Yang are with the Beijing Institute of System Engineering. (e-mail: wangfeng_bise@aliyun.com, bodeng@163.com, sxq546@163.com).

II. Related Work

Testability can be predicted as soon as the system is specified. Freedman proposed domain testability to address the problem of input inconsistency and output inconsistency, which involved use of the concepts of observability and controllability [2]. Voas defined that testability of a program is a prediction of the tendency for failures to be observed during random black-box testing when faults are present [3], [6]. They used DRR to indicate the inexplicit information loss, the bigger the DRR, the more information loss and so the testability is smaller. In object-oriented software, Baudary took the number of class interactions in a UML class diagram as testability measure to indicate the potential conflict that may occur in test, the more class interactions the lower the testability [10], [17].

Software structure has direct effect on test. Some complexity measures are assumed to imply the number of the test cases in term of structural coverage and so can indicate the effort to test the program to a certain degree. Richard defined testability as the number of test cases that needed to satisfy certain test criteria, and computed it on the program control flow [11]. Yeah accurately count the number of the test cases that needed to cover the program and introduced block normalization and structural normalization before the counting that based on data flow [12]. Nguyen used information transfer of between component and its context to indicate the testability of certain component [7].

Fault/failure model reflects the behavioral characteristics of the software during testing. Reference [13] defined testability as a prediction of the probability that existing faults will be revealed during testing given an arbitrary input selection criterion C. *PIE* is proposed to analysis the sensitivity of statement location by statically analysis its execution rate (E), infection rate (I) and propagation rate (P), which can indicate the effort to execute the test to gain certain confidence. But the computation of PIE is quite complex. Lin [14] reduced the estimation of the probability estimate by analyzing the semantic of the code and program structure. Bruce [15] used one sample test suite to estimate the PIE rate. These method decreases the computation complexity with the cost of precision loss.

III. Two Specification-Based Software Testability Measures

Specification defines the problem difficulty while a program implementation is one way to solve it. Addressing the testability problem from scratch may help to find a good

Open Science Index, Computer and Information Engineering Vol:8, No:8, 2014 publications.waset.org/9999083.pdf

answer. Specification specifies in a complete, precise, verifiable manner, the requirement, design behavior, or other characteristics of a system or component [8]. Every item of Software can be viewed as a function or mapping *F* according to some specification *S*, from a set of inputs values (its domain *D*) to a set of output values (its range *R*).Both *D* and *R* are of certain data types or data structures. Programmer has to map the types of the problems to the data types or structures available in certain computer program language. Function *F* is defined in *S* and implemented in *P* which also maps from *D* to *R*:

$$F: D \rightarrow R$$

A. Domain Testability

A software component is observable "if distinct outputs are generated from distinct inputs", a software component is controllable "if, given any desired output value, an extra input exits which forces the component output to that value". Most function and procedure are not a priori observable and controllable. The modifications required to achieve domain testability are called extension. Domain testability refers to the ease of modifying a program so that it is observable and controllable.

Observable extensions are achieved by introducing new input variables so that the component becomes observable. Observability is the ease of determining if specified inputs affect the output and O_b, a measure of observability can be obtained by taking log_2 of the product of the cardinalities of the types of the additional input variables. Controllable extensions are achieved by modifying outputs for the given component so that it becomes controllable, i.e. all claimed outputs are attainable with some input, thus controllability is the ease of producing a specified output from a specified input, C_t, can be measured by taking log_2 of the product of the cardinalities of the types of the modified output variables.

For component *P* implemented function *F* according Specification *S* mapping from domain *D* to range *R*, observability and controllability can be characterized as:

Observable:$\forall x, y \in D\ F(x) \neq F(y) \Rightarrow x \neq y$

Controllable: $\forall r \in R\ \exists z \in D\ F(z) = r$

$O_b = log_2(|T_1| \times |T_2| \times \ldots \times |T_n|)$ (T_1, T_2, … ,T_n are observable added parameter types)

$C_t = log_2(|T_1| \times |T_2| \times \ldots \times |T_m|)$ (T_1, T_2, … , T_m are controllable extended types)

After the modification, the observable and controllable version is achieved. The domain testability of the original version is (O_b,C_t), while the new version is (0, 0), which indicts no extension is needed.

B. Domain to Range Ratio

Voas contended that testability of a program is a prediction of the tendency for failures to be observed during random black-box testing when faults are present [3],[5]. They propose "Internal data state collapse occurs when two different data states are input to some sub-component in a program and yet that sub-component produces the same output state" and "When internal state collapse occurs, the lost information may have included evidence that internal states were incorrect. Since such evidence is not visible in the output, the probability of observing a failure during testing is reduced." Further he contended that "the testability of a program is correlated with the domain-to-range ratio……as the DRR of the intended function increase, the testability of an implementation of that function decrease". In other words, high DRR is thought to lead to low testability and vice versa. According the characteristic of DRR, they classified software into fixed domain/fixed range (FDFR), variable domain/variable range (VDVR), and variable domain/fixed range (VDFR).

$$DRR = \frac{|D|}{|R|}$$

C. The Contradiction between Domain Testability and DRR

According to the definition of DRR and domain testability, Woodward [6] calculated the DRR metric of the domain extended program.

$D' = D \cup \vartriangle D$ and $R' = R - \vartriangle R$

$$|D \cup \vartriangle D| = |D| + |\Delta D|$$
$$|R - \vartriangle R| = |R| - |\Delta R|$$
$$DRR' = \frac{|D'|}{|R'|} = \frac{|D| + |\Delta D|}{|R| - |\Delta R|} = DRR \times \frac{\left(1 + \frac{|\Delta D|}{|D|}\right)}{(1 - \frac{|\Delta R|}{|R|})}$$

So can get:$DRR' > DRR$

Above result means that according to Voas's testability definition, testability decreases while domain testability increases. This is the contradiction between DRR and domain testability. What makes this contradiction? Which one is more reasonable?

IV. The Discussion

A. The Interpretation of DRR

To deduce the probability of random test to find the fault, Reference [4] introduced semantic fault size. Semantic fault size (*SFSZ*) is the "the relative size of the sub domain of *D* for which an output mapping is incorrect".

$$SFSZ = \frac{|D_f|}{|D|}$$

SFSZ is defined as the ratio of the inputs that are mapped to the wrong outputs. It can be used to indicate the probability of random test to find the fault.

For Example: *F(x) =x mod b*

Above is a representative example to illustrate DRR definition and is often used to show the testability variation when b varies. Consider two concrete functions:

$F_1(x)$ *=x mod 2* and $F_2(x)$ *=x mod 5,* both on the same domain D.

So $|D_1|=|D_2|$ and $|R_1|=2$, $|R_2|=5$

Then get $DRR_1>DRR_2$. According to [3] F_2 is more testable than F_1.

Assume that these two functions are both miswrite as:

Open Science Index, Computer and Information Engineering Vol:8, No:8, 2014 publications.waset.org/9999083.pdf

*h(x) =x mod 7*then

$$SFSZ_1 = SFSZ_1(H(x)to\ F_1(x)) = \frac{5}{7} + \frac{2}{7}\left(1 - \frac{1}{2}\right) = \frac{6}{7}$$

$$SFSZ_2 = SFSZ_2(H(x)to\ F_2(x)) = \frac{2}{7} + \frac{5}{7}\left(1 - \frac{1}{5}\right) = \frac{6}{7}$$

So get: $SFSZ_1=SFSZ_2$

It means that both $F_1(x)$ and$F_2(x)$ have the same probability to find the fault by random testing when r>b.

Next assume:

$F_3(x)$ *=x mod 5*, $F_4(x)$ *=x mod 7* and *H(x) =x mod 2*

Follow above procedure, we can get:

$$SFSZ_3 = SFSZ_3(H(x)to\ F_3(x)) = \frac{3}{5} + \frac{2}{5}\left(1 - \frac{1}{2}\right) = \frac{4}{5}$$

$$SFSZ_4 = SFSZ_4(H(x)to\ F_4(x)) = \frac{5}{7} + \frac{2}{7}\left(1 - \frac{1}{2}\right) = \frac{6}{7}$$

So get: $SFSZ_3<SFSZ_4$

It means when the fault value is littler than the right one, the bigger b the bigger probability to find it.

In general, assume *F(x) =x mod b* to *H (x) =x mod r* (r ≠b),

Proof:

Case 1: for *r>b*, and to make the situation simple, make the assumption (*r*, *b*) =1

The semantic fault size is

$$SFSZ_{BT} = \frac{r-b}{r} + \frac{b}{r} \times \left(1 - \frac{1}{b}\right) = \frac{r-1}{r}$$

So the semantic fault size is independent of the parameter b, while depends on r (what kind of error has been made).

Case 2: for *r<b* and (*r*,*b*) =1

$$SFSZ_{LT} = \frac{b-r}{b} + \frac{r}{b} \times \left(1 - \frac{1}{r}\right) = \frac{b-1}{b}$$

Assume |D|=n, then the detection probability of the fault by random testing is:

$$P = \frac{b-1}{n} \times \frac{b-1}{b} + \frac{n-b}{n} \times \frac{r-1}{r}$$

$$\because b < r \Rightarrow \frac{b-1}{b} < \frac{r-1}{r}$$

$$\therefore \frac{n-1}{n} \times \frac{b-1}{b} < \frac{b-1}{b} < P < \frac{n-1}{n} \times \frac{r-1}{r} < \frac{r-1}{r}$$

This means the bigger *b*, the lower bound increase, and so the greater probability the fault to be find by the random testing, which means high testability.

B. The Interpretation of Domain Testability

Observable extension adds new input parameters so it expands the domain of the function. Controllable extension is possible is because the domain of values of the evaluation map for expressions is a subset of the target type. "It is depends on the richness of type-domain definitions" [2]. For f(x) =x mod 5, the domain of f is N. While the range of the function is [0..4], there are no program data type that just coincide the function range, and then define the return result type as integer, this makes it uncontrollable. So we extend it to be controllable by custom a data type as *5_Type*= [0,1,2,3,4] as a subset of integer and $C_t=log_2 5=2.32$. For *f(x) =x mod 501*, its controllable extension must custom data type *501_Type*= [0…501] and $Ct=log_2 501=8.97$. Controllable extension doesn't change the range of the function but just change the range of the implementation data type to make the output range more obvious by explicit bound. The bigger *b* is and the bigger controllable extension needed. This results in low domain testability.

For certain observable extensions, it can find the controllability tendency of *f(x) =x mod b*, $C_t=log_2 b$, decreases as b increases (Because domain controllable is 0).When *b* achieves ∞, no extensions is needed, and its $C_t$is 0. This is a sharply distinct from the tendency.

C. The Discussion

The kernel of testability is observability and controllability. Observability is more important than controllability in terms of finding the latent fault. Both domain testability and DRR view *onto* function as the most testable function and try to make the non-priori *onto* function to or near to 'onto'. Voas listed two main differences between Freedman's approach and DRR as 1) assume observability in his description 2) domain testability bases on extensions that would be required to make the code observable and controllable, while DRR is particularly useful during design when extensions may be difficult to assess".

DRR denote the closeness of the program to onto-function. DRR indicates $SFSZ_{LT}$ became smaller when b decreases, while $SFSZ_{BT}$ is independent of b. This leads to the lower fault detection probability bound increase. So it is reasonable to say when *b* decreases the testability decrease.

Program functions and data types are two basic conceptual units in system design and construction and occur at all levels of abstraction in the description of a system or component. The variables and data structures in a program can be looked at in two ways. One way is as denoting objects of a certain type and the other way is as storage structures for holding objects of the appropriate type [16]. Domain testability takes the first view while [3] the second way as generally do. This makes the contradiction.

D. A New Testability Measure

For a faulty version of program *p*, that is p_f, which maps D_f into R_f, Can get:

$$Input\ semactic\ fault\ size = \frac{|D_f|}{|D|}$$

$$Output\ semactic\ fault\ size = \frac{|R_f|}{|R|}$$

$$DRR_p = \frac{|D|}{|R|} = \frac{|D|}{|D_f|} \times \frac{|D_f|}{|R_f|} \times \frac{|R_f|}{|R|}$$

That is:

$$DRR_p = DRR_f \times \frac{output\ semantic\ fault\ size}{input\ semantic\ fault\ size}$$

Open Science Index, Computer and Information Engineering Vol:8, No:8, 2014 publications.waset.org/9999083.pdf

The result seems wonderful. But carefully study can find now, D_f is not the domain of P_f, but just the fault exposing domain, so it is the domain of another program p_f'.

For clarity, use an example to show the true relation of P_f

If *g(x)* is a wrong version of *f(x)*, limit the domain to [0..100], then:

$$DRR_f = \frac{100}{2} = 50 \;\; DRR_g = \frac{100}{5} = 20$$

$$Input\ semantic\ fault\ size = \frac{4}{5}$$

$$Output\ semantic\ fault\ size = \frac{5}{2}$$

$$DRR_f = 50 \neq DRR_g \times \frac{output\ semantic\ fault\ size}{input\ semantic\ fault\ size} = \frac{125}{2}$$

This is because the P_f contain not only wrong mapping, but still some right mappings while D_f is just part of its domain. So it is another version faulty program that just maps D_f to R_f, but not the version of P_f.

To make the situation more clearly, we make the definition of semantic fault distance to indicate the distance of the faulty version to the correct version as:

$$semantic\ fault\ distance = \frac{output\ semantic\ fault\ size}{1 - input\ semantic\ fault\ size}$$

The tendency of the function is like Fig. 1. The bigger the output semantic fault size and the bigger the input semantic fault size, the big the semantic distance of the faulty program to the correct program, and easier to detect it. Particularly input fault size is near 1 means all the inputs are mapped to wrong output and can be found by any test case from the domain. This also corresponds with our assertion that observability is the more important part of testability.

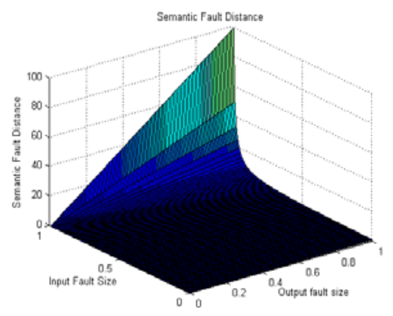

Fig. 1 Semantic Fault Distance

V. Conclusion

In this paper we focus on interpreting the contradiction between DRR and domain testability, both are specification-based testability measures. The contribution of this paper is to use the semantic fault size to calculate the probability of the fault to be found by random test. This is deeper than formula represented in [6] and make DRR measure has more semantic relations to the testability. This kind of analysis can be used to study how the testability of VDVR software varies. Secondly we make it clear that domain testability refers to the data type and its controllable extensions is depended the richness of type-domain definitions, while DRR refers to the domain and range ratio of the function. It may still have type problem in certain implementation language. We propose semantic fault distance to indict the easiness to detect the fault in the software. Compared to semantic fault size, it combined the effect of input and output semantic fault size, and making the result more clearly. This definition extends the restriction of DRR metric only work on mathematical-type problems and can get more wide application.

Software testability is becoming more and more intriguing, and accurate measure is becoming more and more necessary and important. Future work includes clearly and practically defining software testability and proving the validity of semantic fault distance. We hope to engage the testability in the evaluation of software testing.

References

[1] H. Fujiwara, "Logic Testing and Design for Testability," London England: The MIT Press, 1985.

[2] R.S. Freedman "Testability of software components," *IEEE Trans. Soft. Eng.*, vol. 17, June. 1991, pp. 553–564.

[3] J.M.Voas, and K.W. Miller, "A design Phase Semantic Metric for Software Testability", *The Journal of System and Software, Vol.*.20,March 1993,pp:207-216.

[4] A.J.Offutt, and J.H.Hayers, "A semantic model of program faults", in *Proc.Int.Symp. On Software Testing and Analysis (ISSTA'96)*, San Diego, 1996, pp.195-200.

[5] J.M.Voas. "Factors That Affect Program Testability". In *Proc. 9th Pacific Northwest Software Quality Conf.*, Portland, 1991, pp.235-247.

[6] M.R.Woodward, Z.A.AI-Khanjari, "Testability, Fault Size and the Domain-to-Range Ratio: An Eternal Triangle", in *Proc.Int.Symp. On Software Testing and Analysis (ISSTA'00)*, Portland, Oregon.pp.168-172.

[7] T.B.Nguyen, M.Delaunay, C.Robach, "Testability Analysis For Software Components", in *Proc.Int. Con. on Software Maintenance*. 2002, pp.422-429.

[8] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 610.12-1990, IEEE Press, New York,1990.

[9] ISO/IEC 9126:Software Engineering-Product quality.

[10] B.B,Traon,Y.Le,Sunye G. "Testability analysis of UML class diagram", In *Proc. 8th IEEE Symp. on Software Metrics*, 2002. pp.54-63.

[11] R.Bache and M.Mullerburg. "Measures of testability as a basis for quality assurance", *Software Engineering Journal*, March,1990.pp.86-92.

[12] Pu-Lin Yeh and Jin-Cheng Lin. "Software Testability Measurements Derived from Data Flow Analysis", In *Proc. of the CSMR'98*, Florence, Italy, March 8-11, 1998, pp.96-102.

[13] J.M.Voas, "PIE:A Dynamic Failure-Based Technique", *IEEE Trans. Software Eng.*,Vol.18, August,1992, pp.717-727.

[14] Jin-Cherng Lin, Szu-Wen Lin &Ian-Ho. "An estimated method for software testability measurement", In *Proc. 8th International WorkShopOn Software Technology and Engineering Practices*1997.pp.

[15] Bruce W.N.Lo and Haifeng Shi, "A preliminary Testability Model for Object-Oriented Software", In *Proc. Int. Con. on Software Engineering: Education & Practicem*1998,pp.330-337.

[16] W E.Howden, "Functional Program Testing and Analysis", McGraw-Hill Book Company, ISBN 0-07-030550-1.

[17] Ed Adams, Sam Guckenheimer. "Achieving quality by design-part II:UsingUML."White paper by Rational. (http://www.rational.com/).