Software Defect Regions Identification Based on Behavior Mining

Xi Guo[1, a], Pan Wang[2], Yayun Wang[3]

[1] Department of Computer Science, College of Informatics, Huazhong Agriculture University, Wuhan, China

[2] Wuhan Electric Power Technical College, Wuhan, China

[3] College of Public Administration, Huazhong Agriculture University, Wuhan, China

[a]seyeyesx@163.com

Keywords: Software Testing; Software Defect; Software Analysis; Behavior Mining

Abstract. Random testing is a widely used testing technique, and adaptive random testing has enhanced the performance of random testing recently. The research results demonstrate that the improvement depends on the characters of the software defect region. A method that depicts the distribution of software defect region in the input region based on testing is proposed in this paper, and the method that describes the characteristics of the software defect region. The experimental results show that testing constraint reveals the mechanism of software defect trigging and propagation.

Introduction

Reliability is very important to the software system, thus researchers use various testing methods to find the defects in the software, that is observing whether the functions of the software is defect according to a certain input values[1]. However, the current software usually has an enormous scale and complexity, it is impossible to test all the effect inputs. Random testing a widely used black-box testing method, but is has a relatively low efficiency. In order to improve the defects of random testing, Chen proposed the adaptive random testing method[2], which improves the selection approach of test suites according to a better uniform distribution of test suites in the input regions. Theory search also demonstrates that adaptive random testing is an economic technique to save the testing costs[3].

Input constrain is a set of constrains to which the input variables must apply, and it is connect to the input regions[4]. Input constrain also can help the researches to analyze to distribution characteristics of the software defect region relate to the defects.

Input Constraint

During the software development stage, the design defects will evolve to the coding defects, that is to say, when the defect code is executed, the software will defect[5]. In order to find the underlying defects, we must select the test suites belong to the software defect region.

Definition 1. Input Defect Region(IFR). Let P is the program, and its input region is I, the defect d is imported to P, and another program P' is obtained, and the input defect region IFR is defined as follows:

$$\{i | i \in I \wedge P(i) \neq P'(i)\}$$

Definition 2. Input Constraint(IC). Let P is the program, and its input region is I, the defect d is imported to P, and another program P' is obtained, and the Input Constraint relate to d is defined as follows:

$$IC: I \rightarrow IFR$$

The original program P is a triple$\langle I, Stmt, St\rangle$, where I is a set of input, $Stmt$ is a set of statements, and St is a set of states. I can be shown as $\langle I_1, I_2, \ldots, I_n\rangle$, and a state of P can be illustrated as $st_i: \langle i_1, i_2, \ldots, i_n\rangle$, where $i_1 \in I_1, i_2 \in I_2, \ldots, i_n \in I_n$. A statement of P, that is $stmt_i$ where $stmt_i \in Stmt$, is a state transition:$st_1 \xrightarrow{stmt_i} st_2$, where $st_1, st_2 \in St$.

Definition 3. Defect Constraint(FC). The transformed program *P'* is a transition of original program *P*. The transform method *tm* is used in the statement *st*, that is: $st_p \neq st_{tm}$, where $st_p \in ST_p \wedge st_{tm} \in ST_{tm}$, and ST_p, ST_{tm} is the set of *P* and *tm*. Defect constraint relates to constraint *C* is defined as follows:

$$C: I_p \rightarrow \{i | i \in I_p \wedge i \rightarrow Stmt\}$$

Let s_{final} is the final state of program *P*, and the defect program is *P'*, the statement *st* has the feature:$st_p \neq st_{p'}$, where $st_p \in ST_p \wedge st_{p'} \in ST_{p'}$, and ST_p, $ST_{p'}$ is the statement set of *P* and *P'*. The defect propagation is defined as follows:

$$I_p \rightarrow \{i | i \in I_p \wedge i \rightarrow st_i \wedge i \rightarrow st_p\}$$

The computation rules is: as for the original program *P*, and its input regain is *I*, the defect program *P'* is related to the defect *d*, and the Input Constraint(*IC*) is the intersection of the Defect Constraint *FC*(*d*) and Defect Propagation *FP*(*d*).

Defect Region Characteristic

The characteristic of software defect is the key factor that affects the adaptive random testing. The Input Constraint (*IC*) of defect *d* is a set of constraint conditions over input variable[7]. The common operator are disjunction, conjunction and nor, thus an Input Constraint (*IC*) can be transformed into its equal disjunction normal form *IC'*.

The defect ration *r* can be defined as $r = \|I'\|/\|I\|$, where $\|I'\|$ is the number of test cases in the defect region, and $\|I\|$ is the number of test cases in the input region *I*. A software has a higher defect ration, which means the *IC*(*d*) has a larger number of test cases.

Software usually has multiple input variables, thus the software defect ration can be computed as follows. Let the Input Constraint (*IC*) has a defect *d*, and $IC(d) = IC_1 \vee IC_2 \vee \ldots \vee IC_n$. $\|I\| = \prod_{i=1}^{n} |i_k|$, where $|i_k|$ is the region of i_k, where $|i_k| = upper(i_k) - lower(i_k)$ when *i* is a real number, and $|i_k| = upper(i_k) - lower(i_k) + 1$ when *i* is a integer.

The software defect region FR is computed as follows:

$$\|FR\| = \prod \prod |i_k|$$

Let *P* the program, and Input Constraint(IC) of defect *d* is related to the software defect region *fr*, and the test inputs are distributed in the input region of *P*. When a program has two input variables, the defect region has a certain shape, such as point or stripe[8].

Let *P* the program, and its input variable is *i*, the Input Constraint(IC) of defect *d* is *IC*(*d*), and $IC(d)=IC_1 \vee IC_2 \vee \ldots \vee IC_n$. As for the given IC_i, I_D is the region of IC_i. We also can reinforce the checking condition of constraint, that is the input constraint of defect *d* is *IC*(*d*), and *IC*(*d*) is a normal form, IC_i is an element of *IC*(*d*), the software defect region *fr* is a constrain if and on if all the IC_i has a continuous constrain region.

Behavior Distance

Training set $X = \{x_1, x_2, .., x_{|F|}\}$ is the point set in the multiple space, and *F* is the set of characteristic properties. The properties are either discrete or continuous[9]. The classification property is *l*, which is a discrete variable with range of *L*. The target of classification is minimize the misclassification error M_i, that is for each $l_i \in L$:

$$M_i = \sum_{l_i \in L} R_{l_i} p(l_i | q)$$

Where R_{l_i} is the error value of l_i, and *q* is the future position. $p(l_i|q)$ is the probability of *q* for l_i. Behavior distance supposes all the misclassification has the same error limit, that is:

$$R_{l_i} = \begin{cases} 0, i = given\ value \\ 1, i \neq given\ vale \end{cases}$$

However, the behavior distance approach can not anticipate the classification property value of *q* precisely, but give the most probable anticipate value:

$$behavior\ distance = p(l_i|q)$$

The method of behavior distance computation is different from the other classification methods, especially in the definition of prior probability.

Where S_q is the behavior distance of q in the training data set according to the distance function $d(x,q)$, and C is a core function, which is defined as followed:

$$C\big(d(x,q)\big) = \frac{1}{d(x,q)}$$

The prior probability can be defined as followed:

$$p(l_i|q) = \frac{T}{\sum_{x \in S_q} C(d(x,q))}$$

The main steps of behavior distance are usually as follows: firstly, a proper distance mechanism should be selected, and for each data point *dp* in test suite, the number of nearest points to *dp* is *k*, and anticipate the classification property of target point according to the nearest points. When the anticipation is finished, and then computes the classification error.

In practical, the value of property is probably from certain ontology. As for discrete property *f*, if two properties has different values, α equals to 1. But it cannot reveal the real fact, especially when two properties have a close connection in the behavior.

Ontology O is a five tuples, $O := (V, T, F, H, R)$, where V is a set of vocabulary, T is a set of transaction, and F is a reference function: $2^V \mapsto 2^T$, which maps a vocabulary $V_i \subset V$ to a transaction. Many vocabularies can be mapped to a transaction, and one vocabulary can be mapped to many transactions. H is a hierarchy: $H \subseteq T \times T$, $H(t_1, t_2)$ means that t_1 is a sub transaction of t_2. R is the root of transaction. For each $t \in T$, $H(t, R)$ is satisfied.

Let $p(t)$ be the occur possibility of transaction t in the whole transaction set. Let *number*(t) be the number of occurrence of transaction t in the date set D, and *number*(D) is the number of date set D. Transaction usually has several sub-transactions, so during the computation of transaction, the number of sub-transactions must be added.

$$p(t) = \frac{number(t) + \sum_{H(t',t)} number(c')}{number(D)}$$

It is easy to find that $p(t)$ is monotonically increase, and $p(R) = 1$. Let parent(t) be the ancestor set of t, that is:

$$parent(t) = \{t' | H(t,t'), \neg\exists t'', s.t. H(t,t''), H(t'',t)\}$$

Because there is no circle in H, it is obvious that:

$$\forall t \in T, \text{if } parent(t) \neq \emptyset, then\ |parent(c)| = 1$$

Where $|X|$ is the number of elements in set X. The equation means that if t has a ancestor, its ancestor is unique. The information content of transaction t is:

$$I(t) = -\log(p(t))$$

Where k is a constant. Let t_1 and t_2 are two values of property f. As $\forall t \in T$, $H(t, R)$ is satisfied, thus t_1 and t_2 have the same ancestor. Let $M(t_1, t_2)$ be the most minimal ancestor, that is:

$$M(t_1,t_2) = \{t | H(t_1,t), H(t_2,t), \neg\exists t', s.t. H(t_1,t'), H(t_2,t'), H(t',t)\}$$

If t_1 and t_2 satisfy $H(t_1, t_2)$, the link between t_1 and t_2 is $l(t_1, t_2)$, which can be defined as follows:

$$l(t_1,t_2) = \{t_1^i | t_1 = t_1^0, t_2 = t_1^i, i \geq 1, \forall k, 0 \leq k \leq i-1, t_1^{k+1} = parent(t_1^k)\}$$

Because $\forall t \in T, H(t,R)$ and H has no circle, there is only one path between t_1 and t_2 in ontology O, which is recorded as *path*(t_1, t_2), the union set of the link from t_1 to $M(t_1, t_2)$ and the link from t_2 to $M(t_1, t_2)$ is its path, and the definition is:

$$path(t_1,t_2) = l(t_1, M(t_1,t_2)) \bigcup l(t_2, M(t_1,t_2))$$

The definition of behavior distance include two aspects of meaning, the first is the construction of ontology determines the position of $M(t_1, t_2)$, the second is that the value of $p(M(t_1, t_2), p(t_1), p(t_1))$ is from the statistics of the date set. One property in different ontology has different behavior distance, while as for the same ontology; the behavior distance of different data set is also different. Our method integrates the behavior relation and statistic, which can help the researchers simulate the transaction precisely.

Experimental Results

The experimental target is a program that contains 10 modules and 151 lines of executable codes. This program is widely used in the software testing, and researchers can interpolate various defects, that is change several lines of codes.

Because the disjunction normal form of Input Constraint(IC) of α is $T(\alpha)$, which has 5 elements, thus the software defect region φ is multiple. The Boolean element based on $T(\alpha)$, $T_i(i = 1,2,\dots,5)$, and their defect region is $\varphi_i(j = 2,3,\dots,5)$. Table 1 is the results of software defect region of defect α, and the ratio can be computed via: $\|\varphi_i\|/\sum_{i=1}^{5}\varphi_i$.

TABLE I. SOFTWARE DEFECT REGION ANALYSIS

Defect Ratio	Software Defect Ration	
	Defect Ratio	*Percentage*
φ_1	3.1×10^{-5}	9.1%
φ_2	2.3×10^{-5}	12.0%
φ_3	4.7×10^{-5}	7.3%
φ_4	1.9×10^{-5}	6.2%
φ_5	2.7×10^{-5}	8.5%

Acknowledgement

This project is supported by the Fundamental Research Funds for the Central Universities under grant No. 2662015QC009 and the Hubei Province Natural Science Foundation under Grant No. 2014CFB144.

References

[1] Hamlet R. Random testing. In: Marciniak J, ed. Proc. of the Encyclopedia of Software Engineering. 2nd ed., New York: John Wiley and Sons, 2002.

[2] Chen TY, Kuo F C, Merkel R, Tse TH. Adaptive random testing: The ART of test case diversity. Journal of Systems and Software, 2010,83(1):60−66.

[3] Chen TY, Leung H, Mak IK. Adaptive random testing. In: Maher MJ, ed. Proc. of the 9th Asian Computing Science Conf. LNCS,3321, Heidelburg: Springer-Verlag, 2004. 320−329.

[4] Sun CA. A constraint-based approach to identifying and analyzing defect-causing regions. Journal of Software, 2012,23(7):1688− 1701

[5] Chen TY, Kuo FC, Merkel RG, Ng SP. Mirror adaptive random testing. Information and Software Technology, 2004,46(15):1001−1010.

[6] Mayer J. Lattice-Based adaptive random testing. In: Ireland A, ed. Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software. Engineering (ASE 2005). New York: ACM Press, 2005.333 −336.

[7] Andrews JH, Briand LC, Labiche Y. Is mutation an appropriate tool for testing experiments? In: Roman GC, Griswold WG, Nuseibeh B. eds. Proc. of the 27th Int'l Conf. on Software Engineering (ICSE2005). Los Alamitos: IEEE Computer Society Press, 2005. 402−411

[8] YangL，ZuoC，WangYG．K-Nearest neighbor satisification based on behavior distance．JournalofSoftware，2005，16(12)：2054-2062

[9] Barzin R, Fukushima S, Howden W, Sharifi S. Superfit combinational elusive bug detection. In: Leppänen W, ed. Proc. of the 32nd Annual IEEE Int'l Computer Software and Applications Conf. (COMPSAC 2008). Washington: IEEE Computer Society, 2008.144−151.