47

A Model-based Approach for Software Test Process Improvement

Lin Lian[a], Fusayuki Fujita[b], Shinji Kusumoto[a], Ken-ichi Matsumoto[c], Tohru Kikuno[a] and Koji Torii[c]

[a]Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560, JAPAN

[b]Software Laboratory, Sharp Corporation, Nara, JAPAN

[c]Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-01, JAPAN

Abstract

Reducing the effort or cost of testing is a key issue to high productivity in software development. This paper proposes a test process model which aims to grasp the current status of test processes and to provide several metrics to evaluate efficiency of test processes in order to construct the process improvement plan. The model defines test process as an iterated process of four kinds of hypotheses development and verification which deal with: correctness of the program to be tested, location of failure, location of faults, and correction of the faults, respectively. The analysis results of experimental data collected from the test process show the validity of the proposed model and usefulness of the metrics.

Keyword Codes: D.2.5; D.2.8
Keywords: Software Engineering, Testing and Debugging; Software Engineering, Metrics.

1. INTRODUCTION

There are numerous research studies towards the improvement of software development process[1][2]. Basili and Rombach have proposed *Goal/Question/Metric paradigm* (simply called *G/Q/M paradigm*) in the *TAME* project[1]. The *G/Q/M paradigm* has a mechanism to describe explicitly the relation between the goal, to be achieved in the process, and the metric to be applied to the process. Humphrey has proposed SEI Software Process Maturity Self-Assessment[2]. Based on a *process maturity model*, SEI Self-Assessment builds a consensus view of an organization's maturity and the key issues facing it. Then ultimately it presents an improvement plan for software development process endorsed by general management.

The importance of the following key attributes (a)-(c) is commonly stressed in these proposals for the improvement of software development process: (a) software development process is strictly defined, (b) for each activity, quality data are collected and analyzed

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

statistically, and (c) based on the analysis result, improvement of each activity is executed.

This paper proposes a test process model which aims to grasp the current status of test processes and to provide several metrics to evaluate efficiency of test processes in order to construct the test process improvement plan. The key idea of the proposed model is to define test process as an iterated process of four kinds of hypotheses development and verification that deal with (1)correctness of the program to be tested, (2)location of failure, (3)location of faults (that is cause of the failure), and (4)correction of the faults, respectively. Then, we apply the proposed model and metrics to test process in an academic environment.

2. HYPOTHESIS-BASED DEBUGGING

Test process consists of two sub-processes[3]: testing and debugging. Testing is the process of analyzing a software item to detect the differences between existing and required conditions [1] and to evaluate the features of the software items. Debugging is the process to detect, locate, and correct faults [3].

There are two general approaches to debugging: induction and deduction [4]. The inductive approach first formulates a single working hypothesis and then proves or disproves it based on data and the analysis of the data; while the deductive approach begins by enumerating all possible causes or hypotheses and then rules out particular causes one by one until a single one remains for validation.

Araki et al. proposed debugging process model based on deductive debugging approach [5]. They consider debugging process as the process of locating and correcting errors in a program in which errors[2] have been detected. In locating the errors and grasping their causes, programmers develop hypotheses about the errors and their causes, and verify or refute these hypotheses by examining the program. In correcting the errors, programmers again develop hypotheses about how to modify the program and verify or refute them.

3. TEST PROCESS MODEL

3.1. Hypothesis-based approach

We propose a hypothesis-based test process model as shown in Figure 1. In this model, we assume that test data set to be used is given, and test process is completed when we obtain the program finally, which executes what the specification of the program requires for the test data set, in inductive approach.

The model consists of two subprocesses; *testing* and *debugging*, and both of them are further divided into two primitive activities: *Hypothesis Development* and *Hypothesis Verification.*

The Hypothesis Development is an activity to formulate a single working hypothesis by

[1] In this paper, we follow the IEEE standard [3] with respect to the definition of error and fault. In the IEEE standard, an error is defined as a human action that results in software which contains a fault. Examples include omission or misinterpretation of user requirements in software specification and incorrect translation or omission of a requirement in the design specification. A fault is defined as a manifestation of an error in software. A fault, if encountered, may cause a failure (synonymous with bug). Thus "the difference between existing and required conditions" correspond to failure.

[2] "Errors" in [5] corresponds to "faults" in the IEEE Standard.

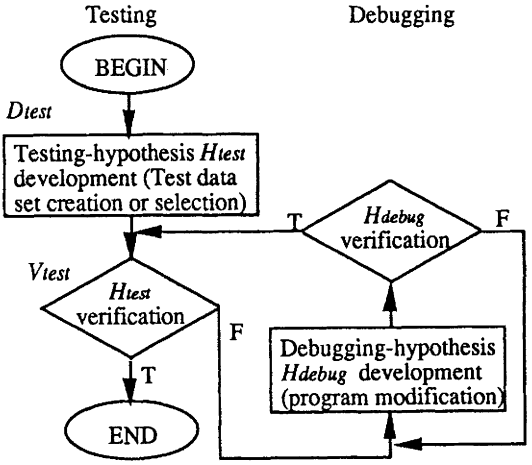

Figure 1: Test Process Model

instantiating *Hypothesis Templates*, i.e., by substituting the parameters of the *Hypothesis Templates* by data. The Hypothesis Verification is an activity to prove or disprove the working hypothesis. In the following part of this section, we shall describe Hypothesis Development and Verification in the test process in detail. Before giving the definition of the test process, we introduce the following notations:

P: A program to be tested.

P_0: An ideal program that executes correctly required specification.

T: A set of given test data.

$t_j \in T$: It is defined to be a sequence of pairs space (input,output) of P_0: $(i_{j_1}, o_{j_1})(i_{j_2}, o_{j_2}) \ldots (i_{j_{n(j)}}, o_{j_{n(j)}})$, where i_{j_τ} is the τth input of t_j, o_{j_τ} is the τth output corresponding to i_{j_τ}, and $n(j)$ is the number of inputs of t_j.

t_j^P: A sequence of pairs space (input,output) of P. It can be also denoted like t_j: $(i_{j_1}, o_{j_1}^P)(i_{j_2}, o_{j_2}^P) \ldots (i_{j_{n(j)}}, o_{j_{n(j)}}^P)$

3.2. Testing subprocess

Testing is a process that checks if program P executes correctly required specification or not for each element of T.

Definition 1: Testing subprocess is verification of hypothesis:

H_{test}: $(\forall t_j \in T(1 \le j \le |T|))\ t_j^P = t_j$

where

$t_j^P = t_j$ iff $(\forall l,\ 1 \le l \le n(j))\ o_{j_l}^P = o_{j_l}$ (See Figure 2).

Testing-hypothesis Template:

$HT_{test}(\rho)$: $(\forall t_j \in T(1 \le j \le |T|))\ t_j^\rho = t_j$

where ρ is a parameter which is a program to be tested. And a working hypothesis for the

testing is defined as $HT_{test}(P)$ where P is a program to be tested. We denote $HT_{test}(P)$ development by D_{test} and $HT_{test}(P)$ verification by V_{test}, respectively (See Figure 1).

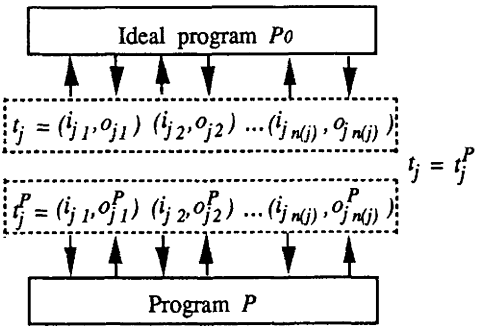

Figure 2: Testing-hypothesis H_{test}

3.3. Debugging subprocess

Debugging process is executed if the working hypothesis $HT_{test}(P)$ is refuted. Debugging is a process that creates a program which could execute correctly required specification for the given test data set T. The program, denoted by P' in this model, is constructed by modifying P.

Definition 2: Debugging subprocess is verification of hypothesis:

$H_{debug} : (\exists P')t_\tau^{P'} = t_\tau$.

In general, in order to construct P', several activities have to be done. In this model, debugging is further divided into the following three subprocesses: (1) Failure location, (2) Fault location, and (3) Fault correction. Each of them is defined as an iterated process of hypothesis development and verification, and has a hypothesis template, respectively (See Figure 3).

(1) Failure location

The failure location is a process that clarifies a statement of the program P which provides incorrect output (or incorrect state of P, e.g., hang up, infinite loop) in the testing.

Definition 3: Failure location is verification of hypothesis:

$H_{fll} : \exists\tau(1 \leq \tau \leq |T|)\exists o_1(1 \leq o_1 \leq n(\tau))\exists\sigma_1(1 \leq \sigma_1 \leq \tau_N)(output(s_{\tau_{\sigma_1}}^P) = o_{\tau_{o_1}}^P, o_{\tau_{o_1}}^P \neq o_{\tau_{o_1}})$.

where $s_{\tau_{\sigma_1}}^P$ is σ_1-th statement of P executed for t_τ. $output(s_{\tau_{\sigma_1}}^P)$ is an output of the statement $s_{\tau_{\sigma_1}}^P$ for t_τ (See Figure 4). The value of $output(s_{\tau_{\sigma_1}}^P)$ is $NULL$ when $s_{\tau_{\sigma_1}}^P$ is not an output statement.

Failure-location-hypothesis Template:

$HT_{fll}(\tau, o_1, \sigma_1) : output(s_{\tau_{\sigma_1}}^P) = o_{\tau_{o_1}}^P, o_{\tau_{o_1}}^P \neq o_{\tau_{o_1}}$.

where τ, o_1, σ_1 are parameters. τ is a test data number, o_1 is an output number, and σ_1 is a statement execution number of P, respectively. We denote $HT_{fll}(\tau, o_1, \sigma_1)$ development by D_{fll} and $HT_{fll}(\tau, o_1, \sigma_1)$ verification by V_{fll} (See Figure 3).

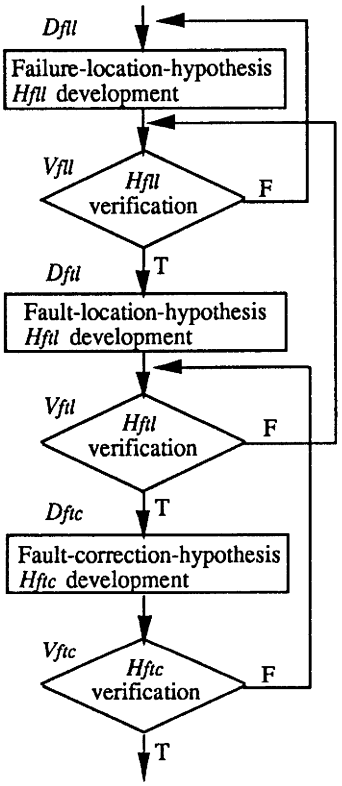

Figure 3: Debugging Process Model

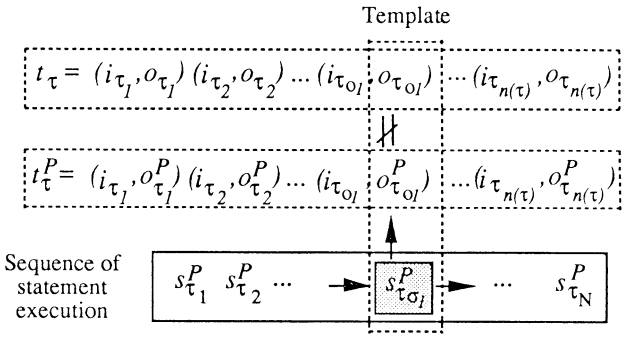

Figure 4: Failure-location-hypothesis H_{fll}

(2) Fault location

A fault is the cause of the failure. And fault location is a process that clarifies statement of the program P which causes incorrect state (an ordered set of internal and/or external variable values) of P during the testing.

Definition 4: Fault location is verification of hypothesis:

H_{fll} : $(\exists o_2(1 \leq o_2 \leq o_1), \exists \sigma_2(1 \leq \sigma_2 \leq M))$ $E^P_{\tau_{\sigma'_1-1}} = E_{\tau_{\sigma_2-1}}(1 \leq \sigma'_1 \leq \sigma_1), E^P_{\tau_{\sigma'_1}} \neq E_{\tau_{\sigma_2}}$, $output(s^P_{\tau_{\sigma'_1}}) = o^P_{\tau_{o_2}}(1 \leq o_2 \leq o_1), output(s_{\tau_{\sigma_2}}) = o_{\tau_{o_2}}, HT_{fll}(\tau, o_1, \sigma_1)$.

where $s_{\tau_{\sigma_2}}$ is σ_2-th statement of P^0 executed for t_τ. And $E^P_{\tau_{\sigma'_1}}$ and $E_{\tau_{\sigma_2}}$ indicate the state (an ordered set of internal and/or external variable values) of P and P^0 after executing $s^P_{\tau_{\sigma'_1}}$ and $s_{\tau_{\sigma_2}}$ for t_τ, respectively (See Figure 5).

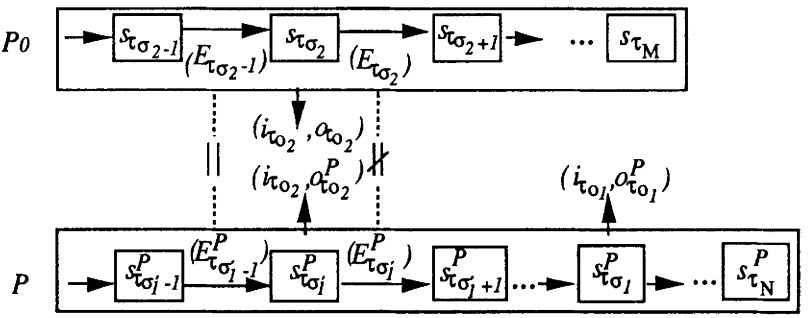

Figure 5: Fault-location-hypothesis H_{fll}

Fault-location-hypothesis Template:

$HT_{ftl}(\tau, o_1, \sigma_1, o_2, \sigma_2)$: $E^P_{\tau_{\sigma'_1-1}} = E_{\tau_{\sigma_2-1}}(1 < \sigma'_1 \leq \sigma_1), E^P_{\tau_{\sigma'_1}} \neq E_{\tau_{\sigma_2}}, output(s^P_{\tau_{\sigma'_1}}) = o^P_{\tau_{\sigma_2}}(1 < o_2 \leq o_1), output(s_{\tau_{\sigma_2}}) = o_{\tau_{o_2}}, HT_{fll}(\tau, o_1, \sigma_1)$.

where $\tau, o_1, \sigma_1, o_2, \sigma_2$ are parameters. τ, o_1, σ_1 are the same as $HT_{fll}(\tau, o_1, \sigma_1)$, respectively. o_2 is an output number, and σ_2 is a statement execution number of P^0, respectively. We denote $HT_{fll}(\tau, o_1, \sigma_1, o_2, \sigma_2)$ development by D_{ftl} and $HT_{ftl}(\tau, o_1, \sigma_1, o_2, \sigma_2)$ verification by V_{ftl}, respectively (See Figure 3).

(3) Fault correction

Fault correction is a process that constructs program P' by replacing the faulty part of program P with a complete one.

Definition 5: Fault correction is verification of hypothesis:

H_{ftc} : $(\exists P')\ t^{P'}_\tau = t_\tau, output(s^{P'}_{\tau_{\sigma_3}}) = o_{\tau_{o_1}}(1 \leq \sigma_3 \leq L), E^{P'}_{\tau_{\sigma'_3-1}} = E_{\tau_{\sigma_2-1}}(1 \leq \sigma'_3 \leq \sigma_3), output(s^{P'}_{\tau_{\sigma'_3}}) = o_{\tau_{o_2}}, E^{P'}_{\tau_{\sigma'_3}} = E_{\tau_{\sigma_2}}, HT_{ftl}(\tau, o_1, \sigma_1, o_2, \sigma_2)$.

where $s^{P'}_{\tau_{\sigma_3}}$ is σ_3-th statement of P' executed for t_τ.

Fault-correction-hypothesis Template:

$HT_{ftc}(\tau, o_1, \sigma_1, o_2, \sigma_2, P')$: $t^{P'}_\tau = t_\tau, output(s^{P'}_{\tau_{\sigma_3}}) = o_{\tau_{o_1}}(1 \leq \sigma_3 \leq L), E^{P'}_{\tau_{\sigma'_3-1}} = E_{\tau_{\sigma_2-1}}(1 \leq \sigma'_3 \leq \sigma_3), output(s^{P'}_{\tau_{\sigma'_3}}) = o_{\tau_{o_2}}, E^{P'}_{\tau_{\sigma'_3}} = E_{\tau_{\sigma_2}}, HT_{ftl}(\tau, o_1, \sigma_1, o_2, \sigma_2)$.

where $\tau, o_1, \sigma_1, o_2, \sigma_2, P'$ are parameters. $\tau, o_1, \sigma_1, o_2, \sigma_2$ are the same parameters of $HT_{ftl}(\tau, o_1, \sigma_1, o_2, \sigma_2)$, respectively. P' is a program. We denote $HT_{ftc}(\tau, o_1, \sigma_1, o_2, \sigma_2, P')$ development by D_{ftc} and $HT_{ftc}(\tau, o_1, \sigma_1, o_2, \sigma_2, P')$ verification by V_{ftc} (See Figure 3).

4. METRICS FOR EFFICIENCY OF TEST PROCESS

In this section, we provide several metrics to evaluate efficiency of test processes and then to construct the improvement plan for the processes based on them. The computations of those metrics are based on the following test process parameters.

$\Gamma = D_{test}V_{test}D_{fll}\ldots V_{test}$: Sequence of operations till termination of the test process.

n_α: Number of operation α in Γ, $num(\delta, \Gamma)$: Number of δ in Γ.

The numbers of repetitions of H_{fll}, H_{ftl}, H_{ftc} for correcting a single fault, denoted by HE_{fll}, HE_{ftl} and HE_{ftc} respectively, are computed as follows:

Failure location error rate: $HE_{fll} = \frac{num(V_{fll}D_{fll},\Gamma)}{n_{V_{test}}-1}$

Fault location error rate: $HE_{ftl} = \frac{num(V_{ftl}V_{fll}D_{ftl},\Gamma)}{n_{V_{test}}-1}$

Fault correction error rate: $HE_{ftc} = \frac{num(V_{ftc}V_{ftl}D_{ftc},\Gamma)}{n_{V_{test}}-1}$

We consider that the nearer the values of HE_{fll}, HE_{ftl} and/or HE_{ftc} approximate 0, the higher the testing efficiency is.

Let VE_{fll} be the number of repetitions of V_{fll} and VE_{ftl} the number of repetitions of V_{ftl} respectively for correcting a single fault, they are calculated as follows:

Verification error rate for failure location $VE_{fll} = \frac{num(V_{ftl}V_{fll}D_{fll},\Gamma)}{n_{V_{test}}-1}$

Verification error rate for fault location $VE_{ftl} = \frac{num(V_{ftc}V_{ftl}V_{fll}D_{ftl},\Gamma)}{n_{V_{test}}-1}$

5. DISCUSSIONS

In order to show the validity of the proposed model and usefulness of the metrics, we apply the model and the metrics to test processes in academic environment[9].

The main characteristics of the test processes in this experiment are: (1) Two graduate students (A and B) designed, implemented and tested a kind of data processing program based on a specification, written by natural language, by using C language. (2) One UNIX workstation was assigned to each of them. While student tested his program, we recorded image on the workstation's display by video camera and VCR, key stroke on the keyboard by *PAM* system [6].

By analyzing image on the display and key stroke on the keyboard, we could obtain a lot of data about the students' activities on testing. From such data, we could know which operation of the model was being performed by the students, and construct the sequence of operations during testing. Almost all parts of the sequence complied with the model. We may, therefore, reasonably conclude that the proposed model can describe the actual test process in student project. We could analyze their test process based on the values of the metrics. Based on results, we advised Student *B* to use tools for fault location, e.g., program slicing tool *SPYDER* [7], and error-cause-chasing tool *CHASE* [8]. Then, we advised Student *A* to use tools for fault location mentioned above, and to use some tools supporting fault correction.

REFERENCES

1. BASILI V. R. and ROMBACH H. D.: 'The TAME project: Towards improvement-oriented software environment', *IEEE Trans. Software Eng.*, 1988, Vol.14, (6), pp.758-773.
2. HUMPHREY W.S.: 'Characterizing the software process: A maturity framework', *IEEE Software*, 1988, Vol.5, (2), pp.73-79.
3. 'IEEE Standard Glossary of Software Engineering Terminology', *IEEE, ANSI/IEEE Std 610.12-1990*, 1990.
4. SHOOMAN M. L.: 'Software Engineering', (McGraw-Hill, 1983).
5. ARAKI K., FUKUZAWA Z. and CHENG J.: 'A general framework for debugging', *IEEE Software*, 1991, Vol.8, (3), pp.14-20.
6. TAKADA Y., MATSUMOTO K. and TORII K.: 'A programmer performance measure based on programmer state transitions in testing and debugging process', *Proc. of 16th International Conference on Software Engineering*, May 1994, pp.123-132.
7. AGRAWAL H., DEMILLO R.A. and SPAFFORD E.H.: 'Debugging with dynamic slicing and backtracking', *Software: Practice and Experience*, 1993, Vol.23, (6), pp.589-616.
8. SHIMOMURA T.: 'Bug localization based on error-cause-chasing methods', *Trans. of IPS*, 1993, Vol.34, (3), pp.489-500.
9. FUJITA F.: 'A software testing process model for evaluating capability of testing in software engineers education', *Master Thesis of Osaka University*, March 1994 (in Japanese).