Conference or Workshop Item:
Huang, Liang, Thomson, Christopher and Holcombe, Mike (2007) How good are your testers? An assessment of testing ability. In: Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, 12-14 September 2007, Windsor, UK.

How good are your testers? An assessment of testing ability

Liang Huang, Chris Thomson and Mike Holcombe
Department of Computer Science, University of Sheffield
{l.huang, c.thomson, m.holcombe}@dcs.shef.ac.uk

Abstract

During our previous research conducted in the Sheffield Software Engineering Observatory [11], we found that test first programmers spent a higher percentage of their time testing than those testing after coding. However as the team allocation was based on subjects' academic records and their preference, it was unclear if they were simply better testers. Thus this paper proposes two questionnaires to assess the testing ability of subjects, in order to reveal the factors that contribute to the previous findings. Preliminary results show that the testing ability of subjects, as measured by the survey, varies based on their professional skill level.

1. Introduction

Sheffield Software Engineering Observatory (SSEO) was set up to empirically study software development projects. In this environment, we conduct experiments with students working with managers and external industrial clients. Our most recent work has been to assess the effectiveness of test first programming in comparison to testing after coding [1] [2].

Test first programming is a software development practice which has been in use since the 1980's [8] [12]. In this method the automated tests are implemented before the object code is written, this is in contrast to the traditional approach where tests are constructed after development. Our ultimate aim is designed to validate Chaplin's test first rule: "If you can't write a test for what you are about to code, then you shouldn't even be thinking about coding" [4].

The results of our experiments so far however are unclear, suggesting that the testing ability of subjects could be an important co-variant in the relationship between testing method and performance. In order to distinguish the good and bad testers this paper describes two questionnaires which assess testing ability. In order to validate the first questionnaire we designed and ran an experiment based on the assumption that second year undergraduates are less skilled than fourth years.

The rest of this paper is organized as follows: Section 2 reviews the literature and motivates the study. Section 3 describes the data collection environment and method. Section 4 describes the process of data analysis and preliminary results. Lastly in section 5 we summaries our findings and suggest future work.

2. Background

2.1 Previous Studies

There have been a number of controlled experiments and case studies aiming to investigate the distinction between test first programming and the test last method [5] [9] [11] [13] [15] [16].

A comparative study [16] with 19 undergraduate students involved in an academic setting showed that test first method did not help programmers obtain higher productivity or program of higher reliability but subjects using test first understood the programs better. A slightly different result was obtained in a formal investigation [5] with 24 students who worked independently. They compared the effectiveness of test first approach with that of test last. It was found that that test first students wrote more tests but failed to deliver software of higher external quality. However the minimum external quality obtained increased with the number of tests. Moreover, students who wrote more tests were more productive regardless of testing strategy employed.

In an industrial setting at IBM, the pair programmers using test first obtained a 40 to 50 percent improvement of code quality, whereas their productivity did not decrease correspondingly [15]. Another structured experiment run by George and Williams [9], delivered an alternative view again. In this study 24 professional pair programmers were divided into two groups. It was observed that Test

0-7695-2984-4/07 $25.00 © 2007 IEEE
DOI 10.1109/TAIC.PART.2007.16

Driven Development (TDD) programmers were less productive but produced code of higher quality because 18 percent more black-box test cases were passed whereas 16 percent more time were consumed in TDD group.

In our early work in this area [13] our colleague compared Extreme Programming (XP) with a design-led traditional method with 96 students, who were divided into 2 groups working as 19 teams on 4 projects. He observed that that XP teams spent more time on testing than the teams using traditional method by statistical methods. This suggests that there was a difference in testing effort applied by the teams.

To further investigate this effect last year we ran a controlled experiment in SSEO with 39 students [11]. The only difference between the development approaches of the subject groups was test first and test last method. In this experiment the students were allocated in two treatment groups working as ten teams competitively on three different projects. Our main finding was that teams using test first method spent a larger percentage of time on testing but failed to obtain significantly higher software external quality, additionally a strong statistical correlation between the testing effort and coding effort was also observed.

2.2 Research Motivation

On comparison of previous studies, we located six published studies [5] [9] [11] [13] [15] [16] on test first programming with differing conclusions. However, the reason for this difference remains unknown. There are a number of factors that could have influenced these results for example: a large number of unnecessary tests or too few tests; differences in testing ability; or differences in test quality. Therefore, it is imperative to analyze the tests written by subjects and to assess the subjects' ability to test. In order to distinguish the good and bad testers, the controlled experiment previously conducted in SSEO [11] was replicated in the spring semester in the academic year 2006-07, and an additional survey was designed to assess the testing ability of subjects involved by assuming that the Genesys group (fourth year undergraduates and masters students) is better than the Software Hut group (second year undergraduates).

3. The Experiment

3.1 Subjects

55 students studying in Department of Computer Science at the University of Sheffield were involved in this study. Some of them registered for the Software Hut module and others for the Genesys module.

The Software Hut module consists of the level 2 undergraduate students. They were required to complete all the courses in level 1 and that in the first semester of level 2 before registering for the Software Hut module. The modules that are related to the Software Hut projects are "Introduction to Programming", "Requirements Engineering", "Object Oriented Programming", "System Design and Testing", "Functional Programming", "Systems Analysis and Design", and "Database Technology". In these modules, they have gained experience of developing software systems using traditional method and different programming languages such as Java.

The subjects involved in the "Genesys Solutions" [10] are the fourth-year MEng and advanced one year MSc students. They play the role of staff and run the software development company themselves. It is assumed that the students in this module have a higher level of professional skill and that they are more socially mature compared with the second year undergraduates. Students in this module are usually divided into several teams, two of which are responsible for marketing and company administration, while other teams are supposed to do the software development using XP. Lecturers in this module play the role of external managers rather than instructor.

3.2The Questionnaire

Two questionnaires were proposed to be designed for the survey with the name of Questionnaire A and Questionnaire B.

Questionnaire A is code based. It is composed of a short piece of Java code and 29 potential test cases to be selected, to assess the testing ability of subjects. The subjects were asked to make a selection from 29 potential test cases to correspond to the Category Partition method of testing [17] and to give Branch coverage [6]. Of all the test cases presented, 22 were required for the Category Partition method, 7 for Branch coverage and 7 for both. Questionnaire A and model answers are presented in Appendix A.

For Questionnaire A, the testing ability of every subject was measured by: the number of correct choices he/she made; the branch coverage obtained; and the number of redundant test cases that were selected.

Questionnaire B which has not yet been issued will be specification based. It will consist of the textual specification for a story [3] to be implemented and a number of potential test cases to be selected. The testing ability of subjects will be measured by 1) the

number of correct choices made, and 2) the number of redundant test cases that were selected.

3.3 Procedure

The subjects received intensive training before allocation to the different treatment groups: test first group and test last group, according to their preference. They were asked to work as teams which were composed of 3 to 6 members. They were required to upload their work including: the code; the tests; and documentation to the repository at least once a week. The academic staff reminded them to conform to the practices throughout the project and ultimately assessed their level of methodology conformance. Furthermore the students were encouraged by a potential reward of up to 50% of the marks being directly related to the methodology conformance.

The questionnaire A was distributed in the week before Easter vacation. From the Software Hut students we obtained 14 responses and 8 were obtained from the subjects working in the Genesys lab. The response rate was fairly low due to the complexity of the questionnaire; however it serves as a trial and will require further investigation. Questionnaire B will be issued at the end of the semester.

4. Preliminary Results

The statistical results in this section are based on the data collected via Questionnaire A. The subjects' ability to test is assessed using three measurements, as described in Section 2.2.

Table 1. Comparison of Marks for Category Partition Method

Subjects	Responses	Mean	Std. Deviation
Software Hut	14	16	5.3
The Genesys	8	18	5.0

As shown in table 1, the mean value of marks obtained by the Genesys students is higher than that obtained by the Software Hut students. However, significance of Mann-Whitney U-test is 0.27, higher than the 0.1 level (Since the sample size is small, we used 0.1 as the alpha value) [14].

According to the statistical results shown in Table 2 and Table 3, all the subjects, regardless of their backgrounds, obtained the same level of branch coverage. But the mean value of redundant tests selected by the Genesys students is lower than that selected by the Software Hut students with a smaller standard deviation. However, significance of Mann-Whitney U-test is 0.92, much higher than 0.1.

Table 2. Comparison of the branch coverage obtained

Subjects	Responses	Mean	Std. Deviation
Software Hut	14	26	4.1
The Genesys	8	26	3.6

Table 3. Comparison of redundant tests selected

Subjects	Responses	Mean	Std. Deviation
Software Hut	14	8.0	7.4
The Genesys	8	6.5	5.8

When we compared Software Hut students with the Genesys in terms of marks for Category Partition method, the result of Mann-Whitney U-test (0.27) is weak but close to the frontier (0.1). Since the sample size is small (22 responses only), and the statistical results exhibit a continuous difference between two groups of subjects, in terms of correct choices made, and redundant tests selected, we used Bayesian approach [7], which is able to provide a numerical probability, for further analysis.

We identify the students as "Excellent", if the numbers of correct choices they made are higher than or equal to 21 (70% * 29), and "Poor", if marks given are less than 15 (50% * 29). In this case, the numbers of "Excellent" and the numbers of those that are not so good in Group A and Group B are presented in Table 4.

Table 4. The Number of Excellent and Poor

Subjects	Responses	Excellent	Poor
Software Hut	14	3	6
The Genesys	8	3	1

With this criterion, the probabilities obtained are listed as follows.

Probability that a Software Hut student is identified as Excellent:
P (Excellent | Software Hut) = 0.21

Probability that a Genesys student is identified as Excellence: P (Excellent | Genesys) = 0.38

Probability that a Software Hut student is identified as the Poor: P (Poor | Software Hut) = 0.43

Probability that a Genesys student is identified as the Poor: P (Poor | Genesys) = 0.13

According to these results, the Genesys students have higher probability to be Excellent (38% for

Genesys whereas 21% for Software Hut), and the Genesys students have much lower probability to be the Poor (13% for Genesys while 43% for Software Hut).

5. Conclusions and Future research

In this experiment, subjects were divided into teams and assigned to two groups doing a number of projects using two testing strategies. During the development process, one questionnaire was distributed and a further one will be used to measure the testing ability of subjects.

The difference between testing ability of Software Hut students and Genesys students measured by Questionnaire A was analyzed using Mann-Whitney test and then Bayesian approach, the results of which showed Genesys students, who has higher level of professional skills and 2 years' more experience, failed to do better using branch coverage, but were more likely to write tests of higher quality when following the category partition method. This could be because the category partition method requires some analysis of the specification whereas branch coverage is based on an analysis of code structure. When using the test first method the tests must be derived from the specification, therefore these results suggest that test first programming requires higher level of expertise.

Testing ability measured via the assessment of code based testing only is not appropriate for test first developers as we have not yet addressed specification based testing directly. And in the experiment, the subjects are asked to select tests from given test sets rather than generate tests. Therefore subjects will be required to complete Questionnaire B, which is specification based, and involve in a more complicated survey in which subjects will be required to generate tests themselves, to assess their ability to test with and without code.

References

[1] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison Wesley, Reading, Mass, USA, 2000.

[2] K. Beck, *Test Driven Development: By Example*, Addison Wesley, Reading, MA, 2002.

[3] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change, 2nd Edition*, Addison-Wesley, 2004.

[4] D. Chaplin, *test first Programming*, TechZone, 2001.

[5] H. Erdogmus, M. Morisio, and M. Torchiano, "On the effectiveness of the Test-First Approach to Programming", *IEEE Transactions on Software Engineering*, Vol. 31, No. 3, March 2005, pp. 226-237.

[6] N. Fenton, S. Pfleeger, *Software Metrics: a rigorous and practical approach*, 2nd edition, International Thomson Computer Press, London, 1997.

[7] A. Gelman, B. Carlin, H. Stern, and D.B. Rubin, *Bayesian Data Analysis*, *Second Edition*, Chapman & Hall/CRD, Boca Raton, Florida, 2003.

[8] D. Gelperin, W. Hetzel, "Software quality engineering", *4th International Conference on Software Testing*, Washington DC, June 1987.

[9] B. George, L. Williams, "A structured experiment of test-driven development", *Information and Software Technology*, Vol. 46, 2004, pp. 337-342.

[10] M. Holcombe, and M. Georghe, "Enterprise Skill in the Computing Curriculum", *Ingenia*, 2003, pp. 56-61.

[11] L. Huang and M. Holcombe, "Empirical Assessment of Test-First Approach", *Proceedings of Testing: Academic & Industrial Conference*, IEEE Computer Society, Windsor, UK, 2006, pp. 197-200.

[12] C. Larman, V. Basili, "A history of iterative and incremental development", *IEEE Computer*, Vol. 36, 2003, pp. 47-56.

[13] F. Macias, *Empirical Assessment of Extreme Programming*, PhD thesis, University of Sheffield, 2004.

[14] H.B. Mann, D.R. Whitney, "On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other", *Annals of Math. Statistics*, 1947.

[15] E.M. Maximilien, L. Williams, "Assessing test-driven development at IBM", *Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, Portland, Oregon, 2003, pp. 564-569.

[16] Müller M., Hanger O., "Experiment about Test-First Programming", *IEE Proceedings on Software*, Vol. 149, No. 5, October 2002.

[17] T.J. Ostrand and M.J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests", *Communications of the ACM*, Vol.31, No.6, June 1988, pp. 676-686.

Appendix 1: Questionnaire A

To administer this questionnaire, please take the potential test cases and place three columns next to them labeled A, B, and C. With a box for free text labeled D.

The tests required for the category partition method are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, and 29.

Branch coverage must be calculated for each response.

Instructions:

In column A please select a minimal set of tests for the code on the right that are required by the category-partition method of testing. In brief: A category is any property of either some input to the function, or the output from it, which can then be used to identify one or more equivalence classes. A partition of a category is any equivalence class which can be identified for that category. So for example if an input (category) is an integer, you may partition it into MAX_INT, MIN_INT, and the values between them, leading to three test cases.

In column B please select a minimal set of tests that will give branch coverage for the code on the right. For branch coverage your test set should execute each line, and each decision in the code. For if for example there was a line "if (i>10 && s.equals("blogs"))". This would require two tests, once where it evaluates to false, and once where it evaluates to true.

In column C please select a minimal set of tests that you think will test the code best. In box D please describe the strategy you used to select these tests (if any).

Potential test cases:

1. Null flowchart; currentNode = -1; P = random
2. One-node flowchart; currentNode = -1; P = not in List or null
3. One-node flowchart; currentNode = -1; P = Node 0
4. One-node flowchart; currentNode = 0; P = not in List or null
5. One-node flowchart; currentNode = 0; P = Node 0
6. 10-node flowchart; currentNode = -1; P = not in List or null
7. 10-node flowchart; currentNode = -1; P = Node 0
8. 10-node flowchart; currentNode = -1; P = Node 5
9. 10-node flowchart; currentNode = -1; P = Node 6
10. 10-node flowchart; currentNode = -1; P = Node 8
11. 10-node flowchart; currentNode = -1; P = Node 9
12. 10-node flowchart; currentNode = 0; P = not in List or null
13. 10-node flowchart; currentNode = 0; P = Node 0
14. 10-node flowchart; currentNode = 0; P = Node 5
15. 10-node flowchart; currentNode = 0; P = Node 6
16. 10-node flowchart; currentNode = 0; P = Node 8
17. 10-node flowchart; currentNode = 0; P = Node 9
18. 10-node flowchart; currentNode = 6; P = not in List or null
19. 10-node flowchart; currentNode = 6; P = Node 0
20. 10-node flowchart; currentNode = 6; P = Node 5
21. 10-node flowchart; currentNode = 6; P = Node 6
22. 10-node flowchart; currentNode = 6; P = Node 8
23. 10-node flowchart; currentNode = 6; P = Node 9
24. 10-node flowchart; currentNode = 9; P = not in List or null
25. 10-node flowchart; currentNode = 9; P = Node 0
26. 10-node flowchart; currentNode = 9; P = Node 5
27. 10-node flowchart; currentNode = 9; P = Node 6
28. 10-node flowchart; currentNode = 9; P = Node 8
29. 10-node flowchart; currentNode = 9; P = Node 9

The code to be tested:

```
public class NodeSearch {
  public int currentNode = -1; // time saving index.
  public int numNodes = 0; // size of the node list.
  public Node[] nodeList; // a list of sorted nodes
  // which represents the flowchart to be searched.

  /* Other functions are included in this class
   * which will manipulate the class variables
   * defined above. These are the default values.
   */

  /* This function will find the "Node to be found"
   * (p) in a flowchart. If found, the wanted node
   * will be returned, otherwise an exception will
   * be thrown.
   */
  public Node findNode (Node p) throws FGItemNotFound
{
    Node n =null;
    boolean found = false;

    if (currentNode == -1) {
      for (int i = 0 ; (i < numNodes) &&
          (found == false); i++) {
        n = nodeList[i];
        found = n.equals(p);
      } // end of for loop
    }
    else if (nodeList[currentNode].equals(p)) {
      // if p is at the index point
      n = nodeList[currentNode];
      found = true;
    }
    else if (nodeList[currentNode].after(p)) {
      // if the Node to be found is after the
      // currentNode, a true boolean
      // value will be returned by the ".after"
      // method
      for (int i = currentNode +1 ; (i < numNodes)
           && (found == false); i++){
        n = nodeList[i];
        found = n.equals(p);
      }// end of for loop
    }
    else { // if the Node to be found is not after
           // the currentNode
      for (int i = currentNode -1 ; (i >= 0)
           && (found == false); i--){
        n = nodeList[i];
        found = n.equals(p);
      }// end of for loop
    }

    if (found)
      return n; //return the Node that has been found
    else
      throw new FGItemNotFound();
      // Error node not found
  } // end of method findNode
  // Other functions…
}
```