Improving Software Testing Education via Industry Sponsored Contests

W. Eric Wong, Linghuan Hu, and Haoliang Wang,
Department of Computer Science
University of Texas at Dallas, USA
{ewong, linghuan.hu, haoliang.wang}@utdallas.edu

Zhenyu Chen
State Key Laboratory for Novel Software Technology
Nanjing University, China
zychen@nju.edu.cn

***Abstract*— This Innovative Practice, Work in Progress Paper presents how we improve software testing education via industry sponsored contests. Over the past decades, we have built software to improve our efficiency, reliability, and safety in production, business, daily life, etc. These goals, however, cannot be accomplished if the software is not properly tested. Some universities provide classes to teach students the fundamental knowledge and techniques of software testing. However, these classes often ignore industry practices and can hardly offer real-world testing experiences to students. To address this, we partnered with industry sponsors to design and host several software testing contests along with software testing tutorials. Through the contests and tutorials, we brought real-world testing and tool experience to the students and provided excellent opportunities for them to practice their learned testing techniques to overcome industry testing challenges.**

Keywords—software testing, contest, code coverage, JUnit

I. Introduction

Software in many service and mission-critical systems has played imperative roles and has even replaced humans in some aspects to improve efficiency, reliability, and safety. However, severe consequences such as property damage and life-loss accidents can happen due to compromised and/or defective software. In the industry, testing remains the most commonly used technique to help engineers ensure the quality of software and to prevent the accidents above from happening. A project can invest a significant amount of resources on testing, but the software produced may still suffer from low quality. The key point is not how much is spent on testing, but how the testing is conducted and who is conducting it. If we were to trace this deficiency in software testing to its source, we would end up at the educational institutions that are responsible for teaching and training people on how software should be properly tested. Thus, if today's software engineers are not sufficiently armed with the knowledge required to test software cost-effectively, it is most likely because they have not been adequately trained to do so.

To fulfill these needs, UT Dallas along with many other universities have provided software testing classes. However, if these classes are generally lecture-oriented, only emphasizing the oral discussion of different testing techniques while ignoring the practical aspect of how these techniques with appropriate tool support can be applied to test real-world software, students might be unfamiliar with the industry practice, and unable to apply the state-of-the-art techniques learned in classrooms at work. In response, we have organized the International Software Testing Contests (ISTC), sponsored by the IEEE Reliability Society, to offer students an excellent opportunity to catch up with industry practices and experience the challenges that testing practitioners may face. Each contestant is required to implement the entire software testing process, starting from understanding the requirements of the software being tested, to generation and execution of test cases, as well as measurement and improvement of their quality. Common adequacy criteria such as branch coverage and mutation score are used. JUnit testing framework [1] and an online testing platform developed by our industry partner are also used. Through these contests, we bring together students, teachers, researchers, and practitioners. It allows all participants to share their ideas of how software testing should be conducted and realize the deficiency of the current pedagogical approach of software testing education. To our best knowledge, we may be the only group in North America that has organized multiple software testing contests to bridge the gap between classroom education and industry practice on such a large scale.

II. Contest objectives

To address the limitations in software testing education and further improve the testing skills of undergraduates and graduate students, we partnered with our industry sponsors and aim to provide the best opportunities for students to use cutting-edge testing tools to overcome testing challenges that they have never faced before. Among many software testing techniques, such as regression testing, integration testing, etc., unit testing [2] has played a critical role in software development life cycle in industry. It is also the lowest level of testing, which is usually conducted first in the testing phase [3]. A recent survey showed that automated unit testing is one of the top five most important principles adopted in today's agile software development [4]. In this context, the contest emphasizes two of the most important testing techniques: 1) unit testing and 2) test case generation for achieving high code coverage. In the future, we will explore the possibilities of including other techniques.

A. *Unit Testing*

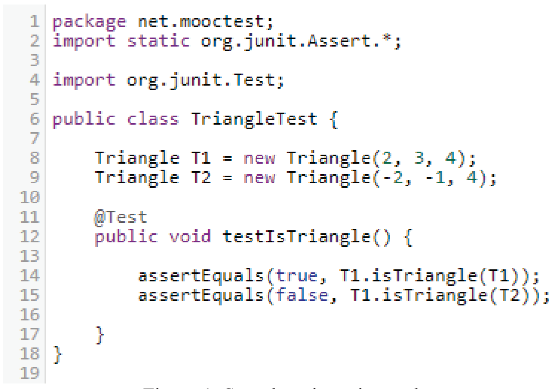
Unlike a decade ago when waterfall software development life cycle was widely used in the industry, agile software development is now adopted by most software companies. One of the reasons is that the software requirements change faster. In response, the developers must constantly revise their software design and implementation. This brings a big problem to the developers as software quality can hardly be

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 29,2023 at 18:10:21 UTC from IEEE Xplore. Restrictions apply.

assured. On one hand, the software was often developed under a tight time constraint, which leads to insufficient testing before the software is released. On the other hand, the constant changes and enhancements on requirements, designs, and implementations make it very difficult to detect the newly introduced bug. To address these challenges, one of the most used testing techniques is unit testing. By focusing on the smallest software module, unit testing can efficiently help detect bugs and, more importantly, easily reuse the generated test cases. Due to these merits, it can significantly reduce testing cost and improve its efficiency.

Although there could be different interpretations of the smallest module, unit testing is often conducted at the class or method level [5]. Figure 1 shows a sample unit testing code using JUnit framework.



```java
package net.mooctest;
import static org.junit.Assert.*;

import org.junit.Test;

public class TriangleTest {

    Triangle T1 = new Triangle(2, 3, 4);
    Triangle T2 = new Triangle(-2, -1, 4);

    @Test
    public void testIsTriangle() {

        assertEquals(true, T1.isTriangle(T1));
        assertEquals(false, T1.isTriangle(T2));

    }
}
```

Figure 1. Sample unit testing code

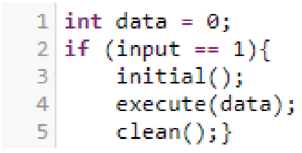
It tests the “isTriangle” method of the “Triangle” class using two test cases –triangle T1 with vertices of 2, 3, and 4, and triangle T2 with vertices of -2, -1, and 4. In the test method “testIsTriangle”, two asserts are used to verify the execution results. In this case, if the “isTriangle” method returns True for T1, and False for T2, the unit test will pass.

Based on our educational experience, most undergraduates or even graduate students from computer science and software engineering are not familiar with this useful testing technique. As for the students who understand unit testing, many of them have not used it. Because of this, we emphasize unit testing in our contest to improve their unit testing skills.

B. *Test case generation for achieving high code coverage*

Although applying unit testing in real-world settings can significantly reduce testing cost and improve the efficiency, other testing techniques are also necessary to improve bug detection effectiveness. In the industry, code coverage is a commonly used metric to determine how thoroughly a software is tested.

For example, statement coverage can be calculated as the total number of statements exercised divided by the number of executable statements in a program [6]. For example, referring to the code shown in Figure 2, if the value of the input does not equal 1, the statement coverage will be 0.4 (two divided by five).



```java
int data = 0;
if (input == 1){
    initial();
    execute(data);
    clean();}
```

Figure 2. A code segment

The reason for its popular application can be implied by a simple heuristic – “You might not be able to detect the bugs if you execute the code. However, you definitely cannot detect the bugs if you do not execute the code”. By including the code coverage achievement, we want to motivate students to apply their learned advanced testing techniques, such as equivalent class partitioning, boundary value analysis [7,8], combinatorial testing [9-11], etc., to generate high-quality test cases that can achieve high coverage. Thus, we included code coverage as one of the contest evaluation criteria.

III. CONTEST SETUP

To successfully bring real-world testing tools and challenges to the students via the contest, we need to appropriately select subject programs and design the ranking criteria. We gained valuable experience from hosting several testing contests at different venues, including an international conference in Europe as well as universities in the USA and China. In this section, we will present the details of subject program selection and ranking criteria.

A. *Subject Programs*

For the software testing contest, we need to carefully select appropriate subject programs since understanding the requirements and implementation can be very challenging for students under a strict time constraint. The subject programs cannot be too simple, but their complexities need to be carefully managed. If the subject programs are too complex and too difficult to understand, it may frustrate students, which might negatively impact our goals.

As most high schools and universities offer courses on Java, we choose Java for our contest. We identify the contest subject programs based on our teaching experience and several complexity metrics, such as lines of code, the number of branches, etc. For each open source subject program that we identified online, we use PITest [12], a popular and powerful mutation testing tool. With its eight default mutation operators [13], as shown in Table 1, we can generate hundreds of mutants. Some sample code before and after mutation operators that are applied are shown in Table 2. The generated mutants are used to measure the mutation scores. More details are explained in Section III. B.

B. *Ranking Criteria*

The score of each contestant will be determined by two criteria – achieved branch coverage and mutation score. As we mentioned previously, code coverage is widely used in the industry to determine the quality of the testing. During the contest, contestants can measure the achieved code coverage of their generated test cases and use this information as hints to generate more test cases.

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 29,2023 at 18:10:21 UTC from IEEE Xplore. Restrictions apply.

Table 1. Mutation operators used for mutant generation

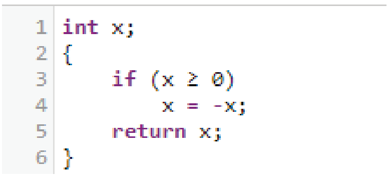
Mutation Operator	Description
Conditionals Boundary	The conditionals boundary mutator replaces the relational operators <, <=, >, >= with their boundary counterpart as per the table below.
Increments	The increments mutator will mutate increments, decrements and assignment increments and decrements of local variables (stack variables). It will replace increments with decrements and vice versa.
Invert Negatives	The invert negatives mutator inverts negation of integer and floating-point numbers.
Math	The math mutator replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation.
Negate Conditionals	The negate conditionals mutator will mutate all conditionals found according to the replacement table below.
Return Values	The return values mutator mutates the return values of method calls. Depending on the return type of the method another mutation is used.
Void Method Calls	The void method call mutator removes method calls to void methods.

Table 2. Sample code before and after mutation operators applied

Mutation Operator	Before Mutation	After Mutation
Conditionals Boundary	if (a < b)	if (a <= b)
	if (a <= b)	if (a < b)
	if (a > b)	if (a >= b)
	if (a >= b)	if (a > b)
Increments	i++;	i--;
Invert Negatives	-i;	i;
Math	a + b	a - b
	a - b	a + b
	a * b	a / b
	a / b	a * b
	a % b	a * b
Negate Conditionals	a == b	a != b
	a != b	a == b
	a <= b	a > b
	a >= b	a < b
	a < b	a >= b
	a > b	a <= b
Return Values	return false;	return true;
	return true;	return false;
	return x;	return x+1;
Void Method Calls	initial(); do(); cleanup();	initial(); cleanup();

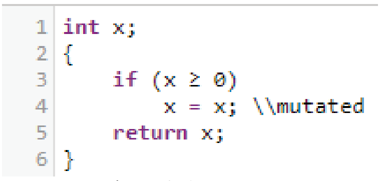
In addition to branch coverage, we also include mutation score as the second criterion to evaluate bug detection strength of the contestants' generated test cases. This is because, in real-world settings, unless practitioners have observed the unexpected exception thrown by the program or unexpected execution failure, practitioners need to spend a large amount of time on output verification to determine whether the test cases detected any bugs. In our three-hour contest, it is not feasible to require contestants to work on both test generation and output verification. Mutation score is a good metric to evaluate the bug detection strength of the test cases, and does not require manual output verification. In mutation testing, a mutant is considered to be killed if its output is different than the output of the original program.

Consider the following code segments, where the code shown in Figure 3 is the original and the code shown in Figure 4 is a mutant with the statement at line 4 that is changed. If an input x of 3 is given, the outputs of both programs are different. Therefore, we say the mutant is killed.



```
1 int x;
2 {
3     if (x ≥ 0)
4         x = -x;
5     return x;
6 }
```

Figure 3. The original code segment



```
1 int x;
2 {
3     if (x ≥ 0)
4         x = x; \\mutated
5     return x;
6 }
```

Figure 4. A mutant

The mutation score is computed by the number of killed mutants divided by the number of all generated mutants. As opposed to the code coverage, we do not provide the mutation score measurement to contestants so that the contestants can experience the real challenges faced by testing practitioners. This is because, in real-world settings, testers estimate bug detection strength of the generated test cases based on their testing skills and their understanding of the subject program.

The aforementioned branch coverage achievement and mutation score are measured by the Mooctest testing tool [14]. Mooctest testing tool is an easy-to-use online testing tool developed by our industry sponsor, which integrates JUnit library with several related software testing measurements, such as code coverage, mutation score, etc.

C. Contest and Tutorial

The software testing contest has three sections: 1) the tutorial 2) the contest, and 3) the experience sharing seminar.

To attend the contest, the contestants only need to have the fundamental knowledge of Java programming, unit testing, and mutation testing. Before the contest, we give contestants a brief introduction to the ranking criteria and subject programs (no source code or details are provided). Comprehensive tutorials of JUnit, tools (Eclipse and Mooctest), and mutation testing are also provided. After the tutorial, we will release a hands-on exercise that includes several sample subject programs to each contestant. Then, we will help each contestant install the contest testing environment.

In the contest, each contestant needs to read and understand three Java subject programs that are randomly selected from our subject program pool and then generate and implement the JUnit test cases. When the contest begins, a timer will appear on their screen, and each contestant can use Eclipse IDE with Mooctest testing platform to download and automatically deploy three testing subject programs. During the contest, a contestant can execute the program using their implemented JUnit test cases and measure the achieved branch coverage. A real-time updated leaderboard will anonymously show the top

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 29,2023 at 18:10:21 UTC from IEEE Xplore. Restrictions apply.

10 contestants' scores. When the timer is up, the contestants can no longer write or change any JUnit test cases, and the contest is finished. After we update and validate their final scores, we will then announce the top 3 contestants in the award ceremony.

An experience sharing seminar will be hosted after the contest so that the contestants can share their testing experience.

IV. Previous contests

The two latest international software testing contests were hosted at the IEEE International Conference on Software Quality, Reliability, and Security at the Czech Technical University in July 2017 (ISTC 17) and the University of Texas at Dallas in January 2018 (ISTC 18).

In ISTC 17, 27 graduate students from different universities participated in the contest. In ISTC 18, there were not only 158 undergraduates and graduates who participated in the contest, but also 35 students from several high schools. High school teachers said their students were very interested in the testing contest, and that this can be an excellent opportunity for students to access software testing rather than just programming. We divided the contest into a college group and a high school group. Based on our teaching experience at UT Dallas, we had not observed a significant difference in the software testing background between undergraduates and graduate students. Therefore, both undergraduates and graduate students are in the college group. The contests for the high school group and college group are the same, except that the subject programs for the high school group are smaller and less complicated.

V. Related Studies

Software testing contest is a relatively new, but fast-growing event. Nanjing University, which also partners with Mooctest, hosted two national software testing contests for college students, including mobile application testing, integrated device testing, and web security testing in China in 2016 and 2017 [15,16]. There were a few other software testing contests for industry practitioners hosted by companies and organizations. The Software Testing World Cup is a series of software testing contests hosted at different locations, such as in Europe, South America, North America, Asia, etc., from 2014 to 2016 [17,18]. The contestants test a subject program and write a testing report about the issues found with severities and suggested fixes. The grade of each team was evaluated by contest judges. The CAST Testing Competition [19] was a software contest in 2011, hosted by Satisfice Inc., that ranked the contestants based on the performance w.r.t testing, test report, bug report, developer relation, and interviews. Inflectra Inc. hosted two software testing contests [20,21] where contestants tested a subject program and reported the found bugs. The rank was determined by the found issues and test report. There were also other similar contests such as BugDeBug [22], International Software Testing Contest [23], and UNICOM Software Testing Contest [24]. To the best of our knowledge, we may be the only group in North America that has organized multiple software testing contests for college and high school students on such a large scale.

VI. Conclusions and Future direction

In this paper, we present how we improve the unit testing skills of undergraduates, graduate, and high school students via software testing contest. Details of industry tools, subject programs selection and ranking criteria are presented. We believe the software testing contest is the next step to take students beyond the lecture-oriented, oral discussion of software testing education. By joining the contests, students can not only improve their testing skills, but also access the latest industry tool support, experience the real-world testing challenges and have the opportunity to apply their learned techniques.

For future contests, we are carefully designing a controlled experiment to quantitatively analyze the impact of the contest. In addition, we will keep revising our ranking criteria and include more programming languages. A team-based contest will also be added in future contests. Additional sections will also be included for students to experience different testing scenarios and new challenges, such as regression testing, performance testing, etc.

References

1. JUnit 5, Available: https://junit.org/junit5/ [Accessed: June 18, 2018]
2. S. Acharya, "Mastering Unit Testing Using Mockito and JUnit," Packt Publishing Ltd, 2014.
3. P. Ammann and J. Offutt, "Introduction of software testing," Cambridge University Press, 2nd edition, 2016
4. L. Williams, "What agile teams think of agile principles," Communications of the ACM, vol. 55, no. 4, pp 71-76, April 2012
5. E. Dietrich, "Starting to Unit Test: Not as Hard as You Think," BlogIntoBook, May 2014
6. S. Desikan and R. Gopalaswamy, "Software Testing: Principles and Practice," Pearson, 1st edition, 2005
7. J. Eo, H. Choi, R. Gao, S. Lee, and W. Eric Wong, "Case Study of Requirements-based Test Case Generation on an Automotive Domain," in *Proceedings of IEEE International Conference on Software Quality, Reliability, and Security – Companion*, pp. 210-215, Vancouver, Canada, August 2015.
8. R. Gao, J. Eo, W. E. Wong, X. Gao, and S. Lee, "An Empirical Study of Requirements-based Test Generation on an Automobile Control System," in *Proceedings of Annual ACM Symposium on Applied Computing*, pp. 1094-1099, Gyeongju, Korea, March 2014.
9. D. Li, L. Hu, R. Gao, W. E. Wong, D. R. Kuhn, and R. N. Kacker, "Improving MC/DC and Fault Detection Strength Using Combinatorial Testing," in *Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security Companion*, pp. 297-303, Prague, Czech Republic, July 25-29, 2017.
10. Ruizhi Gao, Linghuan Hu, W. Eric Wong, Han-Lin Lu, and Shih-Kun Huang, "Effective Test Generation for Combinatorial Decision Coverage," in *Proceedings of 2016 IEEE International Conference on Software Quality, Reliability and Security Companion*, pp. 47-54. Vienna, Austria, August 1-3, 2016
11. Kuhn, D. Richard, Renee Bryce, Feng Duan, Laleh Sh Ghandehari, Yu Lei, and Raghu N. Kacker. "Combinatorial testing: Theory and practice," *Advances in Computers*, vol. 99, pp. 1-66, 2015.
12. H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "PIT: a practical mutation testing tool for Java (demo)," in *Proceedings of International Symposium on Software Testing and Analysis*, 2016, pp. 449-452

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 29,2023 at 18:10:21 UTC from IEEE Xplore. Restrictions apply.

13. *Mutation Operators of PITest.* http://pitest.org/quickstart/mutators/, 2018
14. Mooctest [Online]*, Available:* http://www.mooctest.net/login2*, 2018* [Accessed: June 18, 2018]
15. Mooctest [Online]. Available: http://www.mooctest.org/cst2017/cst2016/ [Accessed: June 18, 2018]
16. Mooctest [Online]. Available: http://www.mooctest.org/cst2017/cst2017/final.html [Accessed: June 18, 2018]
17. Summary of the Software Testing World Cup 2014 Available: http://www.softwaretestingworldcup.com/stwc-2014/ [Accessed: June 18, 2018]
18. STWC 2016 – Software Testing World Cup 2016 Available: http://www.softwaretestingworldcup.com/ stwc-2016/ [Accessed: June 18, 2018]
19. The CAST Testing Competition | Satisfice, Inc. Available: http://www.satisfice.com/blog/archives/605 [Accessed: June 18, 2018]
20. 2nd Inflectra Hackathon – Software Testing Competition Tickets, Available: https://www.eventbrite.com/e/2nd-inflectra-hackathon-software-testing-competition-tickets-44676313055# [Accessed: June 18, 2018]
21. Social Testing – A Software Testing Competition | Inflectra Available: https://www.inflectra.com/Company/Event/social-testing--a-software-testing-competition-579.aspx [Accessed: June 18, 2018]
22. BugDeBug Software Testing Competition at 99tests Available: https://99tests.com/blog/bugdebug-software-testing-competition-at-99tests/ [Accessed: June 18, 2018]
23. E. Alégroth, S. Matsuki, T. E. J. Vos, and K. Akemine, "Overview of the ICST International Software Testing Contest," in *Proceedings of IEEE International Conference on Software Testing, Verification and Validation,* pp. 550-551, Tokyo, Japan, March 2017
24. Software Testing Contest, Available: http://www.unicomlearning.com/software-testing-contest.html [Accessed: June 18, 2018]

Authorized licensed use limited to: b-on: UNIVERSIDADE DO PORTO. Downloaded on October 29,2023 at 18:10:21 UTC from IEEE Xplore. Restrictions apply.