

SPRAE: A Framework for Teaching Software Testing in the Undergraduate Curriculum

Edward L. Jones
Department of Computer Information Sciences
Florida A&M University*
ejones@cis.famu.edu

ABSTRACT

For the past three years undergraduate students at Florida A&M University have been exposed to the practice of software testing, both as part of the senior projects course, and in an elective course in software testing. Because these courses provide a mere introduction to software testing, the students are presented a framework for understanding the theory, management and practice of testing and quality assurance. This paper defines this framework, SPRAE (Specification-Premeditation-Repeatability-Accountability-Economy). We show that SPRAE provides a simple methodology for teaching testing, and it provides an evaluative model for assessing organizational practice of SQA/testing. Finally, we show how this framework is being used to establish the software testing laboratory (TestLab) environment in which students learn the art and science of software testing

Keywords: software testing, partitioning, test automation, specification-based testing.

1 Introduction

Although testing accounts for nearly 50% of software development costs, undergraduate computer science programs typically devote nearly 100% of instruction to the process of developing software, and very little to how to test software. There is a need to increase the quantity and quality of the treatment of testing in the curriculum. Too often, students view testing as merely a demonstration that the program compiles and produces *some* output.

The premise of a recent article by Whittaker [1] is that "testing is arguably the least understood part of the development process." Although expertise in testing has been part of the classical computer science literature for decades [2,3], the subject matter has never been incorporated into the mainstream of undergraduate education. As a result, professionals enter the workforce lacking practical or theoretical foundations in testing. A premise of this work is teaching software developers testing makes them better software developers.

How does one go about learning the basics of testing to establish the foundation for career-long learning? This is the issue discussed in this paper. The SPRAE framework presented in this paper is currently being used to evolve the Florida A&M Software TestLab. The goal of the TestLab project is to provide an environment in which students can learn both the art and science of software testing, and to incorporate lessons learned from the laboratory into courses in the undergraduate curriculum.

* This work received partial support from National Science Foundation grant EIA-9906590, the Dell Computer STAR Program, and Lucent Technologies.

2 The SPRAE Framework

SPRAE is an acronym for major principles of quality assurance that can be applied to software testing. In this section each symbol in the acronym is expanded. A course in software testing should include experience applying each of these principles.

- ❑ **Specification.** *A specification is essential to testing.* The simple example makes this point clear: In order to tell whether 11 is the correct output from a function taking inputs 3 and 7 requires that the tester know what the function is expected to do. The specification states the expected externally visible behavior of software. This principle can be stated alternatively as *No spec, no test.*
- ❑ **Premeditation.** *Testing requires premeditation, i.e., a systematic design process.* Testing is not a random activity, but is based upon principles and techniques that can be learned and practiced systematically. This principle suggests that there is a test life cycle that includes stages of analysis and design. The various test design techniques [4] are based on theoretical and empirical foundations.
- ❑ **Repeatability.** *The testing process and its results must be repeatable and independent of the tester.* The practice of testing must be institutionalized to the extent that all practitioners use the same methods and follow the same standards. In terms of the SEI CMM, the process must *repeatable* and *defined*, and practitioners must be trained.
- ❑ **Accountability.** *The test process must produce an audit trail.* Test practitioners are responsible for showing what they planned to do (premeditation), what they actually did, what the results were, and how the results were interpreted. These results must be reviewable by others. This principle may require the capture, organization and retention of large volumes of data.
- ❑ **Economy.** *Testing should not require excessive human or computer resources.* In order for a test process to be followed, it must be cost effective. Concern for cost provides a counterbalance to the previous elements of the SPRAE framework. The economy principle is often the single driver of an organization's test practice. Economy fuels the pursuit of automated testing tools.

The SPRAE framework represents a way of understanding why testing is a complex process with competing dimensions of theory and practicability. SPRAE is a tool for negotiating these complexities.

3 An Example – Function Testing

In this section we illustrate the use of SPRAE to test a simple function. `Pay`, which computes the weekly gross pay for an hourly employee. `Pay` takes two arguments and returns a single real number. The testing problem is this: Does `Pay` always produce the correct answer? The test process has three steps: (1) develop a set of test cases; (2) execute the function for each test case in the test set; (3) verify that for each test case the computed pay matches the expected pay. The lifecycle (methodology) we use is shown below.

Specification → Test Cases → Test Script → Test Execution → Test Evaluation

3.1 Specification

A specification is a statement of what the function does. The specification is required to determine the expected result portion of a test case. There are various forms a specification can take. We will illustrate three: natural language narrative; semi-formal narrative with pre- and post-conditions; and decision table.

Specification #1 (natural language narrative): Compute pay for an hourly employee, given the hours worked and hourly pay rate. Compute overtime at 1.5 times for hours in excess of 40.0.

A set of test cases for this specification is given below. One case covers overtime, the other does not.

Figure 1. Pay Test Set #1

Rate	10	10
Hours	40	50
Expected Pay	400.00	550.00

This specification is simplistic, as it does not consider practical limits such as a maximum hourly pay rate or the maximum number of hours an employee can work per week (can never exceed 168 hours!). Because this specification is deficient, so is this test set.

Specification #2 (semi-formal narrative): Compute pay for an hourly employee, given the hours (*Hours*), and hourly pay rate (*Rate*). Compute overtime at 1.5 times for hours in excess of 40. Return -1 when *Hours* or *Rate* is invalid.

Preconditions: $(0 \leq \text{Hours} \leq 112)$ and $(5.15 \leq \text{Rate} \leq 20)$

Because this specification is more complex, a more systematic process for test case design is required.

3.2 Premeditation

This section illustrates process for designing functional test cases based on the specification. When the specification is simple, an intuitive process of deriving test cases is often adequate. In general, however, a systematic, repeatable process for translating the specification into test cases is needed. It is also highly desirable that this process be amenable to automation. We illustrate this process using Specification #2. The narrative specification will be converted to a decision table (Figure 2), and test cases will be derived from the columns in the decision table. The upper part of the decision table defines the implicit and explicit binary conditions stated in the specification; the lower (shaded) portion specifies the computation rules defined in the specification. The columns define the different behaviors of the function. For a given column, the combination of condition values ('Y' or 'N') selects the computational rule(s) marked by 'X'.

Each column in the decision table defines a distinct behavior of the function determined by the combination of input conditions under which a subset of computation rules is selected.

The test case design rule, “define one test case to satisfy each behavior,” is a *test partitioning* strategy [4].

Figure 2. Decision Table for Pay

Hours \geq 0	Y	Y	N	-	-	-	-	-		
Hours \leq 112	Y	Y	-	N	-	-	-	-		
Rate \geq 5.15	Y	Y	-	-	N	N	-	-		
Rate \leq 20.00	Y	Y	-	-	-	-	N	N		
Hours \leq 40	Y	N	Y	N	Y	N	Y	N		
Pay = Hours * Rate	X									
Pay = Rate * (Hours + 1.5 (Hours - 40))		X								
Pay = -1			X	X	X	X	X	X		
Function Behaviors (equivalence partitions)	1	2	3	4	5	6	7	8		

The test set in Figure 3 defines eight test cases for testing Specification #2.

Figure 3. Pay Test Set #2

	Test Case							
	1	2	3	4	5	6	7	8
Rate	10	10	10	10	2	3	40	30
Hours	40	50	-3	120	10	50	10	50
Expected Pay	400.00	550.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00

3.3 Repeatability

Testing is repetitive in nature. During initial testing the goal is to achieve a "clean" test run, i.e., execute an entire test set without error. When testing uncovers an error, the cycle "*fix the error(s), then retest*" is followed. When significant changes are made, it may be necessary to do *regression testing*, where previously conducted tests are repeated to ensure that errors were not introduced by the changes. In both these situations, the same test cases must be executed. This requires that they be recorded, along with a definition of the procedures needed to execute the tests.

The primary test product for ensuring repeatability is the *test script*. The test script has two major parts: the test build, and test execution. The goal of the test build step is to ensure that the correct version of the software is tested: copies of controlled versions of the source code and test data files are made. The test execution step gives explicit instructions for executing each test case. When the test requires a human in the loop, the test execution script is a detailed, step-by-step presentation of test cases. When the test is file-driven, the test execution script explains how to run the test script. Figure 4 shows the test execution portion of the test script for Pay.

Figure 4. Test Script Execution Step (Pay)

	User Action		Expected Result
Step	Enter Rate =	Enter Hours	Pay =
1	10	40	400.00
2	10	50	550.00
3	10	-3	-1.00
4	10	120	-1.00
5	2	10	-1.00
6	3	50	-1.00
7	40	10	-1.00
8	30	50	-1.00
9	END		

3.4 Accountability

Executing the test script should produce test results which, like the test script, are recorded and saved in a controlled environment. The *Test Evaluation* stage of the test lifecycle requires that test results be inspected for discrepancies between expected and actual results. Each apparent discrepancy must be recorded in the *test log* file with sufficient detail to determine the cause of the discrepancy: (a) the wrong expected result is stated; (b) the driver or data file contains an error; or (c) the function under test contains an error.

The collection of all the products of the test lifecycle provides an audit trail that answers the basic questions, "What was attempted?", "What were the results?", "What were the errors?" The accountability principle addresses issues of professional responsibility and, in the event of software induced catastrophes, professional liability.

Figure 5. Test Execution Log (Pay)

Step	Test Identification								
	01	02	03	04	05	06	07	08	09
1	P	P	P						
2	P	P	P						
3	P	P	P						
4	P	P	P						
5	F	P	P						
6	P	P	P						
7	P	P	P						
8	P	F	P						
END									

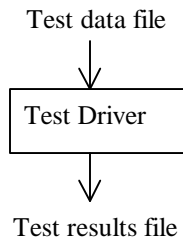
• • •

Discrepancy Log		
TestID	Step	
01	5	16.50 returned. Expected -1.
02	8	222000.00 returned. Expected -1.

3.5 Economy

Economy addresses cost effectiveness. There is a need to work smart rather than work hard. Limitations in testing are accepted, some theoretical, others practical. The impossibility of exhaustive testing is an accepted theoretical result. The practical question, "How much testing is enough?" is often resolved by answering the question "How much resource (time, labor, computer) is available?"

The economy principle motivates practices that lower the cost of performing tasks in the test process. This ubiquitous trade-off involves choices to delete, curtail, automate or manually perform tasks in the overall test process. Repetitious tasks are prime candidates for automation. The principle of economy is best illustrated by the use of a test driver to automate the (repetitious) execution of test sets.



The test driver is a program which reads test cases from a test data file, calls the function with the test inputs as arguments, and captures the actual results from the call; writes the test case and the test results to the test results file.

There are several benefits from using a test driver. It ensures repeatability of tests, the test data input and results files provide an audit trail, i.e., satisfies the principle of accountability. Because having test cases is a prerequisite to using a test driver, the principle of premeditation is also satisfied.

As mentioned earlier, automation can occur at many places in the test process. The trick to automation is the use of formalisms (specifications) from which test products can be derived mechanically. Test automation is a major thrust of current testing research [4].

4 How Big is the Testing Problem?

The purpose of this paper is to explain the SPRAE framework. In this section we make the case that SPRAE is general enough to provide guidance for most testing situations. Figure 6 depicts three dimensions of the software testing problem: purpose, granularity and strategy. This characterization is intended to show the breadth and diversity of testing concerns.

Figure 6. Dimensions of Software Testing

Purpose	Granularity	Strategy
Acceptance Demonstration Defect Identification Reliability Assessment Performance Measurement	Function (unit) Object Component Application	Specification Driven (<i>black box</i>) Implementation Driven (<i>white-box</i>) Usage Driven (<i>operational</i>) Random (<i>statistical</i>) Intuitive (<i>ad hoc</i>)

The author suggests that SPRAE is a generic framework that can be applied to describe the *practice* of testing for most of the instances characterized in Figure 6.

- Some form of specification is always required. For certain strategies (intuitive, random) the specification may be implicit in the tester's understanding of the object under test.
- Premeditation, the application of a design process, is always required. The design approach depends upon the testing situation. The need for a rigorous process is strongest for specification-based testing, weakest for intuitive.
- Repeatability is always required. Automation provides the most reliable mechanism, while contributing to economies of scale.
- Accountability is a management concern spanning technical oversight, contractual demands on the development process, and product liability. Accountability requires record keeping.
- Economy. Testing requires human, computing and time resource. Constraints on these resources driver the choices of what to test, and how to test, and when to stop testing. Automation provides the leverage for accomplishing more in a given amount of time.

5 The TestLab Project

The *goal* of the TestLab project is to provide an environment in which students can learn the art and science of software testing. The *approach* is to have students work on specific testing tasks to produce and disseminate results in the form of completed artifacts, tutorials, experience reports and research publications. The two-fold *benefit* we seek is to interest students in testing as career and graduate study options, and to incorporate the lessons learned from the TestLab into courses in the undergraduate curriculum.

The Internet is the primary dissemination vehicle for the TestLab. The TestLab web site <http://www.cis.famu.edu/~testlab> is under construction. When fully implemented, it will provide the services shown in Figure 7. The TestLab Mission contains marketing products such as the TestLab brochure, back issues of *The TestLab Bulletin*, a bimonthly newsletter with articles about on-going and planned work in the TestLab. The web site also highlights students who have worked on TestLab projects.

Mission →	Marketing, brochure, <i>FAMU TestLab Bulletin</i> issues.
Director →	Picture, professional data, research.
Students →	Pictures, biographical and interesting data.
Sponsors →	Logos, links to corporate sponsors.
Projects →	Links to current, past, future projects.
Research →	Links to current ideas, resources.
Artifacts →	TestLab project results
Test Commerce →	Links to test tools/training companies.
Invitation →	Open proposals for corporate support.
Fun & Games →	On-line testing problems that challenge skills.

The Fun & Games component provides a *testing arcade* in which a player is given a testing problem to solve. The score earned is based on the accuracy of the solution and the time taken. The top scores are retained and posted in the arcade. Unlike an arcade, help is available during a “game” in a form ranging from brief examples to detailed tutorials. As with video games, multiple expertise levels will be supported. Potentially, the full inventory of testing skills supported by TestLab can be delivered through the arcade. The testing arcade can support on-line *testing contests*, an alternative to

programming contests. The testing arcade has the potential of being the most popular and beneficial feature of the TestLab web site.

The mission of the TestLab is to provide supplemental instructional resources and laboratory experiences for existing courses in the CIS curriculum. The testing tools and test products from this project will be adapted into instructional modules appropriate to selected courses, and will be made available as lectures, exams, laboratory exercises, or on-line tutorials.

The TestLab work is funded by corporate partners Lucent Technologies and Dell Computer, and by an NSF grant. The NSF support targets students who plan to attend graduate school. Corporate sponsors are needed to extend student involvement to all students. Corporate partners are encouraged to offer students internships in software testing/quality assurance.

6 Conclusion

We have used an example of testing a simple function to illustrate the elements of a testing process motivated by the SPRAE framework. First, a specification for the function is developed or refined. Next, the specification is transformed into a representation from which the *characteristic behaviors* of the function can be determined. Test cases are defined based on these behaviors. Each test case is a tuple including stimuli and expected responses. Next, a test script is developed which defines a process for repeated execution of a test using explicit test cases. The test script also includes steps to assemble the required data and software components, to build and run the test executable, and to record results. Next, testing is conducted according to the test script. During test execution or upon completion, test results are evaluated, and discrepancies from expected results are recorded in a test log. All test products are placed under configuration control in a form suitable for independent review.

A similar process can be defined for any instance of the testing problem as characterized in Figure 6. There is a standard set of test products: product specifications, test cases, test scripts, test drivers, test results data, and test logs. As part of their undergraduate study, students need to experience generating and using these artifacts for at least one instance of the testing problem. One such experience will provide the basis for tackling other instances of the testing problem.

7 References

- [1] Whittaker, J.A., "What is software testing? And why is it so hard?," *IEEE Software Vol. 17*, No. 1, January 2000, pp. 70-79.
- [2] Myers, G.J., *The Art of Software Testing*, John Wiley & Sons, New York, 1976.
- [3] Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.
- [4] Zhu, H., Hall, P.A.V., and May, J.H.R., "Software unit test coverage and adequacy," *ACM Computing Surveys, Vol. 29*, No. 4, December 1997, pp. 366-427.
- [5] McGregor, J.D. and Korson, T.D., "Integrated object-oriented testing and development process," *Communications of the ACM Vol. 37*, 9, September 1994, pp. 59-77.