Editor: Christof Ebert
Vector Consulting Services
christof.ebert@vector.com

Test Automation

Macario Polo, Pedro Reales, Mario Piattini, and Christof Ebert

Testing is a destructive task in which the goal is to find relevant defects as early as possible. It requires automation to reduce cost and ensure high regression, thus delivering determined quality. This article reviews technologies for test automation. I look forward to hearing from both readers and prospective column authors about this column and the technologies you want to know more about. —*Christof Ebert*

TESTING CONSUMES 30 to 60 percent of all life-cycle cost, depending on product criticality and complexity.[1] It is therefore pivotal to control and reduce test costs. Because we know that testing is important and cannot simply be cut out, test automation plays a key role. This means we must look to test cases, their selections, and their handling. A basic element of testing is the test case, which includes initial conditions to properly set up a testing environment and make the test case reproducible, test data for assigning values to parameters, and operations to be launched against the system under test (SUT) and the oracle (the mechanism that decides whether the test case found an error or not). For creating test data, some well-known techniques exist[2]:

- *Input space partitioning*—the input space is divided into disjoint partitions, and, assuming the SUT will behave the same with all values in the partition, the tester must select only a representative element.
- *Boundary values*—the tester includes in the test data the values in the limits of partitions—that is, if the partitions are positive integers and negative integers, the tester will include the test data 1, 0, and –1.
- *Error guessing*—the tester adds values that have a high probability of highlighting errors.

> Testing consumes 30 to 60 percent of all life-cycle cost, depending on product criticality and complexity.

Test data can be combined in different ways, as Table 1 shows. All combinations get test cases computing the Cartesian product of all proposed values. Pairwise strategies produce test cases visiting all pairs of values for each two parameters; *n*-wise is a generalization of pairwise, now visiting all tuples of *n* values.

Coverage criteria assess the percentage of the tested SUT. Some common coverage criteria include statements, decisions, conditions (respectively fulfilled when test cases have run over all statements, decisions, and conditions—for example, if after we run all the tests, all the statements of the system were executed, the statement coverage is fulfilled by the tests), and modified condition/decision. (This powerful criterion checks conditions and decisions depending on the variable that has influence on every branch of the statement; it depends on the test data used.)

An additional criterion is the mutation score, which defines the quality of the test cases based on their ability to find faults. With a tool, the tester makes faulty copies of the SUT. The idea is to build a test suite that finds all artificial faults via the faulty copies; if the test suite doesn't find any fault on the SUT, then it's likely error-free (because the designed tests have a high ability to find errors).

There are different testing levels, depending on the nature of the element being tested. Figure 1 shows the V model, which includes the most common levels: the unitary level focuses on units (small pieces of software), the integration-testing level focuses on the interaction between units, and the system-testing level focuses on complete functionalities. Up until the system-testing level, all testing is functional. However, in this level and in the next two—the user-acceptance level, which focuses on testing the system to be accepted by the final user, and the release-testing level, which focuses on testing the system in production—new nonfunctional features appear (that is, they appear when the system is complete). This is because performance or security must be tested with other techniques like load testing, stress testing (both of which test the system with heavy loads and concurrency), or black-box security testing (which provides input to the system to check security issues). Moreover, the highest levels of the V model typically use keyword testing (which is based on the keywords found in the specifications) and exploratory testing (which has manual interaction with the GUI).

TABLE 1

Test data combined in different ways.

Parameters A, B, and C with their possible values	
A = {1, 2, 3}; B = {a, b, c}; C = {X, Y}	
Values combination of each parameter	
All combinations (3 × 3 × 2 = 18 test cases)	Pairwise (9 test cases)
{1,a,X}, {1,a,Y}, {1,b,X}, {1,b,Y}, {1,c,X}, {1,c,Y} {2,a,X}, {2,a,Y}, {2,b,X}, {2,b,Y}, {2,c,X}, {2,c,Y} {3,a,X}, {3,a,Y}, {3,b,X}, {3,b,Y}, {3,c,X}, {3,c,Y}	{1,a,X}, {1,b,Y}, {1,c,X} {2,a,Y}, {2,b,X}, {2,c,Y} {3,a,X}, {3,b,Y}, {3,c,X}

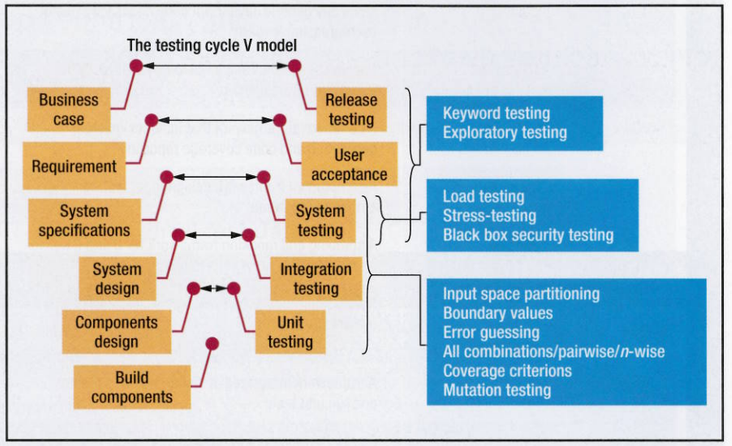


FIGURE 1. The V model and testing techniques.

Testing Methodologies

With other software processes (such as development, maintenance, and configuration management), it's convenient to carry out testing activities with some supporting methodology to identify tasks, artifacts, roles, techniques, and so on. Some methodologies are specifically designed to test systems. TMAP Next,[3] for example, includes three technical processes (preparation, specification, and execution) to design and execute test cases.

A very important process in this is specification, which is where tests are designed and built; at this point, it's important to automate tests to execute in the next process and reuse them in other testing projects. TMAP Next also includes two horizontal phases (control and infrastructure) for monitoring projects and managing the required infrastructure (software, hardware, working places, and so on), and two additional phases for planning and completing the testing project. This methodology is based on a risk and business-driven approach: test cases focus on the system's most critical parts that are under testing and try to find errors as soon as possible.

Test Automation Technologies

Test automation reduces costs and increases the quality of the testing tasks.[4] In practice, XUnit frameworks are the most used technology to automate tests. In such frameworks, test cases are written in an executable language and can be executed automatically. They also provide specific operations to implement the test case oracles.

Capture and replay tools comprise another commonly used technology; they register a tester's interactions with the SUT, save them as a test script, and automatically re-execute them.

0740-7459/13/$31.00 © 2013 IEEE

TABLE 2

XUnit frameworks.

Language	Framework	Description	License	Maturity level*	Expertise level**
Java	JUnit	The most famous XUnit framework for Java	Open source	3	1
	JTest	A commercial tool that includes automated test generation and execution	Commercial	3	3
	JMock	An extension of the JUnit framework to create mock objects	Free	1	2
JavaScript	DOH	Runs tests in-browser or independently	Open source	3	2
	QUnit	Tests any generic JavaScript code and is very useful for regression testing	Free	2	3
	JSTest.Net	Enables JavaScript unit tests to be run directly in other XUnit frameworks	Free	3	3
C/C++	C++ test	A commercial framework that includes unit test generation and code coverage reporting	Commercial	3	3
	Cantata++	A commercial framework designed for testing embedded systems	Commercial	2	3
	Opmock	A stubbing and mocking framework for C and C++ based on code-generation headers	GPL	1	2
	Google C++ Testing Framework	A framework designed by Google to test C++ systems	Free	2	2
.NET	NUnit	A framework integrated in Visual Studio to create and run unit tests	Free	1	1
	DbUnit.NET	An XUnit framework for testing databases	Open source	2	3
	MbUnit	A model-based XUnit framework	Free	2	2
	QuickUnit.NET	Design tests without code and is very useful for TDD	Commercial	3	3
PHP	PHPUnit	An XUnit framework that reports results in XML and HTML, including coverage information	Open source	1	3
	Apache-Test	A PHP implementation of Test::More	Open source	3	3
	Enhance PHP	An XUnit framework that includes mock and stub features	Commercial	3	2
Internet	HtmlUnit	An extension of JUnit that allows testing of HTML code	Open source	1	3
	Selenium	A record and replay framework that works for most Web browsers	Open source	3	2

*1 = low; 2 = medium; 3 = high
**1 = beginner; 2 = advanced; 3 = experts

Finally, specific test management tools are increasingly relevant for distributed software development and collaborating teams from different sites, or even different companies, or for efficiently connecting with test outsourcing providers.[1] Table 2 lists some available XUnit frameworks, most of which are free or open source. Table 3 lists some of the best-known capture and replay tools; you can find more information at www.testingfaqs.org/t-gui.html. Table 4 lists some tools to automate test data combination; find more at www.pairwise.org. Table 5 lists specific tools for test project management; these are tools that provide support throughout the whole testing project.

TABLE 3

Capture and replay tools.

Tool	Technology	Description	License	Maturity level*	Expertise level**
TestComplete	Multilanguage/ multitechnology tool	Runs on Windows	Commercial	3	3
Abbot	Java	Designed for Java interfaces	Open source	1	2
Jacareto	Java	Doesn't generate scripts; can edit tests via a GUI	Open source	1	3
Selenium	Web	Generates scripts that testers can modify	Open source	3	2
TPTP's Automated GUI Recorder	Java	Generates scripts that testers can modify	Open source	3	3
IBM Rational Robot	Multitechnology	Designed for e-commerce, enterprise resource planning, and client/server applications	Commercial	3	3
PesterCat	Web	Generates scripts in XML	Commercial	2	2

*1 = low; 2 = medium; 3 = high
**1 = beginner; 2 = advanced; 3 = experts

TABLE 4

Combinatorial test data tools.

Tool	Description	License	Maturity level*	Expertise level**
TestCover	An online service of combinatorial testing	Commercial	2	3
AETG	A Web-based application that combines test data	Commercial	3	2
CTEXL	A GUI-based application that combines values	Free	1	3
Intelligent Test Case Handler	An Eclipse-based application for generating test values combinations	Commercial	3	3
AllPairs	A command-line application	Free	1	1
CombTestWeb	A Web-based application that implements several combination algorithms, as well as tests from state machine specifications	Free	2	1

*1 = low; 2 = medium; 3 = high
**1 = beginner; 2 = advanced; 3 = experts

An Example of Test Automation

Suppose you have a single Web application that implements the Myers classic triangle-type problem: it consists of a class with three methods accepting three integer values and determines whether such a combination represents a triangle, and, if so, its type—equilateral, isosceles, or scalene. The system structure consists of a single webpage that determines the triangle type using a business class called "triangle" (see Figure 2).

First, the tester should perform a unit test to the triangle class. Test cases will

- create an instance,
- call the **setI**, **setJ**, and **setK** methods to assign values,
- call **calculateType**, and
- read the type returned by **getType**.

Good test values should exercise all possible paths in the triangle class. Numbers from –1 to 7 are good candidates. A pairwise algorithm, such as AETG, gets 64 good test cases that don't find errors in the implementation. One of these unit test cases appears on the left side of Figure 3; the corresponding functional test case is on its right side.

TABLE 5

Test management tools.

Tool	Description	License	Maturity level*	Expertise level**
Enterprise Tester	Can save time in test setup tasks through the reuse of scripts	Commercial	3	2
HP Quality Center	A Web-based application that includes requirement, test, and business process management	Commercial	3	3
IBM Rational Quality Manager	A collaborative application that supports managing the complete software life cycle	Commercial	3	3
Team Foundation Server	A collaborative application that supports source control, data collection, reporting, and project tracking	Commercial	3	3
Test Link	A Web-based application for managing tests	Open source	2	2
Squash Test Management	A test repository manager to manage requirements, test cases, and campaign execution in multiproject contexts	Open source	3	3
XStudio	A modular solution to manage the complete software life cycle	Commercial	3	3

*1 = low; 2 = medium; 3 = high
**1 = beginner; 2 = advanced; 3 = experts

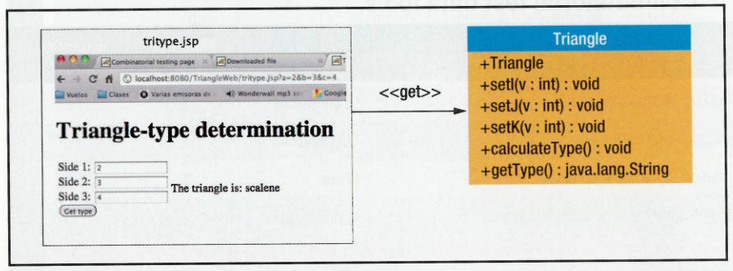


FIGURE 2. The triangle system.

The source code (including the `assert` statements) in Figure 3 has been generated with CombTestWeb (http://alarcosj.esi.uclm.es/CombTestWeb), a powerful tool for test case generation with oracles. Both types of test cases can be executed with additional tools, such as Eclemma, to determine the coverage reached in the system under test. Figure 4b highlights in red the statements that test cases haven't covered; note that no test cases have been produced for the equilateral and isosceles types. The coverage analysis helps the tester to enrich the test suite. The tester can identify areas of the program that haven't been exercised by the tests and, hence, can design new test cases to exercise those areas.

Now it's your turn to apply these principles. We can offer some practical guidance on test automation from our industry and research perspective:

- Carefully select your test tools for scalability and automation because you can't change them for each project, and over time, they bind a lot of capital.
- Always set up clear, measurable quality targets and entry and exit conditions for each test activity.
- Measure and continually improve test efficiency (that is, the effort to detect defects) and test effectiveness (that is, the share of relevant defects found).
- Make your test cases as specific as possible for concrete features, combinations of features, or attack scenarios.
- Carefully design the oracle of the automated tests.
- Avoid contamination of test cases, lest the results of a test case influence others.
- Make an adequate load distribution during execution.
- Maintain the test suite's integrity.

We also have some additional tips specifically for distributed testing:

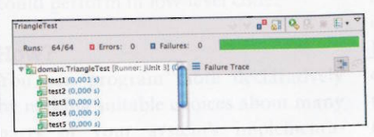
- Maintain control on the execution, including access to the global results from a centralized environment.
- Control test cases' sequence and synchronization, especially when different steps are executed in different nodes.
- Control shared data and variables.
- Align different logs coming from different test teams.

The most important thing to remember is that testing exists to find defects. It must be optimized to balance quality needs with cost. First look to test methodology, then to test automation.

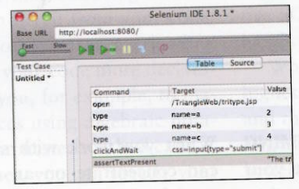

```
public void test1() {
  Triangle t=new Triangle();
  t.setI(-1);
  t.setJ(-1);
  t.setK(-1);
  t.calculateType();
  String result=t.getType();

  assertTrue(result.equals("not a triangle"));
}
```

```
public void testTriangleWeb1() throws Exception {
  selenium.open("/TriangleWeb/tritype.jsp");
  selenium.type("name=a", "-1");
  selenium.type("name=b", "-1");
  selenium.type("name=c", "-1");
  selenium.click("css=input[type=\"submit\"]");
  selenium.waitForPageToLoad("30000");
  assertTrue(selenium.isTextPresent("not a triangle"));
}
```

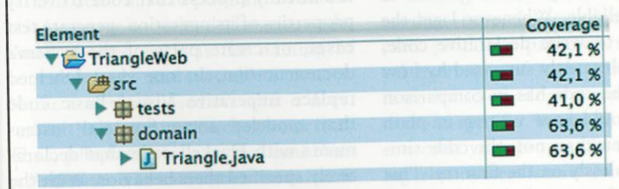


(a)



(b)

FIGURE 3. An example of two types of test cases. (a) A unit test case that tests a single class. (b) A functional test case that tests the complete system.



(a)

```
public String getType() {
    if (this.type==EQUILATERAL)
        return "equilateral";
    if (this.type==ISOSCELES)
        return "isosceles";
    if (this.type==SCALENE)
        return "scalene";
    return "not a triangle";
}
```

(b)

FIGURE 4. (a) A coverage report and (b) covered and uncovered statements.

References

1. C. Ebert, *Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing*, John Wiley & Sons, 2012.
2. P. Ammann and J. Offutt, *Introduction to Software Testing*, Cambridge Univ. Press, 2008.
3. T. Koomen et al., *TMAP Next for Result-Driven Testing*, UTN Publishers, 2006.
4. M. Polo, M. Piattini, and S. Tendero, "Integrating Techniques and Tools for Testing Automation," *Software Testing, Verification and Reliability*, vol. 17, no. 1, 2007, pp. 3–39.

MACARIO POLO is a professor of computer science in the Department of Information Systems and Technologies at the University of Castilla-La Mancha. Polo received a PhD in computer science from the University of Castilla-La Mancha. Contact him at macario.polo@uclm.es.

PEDRO REALES is a PhD student at the University of Castilla La-Mancha. Reales received an MSc in computer science from the University of Castilla-La Mancha. Contact him at pedro.reales@uclm.es.

MARIO PIATTINI is director of the Institute of Information Systems and Technologies at the University of Castilla La-Mancha. Piattini received a PhD in computer science from Madrid Technical University. Contact him at mario.piattini@uclm.es.

CHRISTOF EBERT is managing director at Vector Consulting Services. He's a senior member of IEEE and the editor of the Software Technology department for *IEEE Software*. Contact him at christof.ebert@vector.com.