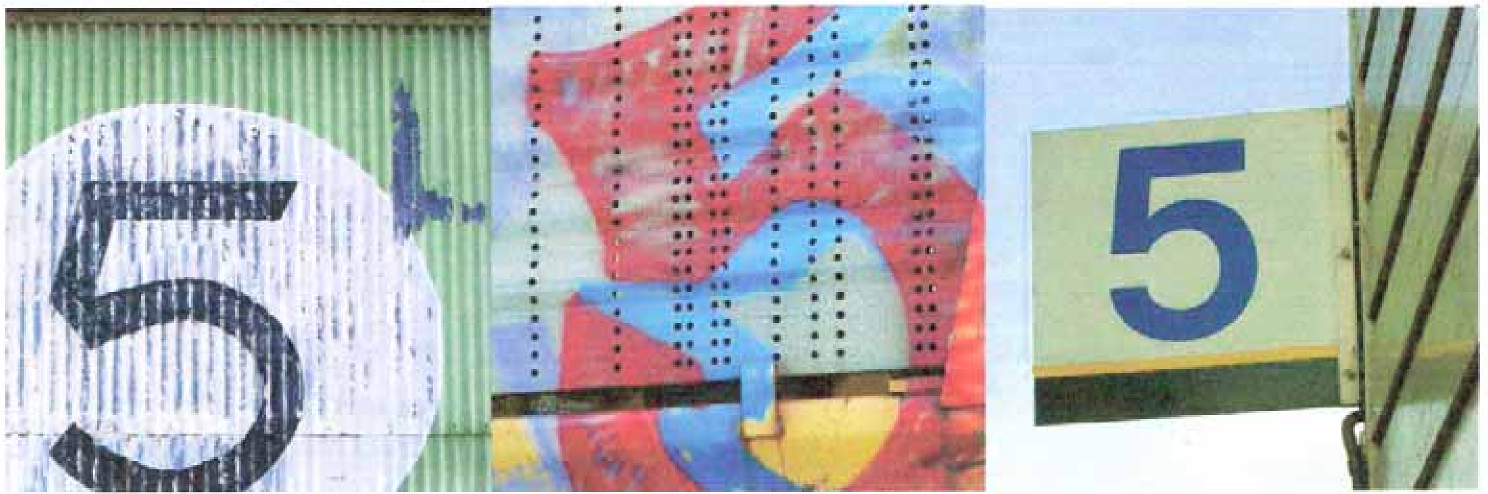


The 5%

By Hans Buwalda

With good test automation, you can have more tests and execute them every time you need to, thus significantly improving the development cycles in your project.

However, test automation is hard to get right. In particular, if you want a high percentage of your tests automated—that is, a high percentage of automation coverage—you may have to invest a lot of effort and costs, and it might take a long time to get there. This gets even worse when changes to the system under test (which always seem to come unexpectedly) make you revisit and revise part or all of your automation.

The testing team can end up spending more time on automation than on testing, and as a result may produce fewer test cases, thus negating a prime potential benefit of automation. And when updating the automated tests takes a long time, you'll also lose another major benefit: instead of shorter test cycles, you might end up with longer ones.

To enhance and focus automation efforts, I've formulated two "5 percent" goals for test automation. Although the 5 percent figure shouldn't be taken too literally, I believe these goals provide a good uphill target to aim your automation efforts:

- No more than 5 percent of structured test cases should be tested manually
- No more than 5 percent of testing efforts should be spent in the automation process

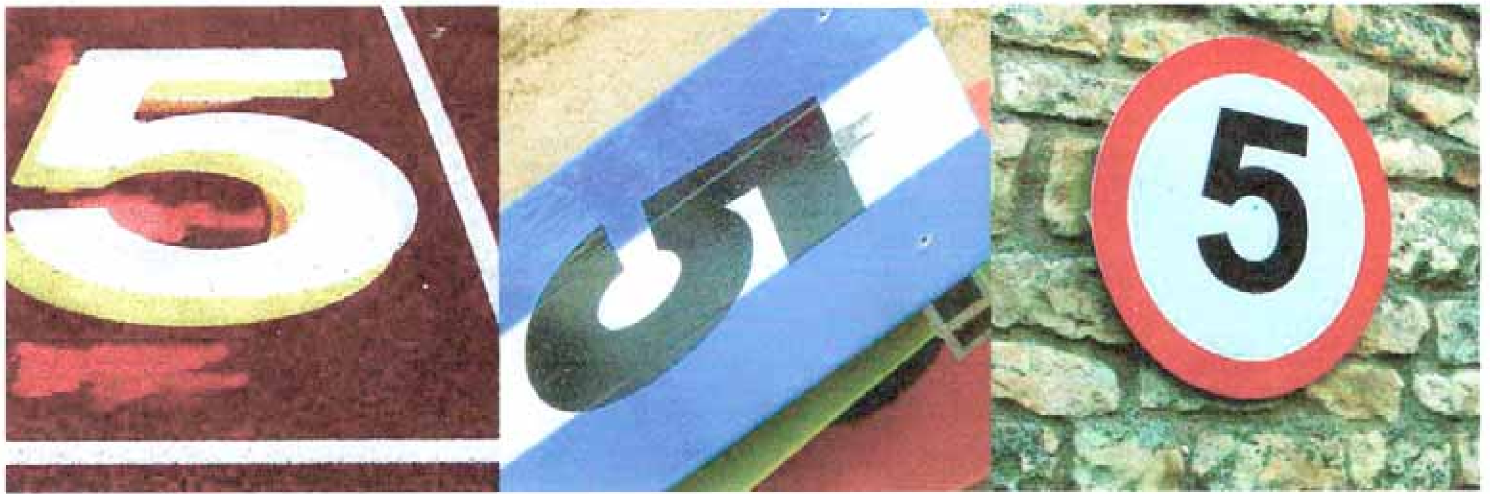
In other words: Aim to have at least 95 percent of your tests automated, and 95 percent of your testers' time spent on test development, not automation.

The first goal ensures that the automation effort has a good payoff. The costs and efforts of introducing automation are usually significant, and the way to cover them is to have adequate automation coverage—a high percentage of the test cases automated.

The second goal is designed to protect testers so they can spend more time in creating more and better test cases. Most test cases I'm seeing in projects are shallow, addressing merely the basic functionalities of the system under test. If testers get more time, they can start writing more elaborate cases, using their imagination (a highly underestimated tool of our trade) and/or testing techniques such as decision tables or soap opera testing.

The two goals are particularly tough in combination. Of course, it's always possible to get more automation coverage by investing more resources, but the second goal prevents that. Conversely, it's easy to decide to spend less than 5 percent of testing resources

Hans Buwalda, coauthor of "Integrated Test Design and Automation" (Addison-Wesley, 2001), leads LogiGear's action-based testing (ABT) research and development.



Solution

Setting the Right Goals Is One Way To Make Sure That You Aren't Spending More On Automation Than On Testing

on automation, but then Goal #1 might not be achieved.

Main Models for Test Automation

Before we examine the feasibility of meeting the 5 Percent Goals, here's a short introduction to test automation. Test automation (which for the purposes of this article means automation of the test execution) comes in numerous variations and hybrids, but here I'll focus on the three main models.

Record and playback: The first, *record and playback*, is probably the best-known, serving as a highlight of sales presentations for testing tools. In this model, user interaction with the system under test, such as keyboard input and mouse movement, is recorded in the form of scripts, which are in fact generated program code. The goal is to play back the interaction on subsequent versions of the system to see if it still works the same way. The user can pause the recording to define checkpoints, keeping status or controls onscreen for comparison when the script plays back.

The record and playback method allows you to learn how a tool represents your interaction with the system under test. Most modern tools work "context sensitively," meaning they understand the classes of the onscreen controls, such as buttons, menus and text boxes, and carry a wide variety of functions to work with them. Since there are many classes and many functions for them, record and playback allows you to see the forest despite the trees. Even more important, record and playback is a simple way to capture a long and complex interaction with the system under test by just doing it. You can then take the generated code and edit it into more robust automation; for example, by replacing the hard values you entered with variables, thus making the script reusable for multiple sets of data.

The record and playback model has its problems and pitfalls. It expects testers to deal with the automation while they're testing, and in the process generates hard-to-understand script code. When the system under test undergoes changes in its behavior, there's a good

chance that many of the recorded tests will no longer work. As I see it, record and playback will never get you even close to the 5 Percent Goals: You'll spend more than 5 percent struggling with the automation, and maintenance problems make it unlikely that any serious automation coverage can be sustained.

Scripted: The second model, usually called the *scripted* or *programmed* approach, is common in more mature testing organizations. As opposed to record and playback, in which scripts are the result of recording mouse and keyboard interactions, in a scripted approach the automation is designed as software. The tests are strung together as a program, with common interactions, such as system log-on, programmed as subroutines. In the case of a log-on subroutine, the user name and password would typically be defined as parameters to the function.

What I like about the scripted model is that the automated tests are regarded as software, not just tests. In most cases the scripts will be built and maintained by experienced automation engineers separate from the testers. This means testers can focus on designing better tests, letting the hassle of "making them work" fall into the hands of engineers who have better skills and experience in making software—which is what automated tests essentially are—work.

The scripted approach tends to be more solid than record and playback. The automation can be carefully planned and built up as a separate component from the tests, and factoring out common tasks and components increases efficiency and maintainability. With the scripted method, you must manage and maintain a substantial amount of software, the collection of automated scripts, since there will generally be a script for every test case. A carefully executed scripted approach can lead to good test coverage if enough resources are committed to it.

If you want a high percentage of your tests automated, you may have to invest a lot of effort and costs.

Action-based: The third model, *action-based* (also called *keyword-based*), is the one best suited to meet the 5 Percent Goals. In this approach, tests are written by testers using actions, such as inputting a value, clicking a button or checking a result on the screen. Tests consist of a series of *action lines* entered into spreadsheets referred to as *test modules.*

Each action line starts with an action in the A column; for example, "enter customer order." Columns B, C, D and so on are used for any data arguments the action requires, such as the customer or product name and the order quantity.

The next action line in the spreadsheet could be "check quantity in stock," with such arguments as the product name and the expected value of the amount of the product that should remain in stock after the order. In this way, test cases are built up as a series of these test lines, as in a short story that creates a test language specific to the system under test. When the test is run, the test tool interprets and executes the test lines one by one, producing a report that reveals whether the actual value of the stock is equal to the expected value.

The advantage of an action-based approach? The automation no longer automates each test case, instead focusing on the actions as generic entities that can be reused in any number of test cases as the test designer sees fit. Although the action-based model is similar to the scripted approach, using actions leads to significantly smaller amounts of automation, which is also much easier to maintain. You can compare this with a language like English: A limited set of words is used to create a virtually unlimited amount of spoken and written language.

Compared to the scripted approach, the action-based model requires more effort to introduce and manage. In particular, testers must learn to describe their tests in the form of action lines, and testers and engineers must agree on well-chosen actions that are highly reusable and easy to maintain.

All three models can be examined in far greater depth, but that's beyond the scope of this article. And they needn't be mutually exclusive: In practice, the main automation models I've described can be combined. For example, record and playback can be used to create a first version of a script that is subsequently adapted into a reusable action.

Meeting the 5 Percent Goals

With these distinctions in mind, how do we move toward meeting the 5 Percent Goals? In my view (which is admittedly biased, since I work with it in most of my projects), the action-based model is the best starting point,

for two major reasons:

Automation coverage is maximized since the testers create all their tests in the action-based format, and it suffices for the automation to cover the limited set of actions to get full automation coverage of virtually all the tests.

Because test designers provide the test "flow," the sequence of actions in the right order and with the proper data arguments, and the amount of actions is limited, the automation efforts are fairly limited, too.

However helpful, carefully applying the action-based method (or any of the other models) is not enough to meet the 5 Percent Goals. In my own work, I've identified three further factors to make test automation successful: test design, automation architecture and organization of the process, with the right team, plan and stakeholder involvement, including subject-matter experts, developers, auditors and managers.

Test Design: Good test design is crucial, even more important than the technical automation effort itself. "Good" test design means not only effective tests, but efficient tests, created in a way suitable for automation.

How do we achieve a good test design that can lead to efficient automation? Following the action-based approach, the major step in a successful test design is to break down the test into test modules, which in actions will eventually become spreadsheets with test lines. Each module should have a clear and focused scope. For example, a module that tests the accuracy of a financial system's interest-rate calculation process should not also test whether the boundaries of an input field are correct or whether the OK button is disabled if not all fields are populated.

Accurate organization of the tests into modules with a clear scope can save you headaches in the successive automation in many situations; for example, when functional tests are run only when lower-level tests have established that the UI is working correctly and that the automation of the functional tests won't encounter unnecessary problems.

The module breakdown should produce a list of modules with information, including:

- name and scope of the module
- relevant stakeholders
- when the module will be developed and reviewed
- when the module will be automated
- when the module needs to run against the system under test

Another important step, which contributes significantly to automation efficiency, is selecting which actions to use and which of their arguments to specify. Make sure to use actions that are adequate for what needs to be tested. For example, "low"-level actions like "click button" should appear only in tests that are actually testing a button. Otherwise, embed them into "high"-level actions like "enter customer," which enters all data for one customer in the dialog or dialogs for that of the system under test.

A major factor for automation success is the quality and structure of the team.

Automation Architecture: Several models for automation are available, but it's not just the model itself; it's the way the model is applied that makes a major difference.

First, you must decide what technology to use to get to the system under test. This choice can substantially influence the difficulty of maintenance for the automation, and also what you will be able to do with it. The technical aspect of test automation is enough to fill another article (or book), but here are some notes from my own experience that I feel support the 5 Percent Goals:

The action-based model, with its

multiple levels of actions, allows you to focus the programming on the elementary actions. This means that great effort can be made to find and apply the best solution you can find, since it is only "done once" and reused at all the higher levels of the actions. It is in fact this capability of automation "to go where no man has gone before" that gives it an edge over manual testing.

Don't hesitate to mix technologies. Typically in projects, you'll have UI testing, but also API calls, TCP/IP messages, parsing of text files and SQL questions. As long as you organize it well, there's no need to hold back at this point.

While you're building the automation, focus on stable, durable solutions. A good example is synchronization, a major bother in many automation projects: The automation must wait long enough for the system under test to finish operations such as displaying a window or enabling an OK button when all mandatory fields have been populated. The simplest method of timing is to put in a fixed wait, but the reaction times of a system under test aren't always the same, and the wait can either be too short, thus stopping the automation, or

too long, slowing it down unnecessary when applied often. Therefore, always find ways to actively detect when a system under test is ready for the next event. This will greatly help to increase stability of the automation, reducing maintenance—and irritation. Fortunately, virtually every tool on the market has good facilities to help you with this.

A major factor influencing the automation maintenance efforts, and the second 5 Percent Goal, is the identification of elements in a graphical user interface, like a dialog or a Web page. In your automation, you'll typically refer to such elements (and the windows containing them) by logical names, like *OK* for an OK button. The automation then needs to map those to the actual elements available onscreen. To do this, the tool typically registers one or more properties that uniquely identify that control. Most of you who work with automation tools will be familiar with this process; for example, the OK button might have a caption *OK*. The trick for you as an automation engineer is to choose the properties to use carefully, picking those least likely to change between builds/versions of the system under test. Captions of buttons or labels of text boxes are good candidates. Even better is to have an internal ID attached to a screen element by the developers. In particular for Web pages, this is easy to do (using the ID property); for Windows applications, the accessible name could work well. However, you may have to negotiate with the development organization to ensure that the use of an internal identifying property is a standard practice. Doing so will greatly enhance the stability of your automation over versions of the system under test, bringing you much closer to meeting the 5 Percent Goal #2.

Organization: The third and final factor in automation success according to the 5 Percent Goals is the organization around your testing and automation efforts: the add-up of people, procedures, practices and other elements that keep your project going. This topic alone is worth far greater attention, but let's review just a few essential elements.

Make sure you get good input and feedback from your stakeholders on your automation. These are typically end users, auditors, infrastructure people, managers, developers and other folks with an interest in the product under test. They can help greatly in getting the tests right, thus avoiding rework in the automation.

Stakeholders can also help in specific areas. As noted previously, developers can use IDs for fields so your automation can detect them. This kind of topic is sometimes called a system's *testability*. Other examples include special class properties to expose otherwise elusive information such as the number of pages in the cache of an Internet server, or infrastructure workers, who can make it easier—or harder—for you to create and manipulate test environments.

A major factor for automation success is the quality and structure of the team. Particularly paramount is the division of labor between test development and automation engineering. This division of roles, a cornerstone of action-based testing, is based on the observation that testers and engineers have very different skills and interests. In general, testers are good at creating tests and finding bugs; they have little affinity with automation.

Also make sure to have best practices in place, like those mentioned earlier, and actively follow and assess the way tests are designed and automation is performed.

Practical Experience

In my experience with the action-based model, meeting the first 5 Percent Goal has always been achievable, for the reasons I've laid out, most importantly that action-based testing automation focuses on actions, not tests. If all tests are written with actions and all actions are automated (which they usually are), all tests are automated.

The second 5 Percent Goal is a tougher challenge. It's possible to get close to the mark after an initial period of setting up an infrastructure and team, and letting them go through a learning curve. After that initial period, the complexity of the UI of the system under test and the frequency and depth of its maintenance cycle are the most significant factors in determining the ongoing degree of automation efforts. However, with good practices and good test and automation design, reducing the automation effort below 5 percent is quite achievable.

You should measure how near the 5 Percent Goals you can reach:

- How many of the test cases are currently automated? If not 95 percent, why?
- How much effort is involved in the automation, and is it reasonable? If the effort is substantially higher than 5 percent, why? And can improvements be made?

The 5 Percent Balancing Act

When you're planning your automation effort, the 5 Percent Goals can offer a clear, though challenging direction. This balance may be difficult, but it will increase your chances of a decent payoff. Just as important, it will free your testers to spend their time more effectively on creative, rather than rote work, ultimately designing more and better test cases.

When you're planning to automate your testing process to any degree, you have many approaches to consider. Of the major test automation models in use in the industry—record and playback, scripting and action-based—I believe the latter model is the most likely candidate to meet the 5 Percent Goals, if based on an effective test design, automation architecture and organization.

Although the ideas presented here are based primarily on the action-based model, the other methods for automation can provide capable alternatives. In testing, regardless of model, it's most important to think things through, aggressively identifying and applying lessons and best practices to achieve a comfortable and effective long-term automation solution. ▣