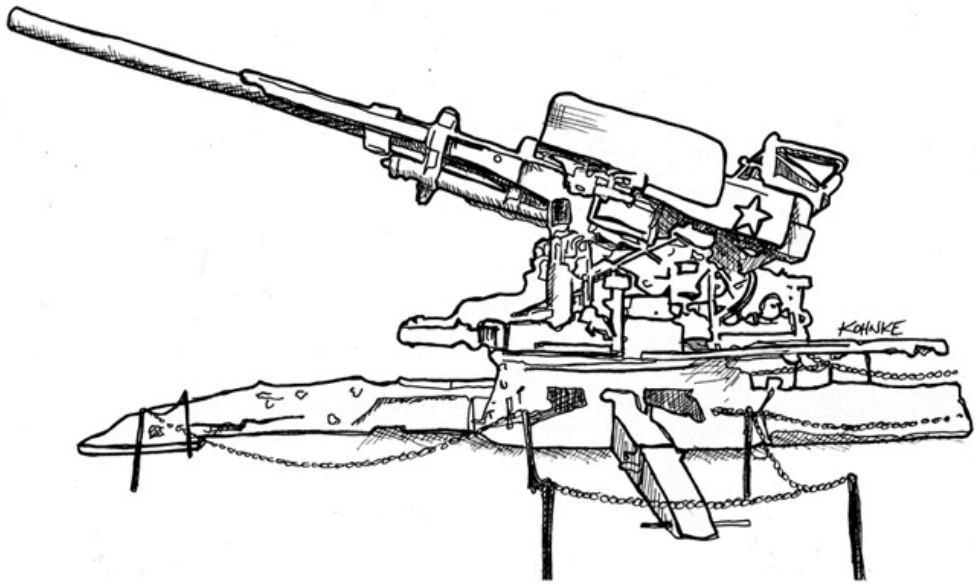
# 16

## Acceptance Testing




Of all the disciplines of clean code, this is the one that programmers have the least control over. Fulfilling this discipline requires the participation of the business. Unfortunately, many businesses have, so far, proven unwilling to properly engage.

How do you know when a system is ready to deploy? Organizations around the world frequently make this decision by engaging a QA department or group to “bless” the deployment. Typically, this means that the QA folks run a rather large battery of manual tests that walk through the various behaviors of the system until they are convinced that the system behaves as specified. When those tests “pass,” the system may be deployed.

This means that the true requirements of the system are those tests. It does not matter what the requirements document says, it is only the tests that matter. If QA signs off after running their tests, the system is deployed. Therefore, it is those tests that are the requirements.

## The Acceptance Testing Discipline

The discipline of acceptance testing recognizes this simple fact and recommends that all requirements be specified *as tests*. Those tests should be written by business analysts (BA) and QA on a feature-by-feature basis, shortly before each feature is implemented. QA is not responsible for running those tests. Rather, that task is left to the programmers; and therefore, the programmers will very likely automate those tests.

No programmer in their right mind wants to manually test the system over and over again. Programmers automate things. Thus, if the programmers are responsible for running the tests, the programmers *will* automate those tests.

However, since BA and QA author the tests, the programmers must be able to prove to BA and QA that the automation actually performs the tests that were authored. Therefore, the language in which the tests are automated must be a language that BA and QA understand. Indeed, BA and QA ought to be able to *write* the tests in that automation language.

There are several tools that have been invented over the years to help with this problem: FitNesse,[1] JBehave, SpecFlow, Cucumber, and others. But tools are not really the issue. The specification of software behavior is always a simple function of specifying input data, the action to perform, and the expected output data. This is the well-known AAA[2] pattern of Arrange/Act/Assert, or the Given-When-Then discipline of behavior-driven development (BDD).

---

1. https://fitnesse.org/

---

2. Credit: Bill Wake.

All tests begin by arranging the input data for the test. Then, the test causes the tested action to be performed. Finally, the test asserts that the output data from that action matches the expectation.

These three elements can be specified in a variety of different ways; but the easiest approach is a simple tabular format.

The following figure, for example, is a portion of one of the acceptance tests within the FitNesse tool. FitNesse is a wiki, and this test checks that the various markup gestures are properly translated into HTML. The action to be performed is "`widget should render`," the input data is the "`wiki text`," and the output is the "`html text`."

| widget should render | | |
|---|---|---|
| wiki text | html text | |
| normal text | normal text | |
| this is "italic" text | this is italic text | italic widget |
| this is '"bold"' text | this is bold text | bold widget |
| !c This is centered text | This is centered text | |
| !1 header | <h1>header</h1> | |
| !2 header | <h2>header</h2> | |
| !3 header | <h3>header</h3> | |
| !4 header | <h4>header</h4> | |
| !5 header | <h5>header</h5> | |
| !6 header | <h6>header</h6> | |
| http://files/x | <a href="files/x">http://files/x</a> | file link |
| http://fitnesse.org | <a href="http://fitnesse.org">http://fitnesse.org</a> | http link |
| SomePage | SomePage<a title="create page" href="SomePage\?edit&nonExistent=true">\|\?\]</a> | missing wiki word |

Following is an example of the Given-When-Then style:

```
Given a page with the wiki text: !1 header
When that page is rendered.
Then the page will contain: <h1>header</h1>
```

It should be clear that these formalisms, whether they are written in an acceptance testing tool or whether they are written in a simple spreadsheet or text editor, are relatively easy to automate.

### The Discipline

In the strictest form of the discipline, the acceptance tests are written by BA and QA. BA focuses on the happy path scenarios, while QA focuses on exploring the myriad of ways that the system can fail.

These tests are written at the same time, or just before, the features they test are developed. In an Agile project, divided up into sprints or iterations, the

tests are written during the first few days of the sprint. They should all pass by the end of the sprint.

BA and QA provide the programmers with these tests, and the programmers automate them in a manner that keeps BA and QA engaged.

These tests become *the definition of done*. A feature is not complete until all its acceptance tests pass. And when all the acceptance tests pass, the feature is done.

This, of course, puts a huge responsibility on BA and QA. The tests that they write must be full specifications of the features being tested. The suite of acceptance tests *is* the requirements document for the entire system. By writing those tests, BA and QA are certifying that when they pass, the specified features are done and working.

In some cases, BA and QA may not be accustomed to writing such formal and detailed documents. In that case, the programmers may wish to write the acceptance tests with guidance from BA and QA. The intermediate goal is to create tests that BA and QA can read and bless. The ultimate goal is to get BA and QA comfortable enough to write the tests.

### The Continuous Build

Once an acceptance test passes, it goes into the suite of tests that is run during the continuous build.

The continuous build is an automated procedure that runs every time[3] a programmer checks code into the source code control system. This procedure builds the system from source and then runs the suite of automated programmer unit tests and the suite of automated acceptance tests. The results of that run are visibly posted, often in an email to every programmer and interested party. The state of the continuous build is something everyone should be continuously aware of.

---

3. Within a few minutes.

The continuous running of all these tests ensures that subsequent changes to the system do not break working features. If a previously passing acceptance test fails during the continuous build, the team must immediately respond and repair it before making any other changes. Allowing failures to accumulate in the continuous build is suicidal.

## Conclusion

Acceptance testing is one of the weakest aspects of software development in the first half of the twenty-first century. This must change. The businesses and the developers must learn to collaborate to create a language and a procedure that allow requirements to be formally specified by BA and QA and that allow programmers to quickly verify that those requirements have been satisfied. The advent of AI makes this need doubly important.