DESIGN OF SOFTWARE TESTS

2018-12 --- 2019-02-08 J.S.

OVERVIEW

1. Project Background and Description

*The primary purpose of the tests is to **verify** the functionality of the applications.*

*However, tests are also a good helper for future edits - for example, it can be **verified** that what worked before is still ok(=e.g. after some refactoring or bug fixing). In addition, the tests can serve as application **documentation**, for training or even - which is even ideal - as a **basis** for programming/implementation itself.*

2. Project Scope

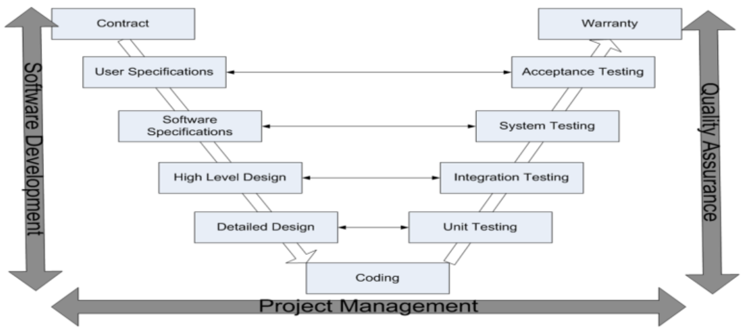
*There are a number of tests. First, we'll give you an **overview** of possible approaches - from different perspectives. Then we come to the description of how to grasp the thing.*

Reliability, Availability:	Validated via a stress test.
Scalability:	Validated via a load test.
Usability:	Validated via an inspection and demonstration to the customer. Is the UI configured to their liking? Did we put the customer branding in all the right places? Do we have all the fields/screens they asked for?
Security (aka, Securability, just to fit in):	Validated via demonstration. Sometimes a customer will hire an outside firm to do a security audit and/or intrusion testing.
Maintainability:	Validated via demonstration of how we will deliver software updates/patches.
Configurability:	Validated via demonstration of how the customer can modify the system to suit their needs.

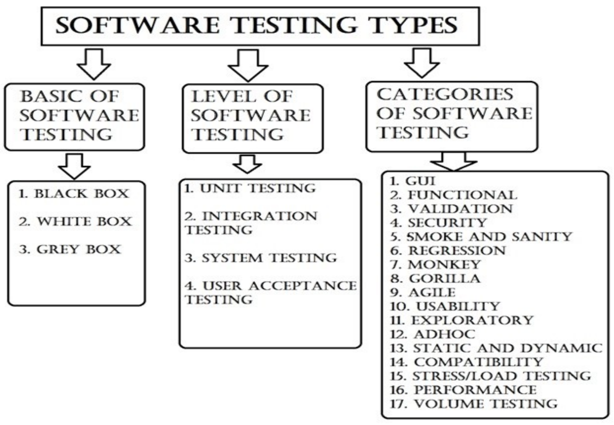
source

3. High-Level Requirements

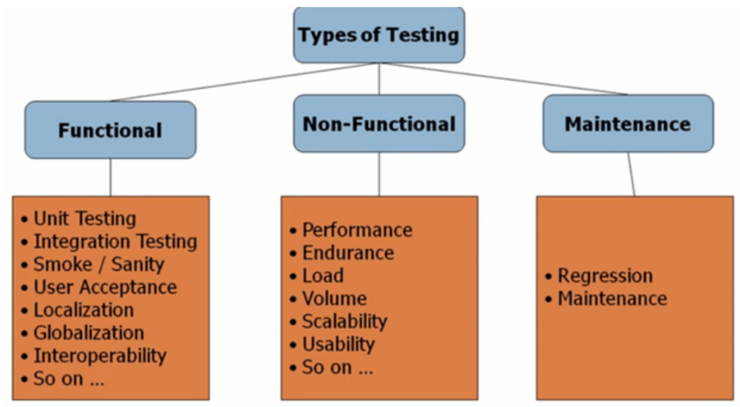
*First of all, again, we will introduce the top-view issues, where we can see **above-mentioned** terms, but now from another perspective. We will still see other possible views - it is important for the final understanding that we do not have to worry about a number of concepts and fear complications. Conversely, the purpose of these "alternative" views is simply to see that it is still the same paradigm - that the requirements, documentation, tests, and implementation(= application code itself) is just a rewriting of the same story, each time in a different language. But what is essential to us is that the story is **the same**. And that's exactly what this storry is all about. And if we do not have this story **at all**, we will probably have a problem. That is long-term experience from the world. And the poet just wants to say that the test was not owed to us because of some story. This is good to realize – some test did not bring any storry, some test can only reuse the story. In other words, it's free and everybody understands it. Such things do not happen everyday - in the world of IT.*



source



source



source

4. Implementation Plan

It is good to get used to tests and make them your friends and helpers. And as we mentioned in point 1 / these tests may be helpers from the very beginning. It is also good to ask yourself how much to test how far to go. And it's best to turn this question around - what happens when we do not test? But we should to imagine the quality of such software. The next step is what are the minimal tests. Essentially, enough music for the least money. It offers: Why not directly test our story?

Let's look at the practical example:

Specification	Documentation	DSL test

A **Name** and **Password** will be required to enter the application.	When someone starts the application, the logon page appears. The user enters the **Name** and **Password** and clicks on the **Sign In** button.	
		```
		Go to     loginPage
		Fill out  Email        johny.test@verify.com
		fill out  Password     paswwordFor-johny.test@verify.com
		clik      Sign in
		```

We see that our story is viewed differently in different stages of development: one is desire, another is a statement. And in our tests it will be verified the Condition that after pressing the button application will follow in right way. The condition (= wish) became a business rule. But it's still the same story. It is clear that we are all on the same ship. If we wanted to program the "classic" test instead of the DSL test, we did the same as with regular programming. Programmer rewrites test entry to low level language. The major drawback is that no one else - except for IT - understands this test. Plus, each change requires a complete round-trip. It's time consuming and costly. On the other hand, DSL is understandable to everyone. DSL is free - it's the language of the company, you can find it in every company. It's just that, on the one hand, you can find industry specialists (= domain) in each industry and IT professionals on the other. DSL is one of the few ways to connect these worlds. DSL is nothing other than using normal language instead of (very) simplified technical languages. It's nothing new. We have been long time talking about formalities or requirements, for example, if we are writing something legal, business, contract, etc. We can use the words formalities or requisites instead a world-`specific term specific. And instead of domain, we can use the word field. So, instead of domain specific, we can talk about business formalities or business disciplines. And our **name** or **password** are definitely such formalities..

You may have noticed grammatical peculiarities in our DSL test. This is another (from many other) benefits of DSL. You do not have to deal with the size of letters, accents, synonyms and even internationalism. That's how it goes in the jargon. Be user friendly.

5. Implementation Plan - Reloaded

The test request is more or less the requirement that was (or should be) already in the assignment. The test only verifies this request, the assignment can also affect the quality of the realization. The test can be seen as an "extra cost", the assignment is always "free of charge". Tests are needed today as an indispensable part of production, because the complexity of the field simply exceeds human abilities. Testing may not be necessary, but then it can not be ruled out the „tests on customers" And that's more expensive. That's all, so the "historical" view of "extra cost" needs to be revised a bit. And it's just a special example specification specification (= which is general) with specific test data (= which are exemplary). Because data is large, tests do not cover everything. But to (practically) it does not matter. Before you enter the Nuclear Power Plant, you will also be testing. It is not possible to test everything - but somehow it works ... So the tests should cover the "enough" area. And do this coverage (= primary decision) without specification (assignment) ...

Let's look at the practical example:

One of the main things to do (except business vision, goals, intent ...) is business logic (= we will call it BL).

And to BL belongs „business rules".

The main principle of „business rules" is:

WHEN → THEN

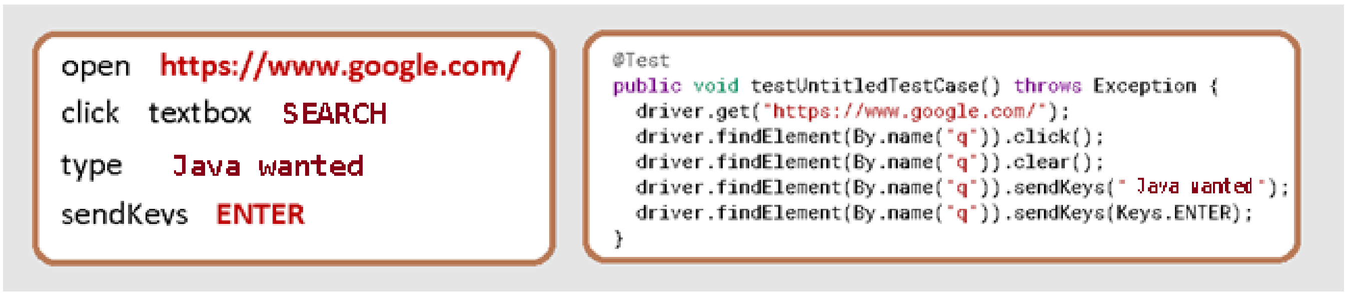
I.e. that something is happening under some assumption (situation, time, quantity etc ...)

In this view, the test principle is:

GIVEN → WHEN → THEN

I.e. we check that for the given example (= GIVEN) the rule just applies.

The manner in which specifications should be formulated should be "human friendly". There is no need to formulate these things low-level in the programming language. This way (and often "nasty") arises from the need for a formal approach. It is already necessary to say **clearly** how things should work. However, the programming language is not intelligible, not even for the IT workers themselves. And the others know what we're talking about. The specification or assignment is something that should be comprehensible to everyone, not just to someone, and to the source code. Not only it is expensive, but the "special" person is not always available, after a while he's working on something else, and with time, it's also a problem for him to read the (own) code. What about future code modifications? Everywhere is called "agile". What steps other programmers will have to do to join such project? Which of these codes are legible? And how much will it cost? Compare the following two programs:

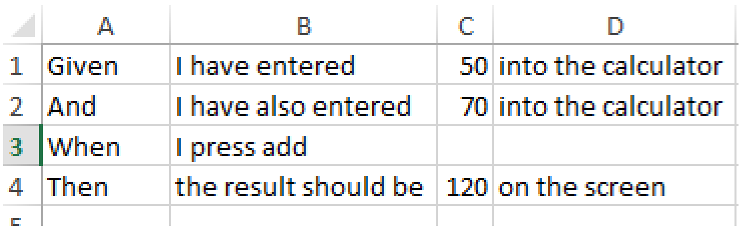


```
open    https://www.google.com/
click   textbox  SEARCH
type    Java wanted
sendKevs  ENTER
```

```
@Test
public void testUntitledTestCase() throws Exception {
  driver.get("https://www.google.com/");
  driver.findElement(By.name("q")).click();
  driver.findElement(By.name("q")).clear();
  driver.findElement(By.name("q")).sendKeys(" Java wanted ");
  driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
}
```

What do you think is the difference above? **None**. Then it's just about what you prefer, what you will like more, and especially if you "want to do it" or you need to see or interfere with it. Or you can delegate it to someone else and he will arrange himself. Still, there will be some demands at one end, if not, so surely some expectations will. And at the other end there will be some implementation that you either understand or not.

And while "ordinary" Excel can be enough:



	A	B	C	D
1	Given	I have entered	50	into the calculator
2	And	I have also entered	70	into the calculator
3	When	I press add		
4	Then	the result should be	120	on the screen

6. Approval and Authority to Proceed

Name	Title	Date