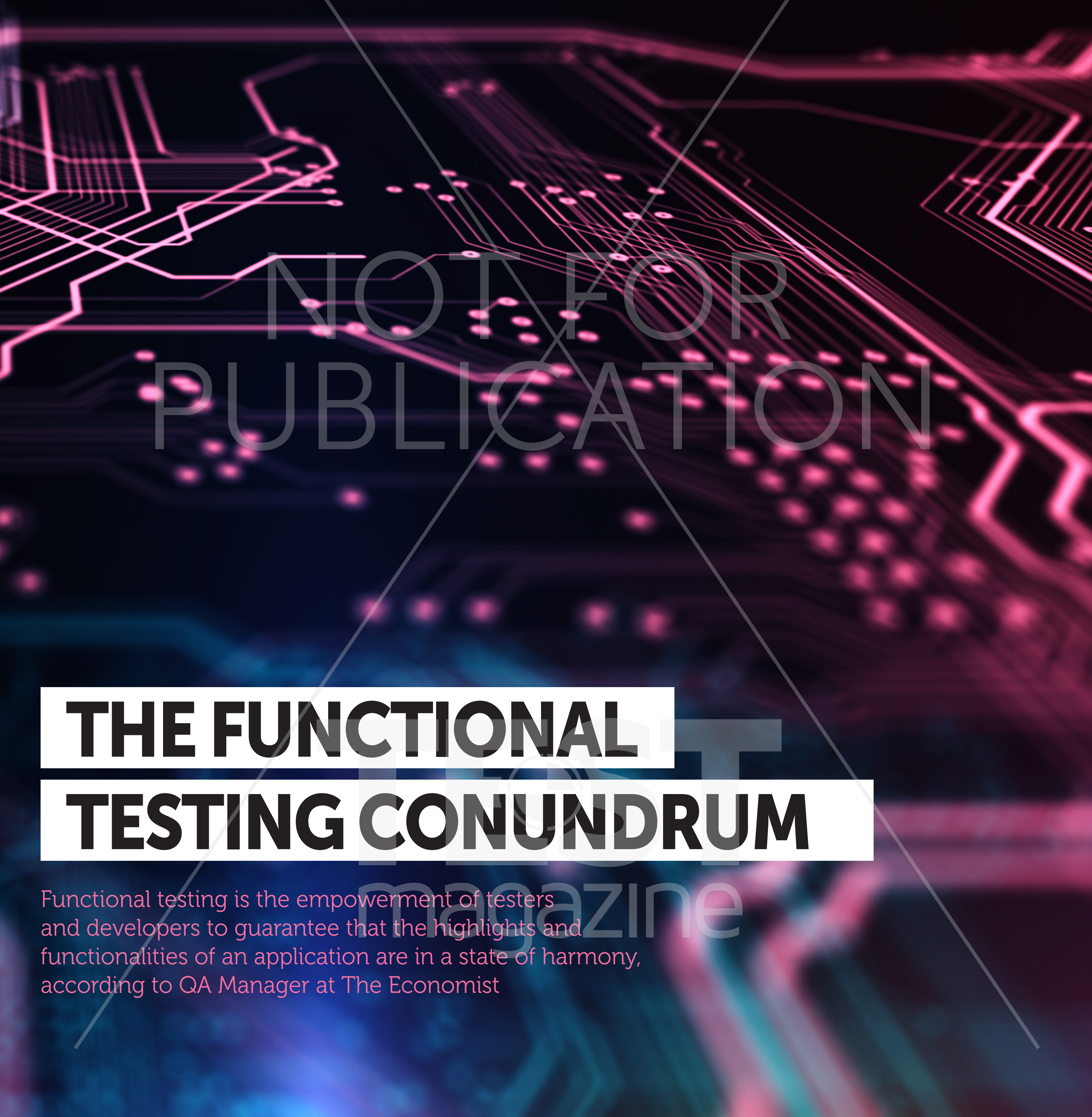


THE FUNCTIONAL TESTING CONUNDRUM

Functional testing is the empowerment of testers and developers to guarantee that the highlights and functionalities of an application are in a state of harmony, according to QA Manager at The Economist

To me, testing is more than just making sure your application works as you expect. It also functions as a form of communication across a team and application, meaning another developer can confidently begin contributing to an application, using the tests as a guide.

The greater complexity of software applications and its increasing competitiveness has created the need for more efficient and exhaustive testing processes. When you have a complex piece of software the importance of functional testing is key, it de-risks going live with a poor product which does not meet customer expectations.

EMPOWERMENT OF TESTERS AND DEVELOPERS

Functional testing (FT) is the empowerment of testers and developers to guarantee that the highlights and functionalities of an application are in a state of harmony. It answers questions like 'can the user do this' or 'does the particular feature work'. Without these, a user could be impacted in a huge way. A black box testing approach where test cases are prepared, keeping compliance with client requirements in mind. Take a new phone, for instance, if you cannot surf the internet it may be deemed worthless, FT would have picked up this flaw with consummate ease if done efficiently.

Non-functional tests look at its overall performance (e.g. testing scalability, reliability, security and compatibility).

If you asked me in the early stages of my career, functional testing would have primarily focused on manual testing and more efficient ways of working. 5 years into my career the big buzzwords were around "automation". Big advancements in the next few years will be "machine learning" and "AI".

WHAT FUNCTIONAL TESTING CAN INVOLVE

- UNIT TESTING: Focuses on a single "unit of code", such as a class method.
- INTEGRATION TESTING: Builds on the unit test by testing multiple pieces of your application together.
- SMOKE TESTING: addresses the most basic functionalities crucial to the working of the product.
- REGRESSION TESTING: Ensures that previously working functionality is still functional.
- USER ACCEPTANCE TESTING: End-users review the functioning of the product.

A key concept is to have a solid set of unit tests, issues can be found early and refactoring can take place with confidence. With FTs there is a safety net rather than the first line of defence. Unit tests should be the car engine and FT's should be the paintwork.

Because of the high scope of tests, this essentially means that there is a low level of abstraction. In an ideal situation, the tests should be defined before the development starts and after the product requirements have been defined.

From experience, tests should not be driven by software development processes but should be driven by what the user expects the function of the software to actually be.

CONDITIONS FOR A GREAT FUNCTIONAL TEST

- Derived from the product requirements.
- Clearly defined and understood by not just a QA person but also the whole team.
- Visible to stakeholders.
- Gives actionable information to developers.
- Audited frequently to ensure they are updated with product requirements.
- QA clear on the functionality of the system.
- The correct set of data.
- Test cases cover all possible scenarios.

CHALLENGES

The risks of seeing the functional test as a repeatable unit mean that they will not be adaptable and issues can be missed. The functional test should be seen as an interactive component of a QA process.

1. Definition of clear and complete test requirements.
2. Managing changes in requirements.
3. Dealing with functional gaps in test plans.
4. The difficulty faced by the development team whilst reviewing large test plans.
5. Ensuring that you have optimum test coverage.
6. Dealing with blocking issues.

TDD APPROACH TO FT

Refers to a process by which you develop and test your code. Under a Test Data Driven (TDD) paradigm, your development process looks something like this:

1. Before coding, you write a test.
2. Run the tests along with any other tests.
3. Write the minimum amount of code necessary to make the tests pass.
4. Refactor.
5. Repeat the process.

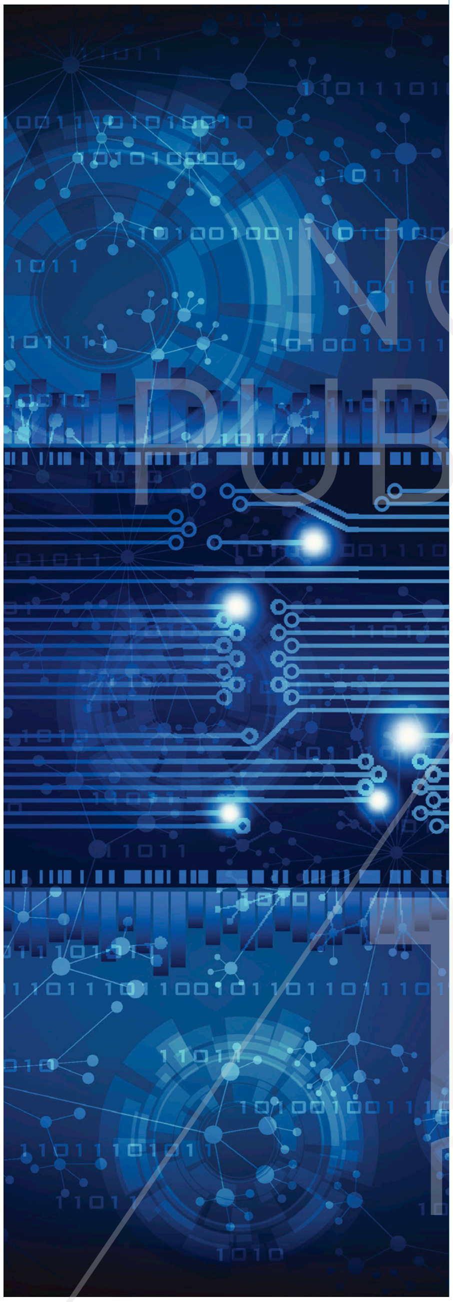
With TDD, you can get broad test coverage of your application. This gives you the confidence

The greater complexity of software applications and its increasing competiveness has created the need for more efficient and exhaustive testing processes

DILEEP MARWAY
QA MANAGER
THE ECONOMIST

Dileep has more than 10 years' experience as a Software Testing Professional and is extremely passionate about quality assurance, with his key principle being "putting customers first".

Advancements in testing processes now mean that more efficient ways of working are now available



to make changes to your application as it grows, as you can see older tests passing and trust that your changes have not regressed.

BDD / BEHAVIOUR-DRIVEN DEVELOPMENT APPROACH TO FT

Behaviour Driven Development is a process by which you can approach development and testing. BDD is more focused on how you test than when you test. With BDD, you focus your tests on behaviour, rather than implementation, ideally starting from your customer's/user's expected experience. In my view leads itself more so to functional testing.

Advancements in testing processes now mean that more efficient ways of working are now available. For instance, the use of the Cucumber tool (BDD) for automating browser tests and Gherkin for human-readable syntax. This allows you to provide a business user with detailed scenarios which outline how someone uses the system.

Also, the use of user stories early in the life cycle gives clear product expectations, making it easier to test.

e.g. User story – Companies are using the job search application and they should be able to create a job application.

FT:

- Open application
- Log in to the company
- Create a job posting

Based on my experience, a user story should have explicit acceptance criteria. Good acceptance criteria are ones that "must" resolve to either a "pass" or "fail" after observations are made.

AUTOMATION VS MANUAL TESTING

Automated test processes have helped to make processes more efficient, especially tasks such as regression testing.

These processes have meant that functional testing can be done in a quicker and efficient way, leaving manual testers to work on edge cases.

UI automation is the act of conducting specific tasks via automation. Automation can help to run tasks more quickly and do so automatically based on conditions being met.

Not all tests can be automated; we still need manual testing in a variety of cases. It makes sense to take a manual testing approach when you're dealing with legacy systems that don't easily support automation when you need to adhere to strict regulations and require documentation as a result.

Automation is more about checking or confirming that something is true than it is for testing or recognising unanticipated problems.

AGILE AND CONTINUOUS TESTING

API testing creates more reliable code. Historically, testing would take place at the graphical user interface (GUI) level. When a developer would finish their work, they would hand it off to the QA engineer. The engineers had limited time so they would test the code at the highest level - the GUI. This would cover both the frontend and the backend development.

This worked for manual testing and for the beginning of automation testing but isn't right for the age of agile and continuous testing. GUI testing is brittle and automated scripts break easily. In addition, teams can't wait for the entire system to be updated and the GUI to be ready before testing occurs.

In the age of agile, testing must take place at a lower level of abstraction (API level). API tests can be created before development is complete. This means developers can validate their code based on pre-written tests (Test Driven Development).

MACHINE LEARNING

In the future, machine learning driven test automation will be the key driver. In most companies, user interface (UI) elements are always changing, which means that test scripts fail as the UI has changed and tests are brittle. Artificial intelligence and machine learning mean that test adaptation can be done easily, this provides a workaround and the user is asked to 'accept' or 'deny' the workaround. To which the piece of test will adapt accordingly.

SUMMARY

Today's users are tech savvy, they use software in nearly every aspect of their daily lives and they have high expectations for how this software should work in order to help them go about their everyday needs better, smarter and faster. As a result, software testing has become increasingly important to and intertwined with the entire software development process.

Whether your team is using hardcore test-driven development (TDD) or simply wants to automate otherwise manual QA, FT benefits from clear, explicit user stories have little room for ambiguity.