By Marc van 't Veer

What is testing in production and why should it be performed?

Based on the experience from a project without TiP, reasons are given why the DTAP model is not suitable for testing modern public online services in the cloud. Apart from TiP being necessary, this paper also argues that TiP should be a regular test level. By comparing a test and the production environment, a view is given on how testing in production should be perceived as normal. Additional to this, a definition of TiP is given and the paper ends with questions for future research.

Introduction

Would you test differently if you could Test in Production (TiP)? Would you follow a different strategy? In a project a public web service was created without TiP. Based on the experiences gained from this project, reasons are given why a test and acceptance environment from the DTAP model (Development, Test, Acceptance, and Production environment) are no longer sufficient (Aalst, 2006), (Pol, 2000). More generally, traditional test environments are not suitable for testing modern public online applications, such as web services in the cloud (Grundy, 2012). Besides that, TiP is a necessity and it is also argued that TiP should be a regular testing level. Based on the comparison between test and production environments, a view is given on how normal it should be to test in production. Additionally, a definition of TiP is proposed and the paper ends with questions for future research.

The natural environment of the new public web service is illustrated using a test project performed at a large Dutch mobile telecoms provider. This service is an API and will be used in the future by external apps and on social platforms such as Facebook (API). An application (app) is typically a small, specialized program downloaded onto mobile devices. With the publication of the API in the cloud, multiple platforms can be supported without creating apps for them all. Using an app, and the web service behind it, end users can access their call and internet credits, address data, and invoices on their handsets in real time. The project needs to ensure the API is stable, has a high performance, is strictly secure, has a re-usable infrastructure and is easily extendable to cope with new functionality. All these requirements determine how the API will perform as a public web service for external users. Users are not only apps but also external developers. A few examples of the risks test management perceives are: unknown integrations, unknown diversity in contract data, and how this data will be presented. The API has no influence on the presentation to the end user. It is possible to use the API wrongly, whether deliberately or unintentionally. Can a tester be held responsible for defects between the API and an external app?

An end user has no clue about the number of steps between the app on his handset, the API, and other back-end systems. For him or her, everything behind the app is the cloud (forum). The end user will judge the quality of the API by the quality of the app. Without an early integration between the app and the API, all possible integration defects will become visible when the API is in production. The test strategy for the API project was "early integration test with complete infrastructure". This strategy was implemented by dogfooding during development, and by system and integration testing. The API was developed using prototypes of the app. Examples of the prototype apps are SoapUI, a web browser, a Windows taskbar app, and a development app which all use the API (dogfooding). During acceptance level, a first prototype of the external app was tested within the internal network. The integration tests were completed with a test of the process to download, install, and log into the app on a private mobile production network.

The moment the app was live and the API was used, the first incidents were reported. For example, a customer had a rate plan that contained two bundle types: calling and text messages. But for each rate plan, only one usage progress bar (only one bundle type) could be shown in the app. A second example was that it was unclear to many end users how to log into the app. Customers were using their existing login account instead of a special access code. A third example was, if the customer had too many wrong login attempts then his account would be temporarily blocked. Here the API was using the standard HTTP error codes like "401 – Unauthorized" (error codes), which is returned if a customer cannot log in. But the app wanted to show a different error message for the cases where the user was inputting a typo and when there were too many wrong attempts. Unfortunately, this difference could not be determined from error code 401. Now the standard error code is extended with specific API error codes: 401 – Unauthorized + 1102 – Invalid username or token of 401 – 1103 – Account has been temporarily locked out.

The app and the API were being fully integrated for the first time in the production environment. Just before going to production, the idea was proposed to perform a number of tests in production because only then would a representative *test* environment become available. People were surprised by the idea and were against it because *"testing in production is simply not done"* (TestNet, 2012), (Groot, 2012). Within the project and with the mobile telecoms provider it was not possible to test in production. In the end, all tests need to provide insight into the risks and find a way to reduce them. By adding TiP as a test level, the behavior and the possible risks of running in production for this service could be made visible.

In the introduction a test and production environment are mentioned, but what exactly are they and what are the differences?

Environments

A test environment is described as "an infrastructure that is needed to perform a test" in (Pol, 2000) and (Aalst, 2006). A test environment should be stable, controlled, and representative. There can be multiple types of environments with different roles. TMap and TMap-Next define a test environment as "a combination of hard and software, connections, environment data, administration tools and processes wherein tests are being executed" (Pol, 2000) and (Aalst, 2006).

During the development cycles of an application, every environment within the DTAP model is used by different consumers. White-box tests are executed on the development environment, the first application tests are performed on the test environment, and the acceptance tests are executed on the "as-it-were-production" environment. Because the usage is distributed over these different environments, development can continue without interfering with an acceptance test (environment). The maintenance organization has the responsibility to keep existing applications up and running on the production environment (Aalst, 2006). With each next level of the DTAP model, more systems are connected to each other and, thus, the environment becomes more comparable to the production environment. As a result, the applications become more robust and mature. The production environment is, thus, protected from low quality and remains stable.

When you look at the context of where the API and app need to operate, as stated in the introduction, a test environment is representative if it is **not** stable. Changes are continuously introduced, connections may drop, users are all external, the devices are very diverse, and there are connections being used that did not exist when the API was developed. The API's natural environment is a publicly controlled domain. In Blokland, 2012 and in Whittaker, 2011 an overview is given of how dynamic, complex, and diverse the environment of a modern public service is. If the goal is that a test should be executed in a representative environment, then this can only be done in production (Blokland, 2012).

A production environment is often compared to a test environment (production-environment). Production is not just an environment but the location where an application should perform "for real". Comparing a test and production environment to each another is like comparing a zoo to nature ("The wild" as referred to in the animated movie Madagascar (Madagascar)). They may have similarities but the differences are plentiful. Also, the notion of *"environment"* sounds too much like a clear ordering of the context where the application is going to be used. In production the service is running "live" and the API and app are servicing the end user. The test environment is a parallel world where changes and new projects are being developed without any impact on the live service. The environment where the service is running is critical for the overall quality. If the test and production environment are more similar, the test results say more about the quality and possible risks. Microsoft's Seth Eliot made a very clear statement when he discussed the "Google Staged Approach" (Eliot, 2012, spring STP). This approach follows four test levels and three of them are in production. The first level is the traditional "upfront tests" in a test environment; the other three levels are user scenario execution, dogfooding, and beta testing in production. Google is going fast to production because there is no added value in more upfront tests. Ten percent more upfront tests does not result in a ten per cent improvement in software quality (Whittaker, 2012), (Eliot, 2011-spring QASIG).

For some it may be "*not* done" to test in production, but what is the value of the test results if the traditional test environment is not representative? Should TiP be a normal test level?

How normal is testing in production?

If software is compared to a "normal" product or service, then it is clear that testing in production is not new. For normal products there are always tests in production. The Dutch Organization for Applied Scientific Research performs production tests for the introduction of the 4G LTE technology for fast mobile Internet in the Netherlands (TNO), (TNO-Huawei). The consumers' union also performs production tests on all kinds of products and services (consumentenbond). When making a little side step into law and liability, then there are more arguments to find why it is normal to test in production. Software is more and more a daily good of the end customer and they expect the software to be reliable. The moment an incident is found, the supplier is responsible. Levy and Bell did research on product liability and what actions can be taken to minimize, transfer, or divide the risks (Bell, 1990). With a legal action against a supplier there is the possibility to make the rights and obligations between the supplier and the customer clearer. The insight into possible risks and liability contributes to *"a positive relation between the supplier and his customers"* (Bell, 1990). By creating jurisdiction about rights and obligations, expectations are more realistic and there is a greater chance that possible problems in the software are discovered, admitted, and corrected before any loss or damage is done (Bell, 1990). The law is different on products and services. The sale of products is subject to many damage and warranty laws (Bell, 1990). These laws do not apply to the sale of a service. The website of ICTRecht (ICT and law) gives basic information for software developers (ICTRecht). This information also includes how to handle liability cases. A developer can be liable for incidents that occur in production (ICTRecht). Software will become a "normal" product and customers expect that the same laws on damage and warranty will become applicable. When looking at product quality, a product should work in production as specified.

Continued product quality control does not stop the moment a product is in production. Within the traditional V-model for software development, each next phase is validated by a test level and so the quality is monitored (Aalst, 2006). The last test level in the V-model is a production acceptance test (PAT), which is generally done by the maintenance organization. In the model there is no test level defined after the software is accepted. ISTQB added a "Maintenance and operation" phase to perform maintenance tests. Maintenance tests are also called "in-service inspection and testing'" (maintenance-testing). These tests focus on regression defects and are comparable to a yearly car checkup (General Periodic Inspection or, in Dutch, "APK"). If a car does not pass the check-up then it needs to be repaired if it is to continue on the road. If you look back at the V-model, these "in-service" tests should be a part of the model. When these tests are added to the model it becomes a circle, since the expected and the real behaviors are compared in production. If you look at the developers at Google, you see that they have much more direct contact with the end users than traditional developers, since Google follows a fundamentally different development model ("Google Staged Approach") for cloud services. Traditionally, the requirements of the end users are constantly translated between the developer and the tester because there is no direct contact between them. According to Whittaker, 2011 there will be

much more testing in production in the future and the whole development model will change.

TiP is necessary to perform good tests for modern public online services such as an API. But what is TiP exactly, and what is its definition?

Definition of testing in production

To have a better understanding of what Testing in Production is, a brief idea is first given of what TiP is and what it is not. TiP is not a replacement for any test level such as system or acceptance tests. This means that TiP should not be done because there was no more time left or because it's cheaper. The goal of TiP is not to check whether an installation was correctly performed on production. TiP is not a production acceptance test, because these tests are done *before* the software goes to production and their goal is also different. A PAT looks at whether the software is easy to maintain rather than how it is used or behaving in production (Aalst, 2006).

TiP is described as testing of software which is installed and executed on the hardware of the end user and is used for real. Wiki defines TiP as follows:

'Production testing is when you are testing a real live system, either about to go live or with live users. It is needed because having software working on the developer's computer is no guarantee that it will work in the client's installation.' (TiP)

Seth Eliot defines TiP as follows:

'Testing in production (TiP) is a set of software testing methodologies that utilizes real users and production environments in a way that both leverages the diversity of production, while mitigating risks to end users.' (Eliot, 2012)

Important elements from these definitions are that tests are done by or for *end users* on *real live systems* with the focus on its *behavior* in its diverse environment.

With the implementation of TiP testing methodologies and techniques that use the diversity of the production environment, the risks of end users can be minimized. The diversity of the production environment functions as a lever to find defects that can only be found there. When going back to the central question, *Why should we test in production?* We can see that there is not only something missing in the definitions (for which types of application is TiP applicable) but they focus too much on methodologies. Eliot is explaining that TiP is meant for (web) services (Eliot, 2011) and defining TiP as a test *type*, while the reasoning in this paper is that TiP is a test *level*. A test level is a group of test activities that are combined, executed, and managed (Aalst, 2006). A test type is a group of test activities that tests the software or a part of it on a combination of quality attributes (Aalst, 2006).

Testing in production can be compared to system testing. A system test is done by a supplier on a lab or test environment, with the goal that the developed system complies with the functional and non-functional specifications, and with the technical design (Aalst, 2006). The difference is that TiP is using the production environment and that behavior is inspected with the possible edge cases that occur naturally there. These edge cases are unthought-of and unpredictable. By using the production environment with real users, this diversity is used for testing. With system testing the focus is on expected behavior and with TiP the focus is on unexpected behavior. Within this new test level, all the test methodologies of Eliot can and should be used like synthetic tests in production or controlled test flight (Eliot, 2012).

When combining all previous points then the total definition of TiP becomes:

'Testing in production is a combined group of test activities that uses the diversity of the production environment and real end user data to test the behavior of the developed service in the live environment, so minimizing the risks for the end users.'

When TiP is introduced into an organization, the whole development model changes. Before TiP is added then it is mainly BUFT or "big up-front testing" that is executed. This means that all tests are done before the software goes live (BUFT). But, with TiP, the developers have more direct contact with the end users and the testers do not need to act like end users. The test environment becomes an internal production environment where testing is done by dogfooding and bringing your own device. When the software is stable, then testing is done in production via beta testing, for example (Whittaker, 2012).

Further research

One of the first questions to answer is: if tests are executed in production, how can this be done without influencing the continuity and stability for the real end users? Other questions are: how can regression testing or test harness be used for monitoring and what does it look like? (Riungu-Kalliosaari, 2012) When a test strategy is put forward for TiP, there should be a discussion with the operations department because the monitoring of all systems can be influenced by TiP. Operations is constantly monitoring the production environment, but when testers are testing in production they are also monitoring both with their own goal. Between test and operations a new role can be created like *"Testops"* (Eliot, 2012). The moment TiP is executed, it must be clear which strategy can be used to cover the risks. If there is a difference in responsibility between fixing defects in the test environments and incidents in the production environment, what will happen when you find a defect in production? Who is going to fix it? Who is paying for this? And when should it be fixed? There should be more research done on how TiP can be introduced into an organization and which techniques can be used to find different types of defect.

End notes

Books

- Aalst, L. van der, Baarda, R., Broekman, B., Koomen, T., Vroon, M., 2006. *TMap® Next, voor resultaatgericht testen*,Tutein Nolthenius. p. 44-51, p. 68, p. 82, p. 412, p. 419
- Bell, S. Y., Levy, L. B., 1990. Software product liability: Understanding and minimizing the risks, *Berkely Technology Law Journal*, Boalt Hall School of Law, University of California at Berkeley, California. Vol. 5, issue 1, 2–3. Available at: *http://www.law.berkeley.edu/journals/btlj/articles/vol5/Levy.pdf*
- Blokland, K., Mengerink, J., 2012. *Cloutest, Testen van cloudservices*. Tutein Nolthenius. p. 162–170

- Eliot, S., 2012. *Testing in Production A to Z, TiP Methodologies, Techniques, and Examples, Software Test Professionals*, p. 5, p. 55, p. 19, p. 20. Available at: *http://www.setheliot.com/blog/a-to-z-testing-in-production-tip-methodologies-techniques-and-examples-at-stpcon-2012/*, *http://www.softwaretestpro.com/item/5477*
- Eliot, S., 2011. *Testing in Production, Your Key to Engaging Customers*, Quality Assurance Special Interest Group, p. 19–20. Available at: *http://www.qasig.org/presentations/QASIG_Testing_in_Production-Engaging_Customers.pdf*
- Groot, D. J. de, 2012. TiP and the iceberg, *Professional Tester*, June. Available at: *professionaltester.com*
- Grundy, J., Kaefer, G., Keong, J., Liu, A., Guest Editors' Introduction: Software Engineering for the Cloud. *IEEE Software*, March–April 2012, vol. 29, no. 2, p. 26-29, Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
- Riungu-Kalliosaari, L., Taipale, O., Smolander, K., Testing in the Cloud: Exploring the Practice. *IEEE Software*, March–April 2012, p. 46–51
- Pol, M., Teunissen, R., Veenendaal, E. van, 2000. *Testen volgens TMap®*, Tutein Nolthenius, 5th edition, p. 495
- Groot, D. J. de, Kramer, A., Loenhoud, H. van, Prins, A., Ro, J., Schoots, H., Vorst, P., 2012. *Bepaal je koers!, Toekomst en trends in testen*, TestNet, p. 75-81

Internet sources

(API)

- API definition: *http://en.wikipedia.org/wiki/Application_programming_interface#Use_of_APIs_to_share_content*
- Project API: *https://capi.t-mobile.nl/*
- Customer API documentation: *https://capi.t-mobile.nl/documentation*
- My T-Mobile app: *http://www.t-mobile.nl/service-en-contact/apps/my-t-mobile-app*

(app)

- Definition of app: *http://dictionary.reference.com/browse/app*

(BUFT)

- *http://blogs.msdn.com/b/seliot/archive/2010/01/20/building-services-the-death-of-big-up-front-testing-buft.aspx*

(dogfooding)

- *http://harveynick.com/blog/2011/08/26/dogfood-nom-nom-nom/* and *http://en.wikipedia.org/wiki/Eating_your_own_dog_food*

(error codes)

- *http://en.wikipedia.org/wiki/List_of_HTTP_status_codes*

(environment)

- Why are there so many environments? *http://askville.amazon.com/environments-software-development/AnswerViewer.do?requestId=2956413*

(forum)

- Comment on the T-Mobile Forum (in Dutch): *https://forum.t-mobile.nl/belstatus-398/de-online-belstatus-gadget-p-edge-werkt-niet-meer-151973/*

(consumentenbond)

- *http://www.consumentenbond.nl/over/Onderzoek/onderzoek_en_verslagen*

(ICTRecht)

- *http://ictrecht.nl/diensten/juridische-producten/starterspakket/softwareontwikkelaars/* and *http://blog.iusmentis.com/2011/09/19/ben-ik-aansprakelijk-voor-de-fouten-in-mijn-software/*

(Madagascar)

- *http://en.wikipedia.org/wiki/Madagascar_(2005_film)*, *http://www.imdb.com/title/tt0351283/*

(maintenance testing)

- *http://en.wikipedia.org/wiki/Maintenance_testing*

(production environment)

- *http://www.head-fi.org/t/167356/what-does-the-word-production-environment-mean* and *http:/www.techopedia.com/definition/8989/production-environment*

(TiP)

- *http://wiki.answers.com/Q/What_is_production_testing_in_Software_development#ixzz266VbdjZh*

(TNO)

- *http://www.tno.nl/groep.cfm?context=thema&content=markt_producten&laag1=897&laag2=191&item_id=377*

(TNO-Huawei)

- *http://www.nu.nl/internet/2601169/tno-en-huawei-testen-lte-in-nederland.html*

(Whittaker, 2012)

- *http://www.computer.org/cms/Computer.org/dl/mags/so/2012/02/extras/mso20120200004s.mp3*

(Whittaker, 2011)

- *http://blog.utest.com/testing-the-limits-with-googles-james-whittaker-part-i/2011/05/* and *http://blog.utest.com/james-whittaker-the-future-of-software-testing/2008/11/*

> about the author

Marc van 't Veer is a test consultant at Polteq and has well over 5 years of experience as a coordinator and system tester. He has gained a lot of testing experience in technically oriented contexts, such as telecoms, SOA, test automation, development of stubs and drivers, and testing API's. In his current project he provides a lot of training and he manages the outsourcing of testing to India. He holds a MSc. from the University of Eindhoven in the Netherlands and holds the ISTQB advanced certificate in software testing.