Building on Success – Beyond the Obvious

A Closer Look at Good Enough Testing

Drs. Erik P.W.M. van Veenendaal*
TMMi Foundation / Improve IT Services BV
erik@erikvanveenendaal.nl

ABSTRACT

On a regular basis I have in recent years delivered an evolving keynote presentation under the title "Building on success – Beyond the obvious". During this keynote I try indicate which basic testing practices are, based on my personal experiences, often key and sometimes even sufficient to "survive" in real-life projects. Being honest and looking at day-to-day practice, I often notice that many structured testing practices, as defined by TMap [1], TMMi [2] and/or ISTQB [3], are not, or at most partly, applied. I often encounter a meaningless test plan, test design techniques not being applied, reviews not being performed and testers not trained and prepared for their job. And this being is the case more than 30 years after releasing the best-seller "Testing according to TMap", and also more than 20 years after releasing the basic ISTQB Foundations in Software Testing syllabus!

The contradiction here is that despite not applying the proposed testing practices most of us are still releasing systems. However, the release is often (a bit) too late, at much higher costs and often not fully according the expectations. At the project retrospective, management typically at first firmly state they are unsatisfied with the result and the situation, and performance shall be better next time. In practice, next time nothing has changed and often it is the same result and situation. I can only conclude that this is apparently acceptable to the management since they don't really act (although they say differently). My personal observation is that there is a sort of minimum set of testing practice and that there are often in practice just enough to get the job done in a project. In this paper, we will explore and present a minimum set of testing practices starting from the concept of "good enough testing".

CCS CONCEPTS

• **Software verification and validation**; • **Software testing and debugging**; • **Use Cases**; • **Walkthroughs**;

KEYWORDS

Good Enough Testing, ISTQB, Reviews, Risk-Based, T-Shaped, Test Techniques, TMap, TMMi, Unit Testing

ACM Reference Format:
Drs. Erik P.W.M. van Veenendaal. 2022. Building on Success – Beyond the Obvious: A Closer Look at Good Enough Testing. In *Federated Africa and Middle East Conference on Software Engineering (FAMECSE '22), June 07, 08, 2022, Cairo-Kampala, Egypt*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3531056.3542764

The statement that there is a sort of minimum set of testing practice and that there are often in practice just enough to get the job and also the corresponding keynote presentation have constituted much discussion, e.g., on social media platforms. A good reason to also devote a short paper to the topic. Apparently some testers and quality professionals are under the opinion that I'm disloyal to TMap, TMMi and ISTQB (nothing could be further of the truth). From a theoretical and literature perspective fully structured testing seems to be the optimal and best solution. However, in practice a professional tester is often capable to select a minimum set of testing practices from a structured testing approach to get the job done. This minimum set of testing practices are the ones that are strictly necessary for a project. We could reference them as the "good enough" testing practices. Note that this will be different when testing in a safety critical environment, but of course most of us are not.

Which ones are the basic or minimum set of test practices that constitute "good enough" testing? In the remainder of this paper we will look at some important candidates to consider. Those that should be used and, based on many personal practical experiences, have shown to be able to make the difference. It's almost like a tool box for agility or to be lean in testing.

Define clear testing prioritiesConsider the testing principle "Testing is Risk-Based" [4]. Whatever you are testing, whether you have a test plan or not, whether you following an Agile or sequential lifecycle, always in any testing there are limited resources, e.g., time and effort. As a consequence one of the most important things in testing is therefore to define clear priorities regarding the items and features being tested. Priorities shall be defined using a risk assessment process. It is important that every blow is worth a drop! A risk assessment process typically has three steps: 1) to identify the risks, 2) to analyse the risks possibly documented in a product risk matrix (figure 1), and finally 3) define an approach to mitigate the risks. It could be as simply as an informal team-based risk session, or a full Failure Modes and Effect Analysis (FMEA) process. There needs to be a common understanding by all of the risks and subsequently of the defined testing priorities.

Perform limited effective reviewsConsider the testing principle "Testing is preventing defects" [4]. Reviews are often not or very

* *Erik van Veenendaal* (www.erikvanveenendaal.nl) is a leading international consultant and trainer from Improve IT Services BV (Bonaire), and a recognized expert in the area of software testing, quality and requirement engineering. He is the author of a number of books and papers within the profession, one of the core developers of the TMap testing methodology and the TMMi test improvement model, and currently the CEO of the TMMi Foundation. Erik is a frequent keynote and tutorial speaker at international testing and quality conferences. For his major contribution to the field of testing, Erik received the European Testing Excellence Award and the ITQB International Testing Excellence Award.

FAMECSE '22, June 07, 08, 2022, Cairo-Kampala, Egypt

ACM ISBN 978-1-4503-9663-9/22/06.
https://doi.org/10.1145/3531056.3542764

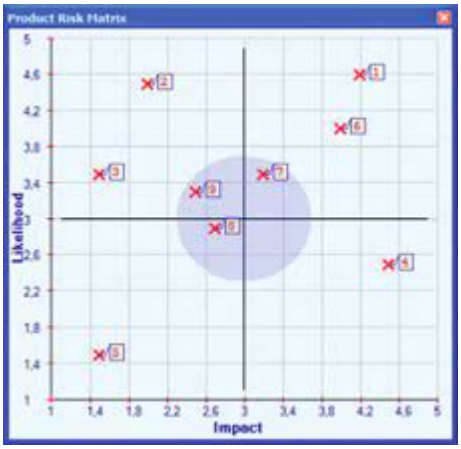

Figure 1: Product Risk Matrix

Figure 2: The T-shaped Tester

poorly performed in projects. Sometimes they are applied highly dogmatic on almost all documents without achieving added value. Make choices (set priorities based on identified risks) and perform a limited number of reviews on documents that are essential to the success of the project, e.g., requirements or user stories. But do make sure that the relative limited number of reviews then are performed in a professional and thorough way. Apply review practices that make the difference, e.g., assign roles, use checklists, not too many pages, apply checking rate, and last but certainly not least ensure the reviews are led by trained moderators.

Get developers doing unit testingConsider the testing principle "Testing requires independence" [4]. Many defects can already be found at unit level. Developers also have a lot of knowledge about the system being developed. Testing for example boundaries and syntax at system test level is really ridiculous and almost like a waste of effort. You can already find these type of defects and do the corresponding rework much more efficiently at unit level. Implement pair testing from XP to achieve independence even at unit level. Ensure that a unit testing process is defined and deployed, preferably supported by code coverage tools. Code coverage provides an independent measure of the quality of unit testing, but it is not only a measure it also provides a great support for the developer doing unit testing getting instant feedback on holes in his or her unit test suite. Unit testing is a many times more efficient and effective approach than setting up a large test team at the very end of the project.

Use test techniques as toolsConsider the testing principle "Testing finding defects" [5]. Test design techniques are a supporting tool to find more and often a certain type of defect. They should by no means be treated as an objective by themselves; finding defects is the goal! Sometimes more formal techniques, e.g., decision table testing, are necessary as a consequence of the complexity or the business risk. However, most often less formal techniques are an adequate way to get the job done. This is especially true for (test) teams with much domain knowledge. One can in this context think of test use cases, classification trees, and of course exploratory testing. An advantage of applying use cases or classification trees is that their way of documenting test cases is more easy to understand by business analysts. This allows for the business analysts to review them and provide valuable feedback. Finally, don't lose yourself in defining a large of test cases, rather start with a test design activity to focus testing. During test design the test conditions (or test situations) that need to be covered are identified. A best practice is to organize test design start-up meetings with the team or with stakeholders, discussing and identifying in collaboration the conditions that need to covered. The output could be something as simple as mind-map, possible to be used as a starting point for exploratory testing.

Build experienced and skilled testersConsider the testing principle "Testing is an extremely complex and intellectual challenging task" [5]. Maybe the most important basic test practice, are the people doing it. To some degree, this paper is especially addressing "level 1" test maturity organizations, which by the way is still the large majority, and as such the testing heroes who are needed in these organizations. Build a great (test) team, almost continuously invest in building the knowledge and skills of the testers. This can be done through formal training, but also by means of training-on-the job and internal sessions where practical experiences are discussed and shared. Following the T-shaped tester concept (figure 2) do not only consider testing knowledge, but also domain knowledge and IT-based knowledge, e.g., requirements engineering and scripting, but do not forget the so-called "soft skills".

Good luck doing "good enough" testing!!

REFERENCES

[1] M. Pol, R. Teunnissen and E. van Veenendaal (2002), *Software testing – A Guide to the TMap Approach*, Addison Wesley
[2] E. van Veenendaal and B. Wells (2012), *Test Maturity Model integration (TMMi) – Guidelines for Test Process Improvement*, UTN Publishing
[3] D. Graham, R. Black and E. van Veenendaal (2019), *Foundations of Software Testing – ISTQB Certification (4th edition)*, Cengage
[4] B. Hetzel (1984), *The complete guide to software testing*, QEB Information Sciences Inc.
[5] G.J. Myers (1979), *The art of Software testing*, Wiley-Interscience