Test Automation Pyramid from Theory to Practice

Andrei Contan
Department of Automation
Technical University of
Cluj-Napoca, Romania
andreicontan@hotmail.com,

Catalin Dehelean
Department of Specialized Foreign Languages
"Babes-Bolyai" University
Cluj-Napoca, Romania
gravedale01@yahoo.com

Liviu Miclea
Department of Automation
Technical University of
Cluj-Napoca, Romania
liviu.miclea@aut.utcluj.ro

***Abstract*—Software professionals employ the disciple of Test automation to create test scripts that validate the technical and functional aspects of a software. A well known test automation model has been defined in the technical literature as, test automation pyramid and is widely referenced when it comes to automated testing within Agile methodologies. Test automation pyramid implies that the interaction with the system under tests should be made on 3 layers, Unit, Service and User Interface. Other interpretations of the model also suggest a quantitative approach, in which the larger surface implies more effort invested in that layer. This paper analyses 5 software projects, developed under Agile principles in which test automation is an important aspect of maintaining high quality standards. The data analysis will focus on the numbers of unit, integration and user interface tests and determines whether the findings comply with the theoretical model. The conclusions attributed to this paper suggests software professionals involved in the development of test automation should put more critical thinking in the models promoted by literature. Also, the paper identified a need for alternative models that are consistent with industry best practices and better correlated with information from real life projects.**

Keywords—test pyramid; testing; test automation;

I. INTRODUCTION

Software testing is a key activity during software development life cycle. The purpose of testing is to validate that code changes applied to software product, does not compromise the quality of the product. In terms of Agile methodology, delivering software testing activities is challenging due to frequent code changes, speed of delivery and market demands. In order to meet such challenges, test automation plays an important role in keeping up the pace with development needs. Running tests faster and providing fast feedback is a key element for delivering successful test automation activities. Successful strategies involve test automation on all software levels, across all application tiers. According to the Software Engineering Body of Knowledge (SWEBOK) [1], the testing levels consist of :

- Unit Testing - verifying the functioning isolation of software elements that are separately testable. is the smallest and simplest form of software testing. These tests are employed to assess a separable unit of software, such as a class or function, for correctness independent of the larger software system that contains the unit. Unit tests are also employed as a form of specification to ensure that a function or module exactly performs the behavior required by the system. Unit tests are commonly used to introduce test-driven development concepts.

- Integration (Service layer) testing - the process of verifying the interactions among software components. Software components that pass individual unit tests are assembled into larger components. Engineers then run an integration test on an assembled component to verify that it functions correctly. Dependency injection, which is performed with tools such as Java Spring [21] is an extremely powerful technique for creating mocks of complex dependencies so that an engineer can cleanly test a component. A common example of a dependency injection is to replace a stateful database with a lightweight mock that has precisely specified behavior.

- UI Testing - is concerned with testing the behaviour of an entire system from an user perspective. It addresses testing techniques such as A/B testing, responsive designed, cross-platform, usability. Tests suites defined at UI level, are usually split into User Acceptance Tests or Acceptance testing. Tests implemented at the top of the pyramid are considered to have the following negative attributes, as suggested by [22]. **Brittle** -A small change in the user interface can break many tests. **Time consuming** - the elapsed time to run UI tests is extensive and can take minutes.

For application tiers, in this paper we will refer to 3 tiers architectures, consisting of: database layer, backend server and consumer service (usually considered User Interfaces).

The testing pyramid was first introduced Mike Cohn [2] and is described as "an effective test automation strategy" in which Unit Testing is the foundation, following the Service (also referred to Integration layer) and UI testing at the top.

978-1-5386-2205-6/18/$31.00 ©2018 IEEE

The software industry provided different interpretations to the testing pyramid [3] , [4] including addressing additional layers [3] or renaming them to improve the model. The most common interpretation of the test pyramid model suggests that the Surface area of each layer, represents the amount of effort to be invested in that specific layer, meaning that unit testing should be the most elaborate one, followed by Integration and Acceptance:

Test Automation Coverage = S(Unit Test Layer) + S(Integration Layer) + S(User Interface Layer),

where S(Unit Test Layer) > S(Integration Layer) > S(User InterfaceI Layer)

However, during the analysis phase of this approach, we couldn't find any industry or academic report to suggest that the model is applicable in real-life software projects. For this reason, the current paper suggest a different interpretation of the testing pyramid along with its applicability in real-life test automation projects. The test automation pyramid is technology agnostic which means that it can be applied across any multi-tier software architecture.

II. Structure

The paper is structured as follows: in section II we provide an overview of the related work in test automation per different layers and other similar models, followed by the study design which describes the research paper that this paper is trying to answer. Next section offers an overview of the results, followed by discussions and interpretation of data. In the final section we provide our conclusions and recommendations of future work.

III. Related work

The area of test automation has gained a lot of attention in the recent years and we were able to find consistent research for specific test automation layers.

The current work has been conducted in cooperation with 3 software companies across 5 projects developed according to Agile methodology where test automation was considered a must.

Unit testing is widely covered by Nguyen et al [5] describing the approach and structure of writing unit test methods. Also Offutt [6] concludes that they were able to *find more faults than edge-paired coverage (EPC) at the unit testing level.* Parameterized UnitTesting is also a critical aspect in terms of code quality. According to Xie, [7], the parameterised unit testing allows the separation of two testing concerns: the specification of black-box behaviour and the generation and selection of white-box test inputs. They follow-up also by highlighting the success stories and future research directions in this area.

On the Integration testing, Orellana [8] covers the differences between unit and integration testing, by concluding that the number of defects reported by the unit tests is significantly higher than those exposed by the integration tests.

Also, in terms of testing activities, the number of developers implementing integration tests is lower than the ones doing unit tests, but also lower than the ones doing functional tests at UI level [9]

On UI testing, the GUI based testing activities have gained lots of traction due to the increased number of mobile devices where usability and user experience is critical [10]. UI testing is categorised as black-box testing method. According to Isabela[[11], UI testing is the process of determining the functional requirements on the software interface whether it has met and done with the right way [12].

IV. Study Design

A. Research questions:

This papers is trying to answer the following research question:

RQ1 - "Does the current development and testing practices reflect the test automation pyramid shape of unit, integration and user interface testing?"

The study was applied in 3 different companies for 5 software products working in an Agile environment. Each software product had a test automation strategy defined to help the development effort by providing fast feedback to code changes and making sure that no regression problems are introduced with every change.

The size of the team varied from 5 to 7 members while all software products are considered Web Applications built on a 3-tier architecture. The area of business of each product varied from online-shopping (2), finance (1), media (1), identity access-management (1).

The main technologies used for each project are presented in table 1:

	DB	Server-side	UI
Project 1	MongoDB	Java	Angular JS
Project 2	MSSQL	.NET	Angular JS
Project 3	MongoDB	Java	React
Project 4	MySQL	Java	Angular2 JS
Project 5	MySQL	Java	Angular

Table 1: Project technologies

This research evaluates the testing pyramid from a different angle. Instead of current interpretation which suggest that test automation should be developed on layers, we propose 2 new views:

A. that test automation should be developed on slices, instead of layers.

B. The test automation pyramid to be interpreted as a set of needs, per Maslow's hierarchy of human needs [13]

The correlation with Maslow's models suggests that for a consistent testing strategy, the testing activities should be addressed from bottom-up. But the amount of effort should not be consistent with the pyramid model meaning that the lower levels (containing the largest surface) does not represent the amount of effort to be invested in that specific testing activity.

A 3 tier architecture consists of database layer, server side and user-interface, the relevant question to ask when considering testing is: *"what is the most efficient and effective level of validating, through tests, a functional area?"*.

In this approach, the test engineer is focusing on efficiency and effectiveness instead of volume and costs. The output of this thinking process is a set of defined test scenarios along with the level of interaction with the system under test.

Our observations during industry experience is that is more to know what kind of risks are being addressed by these sorts of tests and be implemented at the level considered most efficient and effective. We shift the mindset from "Haver you implemented enough tests", to "Are you testing the right thing?"

Example: Considering an application which converts SQL syntax into HTML and back, where the conversion is handled by a 3rd party library. In this case, there's very little complexity at code level, since the qualify of 3rd party library has been addressed since it's an external product. There is complexity, however at integration level where communication with different databases is critical or UI content is relevant for users. Depending on the app and having in mind the 3 layers, the outcome of this approach suggests a totally different shape:

We've identified during project work that implementing tests per slices is a big drawback in managing functional deliverables to client, by the end of each sprint as suggested by the Scrum Values[14]. The challenge was that test automation had difficulties in keeping up the pace with development progress and development changes. Also we've noticed few occasions in which test automation was considered a bottleneck due to order of implementing layers, UI being usually the last part to be implemented in all 5 projects.

V. Results and findings

1. Method Overview

The study was performed by qualitatively and quantitatively monitoring the number of tests per each test level and also the code quality as a result of having test suites addressing application specific concerns. The quantitative results of this analysis are presented in Section V, and a discussion of the results in context of previously published literature is presented in Section VI.

2. Classification of data

The following steps have been followed to acquire the statistical data of this study:

- All data has been acquired from the code repositories of each project and identifying the test specific annotations.
- Tests have been analysed and label according to the type (Unit, Integration or UI)
- All information was centralised and aggregated. The final results are presented in Fig 1, under Results section.

VI. Validity

In this section we describe the various threats to validity that were considered during the study. The purpose of this effort is to make the effort more transparent and trustworthiness on the results. The validity is discussed in this section, since it cannot be finally evaluated until the analysis phase.[15]

Construct validity - addresses if the study succeeded in measuring what was intended to measure.[16]. Our mission was to find out whether the test automation pyramid has industrial applicability and the format suggested can be found in web application projects. To achieve this, we analyzed 5 software development projects across 3 companies where test automation involved a significant part of the overall testing effort. We haven't considered other types of applications, since web application types, having a 3 tier architecture, are the most common ones used within industry. During analysis phase, we also asked developers and testers involved about the missing types of test that could lead to the pyramid shape. In 4 out of 5 situation we were told that the teams followed a risk-based approach to testing and any missing tests are not considered relevant for the scope of the project.

Internal validity - concerns the analysis of information gathered. The causality and correlation implied and stated is reflected in the qualitative part of the paper. We link our qualitative conclusions to data processed to support our statements

External validity - addresses the generalisation of the findings as defined by [15] . A limitation of this study is that only addresses small size projects (5-7 developers and testers) working in Agile-Scrum methodologies for web application projects. We have provided the context in which the research took place to allow other studies to improve the study and enable comparison between results. Another factor towards validity is the complex environment in which all this observations are made.

Reliability - addresses the repeatability of the study[15.]. We have provided definitions of what each testing level means, to make it possible for other researchers to make equivalent interpretations. Introducing new layers such as Components or System Testing [17] could increase the scope of the research, but we're confident from our experience that the new layers will not change the conclusions of the paper.

VII. Results

The analysed data consists of 5 projects, across 3 companies, working on Web Applications for different business industries including: online shopping retailer, media platforms, finance. The timeline for the research was from January 2017 to October 2017.

The RQ1 was addressed through the analysis of the number of tests report.

We found out that only one project (20%) has a pyramidal shape, although does not fully comply with the volumes per layer suggested by the theoretical model. From the remaining 4 (80%) projects, 1 (20%) have a trapezoidal approach meaning that there's no test automation at UI level.

We followed-up on this subject with members of the team and the reason for 0% UI tests is the frequent changes in the user interface or the low risk represented by the potential problems of UI. From the last 2 remaining projects, 1(20%) had a risk based strategy in approaching the testing process which resulted in the 'E'shape while the last represents the 'cone' anti-patterns where the UI tests represent the most effort, followed by Integration tests, while unit tests are the least. Other reasons identified for which User Interface testing is not subject of automation effort consists of:

- Fragility - it was reported that often repeating the same tests multiple tests provides inconsistent results. This is due to the fact that the number of external factors that influence the behaviour of software cannot be controlled during an automated test. A UI test that fails could have multiple reasons: a functional problem, a browser problem, a timeout problem, page load time problem.
- Low return on investment - the effort required to implement the tests doesn't bring the expected benefits. When asking the business persons within the project what are the business risks covered by the UI tests, we couldn't get a consistent answer, concluding that it is more used by the test engineers to avoid repetitive work.

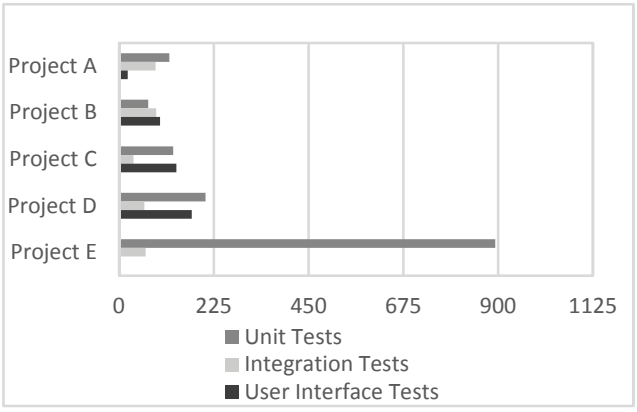

FIG 1 - NUMBER OF TESTS PER SOFTWARE LAYER

	UNIT TESTS	INTEGRATION TESTS	USER INTERFACE TESTS
Project A	119	86	20
Project B	69	88	97
Project C	128	34	136
Project D	205	60	172
Project E	893	63	0

TABLE 2 - NUMBER OF TESTS PER SOFTWARE LEVEL

VIII. FURTHER RECOMMENDATIONS TO PRACTITIONERS

Looking at the results of this study, it is clear that the test automation pyramid does not reflect the industry practices. What it was meant to be just a model for the needs and possibilities of interacting with the system under test, has gained a lot of attention and generated many different interpretations without a qualitative study to support it. Further studies have been conducted in the area of Mobile Applications where it was shown by [23] that the Test Automation Pyramid Model is upside-down. In this version of the pyramid, the automated coverage of unit testing layer is the smallest one. This is due to the fact that not all components of a mobile app can be tested in isolation, due to the complexity of mocking and faking techniques.

We emphasise the importance of critical thinking of such models when test automation strategies are required to be designed. During the study, we noticed that developers are more inclined to write unit tests, while testers are more willing to write UI tests. The topic of integration tests is, at the moment, a common ground where each team manages the responsibility in its own way.

An alternative to Test Automation pyramid, we propose other test models and strategies, as risk-based testing approach [18] or Bayesian Prediction models for risk-based test selection [19] which support the idea that "risk information is used in testing As a suggestion to extend the scope of testing towards 'risky' areas where critical problems can be found, or as a guideline to optimally adjust the focus of testing to 'risky' areas where most critical problems are located" [20].

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we report an investigation on the Test Automation Pyramid. Our efforts are focused in determining the gap between theoretical models suggested by software testing practitioners in literature and the real life project examples. In contrast to the high number of blogs and articles on the internet talking about this model, we were not able to find any scientific paper that support or critique the approach.

Based on this, we conclude that further investigations can be made in several areas of test automation. First is to identify industry well known practices and research its efficiency and effectiveness in the area of test automation. Second, we conclude that more research potential can be found in the area of defining testing strategies models and frameworks that support the analysis, design, implementation and execution of testing activities within given context.

Part of our research, we also consider to have a different approach in the interpretation of the test automation pyramid that can drive the automation testing efforts. The idea behind suggests that each of the software layers can be approach as a pyramid itself. We're looking into the possibility that at database layer, there is unit and integration testing concepts which can also be applied at back-end level where unit/integration/acceptance levels can be addressed, while at the presentation layer, there can also be unit/integration/acceptance levels. As already proposed, the latter implies redefining the top of the pyramid to fit the needs and cover the correspondent risks.

X. REFERENCES

1. Pierre Bourque, Richard E. Fairley. 2014. Guide to the Software Engineering Body of Knowledge (Swebok(R)): Version 3.0 (3rd ed.). IEEE Computer Society Press, Los Alamitos, CA, USA.
2. https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid
3. http://james-willett.com/2016/09/the-evolution-of-the-testing-pyramid/
4. https://martinfowler.com/bliki/TestPyramid.html
5. Memon, A. M. and Nguyen, B. N. (2010) 'Advances in Automated Model-Based System Testing of Software Applications with a GUI Front-End', Advances in Computers. 1st edn. Elsevier Inc., 80(C), pp. 121–162. doi: 10.1016/S0065-2458(10)80003-8.
6. Li, N. and Offutt, J. (2017) 'Test Oracle Strategies for Model-Based Testing', IEEE Transactions on Software Engineering, 43(4), pp. 372–395. doi: 10.1109/TSE.2016.2597136.
7. Tillmann, N. (2016) 'Advances in Unit Testing : Theory and Practice'. doi: 10.1145/2889160.2891056.
8. Orellana, G. et al. (2017) 'On the Differences between Unit and Integration Testing in the TravisTorrent Dataset', IEEE International Working Conference on Mining Software Repositories, pp. 451–454. doi: 10.1109/MSR.2017.25.
9. Kochhar, P. S. et al. (2015) 'Understanding the test automation culture of app developers', 2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings. doi: 10.1109/ICST.2015.7102609.
10. Klammer, C., Ramler, R. and Stummer, H. (2016) 'Harnessing Automated Test Case Generators for GUI Testing in Industry', Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, pp. 227–234. doi: 10.1109/SEAA.2016.60.
11. Isabella, A & Retna, Emi. (2012). Study Paper on Test Case generation for GUI Based Testing. CoRR. abs/1202.4527. . 10.5121/ijsea.2012.3110.
12. D. Siahaan, "Analisa Kebutuhan dalam Rekayasa Perangkat Lunak (Requirements Analysis in Software Engineering)," Yogyakarta: Andi Offset, 2012.
13. https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs
14. https://www.scrumalliance.org/why-scrum/scrum-guide
15. P. Runeson and M. Ho¨st. "Guidelines for conducting and reporting case study research in software engineer- ing". In: Empirical Software Engineering 14.2 (Dec. 2008), pp. 131–164.
16. C. Robson. Real world research. 3rd ed. John Wiley & Sons, 2011.
17. R.C. Martin, "The Clean Coder: a code of conduct for professionals", 2011, ISBN-10 = 0137081073
18. R. Ramler and M. Felderer, "Requirements for Integrating Defect Prediction and Risk-Based Testing," 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Limassol, 2016, pp. 359-362. doi: 10.1109/SEAA.2016.62
19. H. M. Adorf, M. Felderer, M. Varendorff and R. Breu, "A Bayesian Prediction Model for Risk-Based Test Selection," 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, Funchal, 2015, pp. 374-381. doi: 10.1109/SEAA.2015.37
20. M. Felderer, and R. Ramler, "A multiple case study on risk-based testing in industry," International Journal on Software Tools for Technology Transfer (STTT), 16(5), October 2014, pp. 609-625.
21. https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html
22. M. Cohn, "Succeeding with Agile: Software Development Using Scrum",
23. D. Knott, "Hands-on mobile app testing"