

Test Automation with the Gauge Framework: Experience and Best Practices

Vahid Garousi[1,3(✉)], Alper Buğra Keleş[2], Yunus Balaman[2], and Zeynep Özdemir Güler[2]

[1] Queen's University Belfast, Belfast, Northern Ireland, UK
v.garousi@qub.ac.uk
[2] Testinium A.Ş., Istanbul, Turkey
{alper.keles,yunus.balaman,
zeynep.ozdemir}@testinium.com
[3] Bahar Software Engineering Consulting Limited, Belfast, UK

Abstract. While Behavior-driven development (BDD) tools such as Cucumber are powerful tools for automated testing, they have certain limitations. For example, they often enforce strict syntax for test cases, like the "Given-When-Then" format, which may not always be easy to write for a given test case. A new test automation framework named Gauge (gauge.org) addresses that limitation since it does not prescribe the BDD testing process with a strict syntax. In Gauge, writing a test case is as easy as writing down the flow of test cases in several itemized sentences in a natural language, like English.

In the context of Testinium (testinium.com), a large software testing company which provides software testing services, tools and solutions to a large number of clients, we have actively used the Gauge framework since 2018 to develop large automated front-end test suites for several large web applications.

In this paper/talk, the speakers will share several examples and best practices of developing automated tests in natural-language requirements using the Gauge framework. By learning from the ideas presented in the talk, readers (attendees) will be able to consider applying the Gauge framework in their own test automation projects.

Keywords: Software testing · Test automation · Gauge framework · Industrial experience · Best practices

1 Introduction

If planned and executed properly, test automation can increase efficiency and effectiveness of software testing and thus leading to higher quality software products [1, 2]. On the other hand, just like any other software engineering activity, if not planned and executed properly, test automation can result in failure (google.com/search?q = test +automation+failure).

Test automation technologies are rapidly evolving and new methods and tools constantly appear the in industry. One of the approaches is to the Behavior-Driven Development (BDD) approach [3] which "*describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters*" [4].

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12250, pp. 458–470, 2020.
https://doi.org/10.1007/978-3-030-58802-1_33

While the BDD approach and the many tools that support BDD, such as Cucumber, are useful in many contexts for automated testing, they have certain limitations, e.g., [5–8]. For example, they often enforce "strict" syntax for test cases, like the "Given-When-Then" format, which may not always be easy to write for certain test cases [7]. That is one of the main reasons that a recent test automation framework named Gauge (gauge.org) was released by ThoughtWorks in 2018 [5]. Gauge addresses the above limitation since it does not prescribe the BDD testing process with a strict syntax. In Gauge, writing a test case is as easy as writing down the flow of test cases in several itemized sentences in a natural language, like English. Executable test specifications are written in the form of regular natural-language sentences and, using some form of mapping functions, are then directly executed for the purpose of testing.

Development of test suites in such a manner provides various benefits (e.g., enabling testers to write test cases in natural language) and, at the same time, exposes many challenges which have to be addressed, as we will see in the rest of this paper, e.g., flexibility of free text for test scripts can be a demerit as they will not have a defined format.

In the context of Testinium (testinium.com), a large software testing company which provides software testing services, tools and solutions to a large number of clients in several countries, the authors of this paper and their team members have actively used the Gauge framework since 2018 to develop large automated front-end test suites for several large web applications. Our research and development efforts are under the umbrella of a large-scale industry-academia collaboration [9], following the technology transfer guidelines in software engineering [10], and has been conducted in the context and using the funding of a 3-year international European R&D project named the "TESTOMAT – The Next Level of Test Automation" (www.testomatproject.eu), in which 34 industrial/academic partners across six countries are collaborating.

In this paper (talk), the speakers will share several examples and best practices of developing automated tests in natural-language requirements using the Gauge framework. By learning from the ideas presented in the talk, readers (attendees of the talk) will be able to consider applying the Gauge framework in their own test automation projects.

2 Overview of the Industrial Context and Need Analysis

Testinium A.Ş. is a provider of testing services and solutions in Turkey and across Europe. Testinium A.Ş. was founded in year 2010 and has provided automated testing to 50+ clients in several countries so far. In addition to the test service offered to clients, the company has developed and offers two flagship test automation tools: *Testinium* (testinium.com) and *Loadium* (loadium.com). As of Spring 2020, Testinium A.Ş. has about 190 employees. The company works using the agile methodologies in general.

The company has been proactive in adapting novel approaches to increase effectiveness and efficiency of its test activities, and joining the European TESTOMAT project has been one of those initiatives. Almost all of the Systems Under Test (SUTs)

tested by test engineers are the clients' web or mobile applications, e.g., the online ticket sales website of several major airlines in Turkey.

A routine ongoing activity in the company is to utilize the latest test automation tools and approaches in the projects. As it has been discussed in the literature, "*Selecting the right tool for the right purpose is a key to success*" [11]. There have been a lot of guidelines and suggestions for choosing the "right" test automation tool in the grey literature (e.g., blogs posts, white papers), and a recent a grey literature review summarized the body of knowledge on the topic [12]. Due to the reasons discussed in the previous section above, i.e., Gauge addressing the shortcomings of the BDD tools/approaches, the team decided to use Gauge as a main tool for UI-based system testing of web applications.

For the purpose of illustration in this paper, as a the running example, we pick one of the Systems Under Test (SUT), a test tool developed in-house, which is one of the company's main offered solutions, *Testinium* (testinium.com). Testinium is a web application which provides various test management features via a web interface to Selenium. Two screenshots from the *Testinium* are shown in Fig. 1, in which the login page and the "all reports" page are visible. To provide more information about features of the tool, we depict its use-case diagram in Fig. 2.

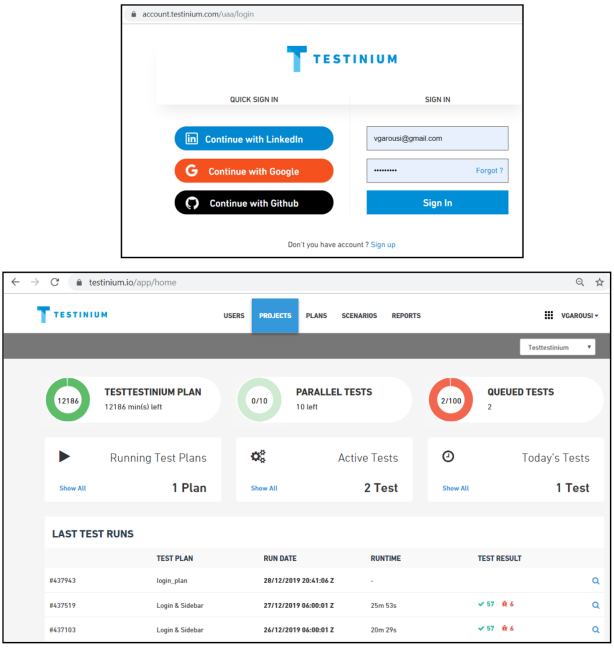

Fig. 1. Two screenshots from the Testinium (testinium.com) test-management system



Fig. 2. Use-case diagram for the Testinium

3 Related Work

Since the Gauge tool is quite recent (released in 2018), except several informal online blog posts and such, there are not any experience reports that we know of, using Gauge.

Various papers have been published on Behavior-driven development (BDD), e.g., [13, 14]. Various books and papers have been published on test automation. One of the books on the topic is entitled "*Experiences of test automation: case studies of software test automation*" [15], which is composed of 29 chapters, all written by industrial test practitioners who reported separate case studies on test automation.

The authors have published a number of experience reports on test automation using different technologies and in different contexts, e.g., [2, 16–21].

4 How the Gauge Tool Works

In Gauge (gauge.org), an automated test case written by writing down the flow of test cases in several itemized sentences in a natural language (like English). Executable test specifications are written in the form of regular natural-language sentences and are then directly executed for the purpose of testing. For illustration, we show in Table 1 the partial test-code listing of a large Gauge test-script that we have developed for the Testinium as the SUT.

To allow test engineers develop test scripts in the varying levels of "abstraction" and to enable highly modular (and higher quality) test scripts [22], Gauge defines five inter-related concepts: test "specification", test "scenarios", test "concepts", test "steps", and step "implementation", the relationship of which are shown in Fig. 3. We also provide an example Gauge test-script for the Testinium in Table 1 in which the above test constructs have been used.

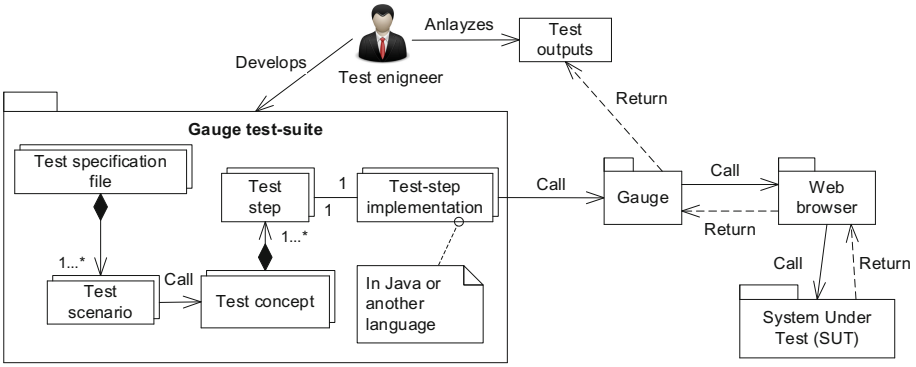

Fig. 3. Architecture of test automation in our approach and building blocks of Gauge test-scripts

Table 1. An example Gauge test-script for the Testinium as the SUT

Level of abstraction: High	1 test "scenarios". Calling 2 test "concepts"	Tags:LoginPage_InputControls * In the Login page, do the controls for the Email field * In the Login page, do the controls for the Password field
	2 test "concepts". 3+4 test "steps"	# In the Login page, do the controls for the Email field * Go to "http://www.testinium.com" address * Check if the "tbEmailOfLogin" element is visible * Login using "testinium" and "passwd**" # Login using <email> and <password> * Inside the field "tbEmailOfLogin", write <email> * Inside the field "tbPasswordOfLogin", write <password> * Check if the "btnSignInOfLogin" element exists * "btnSignInOfLogin" elementine tıkla

Level of abstraction: Low	A step "implementation"	@Step("Inside the field <key>, write <text>")
		public void sendKey(String key, String text){
		ElementInfo elementInfo = Stor
		Helper.INSTANCE.findElementInfoByKey(key);
		sendKeyBy(ElementHelper.getElementInfoToBy(elementInfo),
		elementInfo.getIndex(), text);
		}

A test engineer develops a test suite, which could include several (one or more) test *specifications*. Each test *specification* is a ".SPEC" file and includes a set of test *scenarios* (see the test script example in Table 1). We have learnt by experience that a best practice is to correspond each test specification file to each use-case of the system (Fig. 2). For example, we would have one test *specification* file for the "login" use-case, which has several test *scenarios*, one of them being `LoginPage_Input Controls` as shown in Table 1.

Test *concept* is another feature provided by Gauge to enable modularity of test-code, a practice which has been widely encouraged in test automation [22]. For example, in the example of Table 1, there is one test *scenario* (`LoginPage_Input Controls`), which includes calls to two test *concepts*, which are defined in another scope (a .CPT file). In addition to ensuring varying levels of abstraction, such a modularity also support another proven design pattern: *separation of concerns (SoC)*. Each test *concept* can have one or more test *steps* (see the example in Table 1). Test *steps* are the lowest-level entities in terms of abstraction in Gauge. To enable the executability of Gauge test scripts, a test *implementation* (e.g., in Java Selenium) should be developed for each pattern of test *steps*. For example, test *step* "`Inside the field tbEmailOfLogin, write <email>`" will call the corresponding function in Java code in Table 1, which will send the keys, etc. to realize that test steps. In fact, each test *step* is like a function call, and it can include parameters, as seen in the example.

Other parts of the test-automation architecture diagram in Fig. 3 shows how the rest of the process works. A test *implementation* invokes Gauge and Selenium libraries and, via them, calls (exercises) the UI elements in the web application (SUT). Once tests finish executing, test outputs are provided to the test engineer.

5 Developing Gauge Test Suites for Multiple Projects

As mentioned earlier, we have actively used the Gauge framework since 2018 to develop large automated front-end test suites for ten's of large web applications, as SUTs. The SUTs are those developed by our clients and many systems developed in-house, e.g., Testinium (testinium.com). We show in Table 2 the size metrics of three SUTs, as examples, which we have tested using Gauge in the company so far.

For the Testinium as the SUT, we are showing two rows, which we will discuss in more detail in the next section. As part of the company's mission to continuously improve test practices, as our test engineers and we have developed more Gauge test suites, we have found better ways of developing test scripts. Thus, for the Testinium as the SUT, the two rows in Table 2: size metrics "before" systematic test design (pilot phase), and metrics after systematic test design (details are next).

Table 2. Size metrics for several SUTs, tested using Gauge in the company

Systems Under Test (SUT)	Application domain	Number of use-cases (features) tested	# of Gauge test specifications	# of test scenarios	# of test concepts	# of test step calls in test concepts	# of test steps (unique)	Line of Code (LOC) of test implementations	Development effort (engineer-month)
SUT1-Testinium: Before systematic test design	Testinium-: one of the company's test tools	7/13	162	1,667	518	2,658	119	1,123	4
SUT1-Testinium: With systematic test design		13/13	139	518	129	603	44	380	2
SUT2	E-commerce	14	114	539	67	449	41	3,415	4
SUT3	Telecommunications	8	14	181	5,411	508	58	1,348	5

6 Challenges When Developing Large Gauge Test Suites

We synthesize in this section a summary of the challenges that we have faced when developing large Gauge test suites since 2018, and the best practices that we have come up with to address those challenges to be able to develop high-quality maintainable test scripts [22], in Sect. 7.

As it has been widely discussed in the testing community, a test tool alone is not enough to reach successful test automation. Among other important "ingredients" are "a proven [effective] testing process" and "a skilled test team structured to meet the challenges of automation"[1]. Many online articles and blog posts have also highlighted this fact:

- For "*successful [test] automation*", "*only tool is not enough*"[2].
- "*...while automation may be the answer, a tool is not enough*"[1]
- "*The selection and usage of an automated testing tool do not guarantee success*" [23]

Furthermore, in the context of Gauge, two practitioners have shared the following observations:

- "*Gauge follows Markdown Syntax for scripting the tests in an unstructured way - which means more liberty to make the test like a specific document and also more risk of ending up with scripted tests in many different styles. Having some in-team standards will help to prevent any chaos*." [8]
- "*If you feel that Cucumber is a pain when used in large projects due to its Given-When-Then syntax becoming messy in long run. Gauge can take out that pain to some extent. Even here, the flexibility of free text can be a demerit as they don't have a defined format. But tests may end up looking like a messy text document unless imposed strict restrictions*". [6]

For the case of several initial pilot projects that we were developing Gauge test suites, we observed similar challenges as reported by the two quotes above. An example instance of such an issue is shown in Fig. 4. Note that since almost test engineers working in the company are Turkish, the Gauge test scripts have been have developed in Turkish. Even if the scripts text would not be readable to the non-Turkish reader, we can see that, due to different styles of testers writing the scripts, there were many issues, e.g., a lot of variability in length of test cases, duplication in scripts.

[1] www.mosaicinc.com/mosaicinc/successful_test.htm.

[2] www.softwaretestinggenius.com/successful-automation-only-tool-is-not-enough/.

```
#Login ekranındaki Email alanının kontrollerini yap
*"https://accountdev.testinium.com/uaa/login" adresine git
*"tbEmailOfLogin" elementinin görünürlüğü "true" mu kontrol et
*"testiniumtest1" ve "sifre12345" ile giris yap
*"tooltipMessageOfLogin" tooltipi "Please enter a valid email address." textine sahip mi bekleyip kontr
*"tbEmailOfLogin" elementinin text alanını sil
*"" ve "sifre12345" ile giris yap
*"tooltipMessageOfLogin" tooltipi "This field is required." textine sahip mi bekleyip kontrol et

#Login ekranındaki Password alanının kontrollerini yap
*"https://accountdev.testinium.com/uaa/login" adresine git
*"tbPasswordOfLogin" elementinin görünürlüğü "true" mu kontrol et
*"tbPasswordOfLogin" elementi zorunlu mu kontrol et
*"testiniumtest1@mynet.com" ve "" ile giris yap
*"tooltipMessageOfLogin" tooltipi "This field is required." textine sahip mi bekleyip kontrol et
*Input alanlarini temizle
*"tbPasswordOfLogin" elementine "@ÿ€®Ł¥" değerini yaz
*Input alanlarini temizle
*"tbPasswordOfLogin" elementine "123456" değerini yaz
*Input alanlarini temizle
*"tbPasswordOfLogin" elementine "qwerty" değerini yaz
*Input alanlarini temizle
*"testiniumtest1@mynet.com" ve "1234567" ile giris yap
*"txtDangerOfLogin" elementi "Email or Password is incorrect!" text değerine sahip mi
*Input alanlarini temizle
*"testiniumtest1@mynet.com" ve "12345678" ile giris yap
*"txtDangerOfLogin" elementi "Email or Password is incorrect!" text değerine sahip mi
*Input alanlarini temizle
*"testiniumtest2@mynet.com" ve "sifre12345" ile giris yap
*Şu anki url "https://clouddev.testinium.io" içeriyor mu

#Login ekranındaki LinkedIn butonunun kontrollerini yap
*"https://accountdev.testinium.com/uaa/login" adresine git
*"btnLinkedInOfLogin" elementine tıkla
*Şu anki url "https://www.linkedin.com" içeriyor mu


#Login ekranındaki Forgot butonunun kontrollerini yap
*"https://accountdev.testinium.com/uaa/login" adresine git
*"btnForgot" elementinin görünürlüğü "true" mu kontrol et
*"btnForgot" elementine tıkla ve "https://accountdev.testinium.com/uaa/forgot-password" sayfasina ulas
*"btnTestiniumLogoOfLogin" elementine tıkla
```

Fig. 4. Excerpts from Testinium test suites in the pilot phase, in which the test code is of imperfect quality due to ad-hoc test-case design and scripting

While Gauge is a better tool compared to the competitors, and provides a powerful environment to develop test scripts in varying levels of abstraction, the intentional "flexibility" that Gauge provides for developing test steps in natural language can be a disadvantage, as it is easy to end up with a unorganized text document with a lot of issues (such as duplication) which will be hard to inspect and maintain afterwards [22]. Thus, our initial pilot phase showed to be useful in identifying the following two challenges:

- Challenge 1-Need for systematic test-case design: Systematic design of test cases is fundamentally important and crucial for success of any type of testing, either manual or automated testing [24]. However, many surveys and studies are reporting a general weakness in many context in terms of systematic design of test cases [24–26]. If test-cases are not designed systematically, but instead in purely ad-hoc manner, the resulting test suite will generally suffer from low fault detection effectiveness, since there could duplication test logic among test cases, etc. Also, without systematic design of test cases, the team cannot check the progress and also the completion ratio of tests [24].
- Challenge 2-Need for modular/maintainable test-scripts: Test code is a type of source code and just also any source code, it should be developed with high quality [22], e.g., it should be modular, readable and maintainable. This challenge is

slightly related to the previous challenge in that if test cases are not designed systematically, test scripts will be developed in ad-hoc and chaotic manner.

In the next section, we present a best practice that we have found to be effective in addressing the above challenges.

7 Best Practice: Systematic Design of Test Cases

Since the type of SUTs that we have been working on are all web applications, we identified model-based testing (MBT) [27] as the most suitable approach for systematic test-case design. MBT has been shown to be a suitable approach for testing web apps [28, 29], i.e. nodes would be the pages of the web app and edges would be the transitions among the pages (nodes).

Our ultimate goal has been to use automated MBT tools (such as GraphWalker, graphwalker.github.io) also in the process, together with Gauge, but that would be quite counter-intuitive, e.g. since Gauge scripts are supposed to be written by human testers. Thus, as the first phase, we decided to apply a light-weight manual MBT to design our test cases, and have human testers write Gauge test scripts based on them. The rationale was to have an approach which test engineers will easily use without having a heavyweight and sophisticated tooling.

Figure 5 shows the actual first version of the test model that we developed and of course, it was on paper, to highlight the light-weighted nature of our approach. The additions of new nodes and edges to the original paper version were made later on, as we noticed that new parts had to be added. This shows the "incremental" nature of the work. The light-weight nature of our MBT approach is also called "Just barely good enough"[3] modeling in the Agile modeling context.

In our light-weight model-based test design approach, once a test model is designed, we would derive four types of test cases from it:

1. Transition from the current node (page) to its next neighboring nodes (pages)
2. Node to itself (often, error checking, like: Invalid Login)
3. Input UI tests (in single node/page/unit level)
4. End-to-end testing: testing a path of nodes

For example, applying the above approach on the Login use-case of the system, results in deriving six test cases from the first type above (transition from the current node to its next neighboring nodes), as listed below:

- Node-2-node-1: Login to Dashboard: Valid login using regular login
- Node-2-node-2: Login to Forgot Password
- Node-2-node-3: Login to Signup
- Node-2-node-4: Login using LinkedIn
- Node-2-node-5: Login using Google
- Node-2-node-6: Login using GitHub

[3] www.agilemodeling.com/essays/barelyGoodEnough.html.

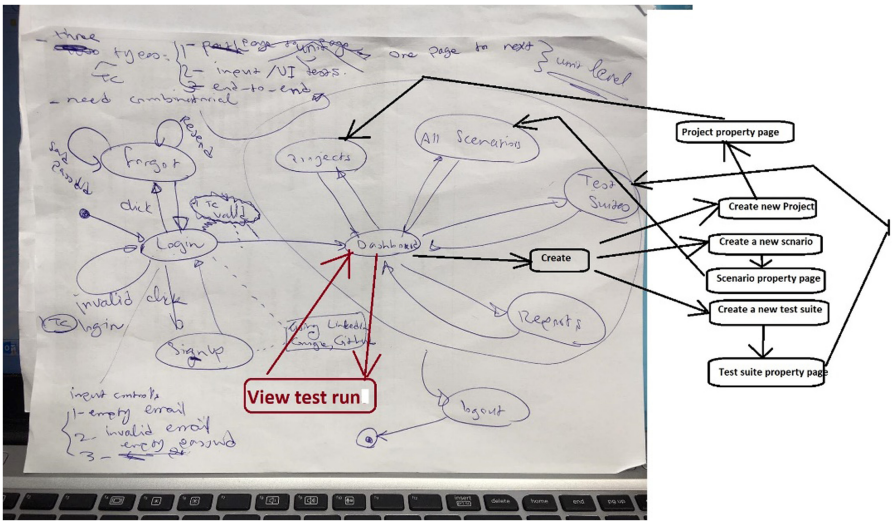

Fig. 5. Systematic design of test cases using light-weight test models

As shown in Table 2, after applying the above systematic approach, we derived 139 Gauge test specifications, and 518 test scenarios for the SUT.

8 Conclusions; Ongoing and Future Works

Now that we have been developing several Gauge test suites using our best practice above, in retrospective, we are able to observe various benefits, compared to our initial phase, in which test-cases were designed in ad-hoc manner.

- **Clarity and road-map for test engineers**: First of all, test engineers have clarity on how to develop Gauge test scripts, since the light-weight model-based test design approach provide a clear road-map for test engineers and makes it easier to know the progress and what additional test cases to derive.
- **Higher quality test script in term of modularity with no/less duplication:** Since the light-weight test design approach breaks the test design problem into smaller parts ("divide and conquer"), test engineers are less likely to develop "spaghetti"-style test scripts with no/less duplication [24].

Our ongoing and future works are in the scope of constantly improving our efforts in developing Gauge test suites, by findings more innovative ways and best practices for these tasks.

Acknowledgements. This work was supported by the European ITEA3 program via the "TESTOMAT (The Next Level of Test Automation)" project with grant number 16032, and by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with grant number 9180076.

References

1. Garousi, V., Mäntylä, M.V.: When and what to automate in software testing? A multivocal literature review. Inf. Softw. Technol. **76**, 92–117 (2016)
2. Garousi, V., Tasli, S., Sertel, O., Tokgöz, M., Herkiloğlu, K., Arkin, H.F.E., Bilir, O.: Automated testing of simulation software in the aviation industry: an experience report. IEEE Softw. **36**(4), 63–75 (2019)
3. Smart, J.F.: BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle. Manning Publications, New York (2014)
4. North, D.: How to sell BDD to the business. https://skillsmatter.com/skillscasts/923-how-to-sell-bdd-to-the-business. Accessed May 2020
5. Maliackal, Z.: Why we built Gauge. https://blog.getgauge.io/why-we-built-gauge-6e31bb4848cd. Accessed May 2019
6. Lakshmi, T.: "Why I chose Gauge over Cucumber. https://medium.com/@divi6.1990/why-i-chose-gauge-over-cucumber-5b3de478d889. Accessed May 2020
7. Matts, C.: The tragedy of Given-When-Then. https://theitriskmanager.com/2019/04/06/the-tragedy-of-given-when-then/. Accessed May 2019
8. Akbari, A.: Cucumber or Gauge? Things to consider when choosing the best Test Automation Framework for your team. https://www.linkedin.com/pulse/things-consider-when-choosing-best-test-automation-framework-akbari/. Accessed May 2020
9. Garousi, V., Shepherd, D.C., Herkiloğlu, K.: Successful engagement of practitioners and software engineering researchers: evidence from 26 international industry-academia collaborative projects. IEEE Software (2019, in press)
10. Gorschek, T., Garre, P., Larsson, S., Wohlin, C.: A model for technology transfer in practice. IEEE Softw. **23**(6), 88–95 (2006)
11. Janicki, M., Katara, M., Pääkkönen, T.: Obstacles and opportunities in deploying model-based GUI testing of mobile software: a survey. Softw. Test. Verif. Reliab. **22**(5), 313–341 (2012)
12. Raulamo, P., Mäntylä, M.V., Garousi, V.: Choosing the right test automation tool: a grey literature review. In: International Conference on Evaluation and Assessment in Software Engineering, pp. 21–30 (2017)
13. Li, N., Escalona, A., Kamal, T.: Skyfire: model-based testing with cucumber. In: IEEE International Conference on Software Testing, Verification and Validation, pp. 393–400 (2016)
14. Sivanandan, S.: Agile development cycle: approach to design an effective Model Based Testing with Behaviour driven automation framework. In: Annual International Conference on Advanced Computing and Communications, pp. 22–25 (2014)
15. Graham, D., Fewster, M.: Experiences of Test Automation: Case Studies of Software Test Automation. Addison-Wesley Professional, Boston (2012)
16. Garousi, V., Keleş, A.B., Güler, Z.Ö., Balaman, Y.: Executable natural-language test specifications: a test-automation experience report. In: Proceedings of the Turkish National Software Engineering Symposium (UYMS) (2019)
17. Shewchuk, Y., Garousi, V.: Experience with maintenance of a functional GUI test suite using IBM rational functional tester. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, pp. 489–494 (2010)
18. Garousi, V., Eskandar, M.M., Herkiloğlu, K.: Industry-academia collaborations in software testing: experience and success stories from Canada and Turkey. Software Qual. J. **25**(4), 1091–1143 (2017)

19. Şentürk, Ş., Akın, A., Karagöz, A.B., Garousi, V.: Model-based testing in practice: an experience report from the banking domain. In: Proceedings of the Turkish National Software Engineering Symposium (UYMS) (2019)
20. Urul, G., Garousi, V., Urul, G.: Test automation for embedded real-time software: an approach and experience report in the Turkish industry. In: Proceedings of the Turkish National Software Engineering Symposium (UYMS) (2014)
21. Akin, A., Sentürk, S., Garousi, V.: Transitioning from manual to automated software regression testing: experience from the banking domain. In: Asia-Pacific Software Engineering Conference (APSEC), pp. 591–597 (2018)
22. Garousi, V., Felderer, M.: Developing, verifying and maintaining high-quality automated test scripts. IEEE Softw. **33**(3), 68–75 (2016)
23. Lewis, W.E.: Software Testing and Continuous Quality Improvement. CRC Press, Boca Raton (2017)
24. Eldh, S.: On test design. Ph.D. thesis, Mälardalen University (2011)
25. Garousi, V., Coşkunçay, A., Can, A.B., Demirörs, O.: A survey of software engineering practices in Turkey. J. Syst. Softw. **108**, 148–177 (2015)
26. Garousi, V., Zhi, J.: A survey of software testing practices in Canada. J. Syst. Softw. **86**(5), 1354–1376 (2013)
27. Kramer, A., Legeard, B.: Model-Based Testing Essentials-Guide to the ISTQB Certified Model-Based Tester. Wiley, Hoboken (2016)
28. Koopman, P., Plasmeijer, R., Achten, P.: Model-based testing of thin-client web applications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES/RV -2006. LNCS, vol. 4262, pp. 115–132. Springer, Heidelberg (2006). https://doi.org/10.1007/11940197_8
29. Monsma, J.R.: Model-based testing of web applications. MSc thesis, Radboud University, The Netherlands (2015)