Web Test Automation: Insights from the Grey Literature

Filippo Ricca[1][0000−0002−3928−5408] and Andrea Stocco[2][0000−0001−8956−3894]

[1] Università degli Studi di Genova, Italy
filippo.ricca@unige.it
[2] Università della Svizzera italiana, Lugano, Switzerland
andrea.stocco@usi.ch

Abstract. This paper provides the results of a survey of the grey literature concerning best practices for end-to-end web test automation. We analyzed more than 2,400 sources (e.g., blog posts, white-papers, user manuals, GitHub repositories) looking for guidelines by IT professionals on how to develop and maintain web test code. Ultimately, we filtered 142 relevant documents from which we extracted a taxonomy of guidelines divided into technical tips (i.e., concerning the development, maintenance, and execution of web tests), and business-level tips (i.e, concerning the planning and management of testing teams, design, and process). The paper concludes by distilling the ten most cited best practices for developing good quality automated web tests.

Keywords: Web Test Automation · Grey Literature · Best Practices.

1 Introduction

End-to-end (E2E) web testing is one of the approaches used for assuring the correctness of web applications. In this context, the tester verifies the correct functioning of the application under test through automated test scripts. Such scripts automate the set of manual operations that the end-user would perform on the web application's graphical user interface (GUI), such as delivering events with clicks or filling in forms, and they are typically used for regression testing [41]. Thus, test cases become software artifacts that developers write resorting to specific testing frameworks. However, the development of complex test suites requires nontrivial programming skills and domain knowledge of the application under test.

An effective code development process must be driven by guidelines and best practices. To the best of our knowledge, the scientific literature has been neglecting the topic of surveying existing best practices or proposing new ones to produce high-quality test code. Instead, researchers have proposed solutions to mitigate specific issues like test fragility [26], or automated repair [40], which are based mostly on anecdotal findings.

On the other hand, the grey literature—constituted by white-papers, magazines, online blog-posts, question-answers sites, survey results, and technical

reports—is a rich source of documents in which practitioners often share their experience matured on the field, and propose best practices, guidelines, and tips related to different quality aspects of test code. Synthesizing knowledge from the grey literature is a contemporary issue in empirical software engineering research [16]. For instance, works have mined the knowledge by practitioners about how to best select the right test automation tool [34], or to highlight the factors behind the choice of what and when to automate [17].

From our experience, the grey literature is still an unexplored gold mine of guidelines for E2E web test automation. However, such insights are still hidden as practitioners lack both the time and the scientific background to distill the most relevant best practices rigorously. For this reason, we have conducted a survey to help structure, curate, and unify the grey literature in E2E web test automation, to understand what best practices are suggested by practitioners, and what are the challenges reported when they are used.

The main contribution of our work is a taxonomy of best practices for E2E web test automation, composed by a rich set of guidelines about different technical and business-level aspects of the testing and development life cycle. We distilled the set of ten most cited best practices that, according to developers, can improve the quality of automated tests. Our taxonomy can be useful to both practitioners and researchers, who can, respectively, use the most quoted best practices to guide the development of better quality test code, and foster future research in this field.

2 Background

E2E web testing is a type of black-box testing based on the concept of test scenario, i.e., a sequence of steps and actions performed by a user on the application under test's GUI. One or more test cases can be derived from a scenario by specifying the input data and the expected results. The test case execution can be automated through test scripts within specific testing frameworks [24].

Selenium is the de-facto ecosystem [14] to support different kinds of E2E web testing [7,27]. For instance, Selenium IDE is used for quick *exploratory testing* as it allows recording the actions performed by the tester on the web application, from which generated test scripts can be conveniently replayed multiple times. With Selenium WebDriver, on the other hand, test scripts are programmed in a high-level programming language using the framework's APIs. As such, developed test scripts become first-class citizens that developers design, maintain, review, and refactor the same way as the production code. For this reason, WebDriver is used mostly to create complex and large test suites for browser-based *regression test automation*. Finally, Selenium Grid offers services that allow distributing test scripts over multiple browsers and platforms, which is convenient for performing *cross-browser and cross-platform testing*.

Even though the web testing community is highly influenced by the Selenium ecosystem, in this paper we focus on the best practices for web test code devel-

opment and maintenance, regardless of the specific testing framework for which they were originally proposed.

3 Related Work

Researchers have long proposed methodologies and techniques to test web based systems, including testing classical [37,12] and modern [6,5,4,29,30] web apps.

In recent years, the increased popularity of tools like Selenium motivated the research community to study the challenges of E2E test automation and propose solutions to improve the quality of test code produced by such tools [36,24,39]. This literature can be broadly divided into two main categories. A first category pertains to empirical studies in web test automation such as studies on the evolution of web test scripts [21], the differences between the approaches [24], economic perspectives in test automation [33], or other secondary studies on web test automation [9,18,28,22]. A second category, instead, refers to solution-based papers that tackle a specific problem in web test automation, such as robustness of test code [26], automated test repair [20,8,40,25], or test adequacy criteria [32,31] Most of this research is built on anecdotal facts, or on knowledge acquired by studies of the first category.

We recognize two main drawbacks. First, both kinds of works are limited in providing a list of best practices for testers to produce good quality test suites. Second, existing works are either too focused on one single problem or not driven by the current state-of-the-practice.

There are works that gathered the practitioners' opinions to improve the overall quality of the testing process. Gamido and Gamido [13] provide a comparative review of open-source and commercial testing tools to help users select the appropriate software testing tool based on their needs. Rafi et al. [10] report academic and practitioner views on software test automation. While publications are biased by positive results, practitioners agreed that available test automation tools offer a poor fit for their needs and generally disagreed that automated testing can be fully replaced by manual testing. Raulamo-Jurvanen et al. [34] identify 14 different criteria for choosing the right testing tool and highlight that practitioners' judgment is highly influenced by related grey literature. Another study by Garousi [17] aims to characterize what industry wants from academia in software testing, by soliciting testers' challenges during their activities.

To the best of our knowledge, no paper provides a curated list of best practices to drive the development of high-quality *test code*, even less by taking into account the developers' perspective. This paper differs from the existing literature as it distills and summarizes best practices for E2E web test automation that can better inform researchers of the developers' desiderata thereby providing actionable feedback and more awareness when devising future testing approaches.

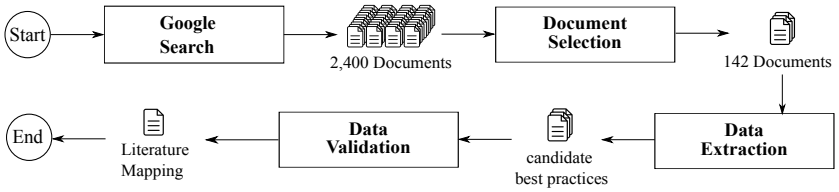

Fig. 1. Overview of the selection procedure

4 Experimental Study

Our study focuses on the *grey literature* for functional testing of web applications with the goals to understand what best practices are suggested by practitioners and, to structure, curate, and unify the grey literature. This section describes the selection procedure we carried out to obtain the relevant documents, which has been designed according to the guidelines proposed by Garousi et al. [15].

4.1 Procedure

Figure 1 graphically illustrates the overall process, which consists of four main phases: (1) Google search, (2) document selection, (3) data extraction, and (4) data validation. In the rest of the section, we provide additional details on each phase.

Google Search To formulate the string for the Google search, the authors identified an initial set of candidate keywords starting from the goal of the study. Each tentative search string was then validated against a list of relevant documents, as suggested in the guidelines by Kitchenham and Charters [23]. This list includes documents that were already known and which were expected to be included in the search results. The process terminated when the authors were satisfied by the search results, i.e., the number of retrieved papers was manageable, all relevant documents known in advance were included, and no candidate relevant keyword was missing in the search string. The final search string is:

```
(("best practices" OR "guidelines" OR "recommendation" OR "tips")
AND
("Selenium" OR "UI" OR "end-to-end" OR "Web" )
AND
("test"))
```

Since we are mainly interested in documents that propose testing guidelines, the first group of terms characterizes words that relate to methods or techniques that have been generally accepted as superior to any alternatives and have become a de-facto standard. The second group of terms defines specific aspects related to the testing phase in the software life cycle, along with Selenium, which is the undisputed framework for browser automation. Finally, we

included the "test" keyword to focus the search on testing-related documents. All relevant documents contained an instance of each keyword from each group (AND operator), whereas keywords within the same group were ORed.

The search was performed from 20 May 2020 to 10 July 2020. For each search query, the first 15 pages of results were scraped, each having 10 documents. The first author conducted 16 queries, which accounted for 2,400 documents that were analyzed overall (150 documents for each query). No more significant documents were found after the 15^{th} page.

Document Selection The Google search is, by construction, very inclusive. This allowed us to collect as many documents as possible in our pool, at the price of having documents that are not directly related to the scope of this study. Accordingly, we defined a set of specific inclusion and exclusion criteria to remove documents not meeting the criteria and ensure that each collected document is in line with the scope of the study.

Inclusion Criteria. First, the document should propose guidelines to help functional test automation of web applications, i.e., such as GUI testing, or acceptance testing. Other kinds of testing such as performance, load, stress, security, or usability testing are not considered. Second, the document should apply to either capture-replay (C&R), programmable, visual, or combinations of these testing approaches. Last, tools' user manuals and presentations are included as long as they specify some guidelines.

Exclusion Criteria. We excluded papers not written in the English language, or that provided guidelines for manual testing or web development. Furthermore, we did not consider videos or books, which are quite difficult to extract information from or to retrieve, respectively. We also discarded websites that required registration for consulting the resource. We excluded sources that provided either too generic, partial guidelines related to only one specific aspect (e.g., documents only explaining the Page Object design pattern), or that were tool-specific and difficult to generalize to other toolsets. We did not consider documents explaining bad practices, even though, in principle, it is possible to infer some best practices, for instance, by the negation of such good practices.

Results of the Document Selection. The studies obtained from the database search were assessed manually by the first author and only those studies that provide direct evidence about the objective of the study were retained. The final number of selected primary documents is 142.

Data Extraction In the data extraction step, the first author read and analyzed in detail the candidate documents, filling out an extraction form with the best practices gathered from each source [23]. A tabular data extraction form was used to keep track of the extracted information. In particular, each row of such form reports a study and each column corresponds to an individual best practice. It is important to highlight that no predefined set of best practices was provided; for each newly retrieved best practice found during the data extraction phase, a new column was added to the form incrementally.

Table 1. Study Selection Process

Search Query	Retrieved Documents	Relevant Documents
best practices selenium test	150	46
best practices UI test	150	32
best practices Web test	150	12
best practices end-to-end test	150	11
guidelines selenium test	150	1
guidelines UI test	150	2
guidelines Web test	150	3
guidelines end-to-end test	150	7
recommendation Selenium test	150	1
recommendation UI test	150	2
recommendation Web test	150	1
recommendation end-to-end test	150	2
tips Selenium test	150	10
tips UI test	150	7
tips Web test	150	2
tips end-to-end test	150	3
Total	2,400	**142**

Data Validation and Taxonomy Construction In the data validation step, the second author independently analyzed each candidate document, with the aim of validating each individual best practice retrieved during the data extraction phase. During this task, 130/142 documents (92%) were marked with a correct and complete extraction. Only 12/142 documents were found to miss some best practices, which were added. The high agreement rate between the authors indicates overall a low degree of subjectivity in the extraction task.

After enumerating all best practices, the authors began the process of creating a taxonomy, following a systematic process [19]. For each best practice, candidate equivalence classes were identified and assigned to descriptive labels. By following a bottom-up approach, the first clustered tags that correspond to similar notions into categories. Then, they created parent categories, in which categories and their subcategories follow specialization relationships.

5 Results

Table 2 illustrates our taxonomy of E2E web test automation best practices from the grey literature. Our study grouped 706 occurrences of best practices into two main categories, namely ***technical aspects*** (80%) and ***business-level aspects*** (20%).

5.1 Technical Best Practices

Technical best practices refer to the development, maintenance, and execution of web tests. That is, testers are already equipped with test requirements, and their task is to translate such requirements into actual test code, with appropriate oracles, or to adapt existing test code to changes and extensions of such requirements, or applications' functionalities.

Table 2. E2E web test automation best practices from the grey literature.

Best Practice	#
TECHNICAL (566)	
Structural Test Script Quality	**163**
Manage the synchronization w/ the web app	67
Keep the tests atomic and short	40
Use appropriate naming and code conventions	31
Focus on reusable test code	25
Test Script Development	**67**
Remove sources of uncertainty (no flakiness)	26
Create tests that are resilient to minor GUI changes	16
Mock external services	15
Write both positive and negative test	10
Monitoring Execution of Test suites and Reporting	**67**
Produce detailed reports	36
Take/use screenshots	17
Use Continuous Integration (CI)	14
Design Patterns	**66**
Use the Page Object Pattern (also Page Factory)	52
Others	14
Locators	**65**
Create robust/proper locators/selectors	48
Preferred locators order	17
Data	**52**
Use data-driven testing	33
Use high-quality test data	19
Test Script Grouping and Ordering	**46**
Make tests independent from each other	34
Group tests, e.g., by functional area	12
Test Execution	**40**
Prioritization	19
Parallelization	14
"Green tests run" policy: All tests must pass	7
BUSINESS-LEVEL (140)	
Planning	**54**
Do not consider test automation as a replacement for manual testing	16
Choose the correct testing framework	15
Mentorships/Experts	12
Test early and test often	11
Design	**46**
Focus on key user flows or process flows or functionalities	24
Understand what test cases to actually automate	14
Test from the end-user perspective	8
Process	**40**
Do not limit to only GUI testing (the testing pyramid)	28
Implement test code review	8
Integrate exploratory testing	4

The most represented subcategory pertains to guidelines on how to achieve a high *structural quality* of the test code (29%). Particularly, the most mentioned tip is careful handling of the synchronization between the web app and the test code [2]. Modern web applications are developed using front-end technologies in which the Document Object Model (DOM) elements are loaded dynamically by the browser and may be ready for interaction at unpredictable time intervals [3]. This is a huge problem for automated testing since there is no universal mechanism to understand when a page is fully loaded and when it is possible to perform actions. As a result, test scripts may encounter ex-

ceptions like `NoSuchElementException`, `StaleElementReferenceException`, or `ElementNotVisibleException`. In Selenium WebDriver, testers can use implicit or explicit waits, which should be preferred to the more generic `Thread.sleep()`. If one fails to place appropriate wait commands in the test code, the associated risks span from having pointless lengthy delays in tests' execution, to having flaky checks due to the waits being non-deterministic. Other best practices pertain to keep the test scripts *atomic* (each test method should concern only one single test scenario), using test *naming* conventions as well as *coding rules*, and focusing on *reusable test code*.

The second most mentioned subcategories pertain to *test development* and *reporting* of the test results. Concerning the former, it is suggested to implement deterministic tests by removing uncertainties that may cause tests to pass/fail nondeterministically [11], as well as implementing GUI-resilient tests, both positive and negative tests, and mock external services to keep the testing environment under full control. Related to the latter, developers suggest providing detailed reporting, making use of screenshots to help visually assess the bugs, and of continuous integration (CI) environments.

Design patterns are suggested as an effective mechanism to isolate the code's functionalities into reusable methods (12%). Developers suggest different design patterns: most of our references mention the Page Object [39], whereas lower occurrences pertain to other patterns such as Bot Pattern, AAA Pattern, and Screenplay Pattern which do not seem yet consolidated within test development.

Other relevant categories pertain to locators (11%). *Locators* are commands that tell the testing framework that GUI elements it needs to operate on. Identification of correct GUI elements is a prerequisite to creating robust test scripts, even if accurate localization of GUI elements can be quite challenging due to the mentioned synchronization issues with the DOM being loaded dynamically. Developers suggest crafting robust locators, which, however, requires in-depth domain knowledge of the web app under test.

Finally, a test suite is cost-effective iff test data are of *high quality* (9%). In a way, a test suite is as weak as the test data it uses, which defines the overall fault-finding capability and hence cost-effectiveness of running it. Among the tips, developers suggest adopting data-driven testing techniques by parameterizing the test cases and using realistic inputs, as well as meaningful real-world combinations that the users may experience.

Other less numerous, yet representative, categories pertain to tests *ordering* (8%) and *execution* (7%). Having independent tests is also a strongly advocated best practice, as well as grouping them, for instance by functional area. Speeding up the feedback to developers by prioritizing the execution of tests that are more immediately impacted by the latest code changes [1] is also a highly suggested guideline. Then, developers care also about performance, and parallelization has been also mentioned as a preferred way to speed up the execution of tests.

5.2 Business-level Best Practices

Business-level aspects are related to the practices of establishing a process that ensures the final quality of the software product and satisfies the customers as well as users. Also, it concerns aspects like resource optimization, communication, cost management, and team building.

The most represented subcategory pertains to *Planning* the process of test code development (39%). The main guidelines in this subcategory are: not considering automation as a replacement for manual testing, choosing the correct/right testing tool/framework for your organization and, hiring a team of experts or a skilled automation engineer.

The second most mentioned subcategory is *Design* (33%), which pertains to guidelines on how design test cases and how to transform them into test code. In this subcategory, the most mentioned tips are the following: (1) focusing on key user flows during test code development, that means to test mainly "happy paths" capturing typical use scenarios and so limit exception testing; (2) creating scenarios and test cases in advance before automating test cases, i.e., having a clear understanding of what test cases to automate, indeed diving straight into automation without a proper test design can be dangerous; (3) conducting testing from the users' perspective, e.g., by getting into the mindset of novice users.

Finally, it is also worth mentioning the best practice of not relying entirely on GUI test automation belonging to the *Process* subcategory. This is one of the main best practices a testing team should consider at first. Ideally, a test suite should be constituted by more low-level unit tests and integration tests than E2E tests running through a GUI (the practical test pyramid[3]). Another best practice that is gaining momentum concerns reviewing the test code [38], similarly to production code. Test code review aims to analyze its quality and to find mismatches or bad practices. For example, a set of tests could be fulfilling their coverage criteria, sufficiently invoking the intended code sections, but if assertions are poorly implemented, the tests will be useless in revealing faults. Lastly, exploratory testing is also suggested as a way to quickly get an intuition of the web app's main functionalities.

5.3 Findings

Based on our analysis, ten best practices emerged as essential for obtaining high-quality test code (Table 3). Nine of them are related to technical aspects, and only one to business-level best practices. This suggests that most sources of grey literature in this domain are predominantly of technological nature. Thus, our final takeaway message to practitioners is to follow, during the planning and implementation of automated tests, at least these top 10 guidelines.

As for researchers, our taxonomy can be useful to foster future research and spot the areas deserving attention. As an example, existing work has been pro-

[3] https://martinfowler.com/articles/practical-test-pyramid.html

Table 3. Top 10 best practices (listed in descending order of references)

Rank	Best Practice	Technical	Business
1	Manage the synchronization w/ the web app	x	
2	Use the Page Object Pattern (also Page Factory)	x	
3	Create robust/proper locators/selectors	x	
4	Keep the tests atomic and short	x	
5	Produce detailed reports	x	
6	Make tests independent from each other	x	
7	Use data-driven testing	x	
8	Use appropriate naming and code conventions	x	
9	Do not limit to only GUI testing (the testing pyramid)		x
10	Remove sources of uncertainty (no flakiness)	x	

posed for creating robust locators [26], automated page objects [39], test independence [4], and test minimization [6]. On the other hand, practitioners suggest removing sources of uncertainties from tests that may cause tests to pass/fail non-deterministically. However, to the best of our knowledge, in the literature, no solutions and tools have been proposed to detect and solve flaky web tests, even less to tackle the synchronization problem.

5.4 Threats to Validity

The main threat to the *internal validity* of this work is the possibility of introducing bias when selecting and classifying the surveyed documents included in our study. We may have missed relevant documents that are not captured by our list of terms. In this paper, multiple Google searches have been performed with no use of private browsing mechanisms (i.e., Google's Chrome Incognito mode) that prevent saving browsing history, cookies, and other site data. Thus, it is possible that Google's search engine provided us with the most suitable results based on our preferences, or our previous searches. We will refine our search procedure in our future work. We do not claim that our survey captures all relevant grey literature; yet, we are confident that the included documents cover the major related best practices.

Concerning the best practices, we manually classified all candidate guidelines into different categories. There is no ground-truth labeling for such a classification. To minimize classification errors, we added a data validation phase to the selection procedure. To reduce the subjectivity involved in the task, the authors followed a systematic and structured procedure, with multiple interactions.

Concerning the *external validity*, we overviewed only the documents available from Google in a specific time frame, and our taxonomy may not generalize to different documents. Also, other relevant classes of best practices might be unrepresented or underrepresented within our taxonomy. Nevertheless, we tried to mitigate this threat by selecting a diverse range of documents using different search queries.

Concerning *reproducibility*, all our results and references are available in our replication package [35].

6 Conclusions and Future Work

The increasing interest of developers and industry around web test automation has fostered a large amount of software engineering research. Novel analysis and testing techniques are being proposed every year, however, without a centralized knowledge base of best practices by professionals, it is difficult to fairly design and implement solutions, or to assess research advancements.

Towards filling this gap, in this paper, we presented a taxonomy of best practices for E2E web test automation derived from an analysis of the grey literature. We manually analyzed several hundreds of documents from which we retrieved many best practices, pertaining to different technical and business-level categories. Moreover, our taxonomy can be used to foster future research in this field, and spot the areas that merit the greatest attention.

As part of our ongoing and future work, we plan to include other sources of grey literature (e.g., arXiv) as well as conducting a thorough literature review of the academic literature that may be useful to validate and improve our taxonomy of best practices. Triangulating our results through surveys and semi-structured interviews with developers is also part of the plan to validate our findings. Moreover, we plan to rate our sources with a credibility score, so as to establish the reliability of a website prior to the data extraction phase.

References

1. Alimadadi, S., Mesbah, A., Pattabiraman, K.: Hybrid DOM-Sensitive Change Impact Analysis for JavaScript. In: Proc. of the 29th European Conference on Object-Oriented Programming. ECOOP '15, vol. 37, pp. 321–345 (2015)
2. Alimadadi, S., Mesbah, A., Pattabiraman, K.: Understanding asynchronous interactions in full-stack javascript. In: Proc. of the 38th International Conference on Software Engineering. p. 1169–1180. ICSE '16, ACM (2016)
3. Alimadadi, S., Sequeira, S., Mesbah, A., Pattabiraman, K.: Understanding javascript event-based interactions. In: Proc. of the 36th International Conference on Software Engineering. p. 367–377. ICSE '14, ACM (2014)
4. Biagiola, M., Stocco, A., Mesbah, A., Ricca, F., Tonella, P.: Web test dependency detection. In: Proc. of 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 12 pages. ESEC/FSE 2019, ACM (2019)
5. Biagiola, M., Stocco, A., Ricca, F., Tonella, P.: Diversity-based web test generation. In: Proc. of 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 12 pages. ESEC/FSE 2019, ACM (2019)
6. Biagiola, M., Stocco, A., Ricca, F., Tonella, P.: Dependency-aware web test generation. In: Proc. of 13th IEEE International Conference on Software Testing, Verification and Validation. p. 12 pages. ICST '20, IEEE (2020)
7. Cerioli, M., Leotta, M., Ricca, F.: What 5 million job advertisements tell us about testing: a preliminary empirical investigation. Proc. of the 35th Annual ACM Symposium on Applied Computing (2020)

8. Choudhary, S.R., Zhao, D., Versee, H., Orso, A.: WATER: Web Application TEst Repair. In: Proc. of 1st International Workshop on End-to-End Test Script Engineering. pp. 24–29. ETSE '11, ACM (2011)
9. Doğan, S., Betin-Can, A., Garousi, V.: Web application testing: A systematic literature review. Journal of Systems and Software **91**, 174–201 (2014)
10. Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Petersen, K., Mäntylä, M.V.: Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: 2012 7th International Workshop on Automation of Software Test (AST). pp. 36–42 (2012). https://doi.org/10.1109/IWAST.2012.6228988
11. Eck, M., Palomba, F., Castelluccio, M., Bacchelli, A.: Understanding flaky tests: The developer's perspective. In: Proc. of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 830–840. ESEC/FSE '19, ACM (2019)
12. Elbaum, S., Karre, S., Rothermel, G.: Improving web application testing with user session data. In: 25th International Conference on Software Engineering, 2003. Proceedings. pp. 49–59 (2003)
13. Gamido, H., Gamido, M.: Comparative review of the features of automated software testing tools. International Journal of Electrical and Computer Engineering **9**, 4473–4478 (10 2019). https://doi.org/10.11591/ijece.v9i5.pp4473-4478
14. García, B., Gallego, M., Gortázar, F., Munoz-Organero, M.: A survey of the selenium ecosystem. Electronics **9**, 1067 (06 2020)
15. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. IST **106**, 101–121 (2019)
16. Garousi, V., Felderer, M., Mäntylä, M.V., Rainer, A.: Benefitting from the grey literature in software engineering research (2019)
17. Garousi, V., Mäntylä, M.V.: When and what to automate in software testing? a multi-vocal literature review. IST **76**, 92–117 (2016)
18. Garousi, V., Mesbah, A., Betin-Can, A., Mirshokraie, S.: A systematic mapping study of web application testing. IST **55**(8), 1374–1396 (2013)
19. Gyimesi, P., Vancsics, B., Stocco, A., Mazinanian, D., Árpád Beszédes, Ferenc, R., Mesbah, A.: BugJS: A benchmark and taxonomy of javascript bugs. Software Testing, Verification And Reliability (2020)
20. Hammoudi, M., Rothermel, G., Stocco, A.: WATERFALL: An incremental approach for repairing record-replay tests of web applications. In: Proc. of 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 751–762. FSE '16, ACM (2016)
21. Hammoudi, M., Rothermel, G., Tonella, P.: Why do record/replay tests of web applications break? In: Proc. of 9th International Conference on Software Testing, Verification and Validation. pp. 180–190. ICST '16, IEEE (2016)
22. Imtiaz, J., Sherin, S., Khan, M.U., Iqbal, M.Z.: A systematic literature review of test breakage prevention and repair techniques. IST **113**, 1–19 (2019)
23. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (2007)
24. Leotta, M., Clerissi, D., Ricca, F., Tonella, P.: Approaches and tools for automated end-to-end web testing. Advances in Computers **101**, 193–237 (01 2016)
25. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Using multi-locators to increase the robustness of web test cases. In: Proc. of 8th IEEE International Conference on Software Testing, Verification and Validation. pp. 1–10. ICST 2015, IEEE (2015)

26. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Robula+: An algorithm for generating robust xpath locators for web testing. Journal of Software: Evolution and Process **28**, 177–204 (03 2016)
27. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: PESTO: Automated migration of DOM-based web tests towards the visual approach. Software Testing, Verification And Reliability **28**(4) (2018)
28. Li, Y.F., Das, P.K., Dowe, D.L.: Two decades of web application testing–a survey of recent advances. Information Systems **43**, 20–54 (2014)
29. Mesbah, A., van Deursen, A., Lenselink, S.: Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web **6**(1), 3:1–3:30 (2012)
30. Mesbah, A., van Deursen, A., Roest, D.: Invariant-based automatic testing of modern web applications. IEEE TSE **38**(1), 35–53 (2012)
31. Mirshokraie, S., Mesbah, A., Pattabiraman, K.: Efficient javascript mutation testing. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. pp. 74–83 (2013)
32. Mirzaaghaei, M., Mesbah, A.: Dom-based test adequacy criteria for web applications. In: Proc. of the 2014 International Symposium on Software Testing and Analysis. p. 71–81. ISSTA '14, ACM (2014)
33. Ramler, R., Wolfmaier, K.: Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost. In: Proc. of 1st International Workshop on Automation of Software Test. pp. 85–91. AST '06, ACM (2006)
34. Raulamo-Jurvanen, P., Mäntylä, M., Garousi, V.: Choosing the right test automation tool: A grey literature review of practitioner sources. In: Proc. of the 21st International Conference on Evaluation and Assessment in Software Engineering. p. 21–30. EASE '17, ACM (2017)
35. Replication Package. https://github.com/riccaF/sofsem2021-replication-package-material/ (2020)
36. Ricca, F., Leotta, M., Stocco, A.: Three open problems in the context of e2e web testing and a vision: Neonate. Advances in Computers (01 2018)
37. Ricca, F., Tonella, P.: Analysis and testing of web applications. In: Proc. of the 23rd International Conference on Software Engineering. pp. 25–34. ICSE '01 (2001)
38. Spadini, D., Palomba, F., Baum, T., Hanenberg, S., Bruntink, M., Bacchelli, A.: Test-driven code review: An empirical study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 1061–1072 (2019)
39. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: APOGEN: Automatic Page Object Generator for Web Testing. Software Quality Journal **25**(3), 1007–1039 (Sep 2017)
40. Stocco, A., Yandrapally, R., Mesbah, A.: Visual web test repair. In: Proc. of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 503–514. ESEC/FSE 2018, ACM (2018)
41. Tonella, P., Ricca, F., Marchetto, A.: Recent advances in web testing. Advances in Computers **93**, 1–51 (2014)