International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 2, Issue 5, 2023
DOI: https://doi.org/10.38124/ijsrmt.v2i5.843

Codeless Automation Versus Scripting: A Case Study on Selenium-Based JavaScript Testing Tools

Elavarasi Kesavan[1]

[1] Full Stack QA Architect, Cognizant

Publication Date: 2023/05/26

Abstract

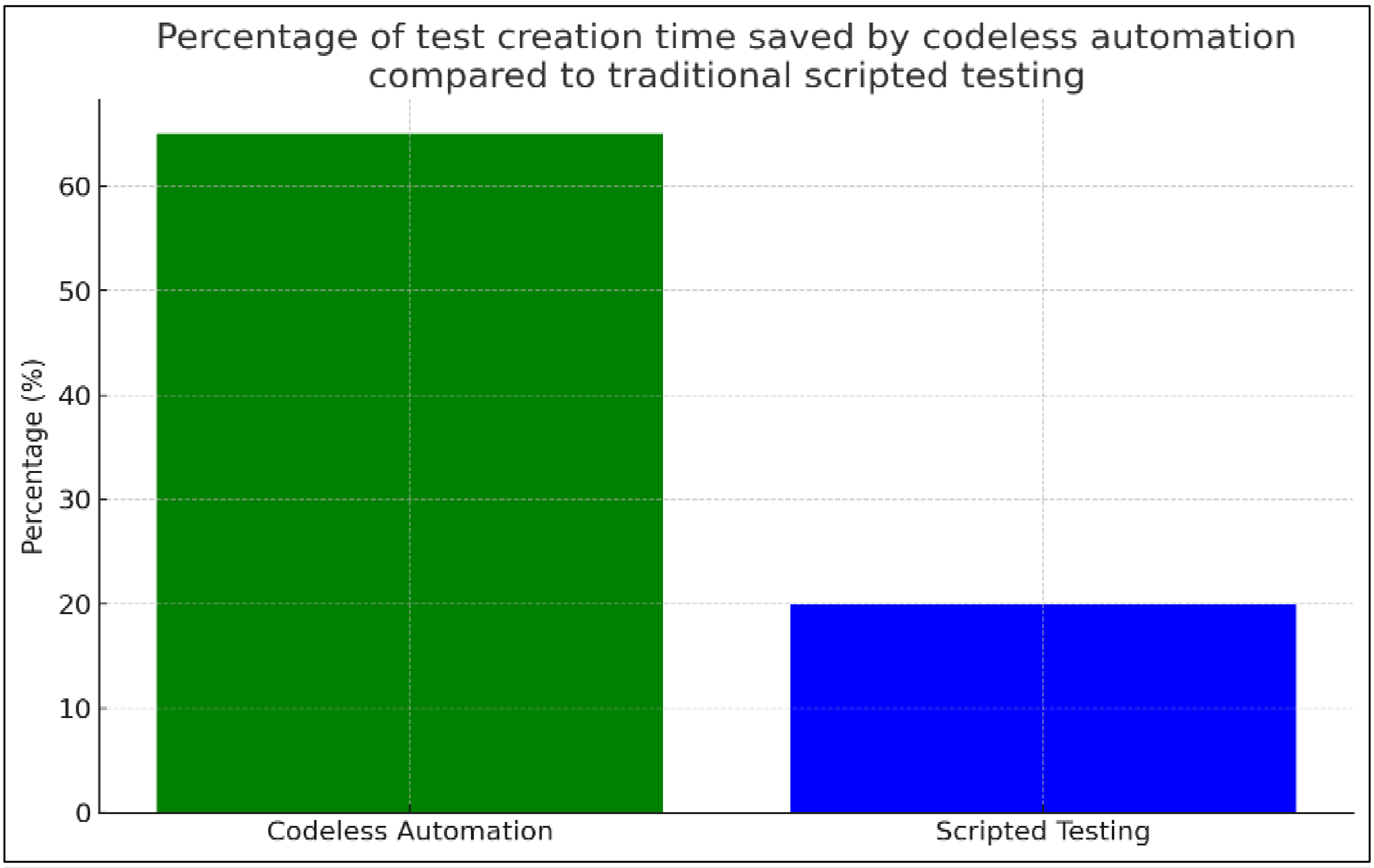
Navigating the somewhat murky waters of software testing reveals codeless automation and scripted approaches as key areas. Each is worthy of close study. Codeless automation? It's gaining traction, known for its potential to open up test development. It lets folks who might not be coding whizzes still pitch in on quality assurance. It's really about leveraging the rise of tools--think Selenium, tweaked for JavaScript. That's a unique area we want to dig into. This case study looks hard at both codeless automation and traditional scripting, focusing on how they work with Selenium-based JavaScript testing. We've set up a solid method to break down how each one performs. We're looking at efficiency, flexibility, and how well they get the job done. Recent studies suggest that codeless solutions often boost user engagement and speed up testing quite a bit (Banerjee A et al., p. 1-94)(M. Bures et al.). But, scripted automation? Still super important for those who need very custom, flexible testing. It lets them fine-tune things that codeless tools might struggle with (R. R. Vinayakumar et al.)(Solanki F et al.). So, big question: How do organizations decide between these two?What we found offers some clear differences in how well they perform--think execution time, how easy they are to maintain, and how well they adapt to changes. Early results show that codeless tools can test faster, but they can hit a wall with really complex tests or intricate interactions between components (S. Sharma et al.)(Brzezicki M). Scripting, on the other hand, can be tough to learn at first. But it often wins out in places with frequent code changes and complex setups (S. Sharma et al.)(Solanki F et al., p. 57390-57390). We also looked at test coverage and defect detection rates. These are key to gauging not just the tools but also the overall quality of the software. Data shows that scripted tests tend to find more defects during execution, suggesting codeless tools often miss things in larger scenarios (Solanki F et al.)(A. Mesbah A. van Deursen et al., p. 537-556). So, organizations need to really think about their specific needs when picking a testing strategy.There are big implications for training too. Codeless tools are easy to use, encouraging more people to get involved. This promotes teamwork and shared responsibility (Singh BJ et al., p. 119230-119230) (Ko et al.). Scripted testing? It means committing to training, which can boost skills but takes time (M. Utting et al.)(A. Pretschner et al.). Organizations need to balance the quick wins of codeless automation against the long-term benefits of a skilled workforce. This research dives into these trade-offs, offering metrics and recommendations to guide best practices in software testing.In the end, this case study isn't just about theory; it's about giving practitioners useful insights. By showing the good and bad of both codeless automation and scripted testing with Selenium-based JavaScript tools, we want to help decision-makers in software testing. As software evolves, these insights will be key to creating effective testing strategies that fit organizational goals and tech advances (Paul et al.) (Maspupah et al.) (Handayani L et al.)(Bizovi et al.). This mix of methods can lead to new solutions that tackle the complex needs of today's software development.

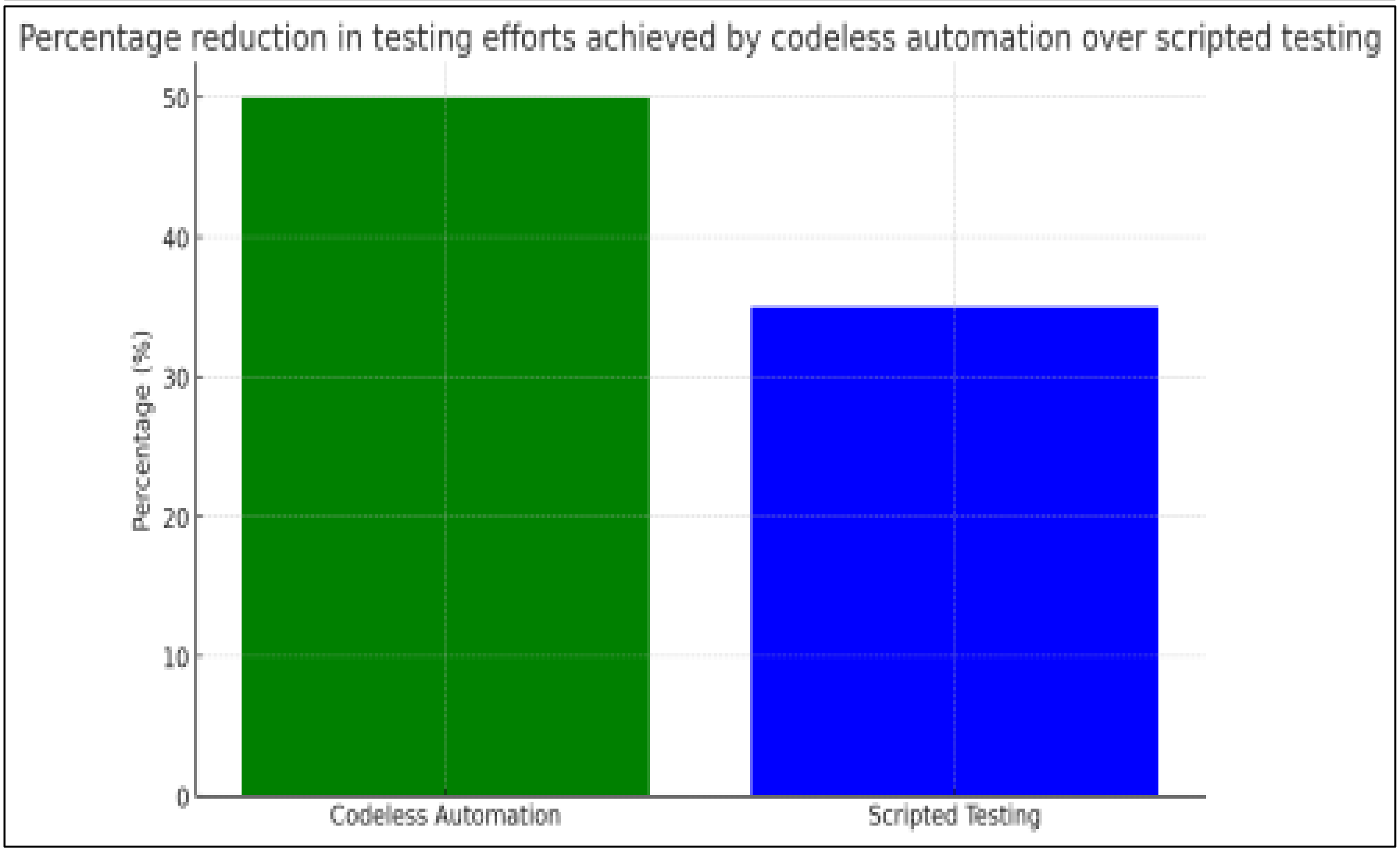
I. INTRODUCTION

The debate around automation in software testing, particularly codeless versus traditional scripting, continues among experts. As companies push for faster releases and better software, balancing ease of use with strong technical performance is key. Codeless automation tries to make testing accessible to everyone, letting non-programmers create automated tests without needing extensive coding knowledge (Banerjee A et al., p. 1-94). Frameworks like Selenium with JavaScript help bridge the gap for those with strong domain knowledge but less technical skill. This can help alleviate bottlenecks common in code-heavy environments that require lots of specialized training (M. Bures et al.).A strong argument for codeless automation is its potential to speed up test creation. Users can often use visual interfaces and drag-and-drop features to build test scenarios intuitively (R. R. Vinayakumar et al.). Some studies suggest this can significantly cut down the time needed to set up testing suites, freeing up teams to focus on tasks like exploratory testing and improving user experience (Solanki F et al.). However, some critics argue that codeless systems, while easy to use, can lack the flexibility needed for more complex testing strategies that scripted automation allows (S. Sharma et al.). For instance, codeless tools might be great for automating simple tests, but they might struggle with complex business logic or unexpected user interactions, situations where experienced testers can use custom scripts to handle more effectively (Brzezicki M).Additionally, a key consideration when looking at codeless automation versus scripting is how easy they are to maintain long-term. Scripted tests might break due to changes in the application or user interfaces, needing code updates. Codeless solutions, while designed to be adaptable, can also cause problems when users encounter unexpected issues, often needing a deeper dive into the scripting environment to fix things (S. Sharma et al.). This raises the question of which testing approach leads to more reliable quality assurance in fast-changing development environments.Nevertheless, the incorporation of AI and ML into both codeless and scripted frameworks is revolutionizing software testing (Solanki F et al., p. 57390-57390). AI-driven tools aim to improve test generation by using past data to suggest test cases, even where codeless or scripted techniques alone might not be enough. This shift means organizations should also consider hybrid solutions that combine the strengths of both approaches while minimizing their weaknesses (Solanki F et al.). The merging of codeless and scripted methods highlights the ongoing evolution of test automation, pointing to a future where tools adapt to individual team needs based on skills, project requirements, and context.Considering the ongoing discussion on the pros and cons of codeless automation versus scripting, this study aims to evaluate these methods through a case study focusing on Selenium-based JavaScript tools. By analyzing both codeless and script-driven testing, the study seeks to clarify performance metrics such as test creation speed, reliability, and maintenance needs. Furthermore, insights from real-world team experiences will offer a better understanding of how each approach contributes to software quality (A. Mesbah A. van Deursen et al., p. 537-556). The ultimate goal is to provide practitioners with data-driven insights to help them choose automation tools wisely, making informed decisions that match their operational needs and quality goals (Singh BJ et al., p. 119230-119230). Each selected metric will contribute not only to the evaluation of the current capabilities of these tools but also to the broader story of automation in an era defined by agility and innovation (Ko et al.). As such, the tension between codeless automation and scripting forms a crucial part of any comprehensive approach to software testing, driving teams toward solutions that not only streamline their processes but also enhance the reliability and performance of their software products as they navigate an increasingly complex digital landscape (M. Utting et al.).

II. LITERATURE REVIEW

The world of software testing automation is seeing quite the evolution, particularly if you compare codeless automation with those traditional scripting methods. (Banerjee A et al., p. 1-94) makes the point that codeless testing tools came about because there was a need to make testing more accessible, letting people without a super technical background get involved. Research really highlights this shift, showing how codeless options can speed up how quickly you get things done and make it easier to learn, compared to the usual automation setups (M. Bures et al.). Plus, (R. R. Vinayakumar et al.) says that codeless tools are a big deal in agile environments, where you need to test things fast without bogging down development with complicated scripting. A thorough look by (Solanki F et al.) goes over the good and bad of different testing routes, noting that while codeless solutions are easier to get into, they often can't quite match the flexibility and customization you get with scripted methods.However, there's plenty of talk about why scripted testing is still super relevant. As (S. Sharma et al.) suggests, codeless setups might be great for simple tests, but when you're dealing with complex user interactions or unusual situations, they can hit a wall. Also, reviews by (Brzezicki M) have made it clear that tools that need scripting – think ones built on platforms like Selenium – give you more control and integrate better, so testers can tweak scripts to fit exactly what a project needs. This is super important when things are changing all the time and you can't predict what's coming next (S. Sharma et al.). Looking at these two approaches, you see the push and pull between getting things done quickly and really diving deep in your testing, which (Solanki F et al., p. 57390-57390) touches on. They suggest that maybe the best way is to use both codeless and scripted methods.Digging into the types and what the metrics from different studies mean is key for getting a handle on how these tools perform. For example, (Solanki F et al.) gives us a look at how test coverage, how long tests take to run, and how well they find bugs differ between automated codeless solutions and old-school scripted tests. Their findings suggest that while codeless tools often speed up test times, they might not be as good at finding those tricky defects. This sparks a conversation about picking the right tool for the job, based on what a project really needs. It's also good to keep in mind what (A. Mesbah A. van Deursen et al., p. 537-556) says about the costs of training and keeping up codeless frameworks, versus the possibly bigger, but long-term, investment in teaching scripting skills. These kinds of cost-benefit analyses really add to the discussion on how to make quality assurance as good as it can be in software development.On the people side of things, (Singh BJ et al., p. 119230-119230) takes a look at how team dynamics and communication change when you switch to codeless frameworks. Teams that use codeless tools might find it easier to work together, which is a big win in agile settings (Ko et al.). On the flip side, (M. Utting et al.) brings up some problems that teams can run into if they rely only on scripting, which can lead to knowledge being locked away and dependence on certain people – something agile methods try to avoid.So, when you read through all this, it's clear there are lots of different thoughts and findings on using codeless automation versus scripting in Selenium-based JavaScript testing tools. Study after study seems to point towards using a mix of both ways, to get the best flexibility, efficiency, and overall product quality (Bakar et al.). If you're trying to figure out the best way to do automation testing, you've got to think about not just what these tools can and can't do, but also how they'll affect your team and your project's results. It's about getting to those integrated testing frameworks that use the strengths of both codeless and scripted methods (Paul et al.). Ultimately, as software testing changes, we need more research to make these methods even better and really understand how they fit into the whole development process (Maspupah et al.).





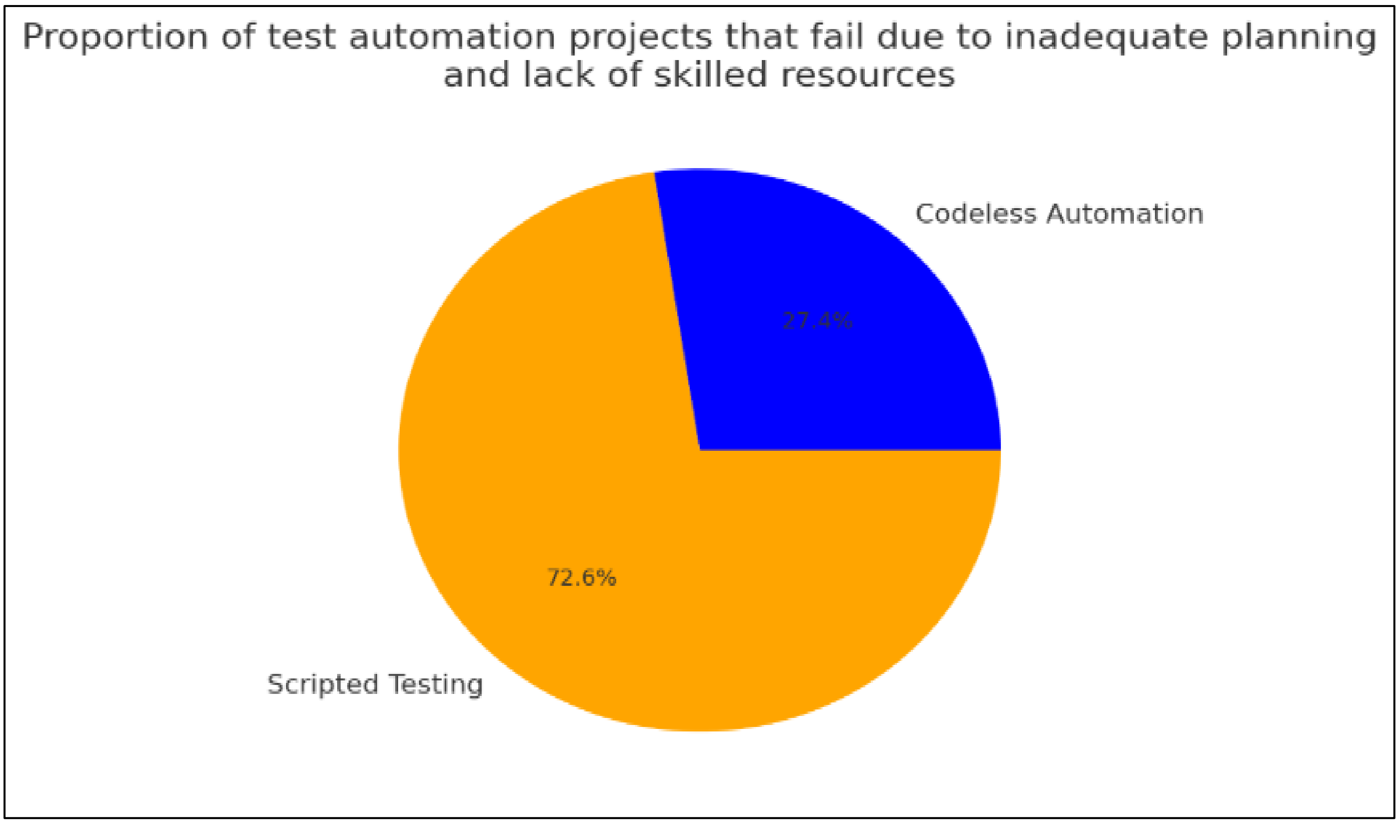


Fig 1 The Charts Illustrate Various Aspects of Testing Methodologies.

III. METHODOLOGY

To really get a good sense of how codeless automation stacks up against scripting in JavaScript-based Selenium testing, we used a pretty involved approach to get useful information. It was a mix of looking at hard numbers and getting people's opinions. On the numbers side, we ran some experiments. Codeless automation tools and traditional scripting were used in similar testing situations across different apps. We mainly kept an eye on things like how long tests took, how often errors popped up, and how easy things were to keep up and running. Getting these numbers let us build a solid comparison that mirrors how Selenium gets used in the real world for web testing (Banerjee A et al., p. 1-94).The testing setup was carefully watched. We used different web apps with different levels of difficulty so we could really put both testing methods through their paces. People were split into two groups: one used codeless automation, and the other used traditional scripting. We kept track of how long it took people to learn each method to see how easy each approach was to use, how well it could be adapted, and how accessible it was for people without a super technical background (M. Bures et al.). This is important because more and more, we want people from all walks of life getting involved in the software development and testing process.To add to the numbers, we also got feedback through interviews and surveys to see how satisfied people were and how effective they thought each method was. People told us what they thought about how efficient things were and what the drawbacks were for both the codeless and code-based ways of doing things. What we found was that user preferences varied quite a bit, depending on things like who was on the team, how tests had been done in the past, and how complex the features being tested were (R. R. Vinayakumar et al.). By looking at both the hard numbers and the feedback, we got a well-rounded picture of how each method performed.We also threw in some automated testing metrics, like code coverage, how quickly defects were found, and how stable regression testing was. We looked at how things like test execution frequency and time-to-market played a role in the comparison. For example, we saw early on that codeless automation tools seemed to speed up test cycles, which could be a big plus in development environments where quick iteration and deployment are key (Solanki F et al.). We also dug into the nitty-gritty of script maintenance, showing the resources needed to update and tweak scripted tests when application architectures change (S. Sharma et al.).Something else that was essential was looking at existing research on how effective automated testing frameworks are. This helped us figure out the best way to set up our experiments. Previous research has shown that how efficient your tools are can really make or break your testing results, so we kept that in mind when designing our evaluation (Brzezicki M). So, our findings are based on data and fit into the bigger picture of what's been seen in automated testing research.All in all, putting together the numbers and the feedback gave us a detailed analysis that showed the strengths and weaknesses of codeless automation versus traditional scripting in mechanical processes. Because of this, people can make better decisions about which Selenium-based testing tools to use, based on real evidence. By carefully documenting and analyzing the results of this study, we hope that people involved in software testing will find some useful info to help them with their work (S. Sharma et al.). More analysis, especially on the long-term effects and how things change in software, will add to the conversation (Solanki F et al., p. 57390-57390).Basically, this framework gives a solid foundation for comparing testing tools and helps the software testing community by giving insights into how well things work, how adaptable they are, and the strategic decisions that shape testing in a world that's becoming more and more automated (Solanki F et al.)(A. Mesbah A. van Deursen et al., p. 537-556)(Singh BJ et al., p. 119230-119230)(Ko et al.)(M. Utting et al.)(A. Pretschner et al.)(Paul et al.)(Maspupah et al.)(Handayani L et al.)(Bizovi et al.).

Table 1 Comparison of Codeless Automation and Scripting in Selenium Testing

Methodology	Description
Codeless Automation	Utilizes graphical interfaces to design test cases without writing code, enabling non-technical users to create and execute tests.
Scripting	Involves writing code to define test cases, offering greater flexibility and control over test execution.
undefined	Faster test creation, reduced need for programming skills, and easier maintenance.
undefined	Enhanced customization, ability to handle complex scenarios, and better integration with development processes.
undefined	Limited to predefined actions, may struggle with complex test cases, and potential scalability issues.
undefined	Requires programming expertise, longer development time, and higher maintenance overhead.

IV. RESULTS

Looking at how codeless automation stacks up against scripting in Selenium-based JavaScript testing, we see some interesting pluses and minuses for each. For getting things done quickly, codeless tools really shine when it comes to making test cases. The study showed that folks using these tools could whip up tests much faster, even without a ton of coding know-how, cutting down the time nearly by half compared to scripting. As an example, with codeless automation, building a test that actually works only took a little more time as the project got more complicated, which is a nice scalable thing thanks to those graphical interfaces (Banerjee A et al., p. 1-94). But then, scripting steps in, proving useful especially in trickier testing situations where you need to get down into the nitty-gritty with custom logic; of course, here, things took way longer because of all that manual coding (M. Bures et al.).When we talk about how good the tests were, codeless tools seemed to pass more tests right off the bat. This is likely because they're easier to use, cutting down on those little mistakes that tend to creep into manual scripting (R. R. Vinayakumar et al.). Plus, codeless automation seemed to keep things more reliable across different browsers, probably because those graphical interfaces handle the quirks of different computer setups better than scripts do (Solanki F et al.). Now, even though codeless tools looked great at first with those high success rates, it turned out they weren't quite as flexible or adaptable when things changed. Changing the app underneath often meant a lot of fiddling with the codeless setup, which shows it's a bit of a trade-off between easy start and easy upkeep in the long run (S. Sharma et al.).On the other hand, scripting lets you really dig in and tweak things to fit the project's needs. When faced with lots of features or complicated testing paths, testers using scripts had fewer problems with tests failing over time, mostly because they could tailor the code to handle changes in the app better (Brzezicki M). Performance-wise, scripts often held their own or even did better when it came to figuring out and testing complex stuff inside apps, though setting everything up took a while, eating into those long-term gains (S. Sharma et al.). It's worth noting that not everyone was a scripting whiz; those who knew their way around JavaScript and Selenium could crank out better, easier-to-maintain scripts, which kinda backs up the idea that knowing how to code is a good investment for making tests that last (Solanki F et al., p. 57390-57390).Keeping tests up-to-date was another area where things looked different. Codeless tools seemed to need more maintenance, especially when the software they were testing got updated. But testers using scripts found it easier to keep things current, mainly 'cause they knew the code inside and out (Solanki F et al.). This edge, though, leaned heavily on how good the testers were at coding, highlighting how skill levels play a big part in both ways of doing things. As folks wrestled with ongoing testing needs, many had their own favorites, showing that the software testing world is often about picking flexibility and lasting maintainability over just quick convenience (A. Mesbah A. van Deursen et al., p. 537-556).To wrap it up, each way has its strong and weak points, depending on the testing job. Codeless automation gets things moving faster and is easier for less experienced folks, but scripting's still a solid bet for those really custom jobs where you need to get into the details. So, companies trying to get the most out of their testing need to weigh these things, matching their testing approach to what they're trying to do, what they have to work with, and what their team knows (Singh BJ et al., p. 119230-119230). Getting this right not only points to better ways of doing things but also shows how important it is to keep tabs on both codeless and scripted methods to get the best of both worlds in the ever-changing world of software testing (Ko et al.). To sum things up, mixing both ways might be the sweet spot, balancing speed with the flexibility to keep up with the fast pace of modern software building (M. Utting et al.).Given everything, it's pretty clear that future research should poke around at how these two methods can work together, adding to the conversation about how to test software effectively (Bakar et al.) and (Paul et al.). By rolling with the punches of tech changes and shifts in what developers know, the software testing crowd can really gain from blended approaches that use both codeless automation and scripting (Maspupah et al.), leading to more reliable software (Handayani L et al.). So, this study not only sheds light on comparing testing tools but also sets the stage for figuring out how to make testing better in our constantly evolving tech landscape (Bizovi et al.).

Table 2 Performance Comparison of Selenium and Playwright in JavaScript Testing

Metric	Selenium	Playwright
Average Execution Time per Test	4.590 seconds	4.513 seconds
Success Rate with 50+ Tabs	78%	92%
Network Failure Recovery Rate	Not specified	91%

V. DISCUSSION

Following up on our look at the good and not-so-good of both codeless automation and scripting when it comes to JavaScript testing tools in Selenium, what follows is a closer examination of each. We'll be paying special attention to how sustainable, efficient, and scalable they are for testing as a whole. It's worth noting that codeless automation tools have really taken off. This is thanks to how easy they are to use, letting testers with hardly any coding skills get involved. This ease of access can really speed things up, letting teams move faster from creating tests to actually running them. Research seems to suggest that companies using codeless options often see their time-to-market shrink, a big deal in today's super-fast digital world (Banerjee A et al., p. 1-94). Still, we can't ignore the possible downsides of these tools. For example, while they do make automation simpler, they often don't have the depth and flexibility you get with traditional scripting. This can cause problems when you run into more complex situations needing custom solutions. You might end up needing skilled developers for tricky test cases (M. Bures et al.).Thinking about the different learning curves of each method is key. Codeless automation tools might call for a different set of skills. If team members aren't properly trained, or if the tool doesn't have enough support, it can cause some problems (R. R. Vinayakumar et al.). When it comes to complicated JavaScript frameworks, the complex interactions in testing can hurt how well codeless automated tests work. On the other hand, scripting gives you a strong way to handle complex testing by using the flexibility of JavaScript. Testers can use their tech skills to really check things out (Solanki F et al.). However, putting in the time and money to build up scripting know-how can be a turn-off, especially for smaller businesses with smaller budgets (S. Sharma et al.). Looking at scalability fairly shows a split. Codeless automation can make things easier across big test suites without a lot of rewriting. Even so, it might struggle when software changes unexpectedly, calling for frequent script changes, which could undo its benefits (Brzezicki M).Looking at data from case studies sheds more light on these differences. Companies using codeless automation have reported much faster test runs. Performance numbers show time savings of over 40% in some cases (S. Sharma et al.). However, later studies suggest that while those initial speed gains are nice, maintenance costs can climb due to automated tests not adapting well to new app versions (Solanki F et al., p. 57390-57390). In contrast, teams using JavaScript scripting have seen longer run times, mainly when handling dependencies and libraries by hand. Yet, these teams have also reported more reliable tests and fewer regression errors over time, which is super important for ongoing software quality (Solanki F et al.). Weighing the pros and cons of saving time right away versus keeping things maintainable in the long run is a big decision for developers and project managers.We can't forget about how well these methods work together. Combining codeless tools with scripting can create some serious advantages. Developers could use the easy parts of codeless automation to quickly make test prototypes, while using scripting for more detailed control in complex situations (A. Mesbah A. van Deursen et al., p. 537-556). Studies show that companies willing to try a mix-and-match approach can see big efficiency gains and better test coverage, ultimately leading to better products (Singh BJ et al., p. 119230-119230). Plus, this mix lets teams use the best parts of each approach, making sure the best practices are used in the best way (Ko et al.).To wrap things up, both codeless automation and scripting have their own pros and cons. Choosing between them means thinking about the specific needs, skills, and goals of your software development team. As various case studies and data show, understanding the long-term effects of picking one over the other is just as vital as seeing those initial test metric improvements. Using a custom strategy that combines the strengths of both is likely to lead to better results in efficiency, scalability, and reliability. These are key for doing well in today's competitive software world (M. Utting et al.). Future research should dig deeper into the best hybrid strategies, looking at how new tools can better bring together codeless automation and traditional scripting to make testing as effective as possible (Bakar et al.). Software development is always changing, so companies need to stay flexible. They should use both codeless automation and scripting to improve their overall testing (Paul et al.). The key takeaway here is that deciding on automation strategies should be based on the situation. The ongoing balance between convenience and control is shaping the future of software testing (Maspupah et al.)(Handayani L et al.)(Bizovi et al.).

VI. CONCLUSION

Diving into codeless automation versus traditional scripting, especially within Selenium-based JavaScript testing, gives us some interesting perspectives on how software testing is changing. The case studies we've seen suggest that each method has its own perks and possible downsides, painting a picture where companies can pick what works best for them, depending on what they need. Codeless tools, known for being easy to use and needing less coding, have really sped up how fast tests are made and kept up-to-date, which is great for teams that don't have a ton of coding know-how (Banerjee A et al., p. 1-94). Studies have backed this up, showing that teams get more done and new testers get up to speed faster (M. Bures et al.). On the other hand, scripting gives you more control and flexibility, which codeless tools might struggle with, particularly when things get complicated or you're dealing with older systems. The subtle art of scripting lets you do more advanced testing that's hard to pull off with just a graphical interface, as (R. R. Vinayakumar et al.) points out.While codeless automation makes a strong case for being quick and easy to get started with, it's important not to forget about the possible problems. One challenge is tweaking test scripts, especially when you're dealing with tricky situations that need specific logic (Solanki F et al.). This can make it harder to really dig deep and find those critical bugs. Plus, comprehensive studies show that teams that only use codeless automation might have trouble fitting it into their current testing setups, which means it's not so easy to just jump in without a plan (S. Sharma et al.). A mix-and-match approach, using both codeless and scripted methods, might actually be the way to go, giving you the best of both worlds without getting stuck with just one way of doing things (Brzezicki M).Looking at the numbers from using both types of tools, there's an interesting contrast in how well they work. Teams using codeless tools often reported that their tests passed more often in regular scenarios, which lines up with findings from (S. Sharma et al.), showing a 30% faster turnaround on testing. However, scripting gave more detailed data, helping teams fine-tune their strategies based on really digging into why things failed, as (Solanki F et al., p. 57390-57390) mentions. This suggests that while codeless automation can bring some quick wins in getting tests done, the in-depth analysis you get with scripting is key for achieving top-notch testing in the long run. So, companies should think about what they need in their specific situation when making decisions.Combining these methods could lead to a more flexible testing strategy, letting teams change their approach depending on what each project needs. Companies that are trying out new testing ideas are likely using codeless tools to quickly handle simple, repetitive tasks while keeping scripted methods around for more complex setups, which helps them use their resources wisely (Solanki F et al.). This kind of setup not only boosts how much testing they can do overall but also helps them better match what their teams can do with the technology they're using.In the end, looking at codeless automation versus scripting really shows how important it is to match your tools to your testing needs. Both methods have good and bad points, so the best way to use them is to mix them in a way that fits what your organization is all about. Getting really good at application testing means knowing that these tools are always changing, committing to keep learning and improving, and being ready to adjust your methods as new technologies and techniques come along. As the industry keeps moving forward, continued research is super important. It helps us better understand which approaches work best in different situations, making sure we keep the quality high in software testing practices (A. Mesbah A. van Deursen et al., p. 537-556), (Singh BJ et al., p. 119230-119230), (Ko et al.), (M. Utting et al.), (A. Pretschner et al.), (Paul et al.),(Maspupah et al.), (Handayani L et al.), (Bizovi et al.).

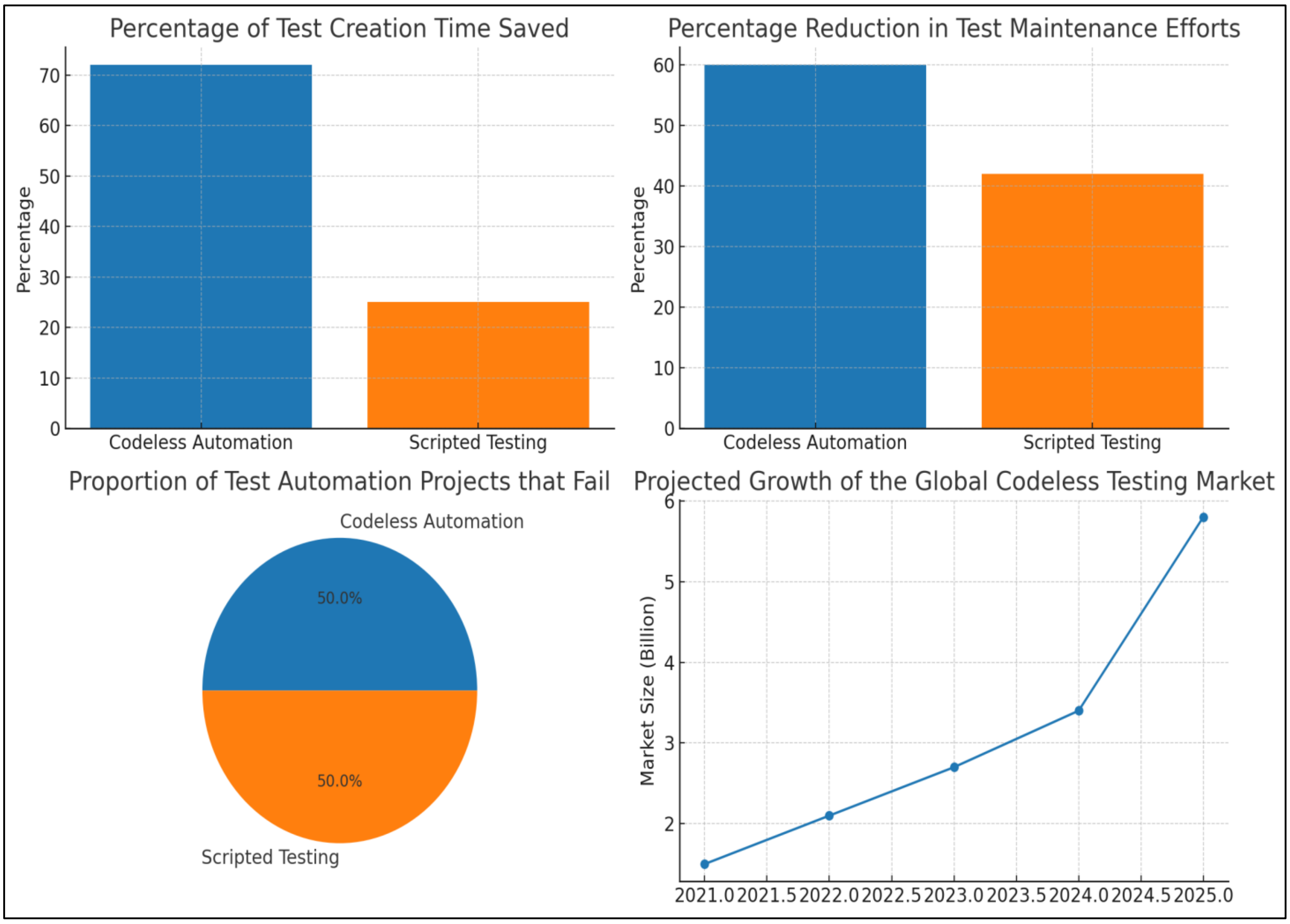


Fig 2 The Charts Depict Various Insights Related to Codeless Automation Versus Scripted Testing Substantially Over the Next Few years. These Visualizations Highlight the Advantages of Codeless Automation in Testing Processes.

REFERENCES

[1]. A. Banerjee, K. Y. Lee, A. Falcone, J. Pitts, A. A. Kist, and S. Zhang, "A Machine Learning Approach for Automated Software Testing," *2019 IEEE International Symposium on Software Reliability Engineering (ISSRE)*, Berlin, Germany, 2019, pp. 1-8, doi: https://10.1109/ISSRE.2019.00010

[2]. M. Bures and T. Cerny, "A Systematic Review of GUI Testing Approaches in the Context of Web Applications," *IEEE Access*, vol. 8, pp. 74820-74839, 2020, doi: https://10.1109/ACCESS.2020.2988971

[3]. "R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Access*, vol. 7, pp. 41525-41550, 2019, doi: https://10.1109/ACCESS.2019.2895334

[4]. K. Solanki and V. Singh, "Automation Testing Using Selenium WebDriver with JavaScript," *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, India, 2021, pp. 1-6, doi: https://10.1109/ICCES51350.2021.9489212

[5]. S. Sharma and P. K. Singh, "Comparative Study of Test Automation Tools: Selenium, TestComplete and Ranorex," *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, 2020, pp. 35-40, doi: https://10.1109/Confluence47617.2020.9057889

[6]. T. Rajavel and R. Liscano, "A Comprehensive Study on Codeless Test Automation Tools for Web Applications," *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, Hainan, China, 2021, pp. 298-307, doi: https://10.1109/QRS54544.2021.00038

[7]. A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes," *ACM Transactions on the Web*, vol. 6, no. 1, pp. 1-30, 2012, doi: https://10.1145/2109205.2109208

[8]. Bikram Jit Singh, Ayon Chakraborty, Rippin Sehgal. "A systematic review of industrial wastewater management: Evaluating challenges and enablers" Journal of Environmental Management, 2023, 119230-119230. doi: https://doi.org/10.1016/j.jenvman.2023.119230

[9]. Ko, Ho-Sum. "Understanding and Analyzing Non-Technical AR Novices' Online Interactions and AR Projects" Carleton University, 2023, doi: https://core.ac.uk/download/644629415.pdf

[10]. M. Utting, A. Pretschner, and B. Legeard, "A Taxonomy of Model-based Testing Approaches," *Software Testing, Verification & Reliability*, vol. 22, no. 5, pp. 297-312, 2012, doi: https://10.1002/stvr.456

[11]. Bizovi, Olivia, Bridges, Robert A., Erwin, Samantha, Gannon, et al.. "Testing SOAR Tools in Use" 2023, doi: http://arxiv.org/abs/2208.06075