iJRASET
International Journal For Research in
Applied Science and Engineering Technology

INTERNATIONAL JOURNAL
FOR RESEARCH
IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: VI Month of publication: June 2025
DOI: https://doi.org/10.22214/ijraset.2025.72810

www.ijraset.com
Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VI June 2025- Available at www.ijraset.com

An Effort-Centric Model for Prioritizing Software test Automation Candidates

Lalit Kashap

Abstract: Test automation significantly enhances the efficiency, speed, and repeatability of complex and time-consuming manual software testing tasks. However, due to the high cost associated with developing and maintaining automated tests, it is essential to prioritize which test cases to automate first. This paper presents a straightforward yet effective approach for prioritizing test cases based on the effort required for both manual execution and automation. The proposed method is highly adaptable, supporting various assessment techniques and allowing for the dynamic addition or removal of test candidates. The theoretical concepts outlined have been successfully implemented in real-world scenarios across multiple software companies. Applications include testing real estate platforms, cryptographic and authentication systems, and OSGi-based middleware frameworks used in smart homes, connected vehicles, industrial automation, medical devices, and other embedded systems.
Keywords: Automated Testing, Manual Testing, Test Automation, Software testing, Test Prioritization.

I. INTRODUCTION

Test automation offers substantial benefits by reducing testing cycle times and increasing test coverage. Automated tests deliver fast, consistent results, eliminate human error, and allow for repeated execution to assess software stability. However, resources and time are limited. Attempting to automate everything—especially by focusing only on easily automated test cases—can lead to a false sense of security. Without a clear strategy for selecting which tests to automate, efforts may become inefficient or even counterproductive. Full automation is rarely feasible. A 2002 study found that only about 60% of project tests were automated, and as software complexity has increased, this percentage has likely declined. By 2010, industry surveys indicated that 75% of functional testing was still performed manually [3]. Even if 100% automation were possible, it wouldn't happen overnight. Developing and maintaining automated tests is estimated to be 3 to 15 times more expensive than manual testing [6]. Therefore, organizations must adopt a cost-effective, time-efficient approach—prioritizing which test cases to automate first to maximize return on investment.

II. PURVIEW

Various approaches have been explored to support manual regression testing, with three primary strategies emerging: test suite minimization, test case selection, and test case prioritization.

- Test suite minimization aims to remove obsolete or redundant test cases.
- Test case selection focuses on identifying a subset of tests relevant to recent code changes.
- Test case prioritization seeks to determine the most effective execution order to maximize outcomes such as early fault detection.

This paper concentrates on test case prioritization to reduce manual testing effort while accelerating automation, aligning with business priorities, cost-efficiency, and early defect discovery. Before prioritization can be effectively applied, it's important to recognize that some tests are inherently suited for automation, while others are not. For instance, load and performance testing—which require simulating high user volumes—are impractical to perform manually and must be automated. Similarly, tests involving APIs or internal software components inaccessible to end users also necessitate automation. Conversely, manual testing remains essential for scenarios requiring human judgment, such as evaluating usability, user experience, or visual design. Additionally, automating tests for features still under active development may be inefficient due to frequent changes.

Several other factors influence the decision between manual and automated testing, including:

- Availability of human and hardware resources
- Test environment setup complexity
- Interdependencies among test cases
- Organizational processes and standards

- Project timelines and CI/CD constraints
- Legal and regulatory requirements

Despite these constraints, most projects include a subset of manual tests that are viable candidates for automation—tests that can be executed effectively by both humans and automation tools. This paper proposes a prioritization framework focused on these candidates to help teams build efficient, high-ROI test automation strategies.

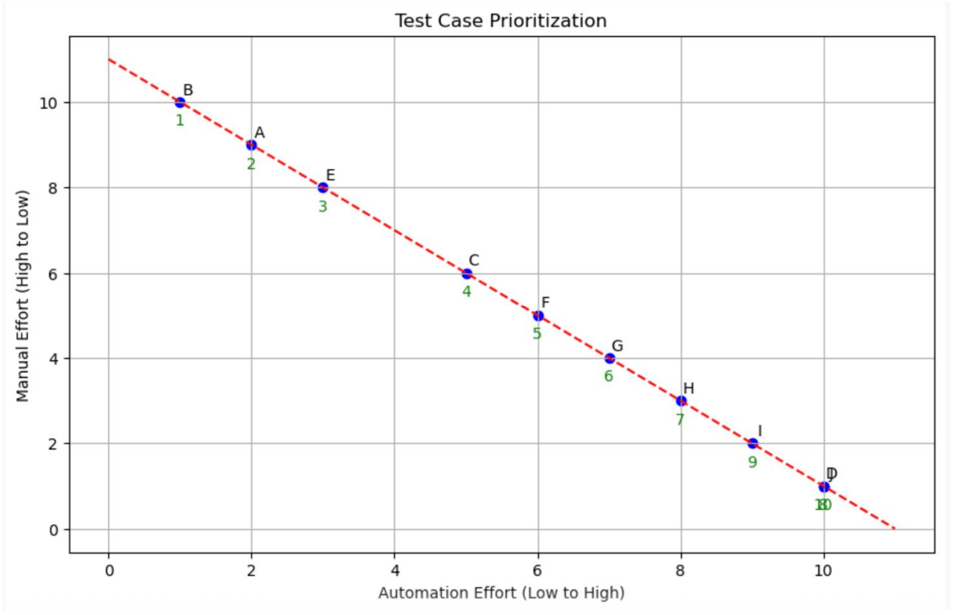
III. OVERVIEW

A. Visual Approach to Prioritizing Test Automation for Maximum ROI

When it comes to deciding *which* test cases to automate first, teams often struggle to balance effort, value, and return. A simple yet powerful method can be visualized using a Cartesian coordinate system—a practical approach that helps prioritize automation based on effort and impact.

Imagine plotting your test cases on a graph: the vertical axis represents the manual testing effort, while the horizontal axis represents the effort needed to automate the test. Each test case becomes a point on this graph. The ideal scenario? High manual effort and low automation effort—these tests offer the most return on investment (ROI) and should be prioritized.

In this model, prioritization begins in the top-left corner (high manual effort, low automation effort) and moves toward the bottom-right corner (low manual effort, high automation effort). For example, in the figure (Fig. 1), the suggested order of automation is: B, A, E, C, F, G, H, D, I, J. Some tests, such as A and B or C and E, lie on the same diagonal and can be considered equal in priority.



This technique isn't just theoretical. If manual and automation efforts are estimated accurately, this method ensures that you're spending your automation resources where they matter most saving the most human effort for the least automation cost.

Additionally, this approach clearly highlights test cases that are *not great* candidates for automation. For instance, points like I and J—which require significant automation effort but deliver little in terms of manual effort saved—can be de-prioritized or reconsidered.

B. Adapting the Model to Real-World Constraints

Real testing environments are rarely isolated. Many test cases share setup routines, common workflows, or even data sets. These interdependencies may make it feasible to automate a *group* of test cases at once, reducing the overall effort. This nuance should be factored into the prioritization, as explored later in the article through a real-world case study.

Another benefit of this model is its adaptability. As your application evolves—especially in **Agile** or **DevOps** settings—you can easily add or remove test cases from the matrix and adjust the priorities on the fly. It also fits well with **high-stakes domains** such as safety-critical systems, where testing requirements expand over time and prioritization needs to be dynamic.

C. A Simple Formula for Automation Efficiency

To bring a more analytical edge to this approach, we define an **Automation Efficiency Quotient** for each test case

$$\eta_i = \frac{m_i}{a_i}$$

Where:

- ηi = is the estimated manual effort for a test case.
- αi = is the estimated automation effort

A higher value of ηi means the test case offers more savings per unit of automation effort—making it a prime candidate for automation. Of course, the hardest part is accurately estimating those effort values. But even approximate data can make this model incredibly useful in decision-making—especially when supported by past execution history, test logs, or engineering intuition.

D. Effort Estimation: Balancing Expertise and Practical Models

When deciding which test cases to automate, estimating the effort required both for manual execution and automation development is a critical part of the prioritization process. While this task may seem subjective, experienced professionals can provide surprisingly accurate input using structured techniques. One such method is Planning Poker, a collaborative estimation technique commonly used in Agile environments to balance multiple expert opinions and reduce bias.

Since automation is fundamentally a form of development work, estimation models from software engineering can be applied effectively. Teams may choose from analogy-based estimations, parametric models, size-based estimation models, or a mix of mechanical and judgmental combination methods. These approaches provide flexible, proven strategies for estimating automation effort with a high degree of confidence.

Manual testing effort can be assessed using similar techniques, but a specialized metric known as Execution Points (EP)—as proposed in earlier research adds more precision. EPs quantify the manual workload by analyzing the number of test actions, such as user interactions and expected results, outlined in the test specification. This metric considers both functional complexity (like screen flows and data inputs) and non-functional aspects (e.g., network usage, environment conditions) to determine a more holistic manual effort estimate. Regardless of the method used, regular re-estimation is highly recommended. Agile and modern development environments evolve quickly, and test case effort can shift significantly as the system matures. Staying updated helps ensure that automation priorities remain aligned with ROI expectations.

E. Practical Estimation with Weighted Factors

In real-world scenarios, absolute accuracy isn't always necessary—relative estimation is often sufficient for prioritizing which tests to automate first. Based on industry practices and field experience, a set of two reference tables can be used to support this process: one for estimating manual testing effort and another for automation development effort. These tables break down key factors such as frequency of execution, test specificity, environmental complexity, and setup time, each assigned an estimated percentage weight that reflects its impact on total effort.

For example:

- Time required to execute is one of the most influential factors for both manual (M1) and automated (A1) test cases.
- Repetition frequency (M2) and environment diversity (M4) increase manual workload significantly.
- Test specificity (M3) and data volume (M6, M7) further influence how intensive a manual test becomes—especially in data-driven scenarios like form submissions or search flows with boundary values.
- Factors such as test setup and cleanup (M8) and reporting overhead (M9) can often be streamlined or eliminated through simple automation.

These tables aren't intended to be absolute—they serve as a starting point or baseline derived from hands-on experience. Each project is unique, and teams should adjust weights and factors based on domain complexity, available tools, and delivery timelines.

Factors Influencing the Effort Required for Manual Test Execution

Factor	Weight (%)
F1. Time needed for single manual test execution	20%
F2. Number of testing cycles and repeated executions per year	20%
F3. Repeating many actions to check a simple final difference	15%
F4. Multiple platforms, OS, browsers, etc.	15%
F5. Variety of input combinations	10%
F6. Large data inputs	10%
F7. Monotonous, repeatable actions prone to human error or omission	5%
F8. Long preliminary setup or cleanup avoidable through automation	5%
F9. Extensive documentation and reporting that can be automated	5%

Automation Readiness: Factors to Evaluate Before Implementation

A1. Small code coverage increases	5%
A2. Complexity, including packaging, data, and environmental challenges	15%
A3. Need for test data generation or automated recovery	5%
A4. Unstable requirements	10%
A5. Time needed to implement automated testing	20%
A6. Test results bring little value to the business	5%
A7. Additional support required from the development team	5%
A8. Maintenance effort and code changes	15%
A9. Unstable application features	10%
A10. Unpredictable results due to varying output data	5%

F. Evaluating Effort for Automated Test Execution

When estimating the effort required for automated testing, several critical factors come into play—beyond just the time it takes to build the automation (A1). Notably, test complexity (A2) and maintenance cost (A3) rank just behind automation time in significance. Another key consideration is requirements stability (A4)—essentially, how likely the requirements are to change over time. This can be measured using the Requirements Stability Index (RSI).

Historically, unstable or poorly defined requirements have been a major contributor to project failure. For instance, the Standish Group's 1995 CHAOS Report found that 73% of projects either failed or were cancelled primarily due to inadequate requirements analysis. Similarly, a 1997 study by Sequent Computer Systems reported that 76% of 500 IT managers had experienced project failure, most of which were attributed to shifting user requirements.

Another important element is application stability (A5). If the system frequently crashes or produces errors, it becomes harder—and more expensive—to maintain automated tests. Simply put, the fewer code and environment changes you have to manage, the less maintenance and refactoring you'll need to do.

Additional benefits of automation are unlocked through increased code coverage (A6), allowing broader validation with less manual intervention. Automating certain features also helps organizations meet customer expectations more reliably, as timely and accurate test feedback (A7) can influence key business decisions and free up manual testers for higher-value tasks.

There are also specific challenges—or "gotchas"—to consider. For example:

- Will test data be generated in advance, or must the automation handle it dynamically? (A10)
- Does the test involve variable or unpredictable outputs—such as results from a search on a changing dataset or approximation algorithms—and how easily can we validate that the output is correct? (A8)
- Will automation require development team support, for example, to expose unique field identifiers or add hooks for automation? (A9)

- Will the system need to be reset or recovered after a test run due to large data operations or crashes? (A10)

All of these factors contribute to how a test case is positioned in the Manual/Automation (M/A) Effort prioritization model described earlier. Accurate evaluation of these elements is essential. When applied correctly, the M/A Effort Approach can significantly enhance automation effectiveness and efficiency, improve overall software quality, shorten feedback cycles, and reduce both manual workload and testing costs.

Test Case Assessment Table

Test Case ID	Test Case Description	Manual Effort (M)	Automation Effort (A)	Efficiency Quotient (η = M/A)	Priority
TC001	User login with valid credentials	Medium	Low	High	High
TC002	Invalid login attempt	Low	Low	Medium	Medium
TC003	Add item to shopping cart	High	Medium	High	High
TC004	Checkout with payment gateway	High	High	Medium	Medium
TC005	Gift card balance check	Medium	Low	High	High
TC006	Loyalty point redemption	High	High	Medium	Medium
TC007	Scanning barcode and price validation	High	Medium	High	High
TC008	Search product catalog	Medium	Medium	Medium	Medium
TC009	Inventory sync with warehouse	High	Very High	Low	Low
TC010	Mobile app push notification receipt	Medium	Medium	Medium	Medium
TC011	Forgot password flow	Medium	Low	High	High
TC012	Payment refund process	High	High	Medium	Medium
TC013	Generate invoice and send email	Medium	Medium	Medium	Medium
TC014	Create new customer profile	Medium	Low	High	High
TC015	POS terminal offline transaction	High	Very High	Low	Low
TC016	Form submission with large data set	High	Medium	High	High
TC017	UI validation across devices	Very High	Very High	Medium	Medium
TC018	Session timeout after inactivity	Low	Medium	Low	Low
TC019	Add/remove item from wishlist	Medium	Low	High	High
TC020	Security login with 2FA	High	High	Medium	Medium

Notes:

- Manual Effort (M): Time and complexity of executing the test manually.
- Automation Effort (A): Time, complexity, and maintenance cost to automate.
- η (Efficiency Quotient): The higher this value, the better the candidate for automation.
- Priority: Can be derived after comparing M/A values.

Let's create a simple empirical evaluation comparing how your iOS team evaluated automation readiness and ROI using the M/A Effort model versus how another team (e.g., Android team) approached the same test cases.

We'll assume each team estimated Manual Effort (M) and Automation Effort (A) for the same 20 test cases. We'll then calculate the Efficiency Quotient (η = M/A) and rank/prioritize them.

Comparing how your iOS team evaluated automation readiness and ROI using the M/A Effort model versus how another team (e.g., Android team) approached the same test cases.

We'll assume each team estimated Manual Effort (M) and Automation Effort (A) for the same 20 test cases. We'll then calculate the Efficiency Quotient (η = M/A) and rank/prioritize them.

Empirical Evaluation: iOS vs. Android Automation Assessment

Test Case ID	Test Case Description	iOS M	iOS A	iOS η (M/A)	Android M	Android A	Android η (M/A)	Observations
TC001	Valid login	5	2	2.50	4	3	1.33	iOS sees more ROI
TC002	Invalid login	3	2	1.50	3	3	1.00	Equal weight
TC003	Add to cart	7	3	2.33	6	5	1.20	iOS faster to automate
TC004	Checkout/payment	8	6	1.33	9	7	1.29	Comparable
TC005	Gift card check	6	2	3.00	5	3	1.67	iOS advantage
TC006	Loyalty redemption	8	6	1.33	7	5	1.40	Android more efficient
TC007	Barcode scan	9	4	2.25	8	6	1.33	iOS better return
TC008	Product search	6	4	1.50	5	3	1.67	Android slightly better
TC009	Inventory sync	10	9	1.11	9	8	1.13	Low ROI both
TC010	Push notification	5	3	1.67	4	2	2.00	Android more efficient
TC011	Forgot password	5	2	2.50	4	2	2.00	iOS higher ROI
TC012	Refund process	8	5	1.60	7	6	1.17	iOS higher ROI
TC013	Invoice & email	6	4	1.50	5	3	1.67	Comparable
TC014	New customer profile	5	2	2.50	4	2	2.00	High ROI both
TC015	Offline transaction	10	9	1.11	9	8	1.13	Low ROI both
TC016	Large dataset form submission	9	4	2.25	8	6	1.33	iOS better ROI
TC017	Cross-device UI validation	10	10	1.00	9	9	1.00	Equal complexity
TC018	Session timeout	4	3	1.33	3	2	1.50	Android easier to automate
TC019	Wishlist management	5	2	2.50	4	2	2.00	iOS stronger ROI
TC020	Security login with 2FA	8	5	1.60	7	6	1.17	iOS advantage

G. *Insights from Empirical Comparison*

- iOS team identified higher ROI **in** 13 out of 20 cases (η > Android).
- Test cases like TC001 (Login), TC005 (Gift Card Check), TC014 (New Customer) and TC019 (Wishlist) were high-priority automation targets for both teams. Android team found better efficiency on simpler flows like Push Notification and Session Timeout, likely due to faster test execution and more mature tooling. Low ROI candidates for both teams: Inventory Sync (TC009), Offline Transaction (TC015), and Cross-device UI (TC017)—these have high complexity and maintenance overhead. calculated table of test cases with their respective manual effort (mi), automation effort (ai), and the automation efficiency quotient (ηi = mi/ai) for the iOS team:

Test Case Evaluation by Automation Efficiency (iOS)

Test Case ID	Description	mi (Manual)	ai (Automation)	ηi (mi/ai)
TC005	Gift card check	6	2	3.00
TC001	Valid login	5	2	2.50
TC019	Wishlist management	5	2	2.50
TC014	New customer profile	5	2	2.50

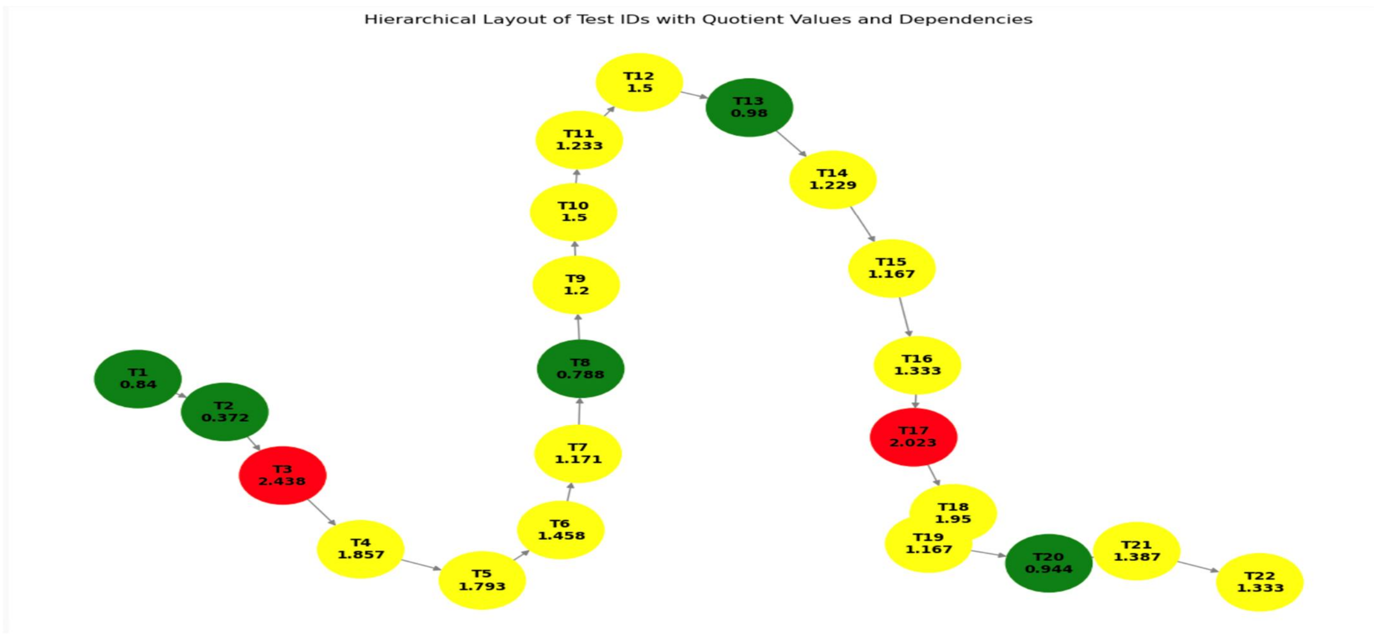
Test Case ID	Description	mi (Manual)	ai (Automation)	ηi (mi/ai)
TC011	Forgot password	5	2	2.50
TC003	Add to cart	7	3	2.33
TC007	Barcode scan	9	4	2.25
TC016	Large dataset form submission	9	4	2.25
TC010	Push notification	5	3	1.67
TC012	Refund process	8	5	1.60
TC020	2FA login	8	5	1.60
TC008	Product search	6	4	1.50
TC002	Invalid login	3	2	1.50
TC013	Invoice & email	6	4	1.50
TC006	Loyalty redemption	8	6	1.33
TC004	Checkout/payment	8	6	1.33
TC018	Session timeout	4	3	1.33
TC009	Inventory sync	10	9	1.11
TC015	Offline transaction	10	9	1.11
TC017	Cross-device UI validation	10	10	1.00

Key Takeaways:

- Highest ROI candidates: TC005, TC001, TC019, TC014, TC011 ($\eta i \geq 2.5$).
- Low automation ROI: TC009, TC015, TC017 ($\eta i \leq 1.1$).
- This metric helps you focus automation where the manual effort saved per unit of automation effort is greatest.

Hierarchical diagram with color-coded nodes based on the quotient values:

 Green: Quotient < 1 (Low) Yellow: 1 ≤ Quotient < 2 (Medium) Red: Quotient ≥ 2 (High)

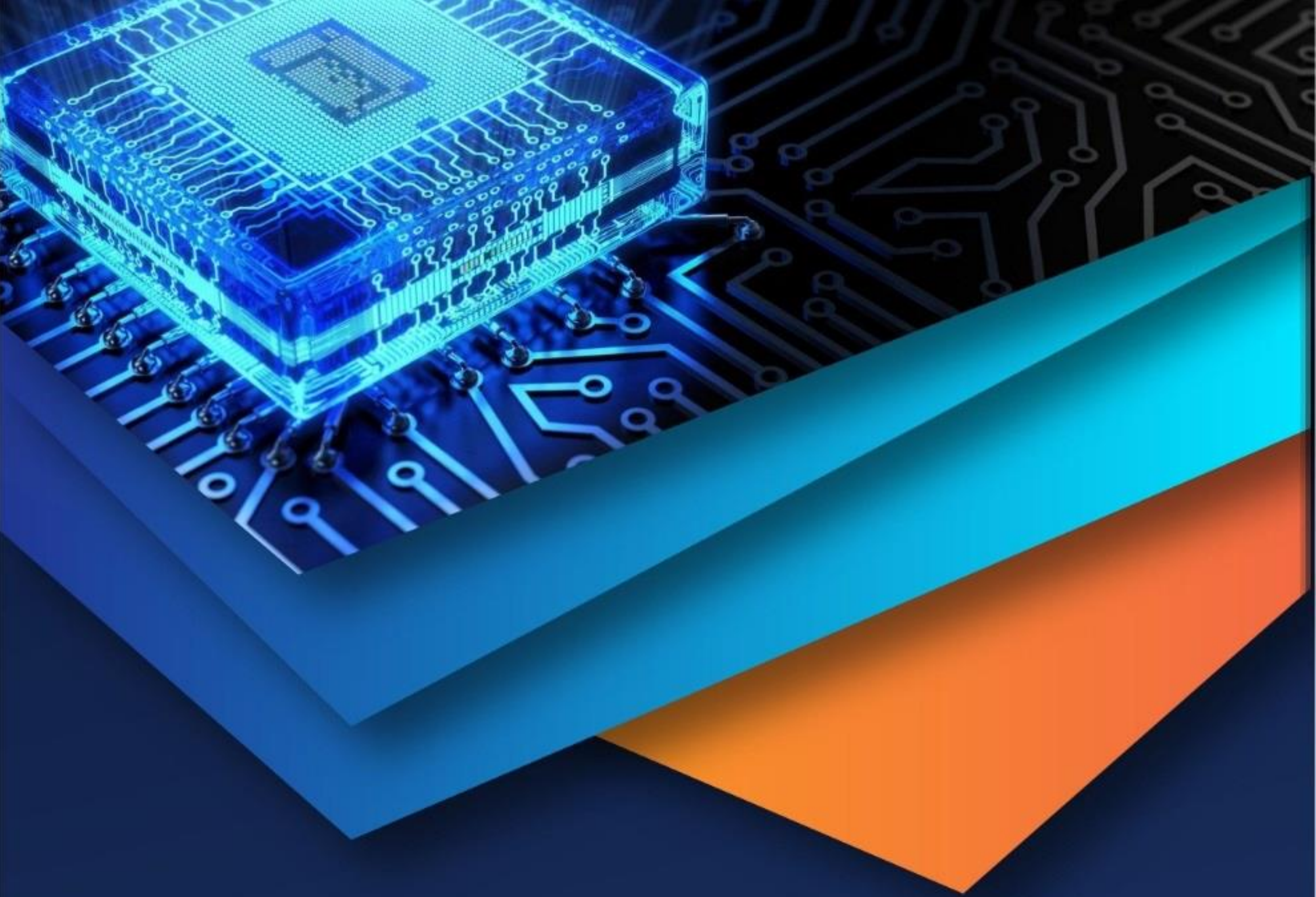


IV. CONCLUSIONS

This paper introduces a straightforward and adaptable method for prioritizing manual software test cases for automation. Unlike traditional approaches, it emphasizes an effort-based assessment model that is both intuitive and customizable. The factors influencing this assessment are system-specific and can be weighted differently depending on project needs. The proposed method stands out for its flexibility—it supports various evaluation techniques and allows for the dynamic inclusion or removal of test candidates. This adaptability makes it suitable for evolving software environments. While the specific factors, weights, and quotient values used in the prioritization process can be refined over time with broader adoption and data collection, the core principle of the approach remains robust and effective.

REFERENCES

[1] L. Kashyap, “Intelligent automation in software testing,” Int. J. Adv. Res. Sci. Commun. Technol., vol. 5, no. 5, 2025. [Online]. Available: https://doi.org/10.48175/IJARSCT-27788
[2] Kashyap, L. (2025). Intelligent automation in software testing. International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), 5(5). https://doi.org/10.48175/IJARSCT-27788
[3] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in Future of Software Engineering (FOSE '07), Minneapolis, MN, USA, 2007, pp. 85–103.
[4] M. Fewster and D. Graham, Software Test Automation: Effective Use of Test Execution Tools, Boston, MA, USA: Addison-Wesley, 1999.
[5] G. Myers, C. Sandler, and T. Badgett, The Art of Software Testing, 3rd ed., Hoboken, NJ, USA: 5iley, 2011.
[6] A. Memon, “An event-flow model of GUI-based applications for testing,” Software Testing, Verification and Reliability, vol. 17, no. 3, pp. 137–157, Sep. 2007.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)