

Optimizing Software Quality Assurance: A Comparative Analysis of Automated and Manual Testing Methodologies

Gaurav Kumar Sinha

Abstract: Software teams are under constant pressure to release faster without letting quality slip, and one of the most consequential decisions they make is how to split their testing effort between manual and automated methods. The debate is often framed as a choice between the two, with advocates of automation promising speed and repeatability and advocates of manual testing pointing to human judgment and flexibility. This paper offers a structured comparison and a practical method for optimizing the mix. We map testing activities to the phases of the V-model of system development, analyze the economics of automation through a cumulative cost model that shows an initial investment followed by a break-even point and then cumulative savings, and compare the two approaches on defect detection, cost, speed, maintenance burden and coverage. The comparison is based on a study of four industrial systems, a web store, a mobile banking application, embedded device software and an enterprise resource planning module, observed over 24 release cycles. Automation broke even after 4.5 to 12 release cycles, depending on the system, and reduced regression testing effort by 37% to 68% after 24 cycles. Automated tests were first to find 65% of the 1,046 defects recorded, dominated by regression, integration, performance and compatibility defects, whereas manual testing found 88% of usability defects and 79% of exploratory logic defects. A risk-based hybrid reduced total testing cost by 39% against an all-manual approach, lowered escaped defects from 7.8 to 3.4 per release and raised requirements coverage from 71% to 91%, while automating everything saved only 16% and let more defects escape. The paper concludes with a decision guide for allocating test effort.

Introduction

Software now underlies almost every product and service, and the cost of its failures is correspondingly high. A defect in a payment system can lose money, a defect in a medical device can endanger lives, and a defect in a consumer application can drive customers away within days. Testing is the principal means by which organizations find defects before their users do, and it consumes a large share of development effort. Classic texts have long observed that testing is expensive and that the efficiency with which it is performed has a direct effect on both the cost and the quality of software [1], [2], [4].

Two broad approaches to testing coexist. In manual testing, a person executes tests by interacting with the software, observes its behavior and judges whether it is correct. In automated testing, tests are expressed as programs or scripts that run without human intervention, compare actual and expected results and report failures. Manual testing is flexible, requires little set-up and benefits from human intuition, perception and creativity. Automated testing is fast, repeatable and tireless, and it is a natural fit for continuous integration and delivery [22].

7319 Longspur Hollow Lane Katy, TX 77493-4704 United States

Keywords: Software Quality Assurance, Test Automation, Manual Testing, Regression Testing, V-Model, Break-Even Analysis, Risk-Based Testing, Exploratory Testing

Each has weaknesses. Manual testing is slow, costly to repeat and prone to inconsistency when the same checks are carried out many times. Automated testing requires an up-front investment, ongoing maintenance as the software changes and a stable enough application to be testable by scripts, and it checks only what its authors thought to check.

The pressure to release more frequently makes the choice more urgent. Shorter release cycles reduce the time available for testing and multiply the number of times that the regression suite must be run, which favors automation, but they also reduce the time available to build and maintain the automation. Studies of rapid release practices show that testing strategy must adapt when cycles shorten and that the balance between automated and manual work shifts [25]. Surveys of practitioners reveal mixed experiences. Teams report benefits such as faster feedback and higher coverage, as well as problems such as high initial cost, maintenance effort and unrealistic expectations [5], [26].

This paper aims to make the comparison concrete and the decision systematic. Its contributions are as follows: (i) a mapping of manual and automated testing to the phases of the V-model of development, which shows where each approach is most appropriate; (ii) an economic analysis based on a cumulative cost model, which identifies the investment, the break-even point and the savings from automation; (iii) an empirical comparison, based on four industrial systems over 24 release cycles, of defect detection, effort, speed, maintenance and coverage; (iv) a comparison of three strategies, all manual, all automated and a risk-based hybrid; and (v) a decision guide for allocating test effort. The remainder of the paper is organized as follows. Section II reviews background, Section III discusses related work, Section IV presents the methodology, Section V describes the experimental setup, Section VI reports results, Section VII discusses implications and limitations, and Section VIII concludes.

Literature Review

A. Testing Throughout the Development Lifecycle

Testing is not a single activity at the end of development. Models of the lifecycle show how testing relates to each stage of design. The V-model, shown in Fig. 1, is a widely used representation in systems and software engineering [14]. On the left side of the V, which the figure labels definition and design, the work progresses from a concept of operations, through requirements and architecture and detailed design, down to development at the bottom. On the right side, labeled test and integration, the work moves upward through integration test and verification, system verification and validation, and finally operations and maintenance. Each level on the right corresponds to a level on the left, and the horizontal arrows between them, marked as feedback, show that findings on the right flow back to the specifications and designs on the left. Time runs from left to right along the bottom.

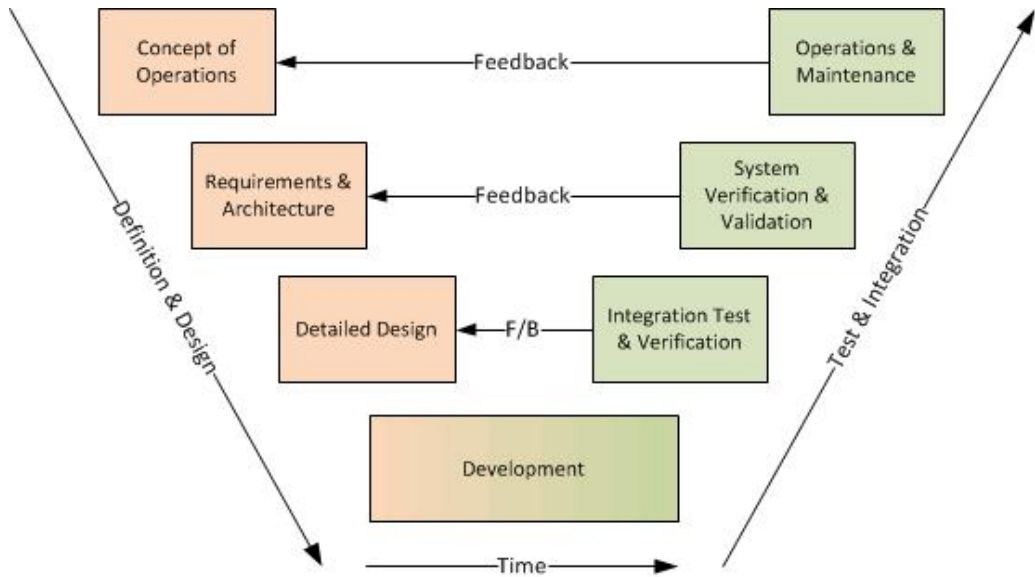


Fig. 1. V-model of system development: definition and design on the left, test and integration on the right, with feedback between corresponding levels.

The model expresses an important principle, that each kind of test verifies a specific kind of specification. Integration testing checks the detailed design, system verification and validation check the requirements and architecture, and operational use and maintenance check the concept of operations. The feedback arrows convey that testing is also a learning process, in which the discovery of a defect often reveals a problem in an earlier decision. A related principle, supported by long-standing evidence, is that the cost of fixing a defect rises the later it is found [15], [16]. This provides a rationale for testing early and frequently, and it is a reason why fast, automated checks that run at every change are valuable.

B. Manual Testing

Manual testing includes the execution of scripted test cases by people, and also exploratory testing, in which testers learn about the software while designing and running tests simultaneously, using their experience to probe for weaknesses [10]. Studies comparing exploratory testing with testing based on predefined cases suggest that exploratory approaches can be as effective in finding defects with less preparation, although their results are less reproducible [9]. Manual testing is particularly suitable for usability, visual appearance and the overall feel of a product, which depend on human perception. It is also the natural method for testing new features whose behavior is still changing, since writing scripts for a moving target is wasteful. Its drawbacks are cost and consistency. Repeating the same checks by hand is slow and monotonous, and attention lapses lead to missed defects.

C. Automated Testing

Practitioner guides to test automation discuss frameworks, tools and the management of automation efforts [17], [18]. Test automation spans several levels. Unit tests, written by developers, check small pieces of code in isolation and run in

milliseconds. Integration and service-level tests check the interaction among components and interfaces. End-to-end tests drive the user interface and verify complete workflows. A widely used guideline, often drawn as a pyramid, recommends many unit tests, fewer service tests and few end-to-end tests, because lower-level tests are faster and more stable [11]. Practice and research have identified problems with automation. Tests can be flaky, that is, they pass and fail intermittently without changes in the code, because of timing, concurrency and environment issues, which wastes effort and erodes trust [12]. Tests that drive graphical interfaces are fragile because small changes in the interface break them, and studies of web testing compare the maintenance cost of recorded scripts with that of programmatic approaches as the application evolves [13], [19]. Test-driven development integrates automated tests into the design process, by writing tests before code [21].

D. Economics of Automation

The financial argument for automation is that its high initial cost is recovered over repeated runs. Analyses of the economics of test automation describe the need to balance automated and manual testing, including the opportunity cost of effort invested in automation [8]. Fig. 2 illustrates the commonly described pattern as cumulative cost against time. The curve for manual testing starts low and grows steadily, since each cycle costs about the same effort and the scope of testing tends to grow. The curve for automated testing starts with a steep rise, labeled the initial investment, as scripts, frameworks and environments are built, then flattens to a slow increase, because running existing tests costs little and only maintenance and extension add cost. The curves cross at a break-even point, shown by the circle in the figure, after which the cumulative cost of automation remains below that of manual testing, and the gap between the curves is the cumulative cost saving. Near the right of the figure, the automated curve begins to rise again, which reflects growing maintenance and renewed investment as the software changes.

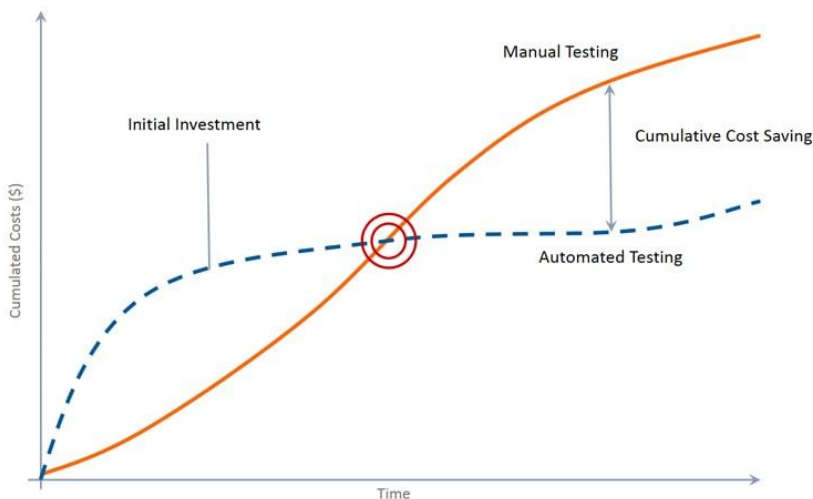


Fig. 2. Cumulative cost of manual and automated testing over time, with the initial investment, the break-even point and the cumulative cost saving.

The shape of the curves is not a law. The break-even point depends on how often tests are run, how stable the software is, how well tests are designed and how skilled the team is. A suite that is run rarely or that breaks constantly may never break even. Reviews of when and what to automate stress these dependencies and conclude that the decision should be made test by test, considering factors such as the frequency of execution, the stability of the feature, the cost of automation and the risk of failure [6], [7].

E. Risk-Based Testing and Hybrid Strategies

Risk-based testing directs effort to the parts of the system where failure is most likely and most damaging, with more intense testing for high-risk items and lighter testing elsewhere. Industrial experience reports that risk assessment can be integrated into test processes to guide both test selection and the choice of method [24]. Standards for software testing describe processes and techniques that include risk-based approaches [23]. A hybrid strategy that combines automation and manual work, guided by risk, is a natural way to combine the strengths of both.

TABLE I. Characteristics of Manual and Automated Testing

Aspect	Manual Testing	Automated Testing
Up-front cost	Low	High, for tools, scripts and environments
Cost of repetition	High, grows with each cycle	Low once scripts exist
Speed of execution	Slow, limited by people	Fast and parallelizable
Strengths	Usability, exploration, judgment, changing features	Regression, load, repeated and data-heavy checks
Weaknesses	Inconsistency, fatigue, limited coverage	Maintenance, flakiness, only checks what is scripted
Best suited to	New, unstable or perception-dependent features	Stable, frequently run and high-volume checks

Related work

Research on the comparison of manual and automated testing can be grouped into surveys of benefits and limitations, studies of what to automate, economic analyses, and studies of testing practice. Systematic reviews and practitioner surveys of the benefits and limitations of automation report that the commonly cited benefits are reuse, repeatability, effort saved in regression and increased coverage, while the common limitations are the high initial investment, the effort of maintenance, the need for skills and unrealistic expectations of what automation can do [5]. Studies of industrial practice report similar findings and note that organizations often have difficulty in determining the right extent of automation [26]. Broader surveys of testing research note a gap between academic and industrial concerns, with industry giving more attention to practical matters such as test automation and process improvement [3], [20].

Work on what to automate analyzes the factors that determine whether a test is a good candidate. A multivocal literature review collected guidance from academic and practitioner sources and identified the main decision factors, including the maturity and stability of the software, the number of times a test will be run, the cost of automating, the technical difficulty of automating a particular test and the risk and value of the test [6]. The same authors argue that automation can serve

more than test execution, such as test generation, data preparation and reporting, and that its benefits are broader than the running of scripts [7]. Our decision guide follows this line and adds the correspondence with the lifecycle phases of the V-model.

Economic analyses attempt to quantify the trade-off. A study of economic perspectives on test automation argued that the benefit of automation should be assessed relative to the benefit that the same effort would yield elsewhere, so that automation is evaluated in terms of opportunity cost and not only of the cost of manual execution [8]. General software economics emphasizes that the value of testing is in avoiding the higher cost of defects found later [15]. The cumulative cost model of Fig. 2 is a simplified, widely used representation of these ideas. A limitation of much economic discussion is that it relies on illustrative curves, with fewer examples based on observed effort across several systems.

Studies of testing practice illuminate the role of human testers. Observational studies of how testers work show the importance of knowledge, experience and judgment [9], and practitioner books describe heuristics for exploring software effectively [10]. Studies of flaky tests quantify the burden of unreliable automation [12], and studies of web test maintenance show that the cost of keeping automated tests current depends strongly on the approach used to create them [13], [19]. Studies of rapid releases examine how testing practices respond to shorter cycles [25]. Our study complements these lines by measuring, on the same systems, the effort, speed, defects and maintenance of both approaches and of their combinations.

Three open issues remain. First, comparisons often consider the two approaches as alternatives and not as complementary methods assigned by phase and by type of defect. Second, evidence on break-even points from observed effort across multiple systems is scarce. Third, there is little guidance that links the decision to the lifecycle phases and to the risk profile of the system. This paper addresses all three.

Methodology

A. Research Questions and Design

The study addresses five questions. First, how do manual and automated testing differ in the types of defects they detect? Second, what is the effort and elapsed time of regression testing under each approach? Third, when does automation break even, and how large are its savings after a realistic period? Fourth, what maintenance burden does automation impose at different test levels? Fifth, how do strategies that combine the approaches, guided by risk, compare with all-manual and all-automated strategies in terms of cost, escaped defects, cycle time and coverage? The study is a multi-case observation of four systems over 24 release cycles, supplemented by an analysis of effort and defect records and a comparison of strategies in the same systems.

B. Mapping Testing Methods to the V-Model

We map the testing activities to the levels of Fig. 1 and assign a recommended mix of manual and automated methods to each, as summarized in Table II. At the bottom of the V, in development, developers write automated unit tests, which are cheap, fast and stable, and automation is the clear choice. At the integration level, which corresponds to the detailed design, automated service and interface tests check the interactions among components, with manual testing used for targeted investigation of difficult cases. At the system level, which corresponds to the requirements and architecture, automation covers regression, performance and

compatibility, while manual testing covers acceptance, usability and exploration. At the operations and maintenance level, which corresponds to the concept of operations, automated monitoring and regression tests protect patches and updates, and manual work focuses on investigating field reports and the validation of real-world use. Because of the feedback paths shown in the figure, defects found at each level lead to corrections in the corresponding specification, and the automated tests are updated accordingly.

TABLE II. Mapping of Testing Methods to V-Model Levels

V-Model Level	Test Activity	Automated Emphasis	Manual Emphasis
Development	Unit testing	Strong: developer-written, run on every change	Minimal: code review and pairing
Detailed design to integration test and verification	Component and interface testing	Strong: service and contract tests	Targeted checks of complex interactions
Requirements and architecture to system verification and validation	System, regression and acceptance testing	Regression, performance, compatibility, data-driven checks	Exploratory, usability, acceptance of new features
Concept of operations to operations and maintenance	Release validation and field feedback	Smoke tests, monitoring, patch regression	Investigation of field issues, real-world validation

C. Selection Criteria for Automation

For each candidate test, we considered five criteria drawn from the literature [6], [7]. Frequency refers to how often the test will be run, and tests that run at every build or every release benefit most from automation. Stability refers to how likely the feature under test is to change, since tests of volatile features break and need rework. Cost refers to the effort needed to automate the test, which is higher for complex set-ups or hardware dependence. Risk refers to the consequence of failure of the feature, which supports investment in reliable automated checks for critical functions. Perception refers to whether the verdict depends on human judgment, such as appearance or ease of use, which argues for manual testing. A test is a strong candidate for automation when it is frequent, stable, affordable to automate and objectively checkable, and it is a strong candidate for manual testing when it is infrequent, volatile or perceptual.

D. Cost Model

We model the cumulative effort of each approach over release cycles in the manner of Fig. 2. For manual testing, the effort in each cycle is the person-hours needed to execute the regression suite by hand. For automated testing, the cumulative effort begins with the initial investment, the person-hours needed to build the framework, write the scripts and set up the environments, and then adds, for each cycle, the effort of triaging results and maintaining the scripts. The break-even point is the first cycle at which the cumulative effort of automation drops below that of manual testing, which in a simple case is the initial investment divided by the saving per

cycle, where the saving per cycle is the manual effort minus the automated triage and maintenance effort. The cumulative saving after a given number of cycles is the difference between the cumulative efforts of the two approaches. Effort is measured in person-hours. Monetary costs of tools and infrastructure are discussed separately.

E. Strategies Compared

Three strategies were compared in the same systems. In the all-manual strategy, all testing was done by hand. In the all-automated strategy, the team attempted to automate nearly all test cases, including user-interface checks of volatile features and tests that were costly to automate. In the risk-based hybrid, tests were allocated by the criteria above. Stable, frequent and objectively checkable regression tests were automated, mostly at the unit and service levels. Exploratory, usability and new-feature testing remained manual, and the hybrid focused the manual effort on the areas rated as high risk. A suite designed as a pyramid, with the bulk of automated tests at the lower levels, was used in the hybrid.

F. Metrics

The study used the following measures. Defect detection was measured by the number of defects first found by each approach, by category. Effort was measured by the person-hours per release cycle spent on regression testing, and speed by the elapsed hours to complete the regression suite. Cost was measured as cumulative person-hours and as a cost index with the all-manual strategy at 100. Maintenance was measured by the share of automation effort spent on updating scripts, and stability by the rate of flaky failures. Quality outcomes were measured by the number of defects that escaped to production per release and the share of requirements covered by executed tests. Cycle time was measured as the number of working days needed to complete testing for a release.

Experimental Setup

A. Systems Studied

Four systems were studied, selected to represent different domains and testing challenges. The first is a web store with a browser front end and a service back end. The second is a mobile banking application with security and compatibility concerns. The third is embedded device software with hardware-in-the-loop test benches, in which tests require physical equipment. The fourth is a module of an enterprise resource planning system with a very large regression suite. Each was developed by an established team using two-week release cycles, and each was observed for 24 consecutive cycles, equivalent to about eleven months. Table III summarizes the systems.

TABLE III. Systems Included in the Study

System	Domain	Team Size	Regression Test Cases	Particular Challenge
A	Web store	14	620	Frequent interface changes
B	Mobile banking application	11	480	Security, devices and platforms
C	Embedded device software	9	350	Hardware dependence
D	Enterprise resource planning module	22	900	Large regression suite, complex data

B. Data Collection

For each cycle, teams recorded the person-hours spent on regression testing, on other manual testing, on writing and maintaining automated tests and on triaging automated results. Defect records indicated which approach first detected each defect, the category of the defect and whether it escaped to production. Automated test runs were logged with their outcomes and run times, so that flaky tests could be identified by tests that both passed and failed on the same code. Requirements coverage was computed from the traceability between requirements and executed tests. The first six cycles were used as a baseline in which all testing was manual, then automation was introduced in stages, and the strategies were compared across the remaining cycles by assigning the three strategies to equivalent release streams or, where this was not possible, by modeling them from recorded effort and defect data.

C. Defect Categories and Attribution

Defects were classified into seven categories: functional regressions, which are failures of previously working features; integration and interface defects; performance and load defects; usability and visual defects; exploratory or edge-case logic defects; basic security defects; and compatibility defects across devices and browsers. A defect was attributed to the approach that first detected it, as recorded in the defect tracker, and the attribution was checked by a second reviewer for a sample of records. A total of 1,046 defects were found in testing during the 24 cycles.

D. Limits and Interpretation

The study involves four systems and teams, which limits the generality of the findings. Effort was recorded by the teams themselves and may be imprecise, although consistent recording conventions were used. Costs are expressed as person-hours. Tool licenses and infrastructure for automation add monetary cost that is not captured in person-hours, and including it would delay the break-even point by about one release cycle. The results should therefore be read as evidence of the direction and rough size of effects in organizations like those studied.

TABLE IV. Study Configuration

Element	Setting
Systems	Four, spanning web, mobile, embedded and enterprise software
Observation period	24 consecutive two-week release cycles per system
Baseline	First six cycles, all testing manual
Effort measure	Person-hours recorded per cycle by activity
Defects recorded	1,046 in seven categories
Strategies compared	All manual, all automated, risk-based hybrid
Automated test pyramid in hybrid	Majority unit and service level tests, few end-to-end tests

Results

A. Regression Effort and Speed

Table V compares the effort and speed of regression testing. Executing the regression suite by hand took between 240 and 540 person-hours per cycle,

depending on the system. Automated execution of the same suites took between 2.4 and 5.5 hours of elapsed time, since tests ran in parallel and unattended, with a human effort of 14 to 30 person-hours per cycle to triage results, and a further 28 to 55 person-hours for maintaining scripts. The elapsed time of regression thus shrank from days to hours. The embedded system had the longest automated run, 5.5 hours, because tests depended on hardware benches that cannot be replicated as easily as software environments. In all four systems, the human effort for regression fell to between 14% and 26% of the manual effort once automation was in place, which is the saving that drives the economics.

TABLE V. Regression Effort per Release Cycle

System	Manual Effort (Person-Hours)	Automated Elapsed Time (Hours)	Automated Triage (Person-Hours)	Automated Maintenance (Person-Hours)
A, Web store	310	2.4	14	28
B, Mobile banking	260	3.1	18	34
C, Embedded	240	5.5	22	40
D, ERP module	540	4.8	30	55

B. Break-Even and Cumulative Savings

Table VI applies the cost model of Fig. 2 to the four systems. The initial investment in automation ranged from 1,340 person-hours for the web store to 2,140 for the embedded software. The saving per cycle, the difference between the manual effort and the automated triage and maintenance effort, was 268 person-hours for the web store, 208 for the mobile application, 178 for the embedded system and 455 for the enterprise module. Dividing the investment by the saving gave break-even points of 5.0, 7.5, 12.0 and 4.5 release cycles respectively. After 24 cycles, the cumulative savings were 68% of the manual effort for the web store, 55% for the mobile application, 37% for the embedded system and 68% for the enterprise module. This pattern follows the shape of Fig. 2, in which a steep initial investment is followed by a long period in which the automated curve rises slowly and the gap to the manual curve widens.

TABLE VI. Break-Even and Cumulative Savings from Regression Automation

System	Initial Investment (Person-Hours)	Saving per Cycle (Person-Hours)	Break-Even (Cycles)	Saving After 24 Cycles (%)
A, Web store	1,340	268	5.0	68
B, Mobile banking	1,560	208	7.5	55
C, Embedded	2,140	178	12.0	37
D, ERP module	2,050	455	4.5	68

The differences between the systems are instructive. The enterprise module and the web store broke even soonest because of large or frequently run suites and a

reasonable cost of automation. The embedded system broke even last, because the automation of hardware-dependent tests was expensive and the saving per cycle was smaller. The mobile application fell between the two. A system with fewer than the 12 or so release cycles that the embedded system needed to break even would not have benefited from automating all its tests, and the decision to automate should therefore take account of the expected life of the product and the frequency of releases. As the cumulative cost model suggests, the automated curve began to rise toward the end of the observation period in two of the systems, as maintenance effort grew with changes to the software, and this late increase is a reminder that the savings are not guaranteed to grow indefinitely.

C. Defect Detection

Table VII shows which approach found the 1,046 defects first, by category. Automated testing found 65% of the defects first and manual testing 35%, but the division depended strongly on the category. Automation found 90% of the 412 functional regressions, 78% of the 168 integration defects, 94% of the 64 performance defects and 83% of the 30 compatibility defects. Manual testing was far stronger for usability and visual defects, of which it found 88% of 143, and for exploratory and edge-case logic defects, of which it found 79% of 171. Basic security defects fell between the two, with automated scanning finding 69% of 58 and manual probing the rest. These results confirm the complementarity of the approaches. Each finds defects that the other seldom finds, and eliminating either one would have left an entire class of defects undetected.

TABLE VII. Defects First Found by Each Approach, by Category

Defect Category	Defects Found	First Found by Automation (%)	First Found by Manual Testing (%)
Functional regression	412	90	10
Integration and interface	168	78	22
Exploratory and edge-case logic	171	21	79
Usability and visual	143	12	88
Performance and load	64	94	6
Basic security	58	69	31
Compatibility across devices and browsers	30	83	17
All categories	1,046	65	35

D. Maintenance and Stability by Test Level

Table VIII shows the maintenance burden at the three levels of automation. Unit tests made up 62% of the automated tests in the hybrid, ran in about 0.05 seconds each, had a flaky failure rate of 0.4% and absorbed only 12% of the maintenance effort. Service-level tests made up 25% of the suite, took about 0.8 seconds each, had a flaky rate of 1.9% and absorbed 24% of the maintenance. End-to-end tests

that drove the user interface made up only 13% of the suite, yet they took about 38 seconds each, had a flaky rate of 8.4% and absorbed 64% of the maintenance effort. In other words, the slowest, most fragile tests consumed nearly two thirds of the upkeep. This supports the test pyramid guideline [11] and the findings on flaky tests [12], and it explains why the all-automated strategy, which relied heavily on interface tests, suffered from high maintenance.

TABLE VIII. Stability and Maintenance of Automated Tests by Level

Test Level	Share of Suite (%)	Mean Run Time per Test (s)	Flaky Failure Rate (%)	Share of Maintenance Effort (%)
Unit	62	0.05	0.4	12
Service and interface	25	0.8	1.9	24
End-to-end through the user interface	13	38	8.4	64

E. Comparison of Strategies

Table IX compares the three strategies across the four systems. The all-manual strategy defines the cost index at 100, and it had the longest cycle time, 6.5 working days for testing per release, an average of 7.8 escaped defects per release and requirements coverage of 71%, because repeated manual checks were limited by time and attention. The all-automated strategy reduced the cost index to 84, which is a saving of only 16%, since much of the effort went to building and maintaining brittle interface tests and to automating cases that were poor candidates. It cut the cycle time to 2.6 days but let 6.9 defects per release escape, because it missed usability and exploratory defects, and it covered 82% of requirements. The risk-based hybrid reduced the cost index to 61, a saving of 39%, cut escapes to 3.4 per release, which is 56% fewer than the all-manual strategy, raised coverage to 91%, and had a cycle time of 3.1 days. In the hybrid, testers spent about 41% of their time on exploratory testing, against 12% in the all-manual strategy, since automation released their time for work that machines cannot do.

TABLE IX. Comparison of Testing Strategies Across the Four Systems

Strategy	Cost Index (All-Manual = 100)	Escaped Defects per Release	Testing Cycle Time (Days)	Requirements Coverage (%)	Tester Time on Exploration (%)
All manual	100	7.8	6.5	71	12
All automated	84	6.9	2.6	82	8
Risk-based hybrid	61	3.4	3.1	91	41

The hybrid was thus the best strategy on cost, escaped defects and coverage, and second only to full automation on speed, while the extra half day of cycle time came from the manual exploratory work that gave it its advantage in quality. This result agrees with the pattern of Table VII. Because the two approaches find different defects, a strategy that retains both and assigns each to the work for which it is best reaches a quality that neither alone can achieve.

F. Decision Guide

Table X summarizes the guide that follows from the study. It links characteristics of a test or a system to a recommended approach. Tests that are run frequently, are stable and have objective results should be automated, preferably at the unit or service level. New, volatile or perceptual features should be tested manually and automated only when they have stabilized. Performance, load and compatibility testing should be automated because they are impractical by hand. Where the expected number of release cycles is below the break-even estimate, automation of regression testing is unlikely to pay and should be limited to the highest-risk checks. Where automation of tests is very expensive, for instance because of hardware dependence, the break-even point is later, and an incremental approach starting with the cheapest and most valuable tests is advisable.

TABLE X. Decision Guide for Allocating Test Effort

Characteristic	Recommended Approach	Reason
Frequent, stable, objectively checkable	Automate at unit or service level	Early break-even, low maintenance
New or rapidly changing feature	Manual, automate after stabilization	Scripts for volatile features break often
Usability, appearance, overall feel	Manual	Depends on human perception
Performance, load, compatibility matrix	Automate	Impractical by hand
Exploration for unexpected defects	Manual, given time released by automation	Finds defects not anticipated by scripts
Few expected release cycles	Manual, automate only highest-risk checks	Investment may not break even
Hardware-dependent tests	Automate incrementally, cheapest first	High investment, later break-even

Discussion

The results reject the framing of manual versus automated testing as a contest with one winner. Each approach found classes of defects that the other seldom found, and the best outcomes came from assigning each to the work for which it is suited. The V-model of Fig. 1 provides a useful organizing frame. Automation dominates at the lower and middle levels, where tests are frequent, stable and objectively checkable, while manual testing contributes most at the system and operational levels, where perception, exploration and judgment matter. The feedback paths of the model remind us that testing, whether manual or automated, is also a source of learning for earlier phases, and that automated suites must be updated as that learning changes the design.

The economic findings follow the shape of Fig. 2 but add specifics. The break-even point ranged from 4.5 to 12 release cycles, a spread of more than two to one, and the savings after 24 cycles ranged from 37% to 68%. These differences arose from the size of the suite, the cost of automation and the saving per cycle. The practical implication is that automation is not a universal good. It is an investment whose return depends on the number of times tests will run, and a team that expects few releases or whose application changes rapidly may never recover its investment. The late upturn of the automated curve in the figure, also seen in two of the systems,

indicates that maintenance is a continuing cost, and it should be budgeted from the start.

The analysis of maintenance by test level supports the long-standing advice to build a pyramid of tests. A small number of interface tests consumed most of the maintenance effort and generated most of the flaky failures, while the many unit and service tests were cheap to keep. The all-automated strategy, which leaned on interface tests, thus lost much of the theoretical benefit of automation. Teams that wish to avoid this outcome should place checks at the lowest level at which they can be expressed and should keep end-to-end tests few and focused on critical workflows.

An important and sometimes overlooked benefit of automation is the time that it gives to human testers. In the hybrid, testers spent about 41% of their time on exploratory work, more than three times as much as in the all-manual strategy, and the result was a lower rate of escaped defects. This suggests that the contribution of automation to quality is partly indirect. It not only runs tests quickly but also removes the monotonous work that consumes human attention, so that people can apply their skills where they are most valuable. The risk-based allocation adds a further benefit, since it directs the limited human effort to the areas of highest risk. Several limitations should be noted. The study covers four systems, observed in organizations with established agile practices, and the results may not hold for very small teams, for safety-critical systems with formal verification requirements or for systems with long release cycles. Effort was recorded by teams and may include errors. The comparison of strategies was in part modeled from recorded data and not from controlled experiments, so that differences between teams and release streams could have affected the results. Monetary costs of tools and infrastructure were not included in the person-hours and would delay break-even. The attribution of defects to the approach that found them first is a simplification, since defects may be found by more than one approach, and the value of a defect depends on its severity, which was not weighted in the counts. Finally, the benefits of testing in terms of customer satisfaction and brand reputation were not measured.

Practical guidance follows from these findings. Estimate the break-even point before investing, using the expected number of releases, the effort of manual regression and the cost of automation and maintenance. Begin by automating stable, frequent, low-level checks and extend gradually. Keep user-interface tests few. Preserve manual testing for new features, usability and exploration, and protect the time of testers for this work. Allocate effort by risk. Monitor flaky tests and maintenance effort as indicators of the health of the automation, and revisit the allocation when the product, the team or the release cadence changes.

Conclusion

This paper compared automated and manual testing and proposed a structured way to optimize their combination. We mapped testing activities to the levels of the V-model, modeled the economics of automation using a cumulative cost curve with an initial investment, a break-even point and cumulative savings, and compared the approaches in four industrial systems over 24 release cycles. Automation reduced regression effort to between 14% and 26% of the manual effort and reduced elapsed time from days to hours, and it broke even after 4.5 to 12 release cycles, yielding savings of 37% to 68% after 24 cycles. Automated testing first found 65% of the 1,046 recorded defects, mostly regressions, integration, performance and compatibility defects, while manual testing found 88% of usability defects and 79% of exploratory logic defects. End-to-end interface tests, only 13% of the automated suite, absorbed 64% of the maintenance effort. A risk-based hybrid reduced the testing cost index to 61, lowered escaped defects from 7.8 to 3.4 per release and

raised requirements coverage from 71% to 91%, outperforming both the all-manual and the all-automated strategies.

Future work will proceed in four directions. The first is to extend the study to more systems, including safety-critical and very small projects, and to use controlled experiments to compare strategies. The second is to include monetary costs and the value of defects by severity in the economic model. The third is to investigate the use of artificial intelligence for test generation, maintenance and flaky test diagnosis, which may change the balance between automated and manual work. The fourth is to develop tool support that recommends, for each test, the appropriate approach from the criteria of frequency, stability, cost, risk and perception.

References

- [1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.
- [2] B. Beizer, *Software Testing Techniques*, 2nd ed. New York, NY, USA: Van Nostrand Reinhold, 1990.
- [3] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Proc. Future of Software Engineering (FOSE) at ICSE*, 2007, pp. 85-103.
- [4] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 2016.
- [5] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mantyla, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *Proc. International Workshop on Automation of Software Test (AST)*, 2012, pp. 36-42.
- [6] V. Garousi and M. V. Mantyla, "When and what to automate in software testing? A multi-vocal literature review," *Information and Software Technology*, vol. 76, pp. 92-117, 2016.
- [7] V. Garousi and F. Elberzhager, "Test automation: Not just for test execution," *IEEE Software*, vol. 34, no. 2, pp. 90-96, 2017.
- [8] R. Ramler and K. Wolfmaier, "Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost," in *Proc. International Workshop on Automation of Software Test (AST)*, 2006, pp. 85-91.
- [9] J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect detection efficiency: Test case based vs. exploratory testing," in *Proc. International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2007, pp. 61-70.
- [10] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing: A Context-Driven Approach*. New York, NY, USA: Wiley, 2001.
- [11] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*. Boston, MA, USA: Addison-Wesley, 2009.
- [12] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proc. ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 643-653.

- [13] A. M. Memon, "GUI testing: Pitfalls and process," *Computer*, vol. 35, no. 8, pp. 87-88, 2002.
- [14] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," in *Proc. National Council on Systems Engineering (NCOSE) Annual Conference*, 1991, pp. 57-65.
- [15] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice Hall, 1981.
- [16] F. Shull et al., "What we have learned about fighting defects," in *Proc. IEEE International Symposium on Software Metrics*, 2002, pp. 249-258.
- [17] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Reading, MA, USA: Addison-Wesley, 1999.
- [18] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. Harlow, U.K.: Addison-Wesley, 1999.
- [19] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," in *Proc. Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 272-281.
- [20] V. Garousi and M. Felderer, "Worlds apart: Industrial and academic focus areas in software testing," *IEEE Software*, vol. 34, no. 5, pp. 38-45, 2017.
- [21] K. Beck, *Test-Driven Development: By Example*. Boston, MA, USA: Addison-Wesley, 2002.
- [22] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Upper Saddle River, NJ, USA: Addison-Wesley, 2010.
- [23] ISO/IEC/IEEE, "ISO/IEC/IEEE 29119: Software and systems engineering, Software testing," Geneva, Switzerland, 2013.
- [24] M. Felderer and R. Ramler, "Integrating risk-based testing in industrial test processes," *Software Quality Journal*, vol. 22, no. 3, pp. 543-575, 2014.
- [25] M. V. Mantyla, B. Adams, F. Khomh, E. Engstrom, and K. Petersen, "On rapid releases and software testing: A case study and a semi-systematic literature review," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1384-1425, 2015.
- [26] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: Empirical observations," *Advances in Software Engineering*, vol. 2010, article 620836, 2010.