servicetrace
SOFTWARE ROBOTICS

THE SOFTWARE QUALITY GUIDE.

Software testing, test automation.

WHY TEST SOFTWARE?

"We have as many testers as we have developers. When we do a new release of Windows, over half the budget goes into quality", Bill Gates tells InformationWeek in an interview in 2002. When CEOs, project managers and IT professionals have to balance tight budgets, software testing tumbles down the agenda and is soon shelved with excuses like "are the test costs reasonably proportionate to the potential effects of any software errors? Is it worth the cost?" The tendency is to plow the entire software development budget into programming – quality assurance doesn't get new software into the customers' hands.

Risk of software bugs

Edzard Höfig, university professor in Berlin, has a YouTube channel with suggested online lectures about software development. In the webcast called Introduction to software testing, he explains the importance of software tests in thought-provoking fashion, telling the story of the maiden flight of the Ariane 5 launch – a costly prestige project of the ESA that was supposed to secure the European space travel organization's leading position and attract umpteen investors.

Technologically, Ariane 5 was a complete new build and accelerated must faster than Ariane 4. The software in Ariane 5, however, was not a new development. Instead, it was based on the whole code of the predecessor version for Ariane 4. A code snippet in the launch system, which had resolved a launch-related problem in Ariane 4 and was no longer relevant to the launch capability of Ariane 5, failed to correctly interpret the high acceleration data and sent false information to the control logic. The rocket veered way off course, and this tripped the self-destruction mechanism.

On June 4, 1996, some 370,000,000 US dollars' worth of material went up in flames with the (unmanned) Ariane. The program code snippet that started the chain reaction that caused the rocket to explode would go down in history as one of the most expensive software errors ever.

The scenario described on a large scale here applies to smaller-dimensioned software projects as well. The message is clear: before new or altered systems are put into service, they must have successfully cleared an adequate test phase. Organizations that forgo software tests make only short-term savings and risk a great deal.

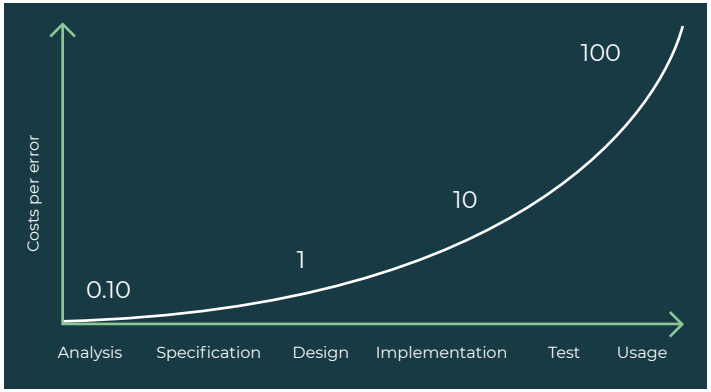
The cost of substandard software often outweighs the cost of failure prevention that would have been incurred by quality assurance. The "rule of 10" from industrial manufacturing, according to which the error costs of a product increase tenfold with every project phase, applies for the cost trend in the software's life cycle.

2.08 billion dollars

Software error costs in the USA in 2020

Some 1.56 billion dollars of this amount were incurred in productive operation. The remainder is accounted for by software that was not put into service due to poor quality.

Source: The Cost of Poor Software Quality in the US: A 2020s Report



The rule of 10 for error costs in software development

What is software quality?

The customer's viewpoint offers a pragmatic definition:

- the software has to fulfill the customer's requirements and support workflows in the best way possible
- the software has to run without errors and cause no disruptions within the operation.

This gives rise to the following priorities for the development of high-quality software:

- focus on clearly and fully defined requirements
- focus on debugging before going live

The measures below are derived from this:

- close and continuous liaison between customer, product owner and developers
- integrate tests into the development process

Failure and bug fixing

The costs of software errors have both direct and indirect effects. Costs for localizing the cause of the error (debugging) and for rectifying the error (bug fixing) are incurred directly. Well-paid software engineers spend, on average, half their working time on unproductive bug fixing procedures they detest – time they could invest in creatively refining the software.

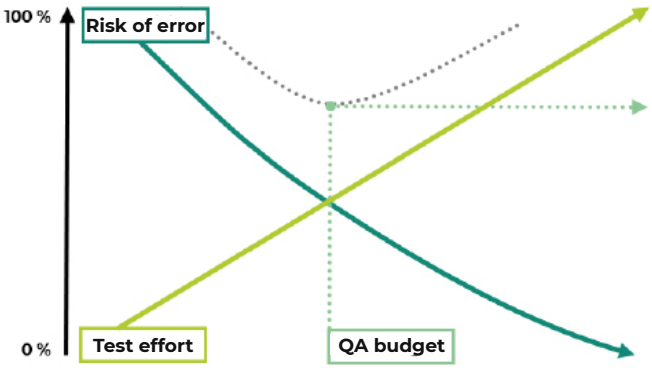
Indirect and incalculably high costs stem from the failures that would occur if the software errors were to make their way into production. These costs include the failure of productive processes, lost customers, waning reputation and contractual penalties.

But it's not just the economic loss that unhinges customers from the competition in the medium-term, the missed opportunity to use high software quality as a unique selling point wreaks even more havoc.

Test expenditure versus error costs

An economically viable investment in quality assurance takes the test effort and error cost ratio into account. The graphic shows that as test expenditure increases, the risk of potential error costs drops. From a certain point, the test costs top the possible error costs. The values for your optimum quality budget settle at the low point of the parabola above this intersection.

Optimum expenditure for software QA



Test costs	Error costs
€ 300.000	€ 0
€ 270.000	€ 100.000
€ 240.000	€ 200.000
€ 210.000	€ 300.000
€ 180.000	€ 400.000
€ 150.000	€ 500.000
€ 120.000	€ 600.000
€ 90.000	€ 700.000
€ 60.000	€ 800.000
€ 30.000	€ 900.000
€ 0	€ 1000.000

At maximum test costs of 300,000 euros and maximum assumed error costs of 1,000,000 euros, the example calculates a value of between 210,000 and 240,000 euros to be invested in quality assurance.

Make software quality a permanent feature of your digital strategy and devote more of your development budget to software tests. Quality prevails and pays off:

- you position yourself among numerous competing providers as a producer of outstanding quality goods
- you establish your permanent position on the market as a reliable digital manufacturer with top quality, long-lasting software products

WHY AUTOMATE SOFTWARE TESTS?

The accelerated dynamic of global market development and the growing demands set by consumers are generating high innovation pressure, especially in the fast moving digital sector. To keep pace with the competition, companies have to put new software products in operation more frequently and at a faster speed than the competition. 2 or 3 releases per year are no longer enough. Especially if the software itself is destined to become the business model, updates at shorter – quarterly or even weekly – intervals are required. Digital giants like Google, Facebook or TikTok make several releases a day.

Agile software development

The trend towards increasingly quick release cycles with more and more frequent updates is aligning to the requirements of the globalized and digitalized 21st Century, an era when consumers, providers and products interact spontaneously, elusively and unpredictably. Long-term planning no longer suffices in our cut-and-thrust world: this first opened the door in software development in the early 2000's to agile frameworks and methods such as Scrum, Kanban and Lean Management.

Agile development (lat. agilis: “lithe, swift) is the polar opposite of a linear development process where all expectations of the software are specified together with the purchaser at the start of the project, the software then being implemented over several months of development with all new functions and features and released with a big bang. This traditional software engineering approach often leads to disappointing wayward developments.

Traditional software development: failure on schedule?

Nowadays, software systems have to support such complex and dynamic business environments that the demands on the product *cannot* be defined completely, reliably and clearly at the outset. Meaning, requirements will change completely or new ones will be added during the course of the project. If the software now continues to be developed as planned “in the silo”, the result at the end is expensive software headed for the trash can.

VUCA describes the altered framework conditions and challenges for organizations in the digital age. Consumers no longer tie themselves to brands for long periods. Instead, they spontaneously swap to providers that offer products to suit their **fluctuating** (volatile) requirements. Future market developments and customer requirements are always **uncertain**: our **complex** and **ambiguous** world with globally and digitally networked economic cycles and omnipresent multi-optionality is proving to be unpredictably dynamic and leaves no room for planning reliability.

This heralds the end of linear thinking and linear strategies, paving the way for agile methods.

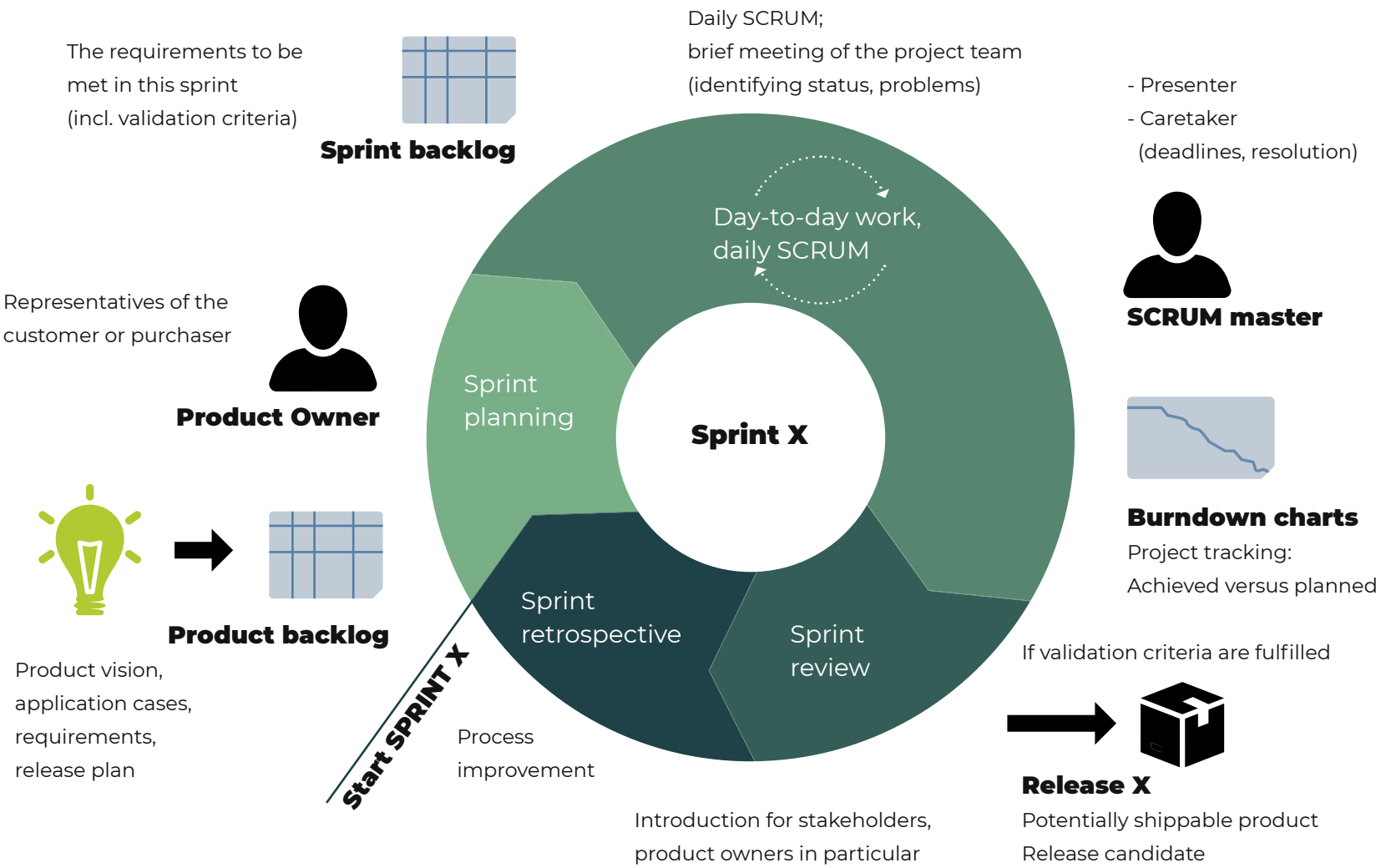
To create the software that the customer will actually use, agile software development according to the Scrum framework takes an iterative (=approaching the solution step-by-step) and incremental (advancing step-by-step, building step-by-step) course.

Scrum: to usable software in partial stages

At project kick-off, the purchaser formulates the desired product features as “user

stories", which list and prioritize the product owners in the product backlog. The developer team gradually implements the new functionalities in "sprints" lasting one to four weeks. At the end of each sprint is a ready-to-use sub-product which the customer validates again. After the "sprint reviews", the demands on the software can be adjusted in further talks between product owner and purchaser.

Thanks to the modularized development process, the rapid release cycles and the continuous feedback loops with the purchaser, agile software development is able to respond flexibly to volatile customer preferences, minimizing the risk of wayward developments in meeting customer requirements.



Agility is a trend

Properly functioning software is more important than comprehensive documentation, ongoing communication with the customer ranks above abstract contract negotiations and responding to change counts for more than rigidly sticking to a plan – the new principles for software development, the Agile Manifest 2001 formulated in revolutionary gestures, are now the gold standard for organizations that want to scale up their innovation pace and stay close to the customer in the process.

The figures in the info box shows two things:

- when introducing agile methods, most companies prioritize speed, closely followed by flexibility
- just on 60% still pay little attention to software quality

Software testing: an expensive bottleneck?

One reason for the reticence is the cost involved in quality assurance. On average, a sophisticated test process accounts for around 30-50% of the total budget for a software project.

Secondly, software testing is a time-consuming process. The all-important end-to-end test, for example, which is designed to check whether the customer can use the software system to successfully implement business-critical processes such as concluding a contract or ordering products, can soon contain hundreds of test cases involving thousands of sub-steps in the case of complex applications in large integrated networks. The colossal amount of time invested in the design, performance and evaluation of the tests thwarts agile teams and processes.

In Gartner's 2018 Agile Survey, 44% of respondents identified working with teams using traditional methods as one of their top-three challenges for adopting agile development. Testing is a common constraint in this situation due to its … reliance on manual efforts; as a result, quality is slow to achieve and ineffective.[1]

When resources are scarce anyway and there is pressure to innovate, the budget is often almost completely deflected from risk prevention to further development.
As the pace of delivery increases … manual testing can't keep up and quickly becomes a bottleneck. As a result, testing often gets sacrificed and quality deteriorates …[2].

The consequences of companies forgoing software tests because they pose a threat to the pace of digital innovation is described in the *Why test software?* section. Only good software quality prevails over the long term and improves the ability to compete. Without tests, however, it's not sustainable.

[1] Christopher Little, Jim Scheibmeir: When Dev goes agile, optimize operations' testing capabilities with automation, Gartner Research 2018
[2] Joachim Herschmann, Thomas Murphy, Jim Scheibmeir: 4 Essential Steps to Implement Test Automation, Gartner Research 2019

Automation for faster, cheaper and better testing

Better to deliver refined quality or put substandard products in circulation quickly? If manual procedures in the test process are automated, software project managers are not required to make such a hard and fast choice between the quality of software and the speed of innovation. With a sophisticated test automation strategy, quality assurance can not only pick up the agile pace of the software development; the automated software tests also raise product quality and cut the cost of testing.

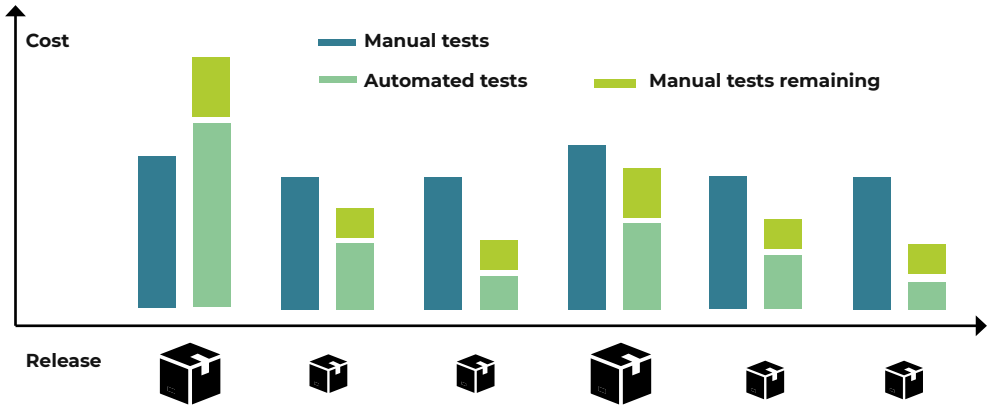
More agility	
• faster test implementation	• test results swiftly reported to development on a regular basis
• more frequent testing	• faster bug fixing and further development
Better quality	
• increased test coverage	• low risk of errors being overlooked
• standardized test execution	• possibility of man-made errors ruled out
Falling costs	
• fewer software errors	• lower error costs
• lower costs for test implementation	• after return on investment, ongoing automation costs a fraction of the salary earned by a software tester
Efficient use of resources	
• liberates qualified and expert testers from time-consuming, tedious and error-prone routine tests	• frees up resources for creative and complex tasks (usability tests, exploratory tests) or re-qualifications (> test automation engineer)

Test automation: Invest and RoI

However, it would be naive to think of test automation as some immediately profitable wizardry, created as if by magic. The conversion from manual to automatic calls for an up-front investment which exceeds the cost of manual testing initially. This involves:

- Conviction and commitment of the test team, company management, IT management, purchasing
- Selectionof test stages and cases to be automated
- Comparison between and selection of automation tools
- Staff training
- Definition of (new) processes, roles and tasks within the test team
- Creation and testing of automated test cases

However, the more often you release software updates and the longer you continue developing the software, the faster and better the initial outlay is returned.



The cost of test automation drops with each release. This is because, once automated, the tests can be easily reused for the next minor release. Then at the next major release, only the new test cases need to be added. For manual tests, by contrast, the cost replicates with each release and even increases as the application becomes more complex.

servicetrace SOFTWARE ROBOTICS

TEST STAGES AT A GLANCE

Cost high

Test stage	Layer	Goal	Implementation	Tools (e.g.)
Acceptance test Acceptance by the customer/purchaser	GUI	Validation and acceptance of the finished software system by the purchaser	Customer, professional user, product owner	Manual tests
System integration test Tests the interoperability with other systems	GUI	To test business processes for error-free practicability with cross-system E2E click trails	Tester	Ranorex Servicetrace: Tricentis
System test Tests the fully integrated system from the user's perspective	GUI	To test the system/software for completeness and functionality of defined system requirements	Tester	Ranorex Servicetrace Tricentis
Integration test Tests the interaction between at least two Components	Interfaces/APIs	To test the functionality of integrated components. Incremental procedure until all components are integrated as a complete system	Developer, tester	Citrus
Component test/ unit test Tests single software modules immediately after implementation	Code	To test the stability and functionality of fragmented and isolated software units. Errors can be quickly assigned to the relevant lines in the code and rectified without a problem	Developer	JUnit MSUnit Pytest

Business perspective

System perspective

Cost low

This overview bundles the test stages that are essential to continuous quality assurance in software development. Only when a build has successfully cleared a test stage at a lower hierarchy will it be incorporated into the next test stage. Cost and time increase as the test stages move up the scale from lower to higher.

Automated test runs are possible at all test stages up to the acceptance test by the customer. A high degree of automation throughout the test process achieves a) a time saving thanks to accelerated test throughput times and fast feedback to development and b) a higher software quality thanks to increased test coverage.

8 STEPS TO THE TEST AUTOMATION STEPS FRAMEWORK

The last section describes best practices for a perfectly efficient and profitable introduction of a test automation procedure.

1) Basics: this is where automation pays off

Use the 4 factors below to verify which tests you should be automating:

- Application life cycle: automate tests for long-life applications that are expected to issue many more releases.
- Test frequency: automate the tests that are required at each release.
- Stability: as long as a defined feature continues to be developed and is subjected to frequent modifications, testing should be manual since automation in this case would involve a high amount of maintenance. Automate if the implementation of the feature is stable.
- Business risk: prioritize the test cases using a risk analysis and automate tests for those features which would generate the most negative impacts on business processes should they fail.

2) Quick win: automate regression tests

Regression tests are run at every test stage. They check whether a test can be successfully repeated after a change to the software or in the software system, or whether such change has brought about undesirable side effects.

The frequency of execution, the repetitive and monotonous nature of the tests and the simple reuse of the test cases make regression tests a top candidate for automation with a high savings potential.

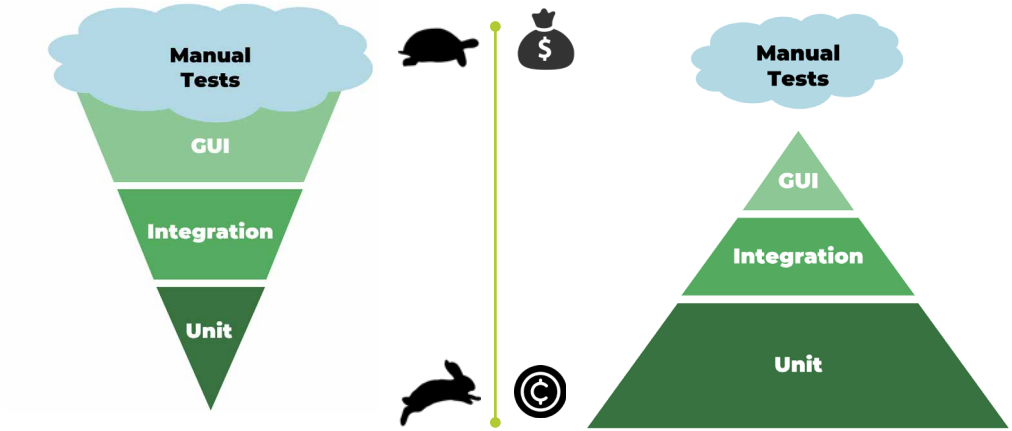
3) Expand: identify more test cases for automation

The following test types are also suitable for automation due to their frequency and repetitive nature:

- Smoke tests ensure the basic operability of the application (e.g. start application, login, selected key functionalities) and should be carried out before all other test activities
- Data-driven tests pass through the same test case with various data sets (e.g. user name, customer number, email, payment method, etc.)
- Cross-browser tests pass through the same test case in various standard browsers (Chrome, Firefox, Edge, Safari)
- Non-functional performance and load tests measure the reaction time of the software with single and parallel access

4) Shift-left: test early, test a lot

The test automation pyramid by Mike Cohn offers valuable orientation for recommended automation outlay according to test stages.

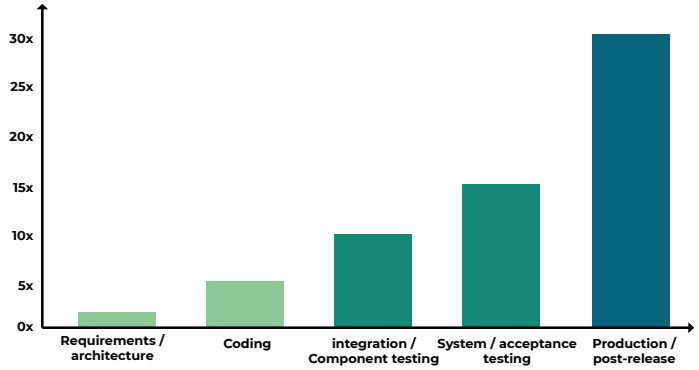


Test strategies: ice cream cone pattern versus test automation pyramid

The common automation practice in software projects is the top-down approach of the ice cream cone pattern: the biggest proportion of the automation budget is invested in GUI tests, the aim being to chase down software errors at the end of the development phase. This method lacks promise and is unprofitable:

- creating and managing the automated GUI tests is relatively expensive and time-consuming
- An error at system level does not provide any proof of which points in the code are affected. Finding and fixing the cause is a tedious and long-winded process
- The further down the development process a bug is found, the more it costs to fix

A bottom-up approach or a pyramid-shaped weighting of the automation outlay along the test stages offers more promise. This approach is based on the premise



The cost of bug fixing increases with every progressive stage of the development process. Regular and early testing significantly reduces these costs.

that software quality should not be “tested” only at the end of the development process at system level, but rather has to be integrated into the product at the outset. This paradigm leads to the “shift left” in software quality assurance: the tests do not begin only at the end of the development phase, instead they run parallel to and support the development process as a whole. With Test Driven Development (TDD), the tests are written even before the software module to be tested. This compels programmers to consistently develop using the cleanest possible code for the required functionalities and/or for the testability of the test item.

The broad basis of the test automation pyramid therefore accounts for the highest number of unit or module tests. The advantages:

- small outlay for creating and managing the tests, fast runtime
- module tests are run in the development environment itself rather than in an expensive test environment, since no dependencies on other system components are tested
- the test itemsare designed with fine-grained modularity and concern a dedicated section in the code, allowing the test results to lead directly to the cause of the error
- module tests reliably validate the functionality and robustness of the code for acceptance into the next test stage

The next biggest volume of tests or automation outlay are the integration tests that validate the interoperability of the individual modules. Only those items that pass the automated integration tests are implemented in the software system.

Afterwards, a small number of selected automated GUI tests can test the most important business-critical E2E click trails for practicability or regression whereas only exploratory, complex GUI tests are run on the new system manually.

5) Shift-right: test the customer’s viewpoint before and after going live

According to the test pyramid, E2E tests from the customer’s viewpoint form the smallest unit of tests to be automated. They are less suited to debugging and bug fixing because one failed function can potentially concern thousands of code sections. Rather, as acceptance tests they are used to validate the practicability of critical business processes from a customer’s viewpoint, i.e. demonstrate the practical suitability of the software.

Automated GUI tests are more costly to create and maintain than unit or integration tests. Here, 3 leverage points make automation more efficient and profitable:

1. You select a tool that extensively simplifies the GUI tests (more on this in the GUI tests section: criteria for the tool selection)
2. You use the automated E2E test cases for functional tests, performance tests and load tests
3. You continue to use the automated E2E test cases after the release goes live and employ the most important business processes from the user's viewpoint for continuous E2E monitoring

"Shift-right" in this context means using the automated quality assurance beyond the development phase throughout the rest of the application's life cycle.

6) GUI tests: tool selection criteria

GUI tests are considered to be extremely costly to create and difficult to maintain. Here, we list some problems that can typically occur during the GUI test automation process – and how you can minimize them by selecting a suitable tool.

Simple and fast test creation and adaptation

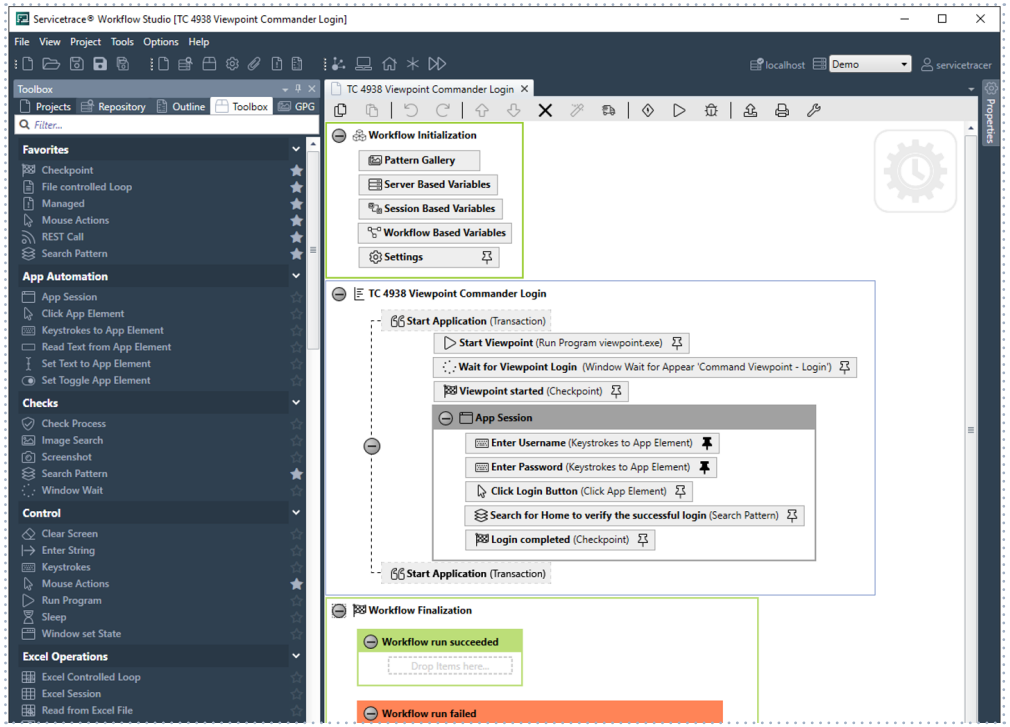
Because GUI tests involve hundreds to thousands of test steps and have to be adapted to new functionalities following implementation, there are two important criteria for tool selection:

a) the procedure used to create the tests must be as simple as possible,

b) the tests must be quickly and selectively adaptable.

For simple test creation, we recommend no-code solutions that allow test cases from predefined modules to be merged into one structured E2E workflow in a graphic editor via drag & drop. Not only is this faster than scripts, it also allows remote testers or professional users from the business world to create automated E2E test cases single-handedly. Business users understand only too well the requirements or the business processes in their department to be tested and hence contribute all professional qualifications to creating appropriate and adequate test cases for E2E acceptance tests.

Indeed, capture & replay approaches are ostensibly more convenient and faster than the drag & drop procedure for creating tests in a graphic editor. The drawback, however, is the monolithic and inflexible nature of the test cases recorded. Each time the GUI is changed, all the test cases have to re-recorded, i.e. completely run again by a manual process.



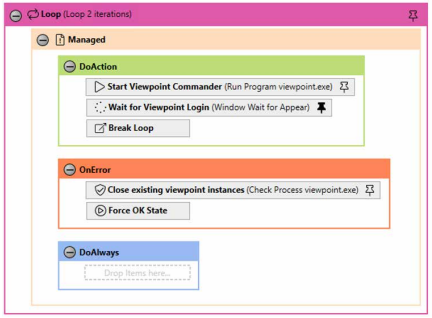
Create GUI tests without programming using the no-code solution, e.g. with the Workflow Studio from Servicetrace Workflow Studio.

Modularly and granularly structured graphic E2E workflows created via drag & drop, however, can be systematically adapted only to the sequence affected by the change to the GUI. If, for example, the icon for starting the application changes, only the image recognition for this icon has to be reconfigured accordingly.

The option of managing elements that occur in multiple workflows in a central repository simplifies and speeds up the process even further. Any adaptations required on an element then need to be made just once centrally and rolled out on all automation runs.

Error handling

Long E2E workflows usually consist of a sequence of aligned test cases. If one test case in the workflow fails during execution, none of the subsequent steps can be carried out and the test automation sticks at the last failed transaction. To ensure that E2E workflows can be fully executed even in case of a sporadic error case, the test automation tool should include a stable error handling function.



Servicetrace Test Automation handles errors by means of managed blocks in the automation workflow: the further behavior of the automation solution in case of a failed sequence is defined here. Any errors that occur are documented with screenshot and analytics package – and the next sequence in the workflow is completed.

Error tolerant: integrated error handling function increases the stability of long E2E tests.

Notifications / pop-ups

Pop-ups that occur spontaneously, during updates of other applications integrated into the workflow for example, can also affect the stability of GUI test automations. You should therefore make sure when selecting your tool that the automation solution is able to intercept and handle such events in automated fashion.

Procedure for waiting times

The test system sometimes reacts more slowly than usual – and the automation process has already run into an error. Your test automation solution should therefore allow configurable waiting times for events in the workflow, e.g. until a browser window opens and displays a certain content.

Dynamic web GUIs

With web applications in particular, GUI elements often change or are displayed differently depending on the user settings. Processes that run on the user interface directly, such as image or pattern searches, are not suitable in this case.

Your automation solution should therefore offer, especially for the automation of dynamic web applications, other processes such as the control of HTML objects.

Divergent system settings

Resolution, color intensity or position of the elements displayed on the desktop may vary, depending on the system settings on the end device.

To ensure your GUI tests are highly stable, the automation tool must behave robustly toward divergent desktop settings.

Hardware and software resources

The test environment setup usually calls for additional hardware and software licenses. You can slash costs if your automation tool allows parallel tests on one end device.

Tool compatibility

Simple integratability into standard test management tools such as Jira or Jenkins is advantageous.

7) Test automation and teamwork: new tasks and roles

If you are planning to introduce a test automation solution, you should assure your employees that they are not about to be replaced by automated processes. Test automation cannot and will not completely replace manual testing. Its purpose is to take the pressure off testers, not make them redundant.

Therefore, you should clearly state from the outset that test automation handles those tedious, repetitive and boring tasks inherent in all software releases which push every test team beyond their stress limit. Automation gives testers much more time for intellectually challenging, exploratory tests that better match their qualifications, lend their work a new sense of purpose and hence increase their job satisfaction.

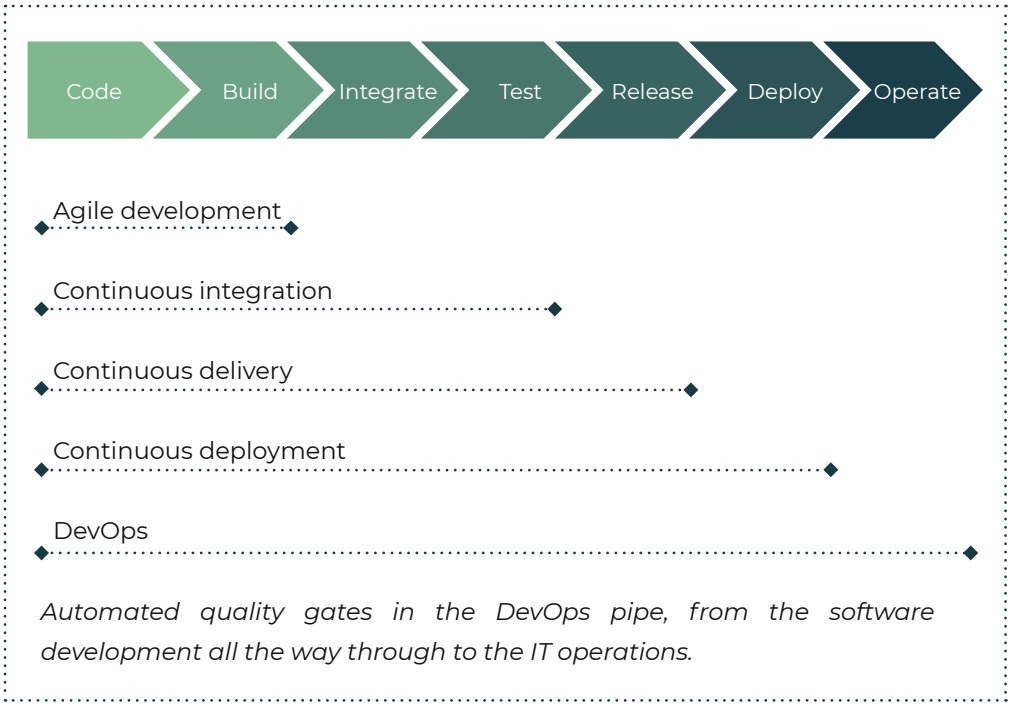
In addition, automation is opening the door to new fields of activity: for test automation engineers, for example, who dedicate themselves fully to the issue all of the time. Testers who have performed tests manually to date could even consider a further qualification – either at the tool provider directly or through a certification at the ISTQB as a "Certified Tester Expert Level –Test Automation".

8) DevOps: continuous delivery, deployment, operation

Test automation is a key element of a comprehensive DevOps pipeline in which a new software build clears various quality gates in automated fashion. The code acceptance process takes place via automated unit tests, with JUnit for example. Integrated modules are accepted using continuous integration tools, such as Jenkins and completely implemented systems are accepted via GUI test tools, such as Servicetrace Test Automation.

The next step in an efficiently automated DevOps pipeline is the automated deployment of the releases that have successfully passed the previous automated tests. Tools like Maven or Docker can then be used to load the new software version into the live productive system at a defined time by an automated process.

The automated DevOps pipeline is rounded out by continuous E2E tests (end-to-end monitoring) of business-critical click trails in the productive environment.



Automated quality gates in the DevOps pipe, from the software development all the way through to the IT operations.

From the field:
Customer stories about test automation

British Heart Foundation counts on performance tests

"The test automation solution from Servicetrace fulfills our exacting requirements for volume and transaction tests to perfection."
Sarah Yates | Project Manager

DSL and mobile radio provider 1&1 scales using software robots

"Without automation, regular and comprehensive regression tests like these would not have been possible."
Andreas Förch | Head of Testing Unit

Change? Of course. Test Automation at Dekra SE

"This is really the only tool that allows us to run complex tests in all applications."
Katharina Hauch | Test Factory Manager

Servicetrace:
No-code solution for automated GUI tests

Digital innovations occur in ever shorter intervals – frequently in the break between agile sprints. You can reliably and cost-effectively manage the huge test volume that arises here using Servicetrace test automation.

Quality takes time – but when time is limited, quality must not be allowed to suffer. For this reason, tester teams rely on Servicetrace Test Automation whenever QA processes need to be accelerated during software development. The solution is suitable for all applications without exception (web / non-web).

Customers value the no-code approach of the solution. The test cases are created and adjusted quickly and easily in the Workflow Studio using an intuitive, graphical drag & drop method. This means that the users from the functional departments can independently automate software tests.

Servicetrace Test Automation provides a comprehensive test suite in which you can manage epics, stories and test cases. The test cases are archived in an audit-compliant repository. Following execution of the test, the test results with a detailed analysis package deliver convenient feedback to development.

Do you manage your work in Jira? That's perfect – Servicetrace Test Automation can be fully integrated.

Sign up for Robotic News & stay up-to-date
www.servicetrace.com/de/robotic-news