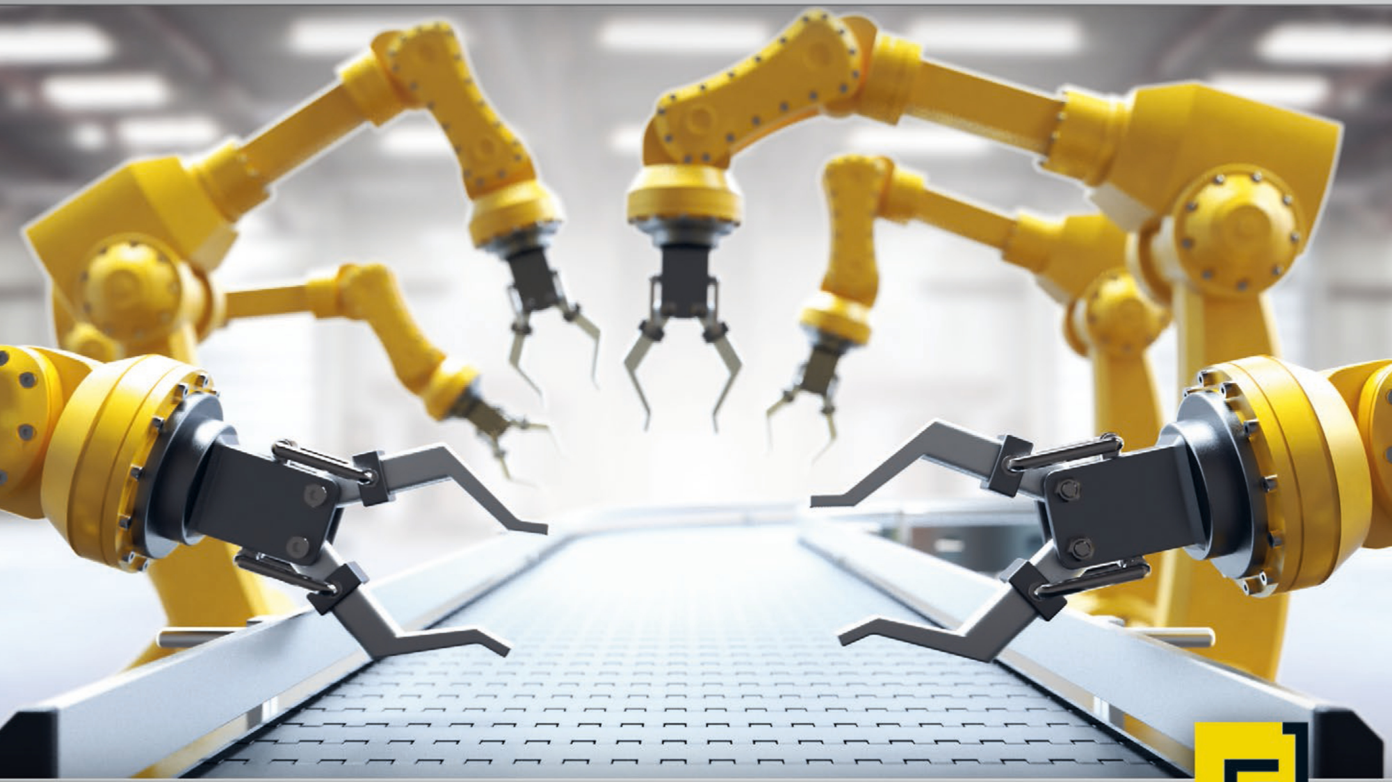
dpunkt.verlag



Manfred Baumgartner · Thomas Steirer · Marc-Florian Wendland
Stefan Gwihs · Julian Hartner · Richard Seidl

Test Automation Fundamentals

A Study Guide for the Certified Test Automation Engineer Exam
• Advanced Level Specialist
• ISTQB® Compliant

About the Authors

Manfred Baumgartner has more than 30 years of experience in software testing and quality assurance. Since 2001, he has established and expanded the QA consulting and training services at Nagarro, a leading software testing services company. He is a board member of the Association for Software Quality and Further Education (ASQF) and the Association for Software Quality Management Austria (STEV). He is also a member of the Austrian Testing Board (ATB). He shares his extensive experience at numerous conferences and in his articles and books on software testing.

Thomas Steirer is a test automation architect, test manager and trainer, and leads Nagarro's global test automation unit. He qualified as an ISTQB® Certified Tester - Full Advanced Level in 2010. He is a lecturer for test automation in the master's program for software engineering at the UAS Technikum in Vienna, and does research into the use of artificial intelligence for increasing efficiency in test automation.

Marc-Florian Wendland is a research associate at the Fraunhofer FOKUS institute in Berlin. He has more than 10 years' experience in national and international, cross-domain research and industrial projects that involve the design and execution of test automation. He is a member of the German Testing Board (GTB) and a trainer for various ISTQB® programs.

Stefan Gwihs is a passionate software developer and tester, and is a test automation architect at Nagarro, where he currently focuses on test automation for agile software development and DevOps.

Julian Hartner is based in New York City. He is an ISTQB® certified quality engineer and a passionate software developer and test automation engineer. He currently focuses on streamlining manual and automated testing for CRM applications.

Richard Seidl has seen and tested a lot of software in the course of his career: good and bad, big and small, old and new, wine and water. His guiding principle is: "Quality is an attitude". If you want to create excellent software, you have to think holistically and include people, methods, tools, and mindset in the development process. As a consultant and coach, he supports companies in their efforts to turn agility and quality into reality, and to make them part of corporate DNA.

Manfred Baumgartner · Thomas Steirer · Marc-Florian Wendland ·
Stefan Gwihs · Julian Hartner · Richard Seidl

Test Automation Fundamentals

A Study Guide for the Certified Test Automation Engineer Exam

- Advanced Level Specialist
- ISTQB® Compliant

dpunkt.verlag

Manfred Baumgartner · *office@manfred-baumgartner.at*

Thomas Steirer · *contact@thomas-steirer.com*

Marc-Florian Wendland · *marc-florian.wendland@fokus.fraunhofer.de*

Stefan Gwihs · *stefan.gwihs@nagarro.com*

Julian Hartner · *julhartner@gmail.com*

Richard Seidl · *office@richard-seidl.com*

Editor: Christa Preisendanz
Editorial Assistant: Julia Griebel
Copyediting: Jeremy Cloot
Layout and Type: Frank Heidt, Veronika Schnabel
Production Editor: Stefanie Weidner
Cover Design: Helmut Kraus, *www.exclam.de*
Printing and Binding: mediaprint solutions GmbH, 33100 Paderborn, and Lightning Source®, Ingram Content Group.

Bibliographic information published by the Deutsche Nationalbibliothek (DNB)
The Deutsche Nationalbibliothek lists this publication in the Deutsche
Nationalbibliografie; detailed bibliographic data can be found on the Internet at *http://dnb.dnb.de.*

ISBN dpunkt.verlag:

Print	978-3-86490-931-3
PDF	978-3-96910-870-3
ePUB	978-3-96910-871-0
mobi	978-3-96910-872-7

ISBN Rocky Nook:

Print	978-1-68198-981-5
PDF	978-1-68198-982-2
ePUB	978-1-68198-983-9
mobi	978-1-68198-984-6

1. edition 2022 Copyright © 2022 dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Title of the German Original: Basiswissen Testautomatisierung
Aus- und Weiterbildung zum ISTQB® Advanced Level Specialist – Certified Test Automation Engineer
3., überarbeitete und aktualisierte Auflage 2021
ISBN 978-3-86490-675-6

Distributed in the UK and Europe by Publishers Group UK and dpunkt.verlag GmbH.
Distributed in the U.S. and all other territories by Ingram Publisher Services and Rocky Nook, Inc.

Preface

"Automatically better through test automation!?"

One hundred percent test coverage, a four-hundred percent increase in efficiency, significantly reduced risk, faster time to market, and robust quality—these were, and still are, the promises made by test automation; or rather by those who make their living with test automation tools and consulting services. Since the publication of our first book on the subject in 2011, test automation has been on the to-do list of almost all companies that produce or implement software. However, the promised and expected goals are rarely achieved. In fact, there is a significant discrepancy between the potential achievements presented in the tool vendors" glossy brochures and the uncertainty in many companies regarding the successful and sustainable use of test automation.

This book provides a broad-based and practical introduction that serves as a comprehensive guide to test automation for a variety of roles in the field. In the fast-moving IT market, test automation has developed rapidly in recent years, both technically and as a discipline in its own right. Scalable agility, continuous deployment, and DevOps make test automation a mission-critical component of virtually all software development.

These dynamics also affect all test automation tools, whether commercial or open source. Therefore, this book doesn't go into detail on specific tools, as any functional evaluation would surely be superseded by the time it goes to print. Additionally, there are so many great open source and commercial sector tools available that picking favorites would be unfair to the other manufacturers and communities. Instead, we list tools suitable to the test automation architecture and solutions discussed in each chapter. Tool comparisons and market research are available quickly and easily on the internet, although you have to remember that these are often not updated regularly.

The importance of test automation has also been confirmed by the international testing community. In 2016, the first English-language version of the ISTQB® *Advanced Level Syllabus Test Automation Engineer* was published—a milestone for the profession of test automation engineers. In late 2019, the German version of the syllabus was released [ISTQB: CT-TAE], which was an important step for the German-speaking ("DACH") countries. This makes test automation more than ever an indispensible core component of software testing in general and provides it with its own certification and educational syllabus.

Previous editions of this book were always ahead of the published syllabus, but we felt the time had come to align ourselves with this established international standard, which is designed to support knowledge sharing and a common test automation language. Furthermore, the book introduces you to the contents of the syllabus and helps you to prepare for the certification exam. The syllabus is highly detailed and is a reference book on its own. However, this book adds significant value by providing a practical context, an easy-to-read format, and real-world examples that make it much easier to gain a firm grasp of the subject matter than you can by studying the syllabus alone.

In short, this book not only prepares you for the certification exam, it also teaches you practical, hands-on test automation.

The contents of the curriculum (currently the 2016 version) are presented in a different order and with different emphases to the syllabus itself. We also supplement the syllabus content with other important topics that are clearly marked as excursus.

Please note that the certification exam is always based on the current version of the official syllabus.

In addition to reading this book, we recommend that you attend an appropriate training course and use the current version of the syllabus [ISTQB: CT-TAE] to prepare for the exam.

Covering the curriculum is only one of several major points that we address in this book and, aside from this, our three main goals are as follows:

Firstly, we want to help you avoid disappointment due to overblown expectations. Test automation is not a question of using specific tools and is not a challenge to implement the marketing buzzwords used by software manufacturers, but rather a resource that enables you to better cope with the constantly growing demands made by software testing.

Secondly, we give you guidance on how to make best use of this resource. We focus on the long-term view, future return on investment, and the real-world business value it provides. These aspects cannot be measured using metrics such as code coverage or the number of test scripts, but rather by the total cost of ownership of application development, evolution, and benefits, as well as user feedback in the marketplace.

Thirdly, we have incorporated key aspects of the test automation process, such as the role of test automation in the context of artificial intelligence (AI) systems and in the DevOps environment.

Does test automation automatically make things better? Certainly not! A manufacturing machine that is set up incorrectly will produce only junk; if it is operated badly, it will produce random, useless results; if it is not properly maintained, it will break down or perhaps even become unusable. Appropriately trained employees, sustainable concepts, a responsible approach, and the awareness that test automation is an essential production factor are the prerequisites for realizing the potential and the real-world benefits of this technology. In most cases, test automation is indispensable for delivering robust quality in agile project environments, making it critical to the success of a project. It is also essential for keeping pace with the speed of modern continuous delivery processes while ensuring the long-term economic viability of software development projects.

We wish you every possible success implementing test automation at your company.

Manfred Baumgartner
Thomas Steirer
Marc-Florian Wendland
Stefan Gwihs
Julian Hartner
Richard Seidl
May 2022

Acknowledgements

For their hands-on support we would like to thank Michael Hombauer, Sonja Baumgartner, Dominik Schildorfer, Anita Bogner, Himshikha Gupta, Christian Mastnak, Roman Rohrer, Martin Schweinberger, Stefan Denner, Stephan Posch, Yasser Aranian, Georg Russe, Vincent Bayer, Andreas Lenich, Cayetano Lopez-Leiva, Bernhard König, Jürgen Pointinger, and everyone at Nagarro.

This book is dedicated to Himshikha Gupta, who worked tirelessly to create the figures and diagrams it contains, and who passed away much too early, shortly before it was finished in early 2022. She will be sorely missed.

Foreword by Armin Metzger

The second wave is here! I believe we are in the middle of the second wave of test automation. The first big wave clearly took place in the early 2000s, and the projects involved were initially very successful in terms of improving the effectiveness and the efficiency of test processes in some specific areas. However, in line with the Gartner cycle, the "trough of disillusionment" was quickly reached and, in my view, most projects didn't actually reach the "plateau of productivity".

What I observed at the time were projects that expended enormous effort over several years to work their way to a high degree of test automation. Then came technology changes such as the switch to .NET platforms, or process changes such as the switch to agile development methodology. A lot of the test automation frameworks didn't survive those transitions. Back then I liked to give talks with provocative titles such as *Test Automation Always Fails*.

We saw two core problems: firstly, companies failed to scale isolated successes to the entire project or organization, and secondly, test automation platforms were not sufficiently flexible to absorb disruptive changes in the technology base.

It is therefore no surprise that, over time, test automation began to lose acceptance. Management aspects also play a supporting role here. In the long run, the great economic expectations of a one-time investment intended to significantly reduce regression efforts were often simply not met.

Since the middle of the second decade of the 21st Century, we see a trending new wave of test automation in large projects. Will test automation once again fall short of its expectations? I don't think so. Both the overall test automation environment and the expectations test automation raises have changed. Test automation has now re-established itself as an indispensable factor for the success of projects in current technological scenarios. What changed?

With the introduction of agile processes, highly automated, tool-supported development has evolved significantly and has now become standard practice. Continuous integration concepts are constantly being refined into DevOps processes to create a seamless platform for the integration of automated project steps—all the way from the initial idea to final production and operation. The end-to-end automation of processes naturally forms an excellent basis for integrating test automation into the overall development process. Additionally, agile processes have helped process scaling to reach a new, higher level of importance. This development is an essential factor for the successful introduction and long-term establishment of test automation solutions.

However, a key factor in the importance (and necessity) of test automation is the current technological platform on which we operate. Disruptive technologies such as IoT (Internet of Things) and AI (artificial intelligence) are rapidly pushing their way out of their decades-old niche existence and into our products. With this comes a significant shift of priorities for the quality attributes we have to test. While 20 years ago, ninety per cent of all tests were functional tests, the importance of non-functional tests for usability, performance, IT security, and so on is slowly but surely gaining ground. The number of test cases required to assess product quality is therefore increasing rapidly, and only automated tests can effectively safeguard quality characteristics such as performance.

The development and maintenance of products takes place in increasingly short cycles. Due to the increasing variance in hardware and software configurations, entire and partial systems need to be tested in an increasing number of variants. Non-automated regression testing thus becomes an increasing burden, and it becomes more and more difficult to achieve the required test coverage while retaining an adequate level of effort.

And—fortunately—we have also learned a lot about methodology: test architectures are one of the most important factors (if not *the* most important factor) influencing quality in the maintainability of automated tests. In fact, test architectures are now so well established that the dedicated role of *test architect* is now being introduced in many organizations. This is just one example of such changes.

But beware: using the right approach and having knowledge of the pitfalls and best practices involved in introducing and maintaining test automation are key to long-term success. Introducing appropriate expertise into projects and organizations is not always easy. This is where the *Certified Tester* certification scheme—long established as an industry standard with a common glossary—can help. The *Test Automation Engineer* training and certification covered in this book are intended for advanced testers and translate the focus and factors that influence the long-term success of test automation into a structured canon of collected expertise—for example, on the subject of test automation architectures. This book clearly shows that these skills are constantly evolving.

We are better equipped than ever and I believe we have taken a significant step forward in the field of test automation. I wish you every success and plenty of creative fun using test automation as a key factor for your professional success!

Dr. Armin Metzger
Managing Director of the *German Testing Board*, 2022

Overview

Contents

1 An Introduction to Test Automation and Its Goals

Software development is rapidly becoming an independent area of industrial production. The increasing digitalization of business processes and the increased proliferation of standardized products and services are key drivers for the use of increasingly efficient and effective methods of software testing, such as test automation. The rapid expansion of mobile applications and the constantly changing variety of end-user devices also have a lasting impact.

1.1 Introduction

A key characteristic of the industrialization of society that began at the end of the 18th Century has been the mechanization of energy- and time-consuming manual activities in virtually all production processes. What began more than 200 years ago with the introduction of mechanical looms and steam engines in textile mills in England has become the goal and mantra of all today's manufacturing industries, namely: the continuous increase and optimization of productivity. The aim is always to achieve the desired quantity and quality using the fewest possible resources in the shortest possible time. These resources include human labor, the use of machines and other equipment, and energy.

Software development and software testing on the way to industrial mass production

In the pursuit of continuous improvement and survival in the face of global competition, every industrial company has to constantly optimize its manufacturing processes. The best example of this is the automotive industry, which has repeatedly come up with new ideas and approaches in the areas of process control, production design and measurement, and quality management. The auto industry continues to innovate, influencing other branches of industry too. A look at a car manufacturer's factories and production floor reveals an impressive level of precision in the interaction between man and machine, as well as smooth, highly automated manufacturing processes. A similar pattern can now be seen in many other production processes.

The software development industry is, however, something of a negative exception. Despite many improvements in recent years, it is still a long way from the quality of manufacturing processes found in other industries. This is surprising and perhaps even alarming, as software is the technology that has probably had the greatest impact on social, economic, and technical change in recent decades. This may be because the software industry is still relatively young and hasn't yet reached the maturity of other branches of industry. Perhaps it is because of the intangible nature of software systems, and the technological diversity that makes it so difficult to define and consistently implement standards. Or maybe it is because many still see software development in the context of the liberal, creative arts rather than as an engineering discipline.

Software development has also had to establish itself in the realm of international industrial standards. For example, Revision 4 of the *International Standard Industrial Classification of All Economic Activities* (ISIC), published in August 2008, includes the new section J *Information and Communication*, whereas the previous version hid software development services away at the bottom of the section called *Real estate, renting and business activities* ([ISIC 08], [NACE 08]).

Software development as custom manufacturing

Although the "young industry" argument is losing strength as time goes on, software development is still often seen as an artistic rather than an engineering activity, and is therefore valued differently to the production of thousands of identical door fittings. However, even if software development is not a "real" mass production process, today it can surely be viewed as custom industrial manufacturing.

But what does "industrial" mean in this context? An industrial process is characterized by several features: by the broad application of standards and norms, the intensive use of mechanization, and the fact that it usually involves large quantities and volumes. Viewed using these same attributes, the transformation of software development from an art to a professional discipline is self-evident.

1.1.1 Standards and Norms

Since the inception of software development there have been many and varied attempts to find the ideal development process. Many of these approaches were expedient and represented the state of the art at the time. Rapid technical development, the exponential increase in technical and application-related complexity and constantly growing economic challenges require continuous adaptation of the procedures, languages and process models used in software development—waterfall, V-model, iterative and agile software development; ISO 9001:2008, ISO 15504 (SPICE), CMMI, ITIL; unstructured, structured, object-oriented programming, ISO/IEC/IEEE 29119 software testing—and that's just the tip of the iceberg. Software testing has also undergone major changes, especially in recent years. Since the establishment of the *International Software Testing Qualifications Board* (ISTQB) in November 2002 and the standardized training it offers for various *Certified Tester* skill levels, the profession and the role of software testers have evolved and are now internationally established [URL: ISTQB]. The ISTQB® training program is continuously expanded and updated and, as of 2021, comprises the following portfolio:

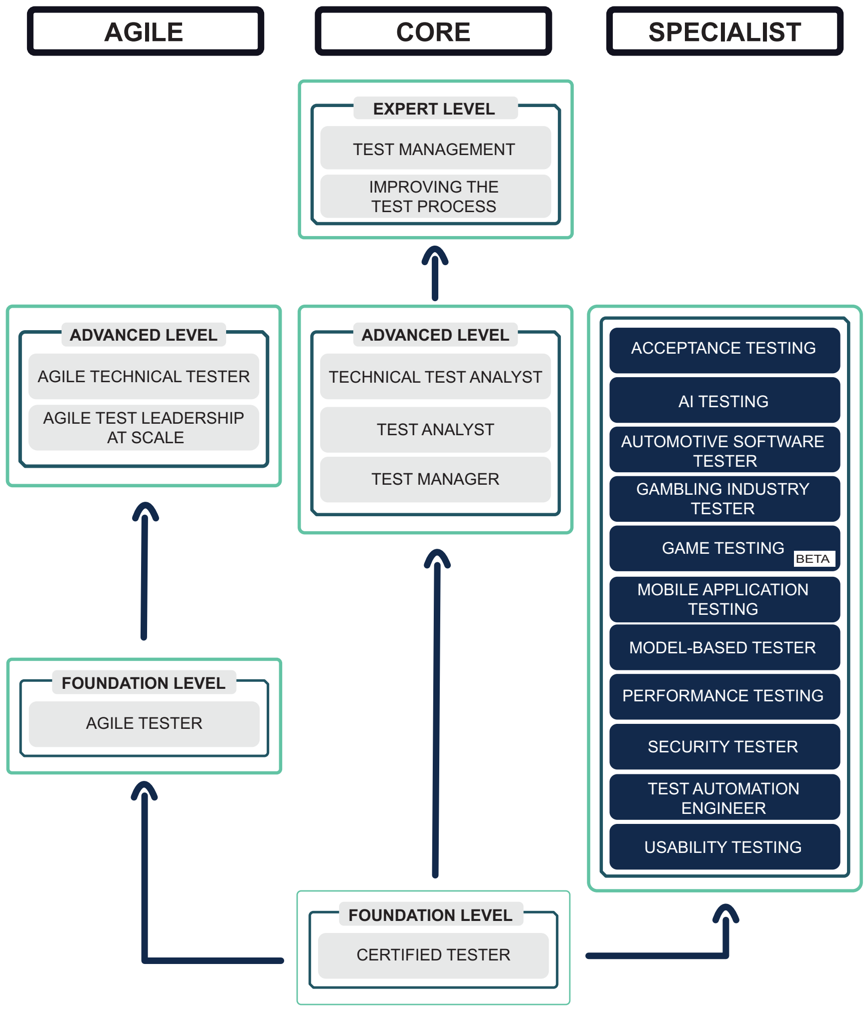


Fig. 1–1
The ISTQB® training product portfolio, as of 2022

Nevertheless, software testing is still in its infancy compared to other engineering disciplines with their hundreds, or even thousands, of years of tradition and development. This relative lack of maturity applies to the subject matter and its pervasiveness in teaching and everyday practice.

One of the main reasons many software projects are still doomed to large-scale failure despite the experience enshrined in its standards is because the best practices involved in software development are largely non-binding. Anyone ordering software today cannot count on a product made using a verifiable manufacturing standard.

Not only do companies generally decide individually whether to apply certain product and development standards, the perpetuation of the non-binding nature of standards is often standard practice at many companies too. After all, every project is different. The "Not Invented Here" syndrome remains a constant companion in software development projects [Katz & Allen 1982].

Norms and standards are often missing in test automation

Additionally, in the world of test automation, technical concepts are rarely subject to generalized standards. It is the manufacturers of commercial tools or open source communities who determine the current state of the art. However, these parties are less concerned with creating a generally applicable standard or implementing collective ideas than they are with generating a competitive advantage in the marketplace. After all, standards make tools fundamentally interchangeable—and which company likes to have its market position affected by the creation of standards? One exception to this rule is the *European Telecommunication Standards Institute* (ETSI) [URL: ETSI] testing and test control notation (TTCN-3). In practice, however, the use of this standard is essentially limited to highly specific domains, such as the telecommunications and automotive sectors.

For a company implementing test automation, this usually means committing to a single tool manufacturer. Even in the foreseeable future, it won't be possible to simply transfer a comprehensive, automated test suite from one tool to another, as both the technological concepts and the automation approaches may differ significantly. This also applies to investment in staff training, which also has a strongly tool-related component.

Nevertheless, there are some generally accepted principles in the design, organization, and execution of automated software testing. These factors help to reduce dependency on specific tools and optimize productivity during automation.

The ISTQB® *Certified Tester Advanced Level Test Automation Engineer* course and this book, which includes a wealth of hands-on experience, introduce these fundamental aspects and principles, and provide guidance and recommendations on how to implement a test automation project.

1.1.2 The Use of Machines

Another essential aspect of industrial manufacturing is the use of machines to reduce and replace manual activities. In software development, software itself is such a machine—for example, a development environment that simplifies or enables the creation and management of program code and other software components. However, these "machines" are usually just editing and management systems with certain additional control mechanisms, such as those performed by a compiler. The programs themselves still need to be created by human hands and minds. Programming mechanization is the goal of the model-based approaches, where the tedious work of coding is performed by code generators. The starting point for code generation is a model of the software system in development written, for example, in UML notation. In some areas this technology is already used extensively (for example, in the generation of data access routines) or where specifications are available in formal languages (for example, in the development of embedded systems). On a broad scale, however, software development is still pure craftsmanship.

Mechanization in Software Testing

Use of tools for test case generation and test execution

One task of the software tester is the identification of test conditions and the design of corresponding test cases. Analogous to model-based development approaches, model-based testing (MBT) aims to automatically derive and generate test cases from existing model descriptions of the system under test (SUT). Sample starting points can be object models, use case descriptions or flow graphs written in various notations. By applying a set of semantic rules, domain-oriented test cases are derived based on written specifications. Corresponding parsers also generate abstract test cases from the source code itself, which are then refined into concrete test cases. A variety of suitable test management tools are available for managing these test cases, and such tools can be integrated into different development environments. Like the generation of code from models, the generation of test cases from test models is not yet common practice. One reason for this is that the outcome (i.e., the generated test case) depends to a high degree on the model's quality and the suitability of its description details. In most cases, these factors are not a given.

Another task performed by software testers is the execution and reporting of test cases. At this point, a distinction must be made between tests that are performed on a technical interface level, on system components, and on modules or methods; or functional user-oriented tests that are rather performed via the user interface. For the former, technical tools such as test frameworks, test drivers, unit test frameworks and utility programs are already in widespread use. These tests are mostly performed by "technicians" who can provide their own "mechanical tools". Functional testing, on the other hand, is largely performed manually by employees from the corresponding business units or by dedicated test analysts. In this area, tools are also available that support and simplify manual test execution, although their usage involves corresponding costs and learning effort. This is one of the reasons why, in the past, the use of test automation tools has not been generally accepted. However, in recent years, further development of these tools has led to a significant improvement in their cost-benefit ratio. The simplification of automated test case creation and maintainability due to the increasing separation of business logic and technical implementation has led to automation providing an initial payoff when complex manual tests are automated for the first time, rather than only when huge numbers of test cases need to be executed or the n^{th} regression test needs to be repeated.

1.1.3 Quantities and Volumes

While programming involves the one-time development of a limited number of programs or objects and methods that, at best, are then adapted or corrected, testing involves a theoretically unlimited number of test cases. In real-world situations, the number of test cases usually runs into hundreds or thousands. A single input form or processing algorithm that has been developed once must be tested countless times using different input and dialog variations or, for a data-driven test, by entering hundreds of contracts using different tariffs. However, these tests aren't created and executed just once. With each change to the system, regression tests have to be performed and adjusted to prove the system's continuing functionality. To detect the potential side effects of changes, each test run should provide the maximum possible test coverage. However, experience has shown that this is not usually feasible due to cost and time constraints.

The required scope of testing can only be effectively handled with the help of mechanization

This requirement for the management of large volumes and quantities screams out for the use of industrial mechanization—i.e., test automation solutions. And, if the situation doesn't scream, the testers do! Unlike machines, testers show human reactions such as frustration, lack of concentration, or impatience when performing the same test case for the tenth time. In such situations, individual prioritization may lead to the wrong, mission-critical test case being dropped.

In view of these factors, it is surprising that test automation hasn't been in universal use since way back. A lack of standardization, unattractive cost-benefit ratios, and the limited capabilities of the available tools may have been reasons for this. Today, however, there is simply no alternative to test automation. Increasing complexity in software systems and the resulting need for testing, increasing pressure on time and costs, the widespread adoption of agile development approaches, and the rise of mobile applications are forcing companies to rely on ongoing test automation in their software development projects.

1.2 What is Test Automation?

The ISTQB® definition of test automation is: "The use of software to perform or support test activities". You could also say: "Test automation is the execution of otherwise manual test activities by machines". The concept thus includes all activities for testing software quality during the development process, including the various development phases and test levels, and the corresponding activities of the developers, testers, analysts, and users involved in the project.

Accordingly, test automation is not just about executing a test suite, but rather encompasses the entire process of creating and deploying all kinds of testware. In other words, all the work items required to plan, design, execute, evaluate, and report on automated tests.

Relevant testware includes:

- **Software**
  Various tools (automation tools, test frameworks, virtualization solutions, and so on) are required to manage, design, implement, execute, and evaluate automated test suites. The selection and deployment of these tools is a complex task that depends on the technology and scope of the SUT and the selected test automation strategy.

- **Documentation**
  This not only includes the documentation of the test tools in use, but also all available business and technical specifications, and the architecture and the interfaces of the SUT.

- **Test cases**
  Test cases, whether abstract or specific, form the basis for the implementation of automated tests. Their selection, prioritization, and functional quality (for example: functional relevance, functional coverage, accuracy) as well as the quality of their description have a significant influence on the long-term cost-benefit ratio of a test automation solution (TAS) and thus directly on its long-term viability.

- **Test data**
  Test data is the fuel that drives test execution. It is used to control test scenarios and to calculate and verify test results. It provides dynamic

input values, fixed or variable parameters, and (configuration) data on which processing is based. The generation, production, and recovery of existing and process data for and by test automation processes require special attention. Incorrect test data (such as faulty test scripts) lead to incorrect test results and can severely hinder testing progress. On the other hand, test data provides the opportunity to fully leverage the potential of test automation. The importance and complexity of efficient and well-organized test data management is reflected in the GTB *Certified Tester Foundation Level Test Data Specialist* [GTB: TDS] training course (only in German).

- **Test environments**
  Setting up test environments is usually a highly complex task and is naturally dependent on the complexity of the SUT as well as on the technical and organizational environment at the company. It is therefore important to discuss general operation, test environment management, application management, and so on, with all stakeholders in advance. It is essential to clarify who is responsible for providing the SUT, the required third-party systems, the databases, and the test automation solution within the test environment, and for granting the necessary access rights and monitoring execution.

  If possible, the test automation solution should be run separately from the SUT to avoid interference. Embedded systems are an exception because the test software needs to be integrated with the SUT.

Although the term “test automation” refers to all activities involved in the testing process, in practice it is commonly associated with the automated execution of tests using specialized tools or software.

In this process, one or more tasks that are defined the same way as they are for the execution of dynamic tests [Spillner & Linz 21], are executed based on the previously mentioned testware:

- Implement the automated test cases based on the existing specifications, the business test cases and the SUT, and provide them with test data.
- Define and control the preconditions for automated execution.
- Execute, control, and monitor the resulting automated test suites.
- Log and interpret the results of execution—i.e., compare actual to expected results and provide appropriate reports.

From a technical point of view, the implementation of automated tests can take place on different architectural levels. When replacing manual test execution, automation accesses the graphical user interface (GUI testing) or, depending on the type of application, the command line interface of the SUT

(CLI testing). One level deeper, automation can be implemented through the public interfaces of the SUT's classes, modules, and libraries (API testing) and also through corresponding services (service testing) and protocols (protocol testing). Test cases implemented at this lower architectural level have the advantage of being less sensitive to frequent changes in the user interfaces. In addition to being much easier to maintain, this approach usually has a significant performance advantage over GUI-based automation. Valuable tests can be performed before the software is deployed to a runtime environment—for example, unit tests can be used to perform automated testing of individual software components for each build before these components are fully integrated and packaged with the software product. The *test automation pyramid* popularized by Mike Cohn illustrates the targeted distribution of automated tests based on their cost-benefit efficiency over time [Cohn 2009].

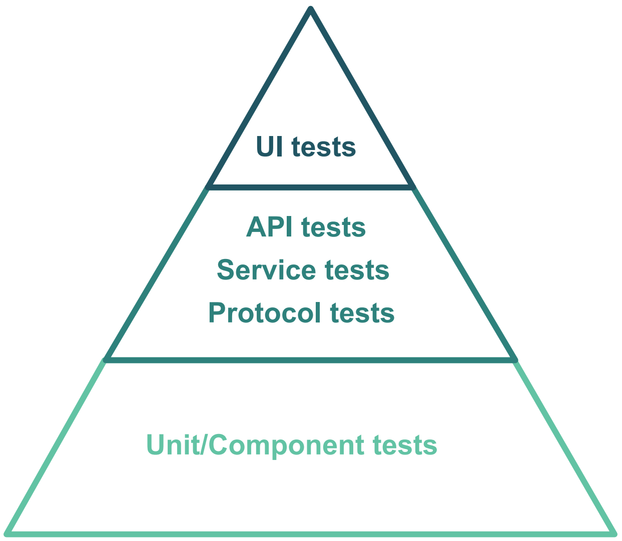


Fig. 1–2
The test automation pyramid

1.3 Test Automation Goals

The implementation of test automation is usually associated with several goals and expectations. In spite of all its benefits, automation is not (and will never be) an end in itself. The initial goal is to improve test efficiency and thus reduce the overall cost of testing. Other important factors are the reduction of test execution time, shorter test cycles, and the resulting chance to increase the frequency of test executions. This is especially important for the DevOps and DevTestOps approaches to testing. Continuous integration, continuous deployment, and continuous testing can only be effectively implemented using a properly functioning test automation solution.

In addition to reducing costs and speeding up the test execution phase, maintaining or increasing quality is also an important test automation goal. Quality can be achieved by increasing functional coverage and by implementing tests that can only be performed manually using significant investments in time and resources. Examples include testing a very large number of relevant data configurations or variations, testing for fault tolerance (i.e., test execution at the API/service level with faulty input data to evaluate the stability of the SUT), or performance testing in its various forms. Also, the uniform and repeated execution of entire test suites against different versions of the SUT (regression testing) or in different environments (different browsers and versions on a variety of mobile devices) is only economically feasible if the tests involved are automated.

Benefits of Test Automation

One of the greatest benefits of test automation results from building an automated regression test suite that enables increasing numbers of test cases to be executed per software release. Manual regression testing very quickly reaches the limits of feasibility and cost-effectiveness. It also ties up valuable manual resources and becomes less effective with every execution, mainly due to the testers' unavoidable decline in concentration and motivation. In contrast, automated tests run faster, are less susceptible to operational errors and, once they have been created, complex test scenarios can be repeated as often as necessary. Manual test execution requires great effort to understand the increasing complexity of the test sequences involved and to execute them with consistent quality.

Certain types of tests are barely feasible in a manual test environment, while the implementation and execution of distributed and parallel tests is relatively simple to automate—for example, for the execution of load, performance, and stress tests. Real-time tests—for example, in control systems technology—also require appropriate tools.

Since automated test cases and test scenarios are created within a defined framework and (in contrast to manual test cases) are formally described in a uniform way, they do not allow any room for interpretation, and thus increase test consistency and repeatability as well as the overall reliability of the SUT.

From the overall project point of view there are also significant advantages to using test automation. Immediate feedback regarding the quality of the SUT significantly accelerates the project workflow. Existing problems are identified within hours instead of days or weeks and can be fixed before the effort required for correction increases even further.