Manfred Baumgartner · Martin Klonk · Christian Mastnak · Helmut Pichler · Richard Seidl · Siegfried Tanczos

Agile Testing

The Agile Way to Quality

Springer

Agile Testing

Manfred Baumgartner • Martin Klonk •
Christian Mastnak • Helmut Pichler •
Richard Seidl • Siegfried Tanczos

Agile Testing

The Agile Way to Quality

Springer

Manfred Baumgartner
Nagarro GmbH
Vienna, Austria

Christian Mastnak
Nagarro GmbH
Vienna, Austria

Richard Seidl
Essen, Germany

Martin Klonk
Sixsentix Austria
Vienna, Austria

Helmut Pichler
Nagarro GmbH
Vienna, Austria

Siegfried Tanczos
Nagarro GmbH
Vienna, Austria

ISBN 978-3-030-73208-0 ISBN 978-3-030-73209-7 (eBook)
https://doi.org/10.1007/978-3-030-73209-7

Translated from German edition: Agile Testing: Der agile Weg zur Qualität by Carl Hanser Verlag, © Carl Hanser Verlag München 2018. All Rights Reserved.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

In the winter of 2001, in a remote ski hut in the state of Utah, a conspiratorial clique of well-known software developers called for a revolution in the software world. They created the Agile Manifesto. With this manifesto, the group defined what they were already doing with extreme programming. However, with their written formulation, they succeeded in gaining worldwide attention for their cause. The development experts gathered in this ski hut were fed up with rigid process rules, nonsensical bureaucratic guidelines, and unworldly approaches of the software engineering discipline at the time. They realized that monotonous development, "done by the book", was outdated in the new fast-paced times. They wanted to free themselves from the shackles of project bureaucracy in order to develop software together with the users, as needed. The previously cumbersome, phase-oriented, document-driven software development was to be replaced by flexible, human-driven development with small, manageable steps. Agile software development should be the approach of the new century.

Agile development is not focusing on the project, but the product. As software development became more and more an expedition into the unknown, the product needed to be created little by little in small increments. Instead of writing long declarations of intent or requirement documents about things that one could not know at the time, one should rather program something that can elicit quick feedback from a future user. It should not take months or even years to find out that the project is on the wrong track, or that the project team is overwhelmed by the task at hand. This should be uncovered within a few weeks.

Thus, the basic principle of agile development is incremental delivery. A software system is to be completed piece by piece. This gives the user representative in the team a new opportunity to participate in every step of the development. After each new delivery, they can compare the delivered intermediate product with his ideas. Testing is thus built into the process. The software is continuously tested right from the start. Whether a tester was also to take part in the agile process was left open at first. The authors of the Agile Manifesto spoke out against strict division of labor. The division into analysts, designers, developers, testers, and managers seemed too artificial for them, and was feared to cause too much loss due to friction. Of course, the project team should have all these skills, but the roles within the team should be interchangeable. The development team should be responsible for everything. It was

only through the contributions of Lisa Crispin and Janet Gregory that the role of a dedicated tester in the team emerged. They campaigned to ensure that someone in the team would take care of quality issues.

Software development requires both creativity and discipline. Toward the end of the last century, proponents of order and discipline had the upper hand, and sometimes thwarted the creativity of the developers with rigid processes and quality assurance measures. But when something is overdone, it will be reversed in a pendulum swing: too much discipline had been imposed onto traditional development projects. The reaction to it is the agile movement, which was designed to bring the spontaneity and creativity back into software development. This is most certainly to be welcomed, but again this aspect must not be overdone. The creativity of users and developers should remain grounded—and an experienced, impartial tester can support that.

Each development team should have at least one dedicated tester to represent the interests of quality. They ensure that the resulting code and product remains clean and meets the agreed quality or acceptance criteria. In the urge to move faster, non-functional quality requirements may fall behind the functional requirement. It is a tester's job to help the team to maintain a balance between productivity and quality. The tester is the good spirit, so to speak, that keeps the team from making progress at the expense of quality. In each release, not only should functionality be added but quality should also be increased. The code should be regularly cleaned or refactored, documented, and freed of all defects. It is the tester's job to enable the whole team and to ensure that this happens.

Of course, agile project organization also has consequences for testing and quality assurance. The people responsible for quality assurance are no longer in a remote office, from where they monitor the projects, check the project results between the phases, and test the product in the last phase, as was often the case in traditional projects. They are now firmly integrated in development teams, where they constantly verify and validate the newest results. It is their responsibility to point out deficiencies in the architecture and code and to detect any errors in the behavior of the system. The tester's role is developing into an agile quality coach, supporting the team in all quality-related matters. They point out problems and help the team to enhance the quality of their software. Contrary to what some said at the beginning of the Agile Manifesto ("Testers are no longer necessary in agile projects"), their role is more important than ever. Without their contribution, technical debt grows and, sooner or later, brings the project to a standstill.

This book describes agile testing in ten chapters. Chapter 1 describes the cultural change that agile development has brought about. With the Agile Manifesto, the course was set for a reorganization of the IT project landscape. Development should no longer be done rigidly according to the phase concept, but flexibly and in small iterations. An executable sub-product should be presented after each iteration. In this way, solutions are explored, and problems are identified early. The role of quality assurance is changing. Instead of acting as an external authority on the projects, testers are embedded in the project so that they can immediately carry out their tests on-site as participants in the development process. Of course, business organizations

have to adapt their management structures accordingly: Instead of waiting on the side for a final result, the business users are asked to actively participate in the project and to control the development via their requirements, i.e., "Stories." On the development side, they work with developers to analyze and specify the desired functionality. On the test side, they work with testers to ensure that the product meets their expectations.

Ultimately, everyone—developers, testers, and users—must adjust to achieve the common goal. Many traditional roles, such as project manager and test manager, are vanishing. Still there are new roles emerging, such as Scrum Masters and Team Tester or Agile Quality Coach. Project management in the traditional sense no longer takes place on a team level. Each team manages itself. The IT world is changing, and, with it, the way people develop software is transforming.

In Chap. 2, which addresses agile process models, the authors focus on the role of quality assurance in agile development projects. They are not afraid to objectively consider the various conflicting goals, for example, between quality and adherence to delivery dates, between quality and budget, and between quality and functionality. The reconciliation of these conflicting goals is a challenge for agile testing.

Contrary to the still common opinion that not much testing is required in agile projects, a lot of testing is in fact necessary. Test-Driven Development (TDD) should not only apply to the unit test, but also to the integration test and system test, according to the motto: first the test cases, then the code. In this case, it means: first the test specification, then the implementation. Test automation plays a crucial role here. Only when a test is automated can the required quality be achieved at the required speed. The whole team should participate in the automation process, as a tester alone will not be able to do it. In addition to the test, audits are also required at certain times during the development of the software product. The aim of the audits is to reveal weaknesses and shortcomings in the software. A good time for this is after every sprint in a Scrum project. The priorities for the next sprint can be set based on the results of the audits. These short audits, or snapshots of the product quality, can be executed by external QA experts in collaboration with the team. The purpose is to help the team identify risks in good time.

In addition to the Scrum process, the second chapter also deals with Kanban and the lean software development process (Lean Software). With the inclusion of examples from the authors' project experience, the reader gets several tips on how to incorporate quality assurance into these processes.

Chapter 3 deals with the agile test organization and the positioning of testing in an agile team. There are quite varying views on this topic. The authors explore which test fits what purpose with the help of the four test quadrants by Crispin and Gregory. On the one hand, the question is asked whether the test is business- or technology-oriented, and on the other hand, whether it relates to the product or the team. Four test types can be derived from this:

1. Unit and component test = technology-oriented/team-supporting
2. Functional test = business-oriented/team-supporting
3. Exploratory test = business-oriented/critique the product
4. Non-functional test = technology-oriented/critique the product

For the explanation of these test approaches, examples from test practice are provided, which show which type of test serves which purpose.

At the end of the chapter, the authors discuss the agile expansion model by Scott Ambler and emphasize how important it is to be able to expand the test process at will. There are core activities that must take place, and marginal activities that are added depending on the stage of expansion. Thus, there are not one but many possible organizational forms depending on the type of product and the project conditions.

The environment in which the project takes place and the product characteristics such as size, complexity, and quality are essential for the selection of the suitable organizational form. In any case, we must not lose sight of the main goal, i.e., the support of the developers. The aim of all test approaches is to uncover problems as quickly and as thoroughly as possible, and to notify the developers in a non-intrusive way. If several agile projects run side by side, the authors recommend setting up a test competence center. The task of this instance is to support the teams in all matters of quality assurance, for example which methods, techniques, and tools they should use. At the end of the chapter, two test organization case studies are presented, one from the telecommunications sector and one from the health sector. In both cases, the test organization is based on the project structure and the respective quality goals.

Chapter 4 raises the question regarding whether an agile tester should be a generalist or a specialist. The answer is, as so often in the literature on agile development, both. It depends on the situation. There are situations, such as at the beginning of an iteration, when the tester might negotiate with the user or product owner about acceptance criteria, in which the tester needs both technical and general knowledge. There are other situations in which testers must deal with automated test tools where the tester needs special technical knowledge. An agile tester must be able to take on many roles, but still the most important thing is that the tester fits into the team as a team player, no matter what role he or she has to take on at the moment. Soft skills are an imperative requirement. In any case, testers are the advocates of quality and they must ensure that the quality is preserved, even when time is running out. For this, they must participate in all discussions about product quality, while simultaneously checking and testing the software. They should uncover problems in good time and ensure that they are resolved as early as possible. Of course, testers cannot do this alone; they need the other team members to contribute. That is why testers must act as a kind of quality coach and help their teammates to identify and solve problems. After all, the quality of the software is a responsibility for the team as a whole.

In the context of the role of the tester in an agile team, the chapter addresses the experience profile of employees. What do the career models look like in the agile world? The fact is that there are no longer fixed roles in agile development. The roles

change depending on the situation, including that of the tester. Employees with exclusive experience in traditional development methods can no longer retreat to traditional roles; they need to adapt. This will not be easy for every employee. The authors suggest a training program which prepares for the role of an agile tester. In the program, they emphasize positive experiences and conclude with a confident note that flexible employees, old or young, can grow into the role of an agile tester.

In Chap. 5, the authors turn to the methods and techniques of agile testing. In doing so, they emphasize the differences from conventional, phase-oriented testing. The process starts with the test planning, whereby the plan is much more non-binding than in traditional projects. It should remain flexible and be easy to update. Agile testing is much more intertwined with the development and can no longer be viewed separately as a sub-project in the project. There should be at least one tester in each development team and fully integrated there. Testers should only be accountable to the team. There may be a project manager outside of the team, who serves as a reference for the testers in several teams, but he or she must not have any influence on the work within the team. At most, a project manager has an advisory role. The previous planning, organization, and control of a separate test team under the leadership of a team manager is no longer necessary. It does not fit the agile philosophy of teamwork.

As for the test methods, the methods that best suit the agile approach are emphasized—risk-based testing, value-driven testing, exploratory testing, session-based testing and development-driven by acceptance testing. Conventional test techniques, such as equivalence class partitioning, boundary value analysis, condition analysis, and decision tables or trees, are still applicable, albeit in a different context. They should be already built into the test tools. The importance of test reuse and test repetition is emphasized. All techniques must meet these criteria. The integration test is a never-ending story, and the acceptance test is repeated over and over. The cyclical nature of an agile project creates a redefinition of the test exit criteria. The test never ends—as long as the product continues to grow. At some point the development is declared finished and the product goes into maintenance.

In Chap. 6, the authors describe which documents the testers still must create in an agile project. This includes a testable requirement specification from the user stories, a test design, user documentation, and test reports. The test cases do not count as documentation, but as test ware. A concern of agile development is to reduce the documentation to a minimum. In the past, documentation was, in fact, exaggerated. In an agile development project, only that which is absolutely necessary is documented. It remains to be seen whether a test strategy or test design is necessary. Test cases are essential, but they are part of the software product as well as the code. Therefore, they are not considered documentation.

The most important document is the requirements specification that emerges from user stories. It serves as the basis for the test, the so-called test oracle. The test cases are derived from it and are tested against it. It also contains the acceptance criteria. The only test reports that are really required are test coverage report and defect report. The test coverage report shows what was tested and what was not. The testers need this document as proof that the testing has been done sufficiently. The user

needs it to gain trust in the product. The defect report records which deviations have occurred and how they are being handled. These two reports are the best indicators of the state of the test.

Testers are predestined to write the user manual because they know the system best and know how to use it. Someone has to write the manual, and the testers are the right candidates for it. They ensure that this document is updated after each release. Otherwise, the book follows the agile principle of restricting the documentation to the essentials. The main aspect is that there is always a solid requirement specification and comprehensible user documentation. A structured, semi-formal specification of requirements forms the basis for the test and no user wants to do without user instructions.

Chapter 7 explains the important topic of "Test automation." Test automation is particularly important in agile development, as it is the main instrument for project acceleration and necessary to support state-of-the-art DevOps approaches. Only through automation can the test effort be reduced to an acceptable level while maintaining product quality during continuous integration. The authors differentiate in this case between unit test, component integration test, and system test. The unit test is shown in detail using the example of JUnit. It shows how developers must work in a test-driven manner, how to build up test cases, and how to measures the test coverage. The component integration test is explained using the Apache Maven integration server. It is important to simulate the interfaces of the integrated components to the components that are not yet available using placeholders. The system test is described by a technical test with FitNesse. The most important thing here is the writing of the test cases in test scripts, which can be expanded and repeated as required. The authors also emphasize how important it is to be able to manage the test ware—test cases, test scripts, test data, etc. —conveniently and securely so that the test runs as smoothly as possible. Tools are also needed for this.

Chapter 8 adds examples from test tool practice, extending test automation with test management functionality. At first, the Broadcom Rally tool is described, which supports the agile life cycle from the management of stories to error management. The agile tester can plan and control this tool in his test. An alternative to Broadcom Rally is Polarion QA/ALM, which is particularly suitable for recording and prioritizing test cases and for tracking errors. Other test planning and tracking tools include the Bug Genie tools, which particularly support test effort estimation; Atlassian JIRA, which offers extensive error analysis, and the Microsoft Test Manager.

For testers in an agile project, the ongoing integration test is crucial. They must integrate the last components as quickly as possible with the components of the last release and confirm that they work together smoothly. To do this, he or she must conduct the testing not only via the user interface, but also via the internal system interfaces. With Tricentis Tosca, external as well as internal interfaces can be generated, updated, and validated. The test messages are conveniently compiled using the drag-and-drop technique. The authors describe how these tools are used and where their limits lie based on their own project experience.

Chapter 9 is dedicated to the topic of training and its meaning. The authors emphasize the role of employee training in getting started in agile development. Training is essential for success in using the new methods, and this applies particularly to the testers. Testers in an agile team need to know exactly what to focus on, and they can only learn that through appropriate training. There are many interpretations of agile approaches; nevertheless, the quality of the product must be ensured, and this requires professional testers who are trained to work in an agile team. Training programs focused on the needs of agile testing are discussed in this chapter.

In summary, this book covers the essential aspects of agile testing and offers a valuable guideline for testing in an agile environment. The reader gets many suggestions on how to proceed in agile projects. He or she learns how to prepare, conduct, and approve the agile testing. As a book written by test practitioners, it helps testers find their way in an often confusing agile world. It gives them clear, well-founded instructions for implementing agile principles in test practice. It belongs in the library of every organization that runs agile projects.

Harry M. Sneed

Preface

The first German edition of *Agile Testing: The Agile Way to Quality* was published in 2013, followed by the second edition in 2018. Since then, we have been asked again and again by our international contacts and colleagues whether there would be an English translation. Thanks to the support of Nagarro, we were able to realize this project, and we hope that it will be well appreciated by the agile community. As mentioned, this book is a translation of the second edition with some necessary updates and not a completely revised new edition. But the next edition is already included as a Theme in our backlog. Until then, we wish you a great time reading this first English edition!

When the "Manifesto for Agile Software Development" was signed in 2001 by a group of software engineers in Utah, USA, it probably initiated the most significant change in software development since the introduction of object orientation in the mid-1980s. The "Agile Manifesto," quasi the Ten Commandments of the agile world, can certainly be regarded as an expression of a countermovement to the strongly regulating process and planning models that spread from the end of the 1980s onwards, such as PRINCE, the V-Model, or even ISO9001. These models tried to counteract the previously chaotic and arbitrary development processes by planning and structuring the processes and documenting them. The Agile Manifesto consciously positions itself in relation to these aspects and gives its central four agile values—interaction, cooperation with the customer, reacting to changes, and finally functioning software—a higher relevance for a successful software development.

The way the Agile Manifesto is formulated in values and principles was one reason why the triumph of agile software development in the years since the publication of the Agile Manifesto has been marked by many dogmas, not to say religious wars. We, the authors of this book, are seeing this not for the first time. In the decades of our professional experience, we have often been confronted with new solutions for the "software problem": structured programming, object-oriented programming, CASE (Computer-Aided Software Engineering), RUP (Rational Unified Process), V-Model, ISO9001, SOA (Service-Oriented Architecture)—a long list of salvation promises, always accompanied by self-proclaimed gurus; some even call themselves evangelists. And many of these innovations were based on very similar patterns: While they presented themselves as *the* solution or were sold by their promoters as *the* saving idea, previous approaches were dismissed as wrong or

outdated. There were also many believers who followed radical ideas, often without reflecting on them and almost without will, because the number of dissatisfied people was and still is large. In this case, the preachers and counselors, who use every hype to make profits, have an easy game—which is a big risk for good ideas.

The last thought was also a central motivation for this book. We, the authors, have always been unhappy with the way people have tried to dogmatically implement new approaches in software development. In most cases the baby was thrown out along with the bath water. In contrast, we see the changes as an opportunity for a process of continuous improvement and optimization. However, we have been confronted in agile projects in recent years with the fact that everything we have acquired and developed as testers in terms of methods, techniques, self-image, or standards (like the test processes according to ISTQB) should no longer apply. This may also be due to the fact that in the past it was mainly the software developers who pushed the agile community. This fact is one of the reasons why the tasks and role of the software tester in agile methods and projects are often not defined or only vaguely defined. Differently interpreted terminologies also contribute to this. For example, when in Scrum we talk about an interdisciplinary development team, some people think that the team only consists of developers (in terms of programmers) who do everything. Others believe that in Test-Driven Development with the development of an automated unit Test Set, the test tasks in development are sufficiently fulfilled and the rest is the responsibility of the user in the User Acceptance Test. So where are the test phases and test levels we are used to? Where and how do we find ourselves as testers in agile projects? The agile approach obviously raises more questions for us testers than it provides answers to previous problems.

These are exactly the challenges we want to tackle with our book, because the book was written by testers for testers. In each chapter we provide answers to the key questions we have come across in our projects. The book deals with general or almost cultural change processes, with questions regarding the approach and organization in software testing, with the use of methods, techniques, and tools, especially test automation, and with the redefined role of the tester in agile projects. A broad spectrum that is certainly not final and comprehensive within the scope of this book, but nevertheless we hope to cover it with ideas and suggestions for the reader.

To make the described aspects even more tangible, the specific topics of this book are accompanied by the description of experiences from concrete software development projects of various companies.

The examples should demonstrate that different approaches can lead to effective solutions that meet the specific challenges of agile projects.

With this in mind, we wish the reader every success in implementing the contents presented here in his or her own projects and at the same time invite him or her to visit us on our Internet platform www.agile-testing.eu.

Practical Examples

The practical example of EMIL in this book comes from a company in the healthcare industry that can look back on 25 years of successful product and software development. However, as the company has grown, new customer requirements and stricter legal regulations have increased, as has the need to optimize development and test processes and make them more efficient. The idea of switching from the traditional to the agile development process has already come up here and there in the company. The software development project EMIL was the starting point for this initiative. The project's goal was the re-implementation of an analysis software that has been successfully used worldwide for ten years and which has been developed by various developers. Particularly from a technical and architectural point of view, the new requirements could no longer be implemented without problems; over the years, many functions were added as "temporary balconies"—but were never removed or integrated. It was estimated that a rough time frame for the re-implementation of all functions of the existing software was about two and a half years. The lack of experience in the target technology and the regulatory requirements that the healthcare industry brings along were identified as the biggest challenges on the way to agile development. The positive and negative experiences, the problems encountered, and the attempts to solve them, from the first year and a half of the project, can be found in this book and are marked accordingly in the corresponding chapters.

Another practical example is provided by OTTO. As an online retailer, OTTO operates in a very agile market environment and uses innovative technologies to ensure a positive shopping experience on otto.de and in the stores. As part of the Otto Group, OTTO is one of the most successful e-commerce companies in Europe and Germany's largest online retailer for fashion and lifestyle in the B2C sector. Over 90% of total sales are generated online. In the practical examples, Ms. Diana Kruse talks about her experiences in her transition from a tester and test manager to Quality Specialist and Quality Coach, visualized by graphics of her colleague Torsten Mangner.

Vienna, Austria — Manfred Baumgartner
Vienna, Austria — Martin Klonk
Vienna, Austria — Christian Mastnak
Vienna, Austria — Helmut Pichler
Essen, Germany — Richard Seidl
Vienna, Austria — Siegfried Tanczos

Acknowledgments

We thank the companies Nagarro GmbH and Otto (GmbH & Co KG) for their contribution to the work on this book.

We would also like to thank our colleagues for their diligent support, and our reviewers, who kept us grounded with critical remarks and made a valuable contribution to this book: Sonja Baumgartner (Graphics), Stefan Gwihs, Diana Kruse and Torsten Mangner (practical examples and graphics Otto.de), Anett Prochnow, Petra Scherzer, Michael Schlimbach, Silvia Seidl, and Harry Sneed. Also, our sincere thanks to our editor, who fixed the countless errors in copy-editing that we had overlooked.

Contents

About the Authors

Manfred Baumgartner Manfred Baumgartner has more than 30 years of experience in software development, especially in software quality assurance and software testing. After studying computer science at the Vienna University of Technology he worked as a software engineer for a large software company in the banking sector and later as quality manager for a CRM solution provider. Since 2001 he has built up and expanded the QA consulting and training offerings of Nagarro GmbH, one of the leading service providers in the field of software testing. He is member of the board of the Association for Software Quality and Further Education (ASQF) and the Association for Software Quality Management Austria (STEV) as well as a member of the Austrian Testing Board (ATB). He contributes his extensive experience in both classic and agile software development as a sought-after speaker at internationally renowned conferences and as author and co-author of relevant technical books: *The System Test - From Requirements to proven Quality* (2006, 2008, 2011), *Software in Numbers* (2010), *Test Automation Foundation* (2012, 2015, 2021), and *Agile Testing: The Agile Way to Quality* (2013, 2017).

Martin Klonk Martin Klonk is a Senior Test Expert at Sixsentix Austria GmbH. Trained as an industrial engineer at the Technical University of Berlin (and the Université Libre de Bruxelles), he started his career in 1996 as a Software Test Specialist at SQS Software Quality Systems in Cologne and Munich. Later he moved to SQS, ANECON, and Nagarro Austria in Vienna. Martin Klonk has worked in a wide variety of industries and has been actively involved in almost all

areas of software testing. As a member of the Austrian Testing Board of the ISTQB he regularly contributes to curricula and their German translation and also conducts training courses. Since he managed to implement successful testing strategies in an agile project in 2007, Martin Klonk has been a strong advocate of agile practices in testing and has led several agile projects as a testing specialist. He is a certified tester, project manager, and scrum master.

Christian Mastnak Christian Mastnak works as principal software testing consultant at Nagarro with over 15 years of QA experience. He leads Nagarro's global "Agile Testing Practice" and implements innovative solutions for international customers in a variety of industries. Pursuing a very hands-on approach in all his projects, he enjoys switching between different QA roles—from Test Manager to Agile Quality Coach, from Test Automation Architect to Test Consultant. These roles allow him to pursue his passion for continuous improvement of QA processes and methodologies. Christian Mastnak has shared his expertise as a popular trainer and speaker at international conferences including 'EuroSTAR' and 'Agile Testing' in Munich and the 'Software Quality Days' in Vienna.

Helmut Pichler Helmut Pichler is responsible for the trainings at Nagarro GmbH and works as a consultant for test and quality management in both domains, traditional and agile. He has actively witnessed the "growth" of agile at conferences and in the community from the beginning and is one of the first trainers of the former Certified Agile Tester (now PAQ) training program developed by iSQI. Helmut Pichler has been President of the Austrian Testing Board, the regional representation of the ISTQB in Austria, for more than 15 years and is an active member of the international testing community, working with experts from Austria and in close cooperation with the Swiss and German Testing Boards, contributing to the updating and further development of international testing standards.

Richard Seidl Richard Seidl is an agile quality coach and software test expert. He has seen and tested a lot of software in his diverse professional career: the good, the bad, and the ugly. The big and the small, the old and the new one. He now combines his experiences into an integrated approach—development and testing processes can only be successful if the most diverse forces, as well as strengths and weaknesses, are balanced. Just as an ecosystem can only exist harmoniously with all aspects in all its quality, the processes in the testing environment must be viewed as a network of different key players. Agile and quality then becomes an attitude that we can practically live for instead of just working it off. He has published various specialist books and articles as an author and co-author, including "The System Test - From Requirements to proven Quality" (2006, 2008, 2011), *The Integration Testing - From Design and Architecture to Component and System Integration* (2012) and "Test Automation Foundation" (2012, 2015, 2021).

Siegfried Tanczos Siegfried Tanczos has been working at Nagarro GmbH since 2004 and has also been a team leader and manager in the software test organization for many years. In addition to his management role, Siegfried Tanczos has been involved in several software testing projects since the beginning of his employment at Nagarro. He has gained a wide range of experience in software testing during his professional career in the banking world, and has been working as a software tester since 1998. Through his work at Nagarro in various customer projects and business units, Siegfried Tanczos has been able to gain extensive experience in dealing with classic and agile process models.

Agile: A Cultural Change

1

In order to better understand the cultural change toward agile software development and to understand "Agile Testing" not only as a buzzword, it is important to take a look into the past. Much of what we perceive today as common knowledge has its justification in the methodical and technical progress of software technology in the last 30 years. Experience is an essential element of innovation and improvement. For example, the average age of the signatories of the Agile Manifesto in 2001 was about 47 years, and back then it wasn't just about doing everything differently; it was about doing it better. This is often overlooked when "agile" is used to simply get rid of unpleasant things or to hide one's own weaknesses. The honest approach to developing software in a cooperative, benefit-oriented, and efficient, economic way is the core of the agile idea.

1.1 The Journey to Agile Development

The transition to agile development in the practice of IT projects has been ongoing for a long time, since the spread of object-oriented programming in the late 1980s and early 1990s. The object-oriented approach changed the way software is developed. The primary goals of object orientation were

- Increased productivity through reuse
- Reduction in the amount of code through inheritance and association
- Facilitation of code changes with smaller, exchangeable code blocks
- Limitation of the effect of errors by encapsulation of the code modules (Meyer, 1997)

These objectives were fully justified, as the old procedural systems were getting bigger and bigger and bursting at the seams. The amount of code threatened to grow

M. Baumgartner et al., *Agile Testing*, https://doi.org/10.1007/978-3-030-73209-7_1

immeasurably. Therefore, a way had to be found to reduce the amount of code for the same functionality. The answer was object orientation. New programming languages such as C++, C#, and Java emerged. The developers began to switch to the new programming technology (Graham, 1995).

However, this technological improvement also had a price—the increase in complexity. By decomposing the code into small, reusable blocks, the number of relationships, i.e., dependencies between code blocks, increased. In procedural software, the complexity lay in the individual blocks, whose flow logic was increasingly nested. In object-oriented software, complexity was outsourced to the architecture. This made it difficult to maintain an overview of the entire system and to plan a suitable architecture in advance. The code had to be revised several times until an acceptable solution was found. Until then the code was often in a messy state.

There were two answers to this challenge. One was modeling. During the 1990s, different modeling languages were proposed: OMT, SOMA, OOD, etc. In the end, one of them prevailed: UML. By representing the software architecture in a model, it should be possible to find the optimal structure and to gain and keep the overview. The developers would—as expected—create the "suitable" model and then implement it in the concrete code (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991).

Software engineering changed from procedural to object-oriented modeling with use cases. Modeling was much more detailed and was supported by new tools like Rational Rose (Jacobson, 1992). However, modeling proved to be very tedious, even with the best tool support. The developer needed quite a lot of time to work out the model in every detail. In the meantime, the requirements had changed and the assumptions on which the model was based were no longer valid. The modeler had to start from scratch, and the customer was getting more and more impatient.

Another answer to the challenge of increasing complexity was "Extreme Programming" (Beck, 1999). Since the appropriate model for the software was not predictable, the developers began to translate the requirements directly into the code in close communication with the user and in short iterations (Beck, 2000).

This approach carries the risk of getting lost in a dead end due to constant change requests in many details but has the advantage that the customer soon sees what is coming. If the code blocks are designed flexibly, they can be reused if a different path needs to be taken. Another advantage of the Extreme Programming was that the user could be taken along on the journey. The user could follow the results of the coding—the real user interfaces, lists, messages, and database content—which he or she could not do in the case of abstract modeling. Thus, in practice, Extreme Programming prevailed, and modeling remained in the academic corner (Fig. 1.1).

The "Test-Driven Development" turned out to be a useful consequence of the Extreme Programming (Beck, 2003). When entering unknown territory, one must protect oneself. The safeguard in code development is the test framework. The developers first build a test framework and then fill it with small blocks of code (Janzen & Kaufmann, 2005). Each component is tested immediately to see if it

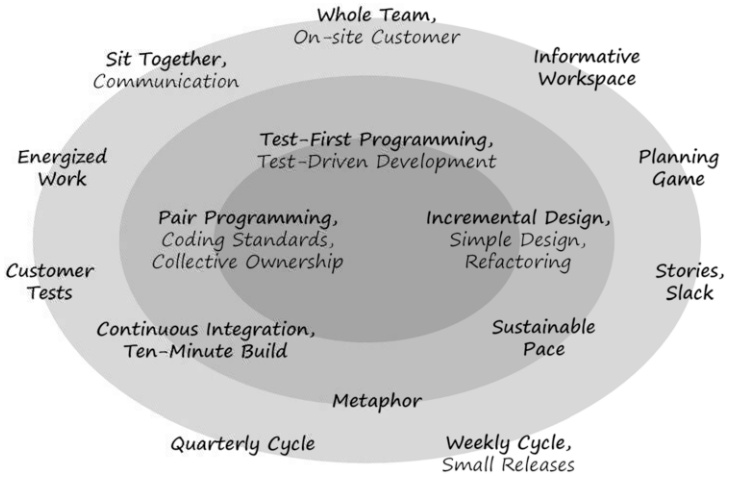

Fig. 1.1 XP practices

works. The developers make their way through the code area and always ensure its status by testing. In this way they finally reach a satisfactory, tested intermediate state which they can demonstrate to the user.

There are only intermediate states in software development, since software, by definition, is never completely finished. Test-Driven Development has proven to be a very solid approach even outside Extreme Programming. This has also been confirmed by several scientific studies, and unit testing and continuous integration have become standard in software development.

It goes without saying that Extreme Programming and Test-Driven Development contradicted the prevailing management methods. The management of software projects requires predictable, pre-dispositioned development where it is possible to determine what will be delivered, when, and at what cost. Systematic Software Engineering should ensure this (see Fig. 1.2).

The 1990s were also the decade of process models, quality management, and independent testing, in short, the decade of software engineering. Software engineering was intended to bring order to software development and maintenance through clearly defined processes with a strict division of labor. Free, unrestricted development was abolished. Many measures were taken by management to finally bring software development under control. The V-Model is representative of these attempts to structure software development (Höhn & Höppner, 2008). Unfortunately, most of these measures were in stark contradiction to the new “extreme” development technology.

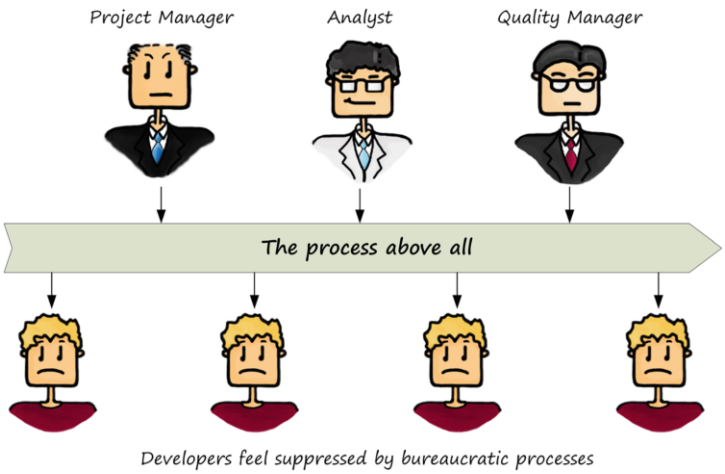

Fig. 1.2 Software engineering creates order in a chaotic software world

1.2 The Reasons for Agile Development

One of the most important reasons for agile development is user proximity. In classical, non-agile development, the gap between developers and users has widened. In the 1970s, this gap was not so wide. When Harry Sneed, who was kind enough to write a foreword to this book, began his career as a programmer, the developer shuttled back and forth between the customers in the engineering department, his/her desk, and the data center every day. The developer discussed the task with the user almost daily, wrote the program, and tried it out in the data center, usually in the evening. Being close to the customer was the most important thing.

This way of working changed in the 1980s and 1990s. In the traditional test world, which was created by Gelperin and Hetzel in the mid-1980s, there is a fundamental distrust toward the developer. It is assumed that the developers—on their own initiative—will produce faulty and poor-quality software (Hetzel, 1988). Moreover, they do not recognize their own errors, and if they do, they are declared as inevitable characteristics of the software: "It's a feature, not a bug." They do not let anyone talk about the quality of their architecture and code. In their eyes, everything is always fine. There is no need to improve it.

With this image of the developer in mind, the call for a separate test organization was raised. For larger projects, there should be a separate test group that supervises several projects. In any case, the test group had to be independent of the developers. This was the prerequisite for the testers to be able to work effectively. A system should be created in which the testers control the work of the developers. The developers produce the bugs and the testers find them. A bug reporting and tracking system should support the communication between the two groups.

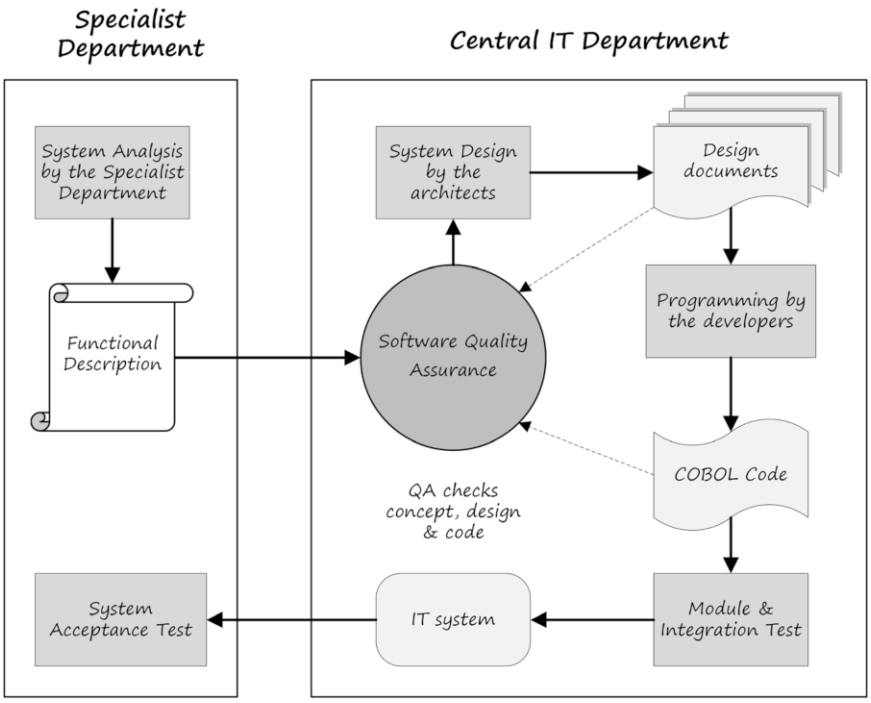

Fig. 1.3 The Software Engineering Model Bertelsmann

This division of labor between developers and testers has been propagated and practiced worldwide. New terms such as "Quality Engineering" and "Quality Management" were created and it was deemed that every larger organization should have a quality manager. This was required by the ISO 9000 standards. And where there is management, there is also bureaucracy. A software quality bureaucracy was established, based on standards and regulations, to guide developers to work correctly (ISO 9000, 2005).

The development process at Bertelsmann AG—the Bertelsmann Software Engineering Model—was a typical example. According to this model, the specialist department should first create a complete functional description of the topic. This was accepted by the quality assurance and development departments and an estimate of the effort required was prepared (Bender, et al., 1983) (see Fig. 1.3).

Based on this cost estimate, an agreement was reached with the specialist department, with a fixed price, a fixed date, and a fixed result. The requirements were then frozen and initially implemented in a system design. This was presented to the customer, who rarely understood it or, more precisely, could have understood it. Most of the time, the users just nodded their heads and said it was okay. The system design was followed by implementation and testing, whereby the testing was always a bottleneck. The finished system was presented to the user many months, sometimes even years, later. The reaction of the user was often that he or she would

not have expected it this way. At Bertelsmann, this well-intentioned but cumbersome process eventually led to the reorganization of IT and the distribution of developers among the departments. Other German companies had also adopted the Bertelsmann Model, but the result was usually the same as at Bertelsmann: disappointed users. The conclusion is that the separation of developers and users has never worked well.

A classic example of a bureaucratic development process is the V-Model or the V-Modell-XT (Rausch & Broy, 2006). This model was primarily designed for software development in the German authorities. It prescribes every step in the process. The demands are collected and defined in a requirements specification. Based on the specifications, a project is put out to tender and offers are gathered. The cheapest or best offer is selected, and the winner of the tender then draws up a functional specification and presents it to the client. If the client understands something about it, the client has the opportunity to make corrections. The system is then implemented and tested. Many months later, the more or less tested final product is handed over to the client for acceptance. It often turns out that the product in the form in which it is delivered cannot be used or cannot be used to the expected extent, and thus the maintenance or evolution process begins. Changes are made in and around the software until it finally meets the user's expectations. This can take years.

In 2009 Tom DeMarco literally wrote the death sentence for such rigid, bureaucratic development processes. He stated that software engineering is an approach "whose time has come and gone" (DeMarco, 2009). From the very beginning, software engineering was aligned to large projects such as those of the United States Department of Defense, which required a strict division of roles. Looking back, we must ask ourselves whether the approach has ever worked for smaller projects. In fact, there was something seriously wrong from the beginning: the long time span between the award of the development contract and the delivery of the final product. During this time, requirements and customer expectations had changed too much. This was already the case in the 1980s and is even more the case now in our fast-moving world. Ergo, proximity to the customer, the principal, must be maintained and development cycles must be shortened.

What applies to the collaboration between developers and users also applies to the collaboration between developers and testers. Here too, the gap had widened over time. When the testing discipline emerged in the late 1970s, developers usually tested their software themselves. At that time, the outsourcing of testing was a revolutionary event. In recent years it has become a matter of course. However, the criticism here is the same as for software development. The time span between the handover to the tester and the first error messages is simply too long. When the first error messages arrive, the developer has already forgotten how they were created. This is the reason why testers and developers should work together on a piece of software and why testers must test the finished component immediately after its creation. Afterwards, the developer can discuss the problems with the tester immediately and fix them until the next component delivery. This fast feedback is the key to agile testing. It must be done quickly, otherwise the agile test will fail (Fig. 1.4).