What Drives Quality

Second Edition

A Deep Dive into Software Quality with Practical Solutions for Delivering High-Quality Products

BEN LINDERS

What Drives Quality

A Deep Dive into Software Quality with Practical Solutions for Delivering High-Quality Products – Second Edition

Ben Linders

This book is for sale at http://leanpub.com/WhatDrivesQuality

This version was published on 2018-12-09

ISBN 978-94-92119-14-8

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Ben Linders by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm reading "What Drives Quality". Get your own copy! benlinders.com/what-drives-quality #DriveQuality

The suggested hashtag for this book is #DriveQuality.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#DriveQuality

Also By Ben Linders

Getting Value out of Agile Retrospectives

Waardevolle Agile Retrospectives

Tirer profit des rétrospectives agiles

Ottieni il meglio dalle tue Retrospettive Agili

Извлекаем пользу из Agile-ретроспектив

Obteniendo valor de las Retrospectivas ágiles

从敏捷回顾中收获价值

アジャイルふりかえりから価値を生み出す – 日本語版

Wartościowe Retrospekcje Agile

Continuous Improvement

Αποκομίζοντας αξία από τα Agile Retrospectives

Khai thác giá trị Agile Retrospective

Jak zvýšit přínos agilních retrospektiv

The Agile Self-assessment Game

This book is dedicated to everyone that understands the value that high-quality software brings!

Praise for What Drives Quality

This book is a succinct summary of what we know about software quality and how to apply it.
Bill Curtis, Director Consortium for IT Software Quality

This book a must-read for every software professional!
Jan van Moll, Director of Quality & Regulatory at Philips

Ben's book is a reflection of his deep knowledge and experience helping teams and organisations identify what quality is in software, why it matters and how to optimise the right qualities for a product.
Shane Hastie, Director of Agile Learning Programs at ICAgile

What Drives Quality is about a lot more than software quality, touching on many of the ideas that have been tried and true for quality in general.
William R. Corcoran, Editor of the Firebird Forum

This book is a lovely collection of ideas for anyone looking to improve their software development process.
Gojko Adzic, Partner at Neuri Consulting LLP

Many pieces of useful advice in an easy-to-read book.
Diomidis Spinellis, Professor of Software Engineering, Programmer, and Technology Author

Whether you are a manager, programmer, programmer, or (naturally) a tester, this book will provide you with plenty of meaningful and practical ways you can enhance and drive quality at your organization.
Ilan Kirschenbaum, Co-Founder, Agile Coach at Practical agile

What Drives Quality is a one-stop shop reference for software developers and testers practitioners.
Eran Kinsbruner, Technical Evangelist at Perfecto

This book is a roadmap to better software with advice for people at all levels of the organisation.
Rik D'huyvetters, Senior Software Engineer at The Reference

Ben's book is practical and actionable. I am already using the ideas in my coaching work with teams and their executive leaders.
Brandi Olson, Agile Strategy, Lean Data, and Coaching, The Olson Group

Contents

Foreword by Bill Curtis

Ben Linders states in his Preface that quality is not a "sexy" topic. In fact, IT executives yawn when you mention "quality", and start checking email on their smartphones. Even so, another term for quality is "risk". Having entered the era of 9-digit defects (not bits or bytes, rather Euros or dollars), the risk exposure of poor quality software has implications for the quarterly financials. The CEO, not the CIO, is now answering for the effects of poor quality software.

Most of the recent €100 million IT outages and security breaches are traced back to defects in the software. Some defects, such as the all too frequent SQL Injection vulnerabilities, have been well-known since the last century. So why do developers still make these mistakes? And why are they not caught before being placed into production?

Too often the answer is an "I need it yesterday" mentality, exacerbated by a lack of professional discipline. This book delves into the factors that affect quality at every step of the software development cycle. It then describes practices that help development teams gain control of them to produce dependable, trustworthy software. Consequently, these practices reduce the risk to operations and the cost of maintenance.

Since most IT organizations report they are implementing some form of iterative/agile/DevOps development process, this book focuses on adopting quality practices in agile environments. In fact, half of the book is devoted to agile quality techniques. Ben's objective is to make sure the speed of delivery is matched by the speed of assurance.

This book is not a compendium of everything known about software quality. Rather it is a succinct summary of what we know and

how to apply it. It can be read in "agile time", and delivers a solid overview that can set readers on course to higher quality, lower risk software. Start here, then follow Ben's recommended readings if you want a deeper dive into specific quality topics. This book is a valuable contribution to professional software engineering practice.

Dr. Bill Curtis
Executive Director, Consortium for IT Software Quality
Fort Worth, Texas

Preface

I'm an active blogger at www.benlinders.com. On my blog, I share my experiences on agile and lean topics, including software quality.

I published my first book on Valuable Agile Retrospectives in 2013. This book – Getting Value out of Agile Retrospectives – has become a huge success. It has many readers all over the world and has been translated by agile teams of volunteers into 13 languages already.

Many readers have told me that they value my blog posts on quality. In 2014 I started writing my second book and decided that the topic should be software quality.

The book started from the blog post series What Drives Quality which is based on the research that I did with the Software Engineering Institute (SEI) and my experience working as quality and process manager in large organizations.

The first editions of this book were a kind of Minimum Viable Product to find out if people are actually interested in software quality. From the feedback that I received, I found out that there is an audience for this book – there are sufficient readers who feel that quality matters.

However, I also found out that quality is not a "sexy" topic. Since it's hard to develop high-quality products many people avoid the topic. Opinions vary on what quality is and isn't and what can be done to ingrain quality into the way products are developed.

A book on quality should be practical. It should help you, reader of this book, to improve the quality of your software and deliver better products. It should inspire you and give you energy to persevere on your quality journey. *What Drives Quality* tries to do just that, and more.

This book is based on my experience as a developer, tester, team leader, project manager, quality manager, process manager, consultant, coach, trainer, and adviser in Agile, Lean, Quality and Continuous Improvement. It takes a deep dive into quality with views from different perspectives and provides ideas, suggestions, practices, and experiences that will help you to improve quality of the products that your organization is delivering.

I'm aiming this book at software developers and testers, architects, product owners and managers, agile coaches, Scrum masters, project managers, and operational and senior managers who consider quality to be important.

I want to thank the many reviewers of my book for investing time and suggesting improvements: Ard-Jan Moerdijk, Ben Liu, Chris Spanner, Heidi Araya, Johan van Dongen, Kevlin Henney, Martin Wiseman, Paul Hookham, Peter Rubarth, Piotr Jachimczak, Rene Ummels, Sharon Bockhoudt, Tom Gilb, Shiv Sivaguru, Srinath Ramakrishnan, Stephen Janaway, William R. Corcoran, and Vasilij Savin. Special thanks goes to Brandi Olson, Rik D'huyvetters, and Yanto Hesseling, for proofreading several versions of the book and providing many suggestions. Your feedback has helped me to make this a better book.

I feel honored having a foreword by Bill Curtis, Executive Director, Consortium for IT Software Quality, in which he emphasized how important quality is and confirms that quality assurance and speed of delivery can go together.

Finally, I would like to thank all the people who invest time to read my blog and comment on the posts. Your feedback helps me to increase my understanding of the topics that I write about and makes it worthwhile for me to keep blogging!

Second Edition

After publishing the first edition as an eBook in September 2017 much has happened. People downloaded the eBook and shared their experiences on delivering high-quality software with me. There were emails, tweets, LinkedIn comments, praise, and other feedback that I received from my readers. It feels really good when people like your book and tell you what they got out of it. Thanks!

I gave several interviews about this book, you can find them on InfoQ, SPaMCast, and Yours Productly.

A book review was published on Software Testing Magazine.

Readers of this book have asked questions, pointed out mistakes in the book and sent me suggestions for improving the book. I've used this to revise the text. A big thank you to everybody who invested time and energy to help me to further improve the quality of my book. Special thanks goes to Ilan Kirschenbaum for providing detailed feedback on the full book!

The second edition contains additions throughout the book. I have added many practices and experience stories to the existing chapters. I also added a new chapter, Driving Quality, where I explore how to measure and analyze quality, describe the Project Defect Model, and show how you can steer quality in agile.

This edition also contains a second foreword by Jan van Moll, Senior Forensic Investigator and Head of Quality & Regulatory at Philips Healthcare. In his foreword he states that "poor software quality can put people and society at high risk" and he emphasizes our "moral obligation to human society and ourselves to ensure our software meets its standards and has the right quality". Thank you Jan for emphasizing how important software quality is!

What more is new in the second edition:

- Using use cases for specifying requirements.
- How personas can be used to understand the users.

- The “dark side” of commitment.
- How burnout impacts quality
- Lean startup, using feedback to learn and develop the right products.
- How Model Driven Software Engineering drives quality
- Refactoring as a technique to improve code quality and find bugs
- Mob programming
- Resilient software design
- More to be added…

The second edition is the first version that will be published in print.

The reactions that I have received make it clear that software quality is considered important by the software development community and by users of the software. One of my missions is to help teams and organizations to deliver high-quality products. This book is my contribution to support this.

Ben Linders
December 2018

Introduction

This book views software quality from an engineering, management, and social perspective. It explores the interaction between all involved in delivering high-quality software to users and provides ideas to do it quicker and at lower costs.

What's in This Book

In this book, I explore how quality plays a role in all of the software development activities. It takes a deep dive into quality by listing the relevant factors of development and management activities that drive the quality of software products. It provides a lean approach to quality by analyzing the full development chain from customer requests to delivering products to users.

The book starts by explaining Why Quality Matters – introducing software quality and explaining what makes it so important to deliver high-quality products.

In the chapter Deep Dive into Quality I explore the factors that drive quality in software activities performed by development teams and explain what managers can do to support teams of professionals in pursuing quality.

In Driving Quality I explain how you can measure quality together with an approach – the Project Defect Model – that helps you to get insight into the quality of a product while it is being developed and use that to drive the delivery of high-quality products.

The chapter Quality Software with Agile Teams contains stories and case studies showing how the quality of software products can be improved. They are based on my experience working with teams and managers, helping them to face and solve quality issues, and improve their performance for sustainable and lasting results.

The chapter Develop your Software Quality Skills provides information about my workshops on software quality and continuous improvement. The Agile Quality Coaching cards is a deck of coaching cards created specifically for the readers of this book which enables them to improve their products and practices.

What Drives Quality doesn't intend to teach you all the theory behind quality or provide detailed descriptions of all possible quality practices. The Bibliography gives you an extensive list of books, articles, and links, that you can use to acquire in-depth knowledge on software quality.

How To Use the Book

This is a practical book with many techniques and ideas that you can apply in your specific situation, within the method or framework that you are using. It aims to help you to improve the quality of the products that you deliver to your users.

There are many suggestions in this book, things which yo can use to improve quality. They are marked as tips with a key symbol:

Try those tips that look suitable and see if they work for you. If they do, great! If not, try another one.

I also added many stories and cases from organizations that I have worked with to share my experience:

Stories and cases are boxed with a user symbol. Use them to get inspiration and think about what you might do to improve quality.

Most suggestions provided in this book are suitable for agile teams and the management of agile organizations given the fact that many software development organizations have adopted or are adopting agile. Specific agile quality practices are also described,

with advice on how to apply them effectively based on an agile mindset.

Register your book today to get access to supporting materials at benlinders.com/what-drives-quality.

With plenty of ideas, suggestions, and practical cases on software quality, this book will help you to improve the quality of your software and to deliver high-quality products to your users and satisfy the needs of your customers and stakeholders.

Why Quality Matters

Many methods for product quality improvement start by investigating the problems and then working their way back to the point where the problem started. For instance audits and root cause analysis work this way.

But what if you could prevent problems from happening, by building an understanding what drives quality, thus enabling you to take action before problems actually occur?

Almost everyone that I have met during my career agrees that it's better to prevent defects or detect them earlier in software development than to have them found by the users of their products.

Still, many companies struggle with changing their processes from "testing or inspecting quality in" to achieving quality from the start – through culture, design, craftsmanship, and leadership.

What is Quality

The quality of a software product or system is primarily how it satisfies the needs of the users, customers, and stakeholders, and delivers value to them. This definition takes an *external view*. It puts quality in the eyes of the beholders: the users, customers, and stakeholders decide if a software product or service has sufficient quality, or not.

If the quality of a software product is insufficient according to the users then they will not use it. If the customers or stakeholders do not get enough value from the product, then they will not buy or support it. Satisfying the needs of all is crucial for software (or any) products to be successful.

How to Deliver Quality

Teams can only deliver quality if they are driven by the needs of the users, customers, and stakeholders. But how can you ensure that you are able to develop products that meet these needs? This is where an *internal view* of quality helps us.

An internal view looks at the architecture and design of the software. It explores the processes and practices that are used to deliver the software. It focuses on the culture of the company, leadership, goals, and reward systems that drive the right behavior.

Agile teams need to decide how they will develop and test their software. My experience working with agile teams tells me that having a solid Definition of Done (DoD) helps to know when products have sufficient quality. Successful teams stick to their agreed upon DoD and adapt it when it turns out that the delivered quality is insufficient. They are supported by their managers, who expect them to define, use, and improve their DoD, and allow them time to do this.

The internal quality view helps to develop software that is understandable and maintainable. Software that can easily be changed when there are new requirements or when the environment in which the software is used changes and the software needs to be adapted.

The Why of Quality

I see a lot of attention paid to the *How* of quality. We have dozens of quality standards like ISO 9000, ISO 15504, ISO 25010, and IEEE 730, models, like TQM, CMMI, People-CMM, Agile and Lean, and Kanban, which tell us how we should develop software, and manage software development. We know how to do agile self-assessments, agile retrospectives, and root cause analysis to improve the way we work. We can even measure quality.

But do we understand and give enough attention to the *Why* of quality? Do we talk to the users of our products? Do we know what is important for them and why? Do we actually know our users? How do they use our software and what value do they want to get from using it?

Quality is in the eye of the beholder, so in the end the users are the ones who judge if we are delivering quality or not and what our product is worth.

A good understanding of why quality is important and what our users consider quality to be is crucial to delivering high-quality software products.

Combining Why and How

To reach quality you need a good combination of understanding why and deciding how.

To deliver quality you have to understand why a user needs certain functionality. Ask yourself why they want the system to be available 24/7? Find out why they want the software to be easy to use, and with fast response times? Why they need it next week, and not at the end of the year, what makes it so important for them?

Just knowing your user's needs (the requirements) is not enough, you must also understand things like:

- why the users need it?
- what is it that they want to accomplish?
- what is important for them? What not?
- where's the value for users?

Only then can you deliver real quality!

How you deliver a quality product or service is about making decisions on how you do it. The way of working, the methods and practices that you use, impact quality.

You have to know how to deliver within budget and time constraints, with the professionals and the knowledge and skills that are currently available. And you must also decide what processes, practices, and tools you will use to develop the product, and manage teams, projects, and products.

Quality Practices

This book explores the role of quality in software development and management bactivities. It takes a deep dive into quality with the Quality Factors Model, a collection of practices and ideas for delivering high-quality software products.

The Quality Factors Model lists the relevant quality factors of each development phase (a bunch of activities usually done at the same time) and for management roles. It describes why and how these factors drive the quality of products and how you can influence them.

Although the term "phase" is used, it doesn't mean that a specific sequence of activities is required to improve quality. The Quality Factors Model doesn't prescribe or assume any specific lifecycle.

Practices and suggestions which drive quality are provided which you can use to improve the quality of the products that you deliver to your users.

They can be applied in waterfall or iterative projects that for example use Prince-2 or RUP, by agile teams using frameworks like Scrum, Kanban, or XP. They can also be used by organizations that are doing large scale agile software development with for instance

the Scaled Agile Framework (SAFe), Large Scale Scrum(LeSS) Disciplined Agile Delivery (DAD), Nexus or Agility Path.

Requirements

With requirements, I mean activities for specifying the products to be developed and supporting activities such as requirements clarification, prioritization, commitment, and requirements management and traceability.

Factors that drive Requirements Quality are (in no particular order):

- *Requirements Management Capability* – Skill and experience level of the professionals performing the requirements management activities.
- *Requirements Commitment* – Agreements between the product owner/manager, the project managers, and team members, where projects/teams commit what will be delivered.
- *Requirements Definition Capability* – The skill and experience level of the people performing requirements definition activities.
- *Requirements Stability* – Inverse of the number of requirement changes over time. The fewer requirement changes you have, the higher requirements stability will be.
- *Requirements Process Capability* – The quality of the defined and baselined requirements processes and practices, including supporting materials such as training and document templates.
- *Roadmap Quality* – Usability of the roadmap with respect to managing the requirements to deliver the right products.
- *Scope Stability* – Impact of major project changes related to the product roadmap, including stability of the products to be developed, development teams, projects, and major changes in team/project funding or product delivery dates.
- *Root Cause Analysis* – Capability to learn from defects found and problems that occurred during development. Analyzing them, determining common causes related to processes, tools, development environment, capabilities, management,

and organization, and defining actions to prevent them from recurring.

Irrespective of which development method (waterfall, iterative, agile, etc) is used, you need to create a common understanding and agreement between the product owner/manager and the development team to deliver the right products.

Let's take a look at these factors in more detail, to see how they drive quality.

Requirements Management Capability

To enable the delivery of products with sufficient quality, you need to define the product requirements before you start to develop the software.

Waterfall projects often start with a requirements phase. In this phase, all requirements are defined in detail and agreed (signed-off) with the end responsible for the product. This can be a product manager, project sponsor/orderer, or client or client representative.

Defining all requirements up front is challenging and often impossible, unless the product is very small. Striving toward completeness is a waste of time.

Based on the fact that you cannot know everything at the start, projects often use change control techniques to manage requirement changes during development.

Agile works differently by defining requirement details just in time before the work starts. Different practices exist for defining and managing requirements, depending on the agile method that is used.

Scrum uses a product backlog which contains product backlog items ("PBI", "backlog item", or "item"). These are units of work which are small enough to be completed by a team in one sprint.

The backlog items need to be defined before sprint, and can be refined by the team and Product Owner at the start of the sprint.

Kanban proposes to use a kanban board with work items. Work items are defined before the team starts to work on them. Replenishment meeting are used to prioritize work items.

Teams can use a Definition of Ready (DoR) to check the quality of the user stories/backlog items/work items. A DoR states the criteria that should be met before accepting an item and starting the work.

Some useful resources to make your own Definition of Ready are:

- The INVEST principle by Bill Wake.
- 10 Tips for writing good user stories by Roman Pichler.
- The book – User Stories Applied by Mike Cohn.
- The book – 50 quick ideas to improve your user stories by Gojko Adzic.
- Using a Definition of Ready on InfoQ.
- Exercise cards for defining your DoR and DoD by David Koontz.

The DoR is not intended as a sign-off, hand-over, or phase check for requirements. As any agile practice, it should be done in an "agile way". For instance by doing the check as part of the planning meeting, or only when the team is unsure about the quality of a user story and then use it as a tool to improve the requirements.

Requirements Commitment

The purpose of requirement commitment is to have agreements between the stakeholders and projects/teams about delivering products with specific functionality to the customers.

Traditionally managed project often emphasize the need for commitment, expecting that it will increase the likelihood that the agreed functionality will be delivered on time and within budget.

Waterfall and iterative projects usually define priorities up front in their project plans. These priorities can help to take decisions during the project if it turns out that commitments cannot be met.

Agile teams use product backlogs to manage their requirements. Product owners prioritize the user stories. These priorities can (and should) change once new insights are gained after delivering the product.

Having commitment on the requirements by all involved stakeholders is important as it ensures that you are developing a product that your customers need and are willing to pay for. It increases the chance of building the right products and reduces waste.

Projects/teams also need to be committed to doing whatever they can to deliver products with the specified functionality and sufficient quality.

You don't need to have a commitment on everything in the backlog for developing products. It is unfeasible, too expensive and takes too long to get. Normally you only need commitment on those requirements for which development has to start.

There's a "dark side" to commitment when it comes to delivery dates and deadlines. When it turns out that it takes more work to

develop the product and delivery dates cannot be met, developers are often asked to put in overtime. Putting in more work hours however increases the chance of making mistakes.

Asking people to live up to their commitment and work overtime to live up to their estimates and meet the deadline is something that I do not advice to do:

My experience is that many estimates are actually the result of negotiations, where developers are put under pressure purposely or feel pressured due to the existing culture. Then there's even more pressure when work obviously takes more time. This doesn't work, literally.

And who's "deadline" is it, who will die when the product takes longer to deliver. The customer or user? The manager? The developers?

I'd like to see research on this, due to burnout I'm afraid the developers category will likely have most casualties. This is not a good thing, and it's causing much damage in the software industry, which is why people like Jason Lengstorf - Getting More Work Done in Fewer Working Hours and John Willis - Burnout in the Software Industry are giving attention to burnout.

Summarizing this: Commitment does matter and can work in many situations but it also tends to put pressure on teams which can lead to non-intended effects.

Forcing people to live up to what they have committed to has high quality risks. Under stress, people tend to make more mistakes and take shortcuts, which results in products with insufficient quality.

Product owners/managers often have to decide with imperfect and incomplete information. In the book Product Mastery Geoff Watts suggests reducing the number of options, be clear about the criteria on which you need to decide, involve people in the decision, and accept that decisions cannot be perfect.

In order that teams can start developing products, you have to make sure that stakeholders agree on the priorities and that there is sufficient commitment to warrant investing time and money.

My advice is to find out and verify what needs to be delivered first. I usually ask the stakeholders the question "What do we need now?"

Having prioritized user stories that are ready at the start of an iteration helps to increase commitment from the stakeholders and the development team, resulting in higher product quality.

To be able to act upon changing requirements a good approach is to commit to as little as possible. Olav Maassen and Chris Matts suggest in their book Commitment to "never commit early unless you know why". This is a good approach to deal with change.

Requirements Definition Capability

The requirements definition capability has to do with how good you communicate requirements. The depth, breath and quality, of the communication has a large impact on the quality of the products that are delivered.

It's about communicating and collaborating over documenting requirements – ultimately the specification of the products has to be in the head of the developers and testers.

Using a requirement specification document (or any other written format like use cases or user stories in agile) to communicate requirements often leads to confusion, resulting in developing wrong products that the users don't need.

In agile, the sprint planning is a meeting held at the start of an iteration where all involved discuss what needs to be developed. It's important is that assumptions made by the developers and testers are communicated and checked with those responsible for supplying the requirements. In agile, those are usually the product owners.

It is essential to have frequent in-depth communication between development teams and the product owners or managers and (future) users about what the software should do to ensure that the right products are developed. Development teams should be able to ask for clarification if something is not clear. They should develop a good understanding of how the products will be used.

Agile suggests to use user stories to communicate requirements. User stories are short descriptions of features from the view of a user of the product. The format often used is:

As a < type of user >,

I want < some goal >

so that < some reason >.

User stories should support discussions and collaboration, not replace them. Hence they should be kept small, not become to detailed.

Acceptance criteria in user stories provide space for precise details. These are used to check a story before delivery.

Where the Definition of Done is applicable for all user stories, acceptance criteria are specifically for one user story.

In agile, the ability to write effective user stories enables teams to deliver the right products fast. Effective user stories express the needs of users and support effective communication and collaboration between product owners and agile teams. They are prompts for communication which help to understand the needs of users and give clarity to ensure the right products are built.

Richer communication techniques have proven to significantly reduce requirements ambiguity and improve clarity. Examples are face to face discussions, requirement clarification workshops, visiting users and involving them, and agile planning and backlog grooming. They serve to verify the requirements with product owners and users and help to map them to engineering tasks.

More about user stories can be found in the books User Stories Applied by Mike Cohn and Fifty quick ideas to improve your user stories by Gojko Adzic.

Use cases can be used to define the interactions between a role and a system to achieve a goal. The main flow of a use case list the most common way that the system is used; alternative or optional flows describe other possible interaction paths. Exception flows can be used to define how the system should react if users use it in a way which was not intended.

The article Applying Use Cases in Agile: Use Case 2.0, Slicing and Laminating explores how use cases fit into an agile approach.

Defining the requirements may include activities like user experience (UX) design or user interface (UI) design. The aim of such activities is to help teams to produce a software product that is easy to use and does what users expect that it would do. Developers and testers need to communicate closely with the designers doing UX/UI activities to understand how the product should look and how the users will be using it.

Acceptance Test Driven Development (ATTD) is a practice where the acceptance criteria are discussed and acceptance test cases are defined before code is produced. It's primarily intended to increase the understanding of the requirements using different views: what do the users need, how can we solve that, and how can we test it.

On many occasions, I've seen the value of having testers involved when defining and clarifying requirements. Their questions have often lead to a better understanding and earlier identification of flaws and risks.

Behavior Driven Development (BDD) is a practice to describe the required behavior of the software. Using conversations, concrete examples, and automated tests, it helps to streamline the communication between product owners/managers, domain experts, and teams about what the software should do. It can also be used to define acceptance criteria which can be automated as described earlier in ATTD.

Personas can be used to represent and understand typical users of your system. Each persona describes a specific category of users and explores how they use the system, their values, expectations and needs, background, role and responsibilities, etc.

My experience is that personas make it easier for teams to identify with the users, ask questions, and understand the user's needs.

The use of personas is described in the book Product Mastery by Geoff Watts.

The perfection game is a general purpose feedback technique. I use it in agile retrospectives and also when I coach or train teams in writing effective user stories.

To get feedback on a user story, ask the following questions:

- I rate the user story xxx on a scale from 1-10
- What I liked about it?
- To make it perfect I would?

You should rate the user story on a scale from 1 to 10, based on how much value you think you can add yourselves by improving it. For example, when there is nothing you can think of to improve, rate it a 10. If you think you can make it twice as good and valuable, give it a 5.

Answer "what I liked about it?" by mentioning the qualities and strengths that the user story has, and answer "to make it perfect" with concrete things that you would do to improve the user story to make it perfect (a 10).

Using it this way, the perfection game helps you to quickly get actionable feedback on your user stories to improve their quality.

Requirements Stability

Requirements stability is the inverse of the number of requirement changes over time. The fewer requirement changes you have, the higher requirements stability will be.

The aim of requirements stability is not to prevent changes to requirements from happening – they will happen. Discouraging change or (even worse) trying to ignore it is no solution either.

But to be able to write code you need some stability in your requirements. Which is also true when using an agile approach.

Scrum expects requirements to be stable during a sprint.

It matters that projects and teams are sufficiently capable to deal with changes and can maintain stability during development. Both are needed.

Clarifying the requirements increases stability.

Developers and tester should use the available possibilities to ask for clarification if something is not clear with the requirements. Agile teams often do this during the sprint planning, product backlog refinement or backlog grooming meetings.

The purpose of product backlog refinement or backlog grooming meetings is to keep the backlog up to date and orderly. These meetings are also often used to discuss the business value and priority of the backlog items.

My suggestion is to mark requirements which are insufficiently clear (except of course for the ones which are clarified during meetings) so that it is clear for product owners and team members that more work needs to be done before they can be pulled into an iteration.

Time and money are invested in an iteration. Every decision on adding a user story to the iteration backlog of a team is actually

an investment decision – which is something many teams and organizations are not aware of.

Agile teams using Scrum treat requirements as being stable during an iteration (sprint). When a user story is added to an iteration the assumption is that there is a real need for software that fulfils the requirement described in the user story.

In agile, the requirements are fixed during an iteration and flexible over iterations (more on this in fixing scope in agile projects).

If there is a risk that a requirement underlying a user story may change at short notice then it may be better to select a different high priority user story for the next iteration.

It's a good practice to always have user stories ready for 2-3 iterations or a couple of weeks so that it is possible to switch user stories during the sprint planning or even in the daily stand-up when required.

With Scrum, having sufficient user stories ready also helps if teams finish all user stories before the end of an iteration. At that time they can agree with the product owner to pick another high priority user story and add it to the ongoing iteration.

Iterations can also be used to clarify requirements. You can use a spike, a practice from eXtreme Programming, to research a requirement or to investigate the feasibility of a technical solution which helps you to drive out risk and uncertainty.

This is somewhat similar to using a Minimum Viable Product (MVP) in Lean Startup to increase the knowledge of what users really need.

Requirements Process Capability

Processes and practices reflect the way that work gets done in the organization. They govern how professionals work together to deliver value.

Processes include supporting materials such as training and document templates. A process is not a document, process documentation exists to support professionals doing the work. The actual way of working is the process.

I consider the "Definition of Done" in agile to be a process. It is the way that teams agree to work together to deliver value.

Process capability defines the ability of a process to deliver valuable outcomes. As a process is the way professionals work, capability is mainly determined by people and their skills and abilities.

Process improvement is about improving the way of working. It's not about changing a document but has to do with behavior change, professionals changing the way they do their work.

Requirement activities are a part of many roles. They are not only done by product managers and product owners, but are also part of the work that is done by developers and testers, and people in operations. If you want to improve the Requirements Process Capability then you need to look at all activities and roles, end-to-end.

Improvements are often realized by changing the way that professionals in different roles work together.

Roadmap Quality

The product roadmap should visualize how the product is expected to evolve over time.

Product roadmaps typically contain information about:

- When to develop which product versions.
- Business cases and value propositions for product versions.

- Allocation of product scope to versions.
- Product and feature introduction dates and plans.
- End of maintenance dates.
- Phase out dates.

Creating a roadmap typically starts by discussing and agreeing on the products vision and goal. Knowing the why of your products helps to decide what should be included, and what not.

Roadmaps created by multidisciplinary teams often have higher quality. Having people with different views who collaborate and challenge each other's thoughts leads to fresh and better ideas for products.

The purpose of roadmaps is to synchronize and align activities of all involved. They should reflect current insights, which means that they will change frequently based on feedback from users, customers, stakeholders, and development teams.

It's important to keep roadmaps up to date and communicate changes to keep everyone involved. You may think that that's a no-brainer, but I've seen many organizations where only senior product managers worked with the roadmaps and didn't involve others – which is (literally) not a workable solution.

Transparency is essential if people want to work together effectively. My advice is to make roadmaps accessible for everyone and use them as information radiators.

Lean Startup is an approach that helps you to deliver more value to customers in a short time and to grow your business.

How much quality a roadmap has will become clear once you start delivering your products. The feedback that you get helps to decide if you should persevere or pivot.

The lean startup approach has been described by Eric Ries in his book The Lean Startup: How today's entrepreneurs use continuous innovation to create radically successful businesses. As Eric states:

> The method is designed to teach you how to build a startup. It also offers ways to scale up and grow a business with the greatest possible speed.

The most important and most distinctive concepts of the approach are the build-measure-learn cycle, the minimum viable product, validated learning, and persevere or pivot. These concepts can be used to create and test your product roadmap.

The goal of the lean startup approach is to learn as much and as quickly as possible what customers want. With that knowledge you become able to make products that increasingly match the needs of the customers.

With lean startup, Build-Measure-Learn is the cycle that you go through. You make a product version, deliver it to your customers, and use the feedback to learn what the customers want, and to improve the product (persevere) or decide to create another product (pivot).

Learning is important, but it is not the only goal. It is essential to deliver the smallest possible product (Eric Ries calls it Minimum Viable Product or MVP) which has maximum value for customers, what they can use and will pay for.

If the product has no value, if the customer does not find it interesting to use, then you will not receive feedback and you will learn nothing.

The goal of a minimum viable product is to learn. This is possible, for example, by offering different product variants (for example different websites or user interfaces, different execution, various contract formats or licenses, etc.) and measuring which variant is most appreciated by customers (number of hits, products sold, etc).

To get results with lean startup, you need to define the variants and the measurements beforehand. Also you need to achieve a clear understanding of what you want to learn from the customer.

With everything you learn from your customers and the feedback you receive, you are better able to decide what you can do best with the product. As long as everything indicates that you are on the right track, then go can go ahead (persevere). If you come to the conclusion that it is not the right product, not something that customers need, then you can choose to stop and develop another product (pivot).

With lean startup you can improve the quality of your product roadmaps to make the right product.

Scope Stability

Many organizations struggle with changing requirements. The scope of their projects is unstable, which can have a major impact on the quality of the developed products.

Managing scope stability increases the quality and effectiveness of development projects.

One solution that is used in agile is to stabilize the requirements for an iteration. This helps teams to focus and deliver working software in small chunks.

Another agile solution (often used with Kanban) is reducing the Work In Progress (WIP) to increase organizational focus.

Risk management techniques can be used to identify potential changes, and to take actions to limit impact, for instance by clarifying requirements or reducing the project scope before starting development.

Story mapping can be used to visualize the product that needs to be developed. It uses a matrix to horizontally group the functionality and vertically slice it up into iterations. Discussing the story map helps to discover missing functionality and to prioritize in which order the product will be delivered (most valuable parts first).

Root Cause Analysis

Many defects found during reviews or in testing have their origins in the requirements activities. Root cause analysis is a practice for finding the deeper causes of such problems.

I consider the ability to do root cause analysis to be an important driver for software quality. Major or frequently occurring defects often provide valuable information about flaws in your products and/or development processes, which you can use to improve the quality of your products.

For root cause analysis I prefer to use the Apollo method described in the book Apollo Root Cause Analysis by Dean L. Gano. Strong points of this methods are that it is drive by facts (not assumptions) and that timing is taken into account when looking for cause-effect relationships.

Note: The above mentioned book is out of print. Dean's recent book RealityCharting provides a causal analysis process which can be used to visualize all causes, the interrelationships between causes, and effective solutions to prevent recurrence.

Over the years I have analyzed many defects caused by requirements activities. The root causes that I found often were related to communication and collaboration between those involved, lacking customer focus, and insufficient skills for defining and managing requirements.

These are the root causes that I found. Yours will be different! Therefor I suggest to analyze your defects to find out what is causing problems.

Increasing Requirements Quality

To deliver high-quality products to customers, quality practices have to be ingrained throughout development – quality starts with ensuring the quality of the requirements!

Quality Software with Agile Teams

Agile teamwork has shown to be a great approach to deliver high-quality software products. The agile values favor quality, and there are lots of agile practices available that teams can apply to develop high-quality software. Users are happy with the early and frequent deliveries of working software by those agile teams.

What is Software Quality?

I define high-quality software as "software that satisfies the needs of the users and delivers value to them." Quality is in the eye of the beholder – the users are the ones who decide if a software product or service has adequate quality, not the agile teams.

Teams can only deliver quality if they are driven by the needs of the users. In agile, this is supported by the agile values, and by intense collaboration between the product owners and the agile teams.

Software has to be "fit for purpose" – users need to be able to do their work using the software.

The book Fit for Purpose by David Anderson and Alexei Zheglov explores how companies can understand their customers and develop products that fit with the purpose(s) their customers have.

Agile Values Support Quality

The manifesto for agile software development describes the values that agile methods consider important.

In my opinion, these values support the delivery of quality software:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Some examples how the agile values support quality are:

Working software over comprehensive documentation focuses on delivering products to users. It encourages early and frequent delivery, enabling users to use the software and start getting value.

Responding to change over following a plan results in higher quality, as it urges agile teams to adapt software that does not satisfy the needs of the users.

It is no surprise that agile teams deliver high-quality software and services to their users:

There's data on the business benefits of agile that confirms that agile supports quality. The State of Agile Report from VersionOne also mentions enhancing quality as one of the main reasons why organizations adopt agile.

Stories on Agile Quality

This chapter – Quality Software with Agile Teams – explores how agile principles and practices can be applied to deliver high-quality software. It contains stories and case studies from my experience in working with teams and managers, helping them to face and solve quality issues, and improving their performance for sustainable and lasting results.

Agile provides significant benefits when it comes to quality. If you want to improve quality then it helps to make a business case for quality with agile.

As Philip Crosby stated years ago: quality is still free. But you need to have a quality mindset to delight your customers, and you will have to sell quality to your managers by showing them how it makes the business successful.

Agile quality practices like sprint planning meetings, daily stand-ups and retrospectives, and technical practices like pair programming or Test Driven Development all support the delivery of quality software.

Let's explore what happens when quality problems arise in software in a culture where people don't dare to speak up, being afraid to get punished if their managers finds out that their software isn't working properly: What if we fail?

Teamwork enables agile teams to deliver high-quality software; it enables them to decide how to do their work, and helps to continuously learn and improve their way of working.

Agile supports empowering teams, which is a more effective and quicker solution than adding people when quality becomes a problem. Empowered teams have what it takes to increase the quality of products.

My experience is that when teams are working under too much pressure, technical debt will increase and the velocity of teams will go down. Also often the cycle time goes up, it will take more

time for teams to deliver. Agile promotes that teams work in a sustainable pace, delivering value to their customers.

Teams can increase the quality of software with visual management. Making things visible helps teams to deliver products that their customers need and collaborate effectively with their customers and stakeholders.

Agile retrospectives can be used to investigate quality issues or to agree upon actions that can improve the quality of the software that is delivered.

The stories in this chapter are meant to inspire you and give you ideas and energy to improve quality in your organization.

The stories also show how the practices described in Deep Dive into Quality have been used to deliver high-quality software.

Agile teams are driven by values that favor quality. They collaborate intensively with the users of the software and use practices to develop high-quality software products.

Deliver High-Quality Software with Agile Teamwork

Effective agile teams are able to decide how to do their work and to continuously learn and improve their way of working. Teamwork enables them to deliver high-quality software that satisfies the needs of their customers. They can use techniques like swarming and pair working to solve complex problems quickly and reduce their technical debt.

Quality through Teamwork

How does agile teamwork help to deliver high-quality software? It has to do with multidisciplinary teams that are able to solve complex problems, driven by their motivation and empowerment.

My opinion is that the whole team owns quality, which means that everyone on the team is responsible and accountable for their contribution towards delivering high-quality products. Give your teams the means to deliver high-quality software and allow them to self-organize and find out what works for them.

Most often quality issues are complex problems. They need to be viewed in multiple ways to solve them. Agile teams are multidisciplinary, they consist of professionals with different skills, knowledge, experiences, and backgrounds. Such teams have a diversity that helps to find innovative solutions and the know-how to collaborate for effective and lasting solutions.

Giving people freedom to decide how they will do their work will empower them. Self-organization gives freedom to teams.

In self-organized agile teams, motivation is often high since people feel that they are in control. Reaching the goal is what counts for teams and every team member will do the best (s)he can to get there.

Team Techniques for Quality Software

When there is a major issue, swarming can be used to address it effectively and quickly. The whole team focuses on a single issue and together they will do whatever it takes to solve it.

Agile teams should have all the skills and experience that is needed to deal with problems effectively. They should know how to communicate and collaborate to get the job done.

Team members can work in pairs to increase the quality of the software while writing it. They can take turns on the keyboard, and switch to remain sharp and spot problems or opportunities to improve code and reduce technical debt. Pair working makes it possible for professionals to learn new skills or sharpen existing ones.

If you're very deep into something, chances are that you start to overlook stuff – this is where pair working can help you. Also, two pairs of eyes can see more than one :-).

Managing Teams for Quality

Agile teams should be self-organized. They don't need managers to decide for them – telling them how to do their work isn't needed and doesn't work.

What managers can do to enable their teams to deliver high-quality software is:

- Make it crystal clear that quality matters to them.
- Reward people and teams that deliver high-quality software.
- Work together to remove any impediments that teams bring up.
- Act as a servant leader to help their teams to be successful.
- Arrange for coaching and mentoring for teams.
- Encourage and provide time for teams to learn and improve.
- Encourage people to set goals for themselves related to improving and maintaining quality.
- Discourage shortcuts that lessen of damage quality.
- Be an example: Act on quality issues and invest time in learning.

Rewarding people can be done in many ways. Financial rewards are one way, but often giving compliments, praise people, or publicly reward people for their behavior has much more effect.

Rewards are individual. Where some people thrive on attention and compliments, others appreciate opportunities to take a course or go to a conference. Another person would like to receive a bonus or go out and have dinner with their partner.

When you are unsure what would work for someone, my suggestion is to ask the person.

Learning to Deliver High-Quality Software

Learning enables improvement. Peer-learning can be very effective, teams can provide an environment where people can learn from

each other. When people team up, the result is greater than the sum of the individual parts.

Teamwork enables teams to deliver high-quality software.

Software quality is free, teams that invest time and energy in learning how to build the right products with good quality will save money.

Develop Your Software Quality Skills

I regularly provide public workshops and training, and in-house classes tailored to specific situation and needs of organizations.

Effective Root Cause Analysis

In the *Workshop Effective Root Cause Analysis* you will learn practical and effective techniques to analyze problems. You will practice these techniques using major defects or problems from your own organization.

What will you get out of this workshop:

- Understanding the why and how of root cause analysis.
- Experience how to do root cause analysis.
- Learn how to define actions to prevent problems.

Valuable Agile Retrospectives

In the Workshop Valuable Agile Retrospectives you will practice different kinds of retrospective and learn how to adopt and apply retrospectives in your own organization.

What will you get out of this workshop:

- Understand the why, what and how of agile retrospectives.
- Practice different retrospective exercises.
- Learn how to create a safe environment to run retrospectives.
- Practice skills for facilitating retrospectives.

Making Agile Work for You

Learn how to apply agile practices to develop the right products, deliver faster, increase quality, and become a happy high-performing team in the Workshop Making Agile Work for You.

What will you get out of this workshop:

- Practice working with Agile and Scrum/Kanban: planning, daily stand-ups, product reviews and retrospectives.
- Get ideas for improving collaboration and communication with stakeholders to deliver more value.
- Find out how to deal effectively with impediments to take control and truly become self-organized.
- Learn tips and tricks to improve your agile way of working.
- Get advice on selecting and applying agile (Scrum, Kanban, XP, Lean) practices effectively for you.

Agile Self-assessment Game

The Agile Self-Assessment Game is used by teams and organizations to self assess their agility. Playing enables teams to reflect on their team interworking and take the next steps in their agile journey.

With this game, teams discover how agile they are and what they can do to deliver more value with high-quality software.

The card of the Agile Self-assessment Game are based on the manifesto for agile software development and generally accepted agile principles and practices. This makes the game useful for all agile team, whether using Scrum, Kanban, XP, Lean, DevOps, SAFe, LeSS or any other agile framework.

You can download the game and expansion packs in my webshop.

Attend a Workshop

See my upcoming workshops for attending a public workshop.

Contact me if you want to have an in-house workshop tailored to the needs of your organization.

More information, see Services Ben Linders Consulting.

Agile Quality Coaching Cards

The Agile Quality Coaching cards can be used to improve the quality of software products and services. It's a deck of coaching cards created specifically for the readers of this book: Agile teams, Tech leads, Architects, Product Owners, Scrum masters, Agile coaches, consultants, and anyone involved or being responsible for delivering high-quality products.

The cards can be downloaded in my webshop. Readers of What Drives Quality get a discount of 25% with the coupon "What-DrivesQuality". Use this Download Link.

The texts on the Agile Quality Coaching Cards are based on the book What Drives Quality – 1st edition. It's a deck of 52 cards in PDF format with statements about agile quality mindset, values, principles, and practices. These statements can be used to investigate the quality of products and the practices used to deliver high-quality software.

A suggestion for using these cards in agile coaching is included. Agile coaches and Scrum masters can also use their own coaching techniques with these cards, or use exercises from the Agile Self-assessment Game.

About the Author

Ben Linders: Trainer / Coach / Adviser / Author / Speaker

Ben Linders is an Independent Consultant in Agile, Lean, Quality and Continuous Improvement, based in The Netherlands.

Author of Getting Value out of Agile Retrospectives, Waardevolle Agile Retrospectives, What Drives Quality, and Continuous Improvement. Creator of the Agile Self-assessment Game.

As an adviser, coach, and trainer I help organizations with deploying effective software development and management practices. I focus on continuous improvement, collaboration and communication, and professional development, to deliver business value to customers.

I'm an active member of networks on Agile, Lean, and Quality, and a well-known speaker and author.

I share my experiences in a bilingual blog (Dutch and English), as an editor for Culture and Methods at InfoQ, and as an expert in communities like Computable, Quora, DZone, and TechTarget.

Follow me on twitter: @BenLinders.

Bibliography

My Blog and Books

Ben Linders - Sharing my Experience - www.benlinders.com

Getting Value out of Agile Retrospectives

What Drives Quality

Register your book at benlinders.com/what-drives-quality

Books (Ordered on Title)

Accelerate: Building and Scaling High Performance Technology Organizations by Nicole Forsgren, Jez Humble, and Gene Kim.

Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior by Tom DeMarco et al.

Apollo Root Cause Analysis by Dean L. Gano.

Code with the Wisdom of the Crowd by Mark Pearl.

Commitment: Novel about Managing Project Risk by Olav Maassen, Chris Matts, and Chris Geary.

Competitive Engineering by Tom Gilb.

Continuous Delivery by Jez Humble and David Farley.

Economics of Software Quality by Capers Jones and Olivier Bonsignour.

Effective Debugging: 66 Specific Ways to Debug Software and Systems by Diomidis Spinellis.

Fifty quick ideas to improve your user stories by Gojko Adzic.

Fit for Purpose: How Modern Businesses Find, Satisfy, & Keep Customers by David Anderson and Alexei Zheglov.

Getting Value out of Agile Retrospectives by Luis Gonçalves and Ben Linders.

iTeam: Putting the 'I' Back into Team by William E. Perry.

Management 3.0 by Jurgen Appelo.

Managing for Happiness by Jurgen Appelo.

More Agile Testing by Janet Gregory and Lisa Crispin.

Peopleware: Productive Projects and Teams by Tom DeMarco and Timothy Lister.

Product Mastery by Geoff Watts.

RealityCharting by Dean L. Gano.

Refactoring by Martin Fowler.

Reinventing organizations by Frederic Laloux.

Scrum: The Art of Doing Twice the Work in Half the Time by Jeff Sutherland.

The Business Value of Agile Software Methods by Dr. David F. Rico et al.

The Clean Coder by Robert C. Martin.

The Digital Quality Handbook by Eran Kinsbruner.

The Lean Startup by Eric Ries.

The Mythical Man-Month: Essays on Software Engineering by Fred Brooks.

The ROI from Software Quality by Khaled El Emam.

The Software Craftsman by Sandro Mancuso.

Turn the Ship Around! by David Marquet.

User Stories Applied by Mike Cohn.

Visualization Examples by Jimmy Janlén.

Links

Manifesto for Agile Software Development

retrospectives.eu

Retrospectives Exercises Toolbox

Agile Self-assessment Game