Agile Documentation

A Pattern Guide to
Producing Lightweight Documents
for Software Projects

Andreas Rüping

JOHN WILEY & SONS, LTD

Agile Documentation

Agile Documentation

A Pattern Guide to
Producing Lightweight Documents
for Software Projects

Andreas Rüping

JOHN WILEY & SONS, LTD

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Rüping, Andreas.

Agile documentation : a pattern guide to producing lightweight
documents for software projects / Andreas Rüping.

p. cm.

ISBN 0-470-85617-3 (Paper : alk. paper)

1. Flexible manufacturing systems. 2. System design. I. Title.

TS155.65.R87 2003

005.1'5–dc21

2003011756

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-470-85617-3

Typeset in Garamond Light and Frutiger by WordMongers Ltd, Treen, Cornwall TR19 6LG, England

Printed and bound in Great Britain by Biddles Ltd., Guildford and Kings Lynn

This book is printed on acid-free paper responsibly manufactured from sustainable forestry,
in which at least two trees are planted for each one used for paper production.

Contents

Foreword

As Jerry Weinberg says in his classic text *The Psychology of Computer Programming*:

> *Documentation is the castor oil of programming. Managers think it is good for programmers, and programmers hate it! The value of documentation is only to be realized if the documentation is well done. If it is poorly done, it will be worse than no documentation at all.*

Nothing in the Agile Manifesto (`http://agilemanifesto.org/`) states 'Thou shalt not do any documentation', but since many developers have a genetic reluctance to any form of writing that isn't expressed in a programming language, they have clasped the following principle to their collective bosoms:

> *…we have come to value: Working software over comprehensive documentation*

and proclaimed to the world that documentation is out.

My software development career has been mostly on large projects, like the one that developed the software for the Boeing 777. There is no way that projects like that can dispense with documentation. I would be the first to admit that the 777 project and all the others I have seen close up could have been done better. They could have been completed just as well in a less ponderous fashion. Not only do I believe that there's always room for improvement, but I also believe that we should strive continually to improve – especially on safety-critical projects. So either we admit that projects of this magnitude are hopelessly 'non-agile', or we agree that, when it's appropriate – that is, when it adds value – there is a need for documentation. I vote for the latter.

Now, however, we're faced with the dilemma – what does that mean on an agile project? Can a project really follow agile principles and still produce documentation? This is the question Andreas Rüping addresses in this book. Andreas has documented his experiences of successful and unsuccessful project adventures with documentation. He shares an array of encounters with diverse projects: small to large, old technology to new, spanning the past 12 years. I've become a believer in the power of stories – nothing is better than hearing what others have done. When we share our successes and failures, we all learn. There are lots of good stories here, real projects all, with some instructive lessons learned and some pitfalls to be avoided.

Andreas does a good job of explaining the trade-offs: when documentation is better than face-to-face, when on-line is better than hard copy, when diagrams are more useful than text. In a discussion near and dear to my heart, he also shows how documentation impacts the customer.

All this information is captured as a set of related patterns. Just like stories, I'm a believer in the power of patterns. I've written many myself and I can attest that they provide guidance in a useful form. Andreas provides sufficient information for us to apply this guidance and benefit from his experience. This is useful, practical stuff.

The book follows its own principles. It is lightweight and presents the useful ideas without burdening the user. It is easy to read and understand and presents solutions that are clearly based on real project experience. I found myself nodding in agreement or tilting my head in consternation as I read something surprising. I learned a lot by reading this little book, and I'm sure you will, too.

Linda Rising

Preface

If you work in the software industry, you will know that documentation plays an important role in many projects. Among other things, documents describe user requirements, software architectures, design decisions, source code and management issues.

There can be a lot of value in such documents. Documentation can contribute to the success of a project by making necessary information available to the team members. Documents can preserve knowledge within a team, and prevent the team from re-inventing things when team members leave and new people join. Documents can capture expertise gained in one project and make it available to future projects. When knowledge has been committed to paper, it cannot be lost.

However, we are living in the information age. We are surrounded by much information, often too much. It can become difficult to filter what we really need. Projects sometimes suffer from too many documents and too long documents. If this is the case, team members looking for specific information can easily get lost. Some things are also much better communicated face-to-face than via written documents. Too much documentation is as bad as no documentation at all.

It is also hard to keep documents up to date when their subjects undergo change. Keeping documents up to date is especially hard when a project is busy and many other things require attention. But outdated documents can easily lead readers onto the wrong track – outdated documents often do more harm than good.

This book takes an *agile* approach to documentation – an approach that is both lightweight and sufficient, in the same vein as the agile approaches to

software development that recently have become popular (Cockburn 2001, Highsmith 2002, Ambler 2002).

This book presents a collection of patterns – guidelines that offer solutions to the recurring yet multi-faced problems of documentation. These patterns are governed by the following overall principles:

- Project documentation is most effective when it is lightweight, without any unnecessary documents, yet providing all the information relevant to readers.
- Documents that are considered necessary can only prove useful if they are of high quality: accurate, up-to-date, highly readable and legible, concise and well structured.
- Tools and techniques are useful only if they facilitate the production of high-quality documents and make their organisation and maintenance easier.
- The documentation process must be efficient and straightforward, must adapt to the requirements of the individual project and must be able to respond to change.

It is important to emphasise that this book does not prescribe a standard method that claims to solve all the problems associated with software project documentation. First, such a method is virtually impossible, as no two projects have the same documentation requirements. Second, a heavyweight method is the last thing I would want to propose – a fully-fledged 'standard' documentation method would be too inflexible and would involve too much bureaucracy to be useful. It would certainly not be agile.

This book focuses rather on the elements and processes that can repeatedly be found in good project documentation, and that express an agile attitude. Such elements and processes have been shaped into patterns that you can use to design the documentation that fits your individual project best, and that contributes to the expertise held in your organisation.

Scope

This book is meant for people who work in the software industry and whose job includes writing software documentation at some point. This is true for most software engineers, designers, consultants and managers. If you belong to any of these groups, then this book is for you.[1]

Perhaps you enjoy documentation, or perhaps you see it as a burden. In either case, this book will give you hints on how to focus on what is important in documentation, it should make your documentation process more efficient, and it should lead you to better results.

You can use agile documentation in different kinds of projects. First, agile documentation is targeted at software development projects. Development projects have an overall goal of delivering working software that satisfies the customer's requirements. In a development project, documentation is a means, not an end: documentation is supposed to help the team accomplish their tasks. This book recommends documentation that is as lightweight as possible, but no lighter.

Consultancy projects are also within the scope of this book. Consultancy projects place slightly different requirements on documentation than development projects, since consultancy projects sometimes have documentation, rather than software, as the desired project output. Consultancy projects can profit from an agile approach, as such an approach makes the documentation process more efficient and the resulting documents more compact and straightforward.

Organisation

Before presenting the actual patterns for agile documentation, this book begins with some introductory remarks on agile development and on patterns. If you would like to read about the *Agile Manifesto* and how it relates to documentation, this introduction will be useful to you. If you would like to learn what patterns are and how they can be used, you will also find answers in the introduction. A section follows that briefly describes the projects in which the patterns in this book were observed.

The actual collection of patterns is found in the five main chapters of the book, each of which deals with a particular topic of software project documentation. Specifically, the main chapters address the following areas:

1. *Finding the Right Topics*

   Documentation is important: some aspects of a project require documentation desperately, while others do not. So which documents are necessary in your project, and what topics should they cover? What level of detail is

1. The book, however, is not about the sort of user manuals that come, for example, with standard software packages, software installation guides or the like, nor is the book targeted at documentation that is produced by professional technical writers.

necessary? What documents are perhaps unnecessary? This chapter presents some guidelines on how to find out what documentation your project requires.

2. *Structuring Individual Documents*

   Well-structured documents give readers better and quicker access to information than poorly-structured documents. But what does a document structure look like? How can you make sure your readers easily find the information they're looking for? This chapter offers suggestions about how to increase the readability of project documents.

3. *Layout and Typography*

   Readability is one thing, legibility is another. How can document layout support the readers' ability to grasp a document's contents quickly and reliably? How can such a layout be achieved with standard word processors? This chapter tells you how to improve the appearance of your documents easily.

4. *Infrastructure and Technical Organisation*

   This chapter talks about how you can manage your project documents. The chapter begins with organisational issues: how can you obtain an overview of the project documentation? Are the documents supposed to be printed on paper? What about on-line documentation, which is becoming more and more popular? Solving such issues quickly leads us into more technical topics: how can documents be processed and stored? How can you make sure that individual documents can be found easily? What steps need to be taken to make project documentation easily maintainable? What tools are necessary for this?

5. *Management and Quality Assurance*

   The final chapter addresses management issues such as budget, responsibilities and priorities, as far as project documentation is concerned. The questions to ask here are: what does an efficient documentation process look like, or, how can bureaucracy be avoided? Being agile means putting people in the foreground, so this chapter emphasises the roles people play in the documentation process and stresses the importance of feedback and reflection.

How to read this book

There are different ways to read this book. You don't necessarily have to read the book in sequential order:

- If you are interested in a quick overview, just go through each pattern quickly and read the boldface sections. These form thumbnail sketches that give you an overall impression of the actual pattern. In addition, a summary of all such thumbnails is given at the end of the book.
- Read the complete patterns if you want to gain deeper insight, and particularly if you're interested in the rationale behind the individual patterns.
- Begin with the experience reports, if you'd like to take a journey through several real-world projects. The reports explain how the patterns were used in those projects.

It's a good idea to combine these approaches. You can start with the thumbnails, so you get an overview of what the book has in store, and read the complete patterns when you become interested in the details or the background of a pattern. You can then use the thumbnails as a checklist when you work on the documentation of your project, using the complete patterns when dealing with more detailed issues. Alternatively, you can begin with the experience reports, and follow the references to the individual patterns whenever you feel a pattern is of particular interest to you.

If you are interested in some topics more than others, you can concentrate on the chapters that are of particular interest to you. Pointers will occasionally refer you to related material in other chapters.

This is a relatively short book: it is intentionally lightweight and aims to follow the approach it proposes – you don't have to read many hundreds of pages. Many of the patterns fit on two or three pages, and you can use the thumbnails if all you need is a short overview. It won't take you too long to make yourself familiar with an agile approach towards the documentation of software projects. I'd like to invite you to take this approach with the goal of making documentation more effective for authors and readers alike.

I am interested in receiving your feedback on this book. If you have any comments, feel free to contact me at `rueping@acm.org`.

Andreas Rüping

Acknowledgements

Project thanks

My first ideas on agile documentation (though I didn't refer to it as such at the time) date back several years to a time when I was working at FZI (Forschungszentrum Informatik, Research Centre for Information Technology) in Karlsruhe, Germany. During a few research projects and several industrial collaborations, I had the chance to learn a lot about what characterises good project documentation. But there was more to it than this: the team spirit among the group allowed me to enjoy those years a lot. My thanks go out to everybody in the group, especially Gerhard Goos, Claus Lewerentz, Simone Rehm, Franz Weber, Dieter Neumann, Walter Zimmer, Thomas Lindner, Eduardo Casais, Annette Lötzbeyer, Achim Weisbrod, Helmut Melcher, Oliver Ciupke, Benedikt Schulz, Rainer Neumann, Artur Brauer, Jörn Eisenbiegler, Markus Bauer and Holger Bär.

My understanding of good documentation was refined when, a few years later, I joined sd&m software design & management AG, Germany. I had the chance to look at the documentation produced in several projects in which I was involved. Many of the patterns included in this book came to my attention when they were successfully applied in sd&m's projects. Thanks go out to my colleagues for being a good team, for the fruitful collaboration throughout many projects and for many insightful discussions.

Over the last few years, EuroPLoP – the European conference on software patterns – has been an excellent forum for discussing all sorts of topics around patterns, for me and for others. Thanks to everybody with whom I was happy to collaborate in our efforts to organise the conference, especially Frank Buschmann, Jens Coldewey, Martine Devos, Paul Dyson, Jutta Eckstein, Kevlin Henney, George Platts, Didi Schütz and Christa Schwanninger.

EuroPLoP turned out to be particularly helpful when I submitted papers on various aspects of documentation. First of all, I'd like to thank those who acted as shepherds for my papers: Ken Auer, Ward Cunningham, James Noble and Charles Weir. Their comments and suggestions for improvement had a lasting influence on the patterns that would make it into this book. Moreover, many people offered valuable feedback and loads of good ideas in the EuroPLoP workshops. They are too many to name in person, but their help was greatly appreciated.

A workshop on 'Patterns for Managing Light-Weight Documentation' at the OT 2002 conference in Oxford also generated helpful ideas. Thanks to all participants.

When I put the manuscript for this book together, several people volunteered to work as reviewers. Scott Ambler, Wolfgang Keller, Klaus Marquardt, Linda Rising, Peter Sommerlad, Markus Völter and Egon Wuchner took the time to read the draft, offered their insight and made valuable suggestions for improvement. This book has profited a lot from their generous help.

Several people have provided a lot of support throughout the publishing process. First of all, I'd like to thank Gaynor Redvers-Mutton of John Wiley & Sons for her work as the editor of this book. She provided a lot of help in making the book come to life. Thanks also to Karen Mosman for her support in the early stage of the publication process, to Jonathan Shipley for taking care of many organisational details, and Juliet Booker for her work as the production editor. Last, but certainly not least, I'd like to thank Steve Rickaby of WordMongers for the smooth ride through the copyediting stage. This was a very enjoyable process that spawned fruitful discussions on the contents, language and layout of the book.

Family thanks

I'm happy to acknowledge that this book has also profited greatly from people who weren't directly involved. My final thanks go out to Gerhard, Hiltrud, Jutta, Sven-Folker, Magnus, Nils Johann and Mareike for encouragement, support and those moments of balance that you need when you go through the process of writing a book.

Introduction

Agile development

Agile documentation has borrowed its name from the ideas of *Agile Software Development*. Agile software development was originally proposed by the *Agile Alliance* – a group of 17 software practitioners who first met in February 2001 to collect ideas for better ways of software development.

These ideas are described in the *Agile Manifesto*, which can be found on the Web (`www.AgileAlliance.org`) and which is also cited in a number of books (Cockburn 2001, Ambler 2002, Highsmith 2002).

Here is the central part of what the Agile Manifesto says:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

The manifesto continues with a number of more detailed statements and concrete recommendations.

Agile development is not one specific method of developing software. Agile development comprises several methods proposed by different people, which apply in different contexts and have different characteristics. All these

methods have in common, however, the fact that they are centred on the core values expressed in the manifesto.

Some of the best-known agile methods have been described in books:

- In his book on *Agile Software Development* (Cockburn 2001), Alistair Cockburn speaks about the central role that teamwork plays in software development projects, and about the communication issues that arise in development projects of different sizes and at different levels of rigour.
- Jim Highsmith's book on *Adaptive Software Development (*Highsmith 2000) views software development issues from the perspective of complex adaptive systems. His new book on *Agile Software Development Ecosystems* (Highsmith 2002) gives an overview of the principles of agile development, and includes interviews with several noteworthy figures from the agile community.
- Scott Ambler's book on *Agile Modeling* (Ambler 2002) addresses the modelling part of the software development process. It details practices that lead to effective and lightweight modelling, placing special emphasis on the human aspects of software development.
- *eXtreme Programming* (Beck 2000) was proposed by Kent Beck. XP, as it is usually known, is an agile method centred on programming in its social context. XP welcomes changing requirements and places much emphasis on teamwork.
- Another agile method is *Scrum* (Schwaber Beedle 2001), put forward by Ken Schwaber, Michael Beedle and Jeff Sutherland, who draw on the importance of self-organisation and reflection.
- Mary Poppendieck's forthcoming book on *Lean Development* (Poppendieck 2003) describes a number of principles of lean thinking, targeted at software development leaders.

As the Agile Manifesto is still rather new, we can expect more agile methods for software development to arise in the near future.

The role of documentation

What role does documentation play in an agile project?

The first thing to understand is that documentation appears on the right-hand side of the value statements in the Agile Manifesto. This means, in short, that the best documentation in the world is no excuse if the project is supposed to deliver software, but fails to do so.

This does not mean, however, that documentation is generally unimportant or that documentation need not be provided.

Let's take a look at what the authors of some of the agile methods have to say about documentation:

- Alistair Cockburn recommends that documentation be 'light but sufficient' (Cockburn 2001). He introduces the *Crystal* family of methodologies, which is targeted at projects of different size and criticality. The *Crystal* methodologies require documentation to be created, but let the individual project decide what that documentation should consist of.
- Scott Ambler's book on *Agile Modeling* (Ambler 2002) includes a chapter entirely devoted to documentation. This chapter is named *Agile Development*, just like this book. Scott Ambler's chapter and Chapter 1 of this book were parallel efforts. They follow different presentation styles, but they come to similar conclusions. Scott Ambler compares the agile approach to documentation with 'travelling light': to 'create just enough models and just enough documentation to get by'.
- Jim Highsmith, in *Agile Software Development Ecosystems* (Highsmith 2002), warns us not to produce documentation for documentation's sake, but calls for a balance: 'Documentation, in moderation, aids communication, enhances knowledge transfer, preserves historical information, and fulfils governmental and legal requirements'.

My view is that a light-but-sufficient approach is favourable for two reasons. First, such an approach prevents the project team from expending unnecessarily large effort on documentation. Second, light-but-sufficient documentation is more accessible, and therefore more useful, for a team than voluminous documentation. I think Scott Ambler asks the right question: 'What would you rather have, a 2000-page system document that is likely to have a significant number of errors in it, or a 20-page, high-level overview?' (Ambler 2002)

Certainly, detailed documentation is sometimes necessary, but usually the more concise and accessible documents resonate most among readers. Details often change more quickly than documentation can be updated, and are better communicated face-to-face. (There is more on written, as opposed to face-to-face, communication at the beginning of Chapter 1.)

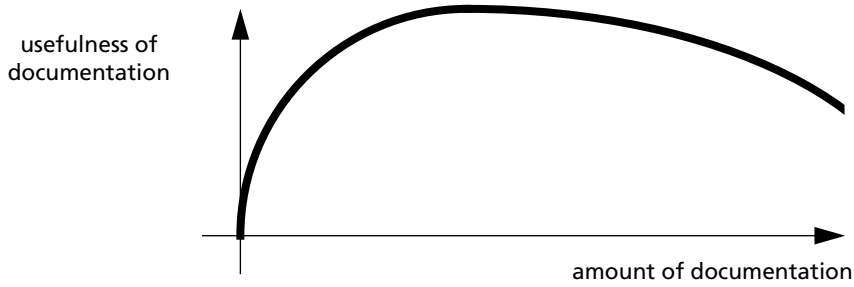

Figure 1. The usefulness of documentation

Figure 1 demonstrates the relationship between the amount of documentation and its usefulness. Beyond a certain point, the usefulness of documentation decreases when more information is added, because finding relevant information becomes more and more difficult as the overall amount of documentation increases.

I think I can summarise this by saying that quality is more important than quantity in project documentation. A certain level of detail and comprehensiveness is necessary – and depends greatly on the individual project – but it is the concise documents that contribute most to communication in a project team. The effort that you can save by producing *light* documentation is better spent on the *quality* of the documents that you do create, making those documents accurate, up-to-date and well organised.

People sometimes get the impression that, in an agile context, not only is lightweight documentation given preference over comprehensive documentation, but also that quality isn't so important. I think this is a misconception, and clearly I disagree. If you decide that a document is necessary, then it must have a purpose, otherwise you wouldn't make the decision to create it. But to fulfil that purpose, a certain quality is essential.

As with so many other things, you can choose to do something or you can choose not to, but if you choose to do it, then it's best to do it 'right'.

The patterns in this book invite you to deal with documentation in an agile way. They don't prescribe a strict process, but offer best practices for defining the right amount of documentation in your project, and for making that documentation flourish.

Patterns

So what are patterns? Let me explain.

This book deals with a variety of questions about documentation. What documentation is necessary and useful? Which topics should be covered? How should individual documents be structured? How can the project documentation as a whole be organised, and what tools are necessary to do so? How can you organise the documentation process?

If you have been responsible for aspects of the documentation of a software project, you have probably faced at least some of these questions. Such questions aren't new – whoever contributes to the documentation of a software project faces them over and over again.

Lurking behind such questions are recurring problems that have recurring solutions. These recurring solutions, or patterns, can be used as guidelines for the documentation of future projects.

A *pattern* in this sense is essentially a well-proven problem-solution pair, presented in a structured form. Users can look up patterns for their particular problems, apply the solutions, and thereby draw on the general expertise available.

In fact a pattern is a little bit more than this. A good pattern also describes the *forces* that are associated with a problem – all those issues that influence or constrain possible solutions. A pattern therefore not only presents a solution, but also offers the rationale behind that solution.

Finally, patterns normally don't stand alone. A single pattern solves a single problem, but when we approach a topic in its entirety, more often than not we are faced with a set of related problems. So what we need is a set of related patterns. The degree to which patterns are related differs. Some collections of patterns are loosely coupled and take the form of a catalogue, while others are more strongly interwoven. In the latter case, we speak of a *pattern language*.