Who Will Read My Patterns?

On Designing a Patterns Book for Target Readers

By Rebecca and Lise  
For PLoP 2019

## Abstract

*As patters authors, we spend a lot of time writing. Usually our work is driven by what we want to share, by experiences, working solutions, and ideas we are excited about. But how often do we think about the readers other than initially stating our intended audience? Who is going to consume our writing, and are we really putting in the effort to reach these readers in the best way possible?*

*We first did an informal analysis of books about software process and methods to gain a better understanding of characteristics and book design practices for this kind of book. To learn about our potential readers, and how to design our writing for better consumption, we created personas reflecting the readers. We used a combination of interviews and surveys to gather feedback on what readers are looking for when accessing information about software engineering. We then used this information combined with what we learned from the book study to create guidance for our future work of turning a set of patterns papers into a cohesive whole.*

# Introduction

Since 2015, we have worked on a collection of patterns for creating and managing a product backlog for software development. A product backlog is a set of work items that constitute the work needed to build a software product, e.g. item representing features, bugs, tests, etc. Our work is focused on larger software projects that have backlogs of a size and complexity that require digital tooling to manage them. At this point, we have documented the patterns in the collection that we have found so far, and we are starting to integrate what is today a number of patterns papers presented at EuroPLoP and PLoP [refs] into a single, cohesive work.

In our papers we deliberately experimented with different techniques to illustrate the application of the individual patterns through examples. In one paper we used e-mails and text messages to and from a main character. In other papers, we used storytelling or a set of interconnected examples about building an automated pool cleaning system. Although these experiments seemed to work in individual papers, based on feedback from our pattern readers at PLoP and EuroPLoP conferences, we do not really have any good measures to tell us what worked best. We also do not know how these techniques will work out for a consolidated group of patterns since the writing will be so much longer than a typical patterns paper.

When we started writing our patterns we were driven by a desire to share the practices we had come to trust. Looking back over the five years working on these patterns, the whole process around the pattern papers was also a massive learning process on our part, with critical feedback from shepherds and workshop participants. Not in the least, was learning how to effectively put our ideas into writing and to blend our experiences. Although we believe the publication of papers significantly helped us to clarify our concepts and make the work more accessible to readers, so far we have not really put ourselves into the shoes of our intended reading audience.

It is clear to us that we want and need to spend time and effort on designing our final product. In doing so, we want to drive the structure and style of our work based on the needs of our readers rather than from our desires and aspirations as writers.

# Approach

Starting from our motivation as explained in the introduction, we decided to follow a method adapted from the user experience (UX) community using personas and stories. Personas are fictional characters created to represent users (in our case to represent readers) and explore their needs and goals, as well as their personalities and behaviors. Stories or scenarios are then created to illustrate use of the product by the defined personas. This builds a better understanding of the use of the product. It also allows the designer to get some distance from the product itself and gain a broader perspective (e.g. the persona is using the product and the designer is “observing” them). UX techniques are increasingly used to design software products, and since our book is a product in the software domain this was a natural way of thinking for us.

In summary, the flow of events we pursued (and will continue to pursue) is as follows: data collection and analysis (research), creating persona descriptions, developing scenarios or stories, followed by developing design prototypes. A part of this flow should be approvals of those design prototypes (e.g. creating a hypothesis with a design and testing this out by sharing our writing with select user representatives). This latter activity is part of our future work and outside the scope of this paper.

We set out to do informal research in two areas: reviewing books in the same domain as the one we plan to write, and getting feedback from colleagues and friends who are readers and authors of such books. We stress the word informal, as we are both practitioners and not academics. Our approach follows no scientific method. Our fundamental goal is to gain a better understanding of how books connect with and bring value to readers, and then use this to guide our own work.

The purpose of the book review was to analyze what we think is a representative collection of books to extract ideas from which can help us with our own design. Kind of like apprentices studying the masters. To get feedback from people we created a survey with rather broad questions, and sent this out to a number of contacts. We ended up with around twenty responses that we then tried to summarize and compare to gain further insights. We now had ample materials to work from, and could compare the readers’ preferences with how various authors had designed their books.

We then used the input from our real readers to create the personas that represent our potential readers, and wrote stories of these personas reading our (so far imaginary) book and using what they read to do their job. Finally, we create a first outline of our book and agreed on the most important principles of book design that we will apply for our own writing, hopefully creating a book that real readers will find a decent piece of work. [So far this is aspirational, as we have not yet written persona scenarios]

To keep the paper shorter, and help readability, there are two appendices to this paper. The first is a description of the books we used for our research. The second is a more complete list of the design principles that we gleaned from our research. See, we are also applying our design thinking to this paper!

# What we learned from the Books

The purpose of the book review was to analyze a representative collection of books to extract ideas that can help us with our own design (like apprentices studying the masters). Since our work falls into the category of software processes and practices, the kinds of books that are most relevant to our work fall within this domain. The selection is a mix of books that we perceive to be fundamental books in our industry, as well as recent writings. We also looked at business book that software professionals read that describe practices applicable to software companies and organizations. We included some writings that are not full-fledged books but shorter pieces, for example the Scrum Primer. The selection of books was influenced by what we have access to. Since we have both been piling up books for years we do not think that our final list was too limited. We see that many of the books that were mentioned when gathering feedback from readers were books we also had selected for our review.

Let us first quickly introduce the books, before discussing various design elements and solutions as the outcome of our research. Note that a lengthier introduction of the books can be found in Appendix A. We grouped the books into the following categories: fundamental books, books on specific processes, books for specific professions within the software industry, books on DevOps (representing a current hot topic), patterns books, and business books.



Figure 1 Books we studied

Fundamental books (classics): *The Mythical Man Month, Software Requirements, The Deadline: A Novel about Project Management, Retrospectives*

Books on specific processes and methods: *Extreme Programming*, *The Scrum Guide*, *The Nexus Framework*, *Large-Scale Scrum*, *Writing Effective Use Cases*

Books for specific professions (architects, developers, and testers): *Code Complete*, *The Pragmatic Programmer, Refactoring*, *Developing and Communicating Software Architecture, Agile Testing, How to Break Software*, *Lessons Learned in Software Testing*

DevOps books (representing current hot topics): *The Phoenix Project, The DevOps Handbook, Site Reliability Engineering, The Site Reliability Workbook, Accelerate*

Patterns books: *Design Patterns*, *the Patterns of Software Architecture (POSA) Series*, *Analysis Patterns*, *Organizational Patterns*, *Fearless Change*, *Business Patterns*

Business books: *Agile and Lean Program Management*, *The Goal*, *The Lean Startup*

By no means do we argue that this is THE list of books to read (if you want that list there are several web pages that recommend these lists [refs]). It is purely the list of books that we ended up processing. We are also not looking at classifying individual books as “good” or “bad.” Every book had something to teach us and gave us ideas of book design practices. And each book had something that could have been done differently and arguably better, although that judgment is based on individual readers’ opinions. Finally, I, Rebecca, didn’t argue for including my own books in this list, although they certainly influence the lens through which I examine other books.

Technical books are certainly different from novels, and while (we think) that writers of fictional work are in the profession because they somehow enjoy writing, authors of technical books are more driven by the desire to teach and share knowledge, possibly with a pinch of desire for peer recognition added to the mix. The wish to share does not necessarily make one into a good author, and authors of technical books normally write one or a few books while spending most of their time working in their profession. This may be one reason some books can be a hard read even if the content is very valuable to the reader. This brings us to the problem of trying to classify what makes a good software process book. Is it the readability or the contents? Is it its timeliness or its ability to survive over time? We are not trying to reach a conclusion to these questions in this paper, but rather to look at elements and practices in the design of books on software process and practices that can help guide us as we write a book that hopefully is a good read to a decent audience and that stands up over time.

We need to clarify one point—when we describe books as belonging to the software processes and practices domain we mean this in a broad way to differentiate between books about mindset, workflows and practices, team collaboration and related topics from those software books that are about specific technologies, programming languages, or detailed coding practices. Looking at our selection of books, this should be evident, but we still wanted to state this to avoid any misunderstanding.

The first thing we realized is that there are different styles of books.

Story-telling  
In these books, the goals of a methodology gradually become clear through stories, either as examples or introductions or authors personal experiences, or in the ultimate way as full-blown novels. This style may make the advice of what to do more obscure—the reader needs to extract these essences out of the story and find out how these ideas might work in their own context. But this style allows for a more intuitive and personal understanding of the whys, and it leaves more to the imagination of the reader, which is more fun and challenging than books that are very prescriptive.

Mindset/philosophical  
These writings (whether the whole book or the introductory part of a book) are focusing on the way of thinking rather than specific activities.

Practical Guides  
Most of the software process books fall within this category, describing a number of activities in enough detail that the readers can practice them. In many cases, these books become tools that the reader repeatedly refers to when performing the practices.

References  
Some books really are collections of a specific type of content, as for example some of the patterns books. They differ from the practical guides in that there is no story involved in the sequencing of their contents, nor goal of building up a methodology through following the practices.

Empirical  
Contents of these kinds of books are built on a certain amount of research, by studying a number of teams or organizations or systems to extract out essential information. So rather than being the ungrounded ideas or opinions of the authors, they are based on empirical studies.

The next insight we gained was that there are a number of topics where the author needs to make conscious decisions when creating the design of a book. The type of book will impact these decisions, and we have tried to summarize this thinking in table 1. But first let us explain the design topics:

#### Overall structure

When writing a theater play, there is a certain recipe to follow: introducing the characters, setting the context, building up the suspension in the first act, and resolving it in the second, creating the finale. The design of technical books may be less evident and there is a lot of variation. But good technical books typically have a structure that helps the reader navigate their contents. They often provide both an overview of the communicated knowledge as well as the details.

A common practice that works well is to start with an overview that introduces the concepts and major ideas. If this is written as a “standalone” section, readers who are not ready to dig into the details can still appreciate the book, and possibly recommend it to others and/or put it aside for further study later. This method is often used in Practical Guides to first create an understanding of the overall domain or methodology, and then followed with the more direct advice or detailed techniques. The same method can be applied within each chapter, starting with explaining the more detailed concepts and then going into the details of their application.

Both a book’s contents sorted into chapters, and the order of those chapters have a big impact on readability. Although not a novel, there is still a story to be told or a methodology to be built, and the reader needs to be exposed to the concepts and ideas in a logical sequence. Typically, this proceeds either from the basics to more advanced topics and/or from the start to the end of a development process timeline.[[1]](#footnote-1) Books written by multiple authors, and not least where authors write individual chapters are especially challenging with a higher risk of repetition and even potential conflicts or inconsistences in terminology and practices.

#### Navigation

Navigation is related to structure in that a book with a good flow of content is also easier to navigate. But help with navigation can also be accomplished by explicitly explaining the structure of the book in the introduction[[2]](#footnote-2), pointing the reader to particular sections targeting special roles or interests. The essential navigation tool is the table of contents (TOC). For the TOC to work well, it is important to give descriptive names to the chapters so that it is clear what they are about. Cool or catchy titles increase may the entertainment value, but at the cost of navigational support.

#### Length

After years of reading, and being avid readers at that, we observe that the length of the book is a key element of its design. Too short, and a book is more like a booklet and may not gain the full respect as a book would. But we both have read too many books that are far longer than they needed to be. Some carefully designed repetition to help readers remember the essence (like a summary at the end of each chapter) is good. But many books readability would improve if the authors focused on removing repetition and in some cases remove sections that are not that relevant to the main ideas of the book.[[3]](#footnote-3)

#### Use of illustrations

Illustrations are good for two reasons: they add to the understanding and they break up the monotony of the text. They need to be relevant, and they need to be accompanied by explanatory text. They also need to be intuitive enough not to require in-depth study to make sense. Finally, they need to work in color as well as in monochrome.

Illustrations in a book are part of creating a book’s identity. It works really well when illustrations following the same style throughout.

#### “How to” examples

When writing about software methodology rather than software programming or design, examples are probably best done as small stories (e.g. the examples are not code snippets). The challenges are that the story must be long enough to provide the reader enough information to understand how to use a practice, but without taking up too much space. The story must also be able to connect to core readers using a simple context and not require implicit knowledge to follow it.

One practice is to use a running example throughout the book. Thiss makes it easier for the reader to follow (e.g. not changing the context for each example). But speaking from experience, this is much more difficult for the writer who has to creatively think how to weave the topics she wants to cover into a coherent story. Technical books that are written as novels have accomplished this, as the ideas are explained through one extensive example. However, the downside of this one example approach is that not all the patterns may be so easily woven into a single storyline, especially if there are competing pattern sequences that could be followed.

#### Choice of author’s voice

Authors who share some personal stories tend to increase the credibility of their book’s contents. It builds trust in the reader as they learn about the author and her personal experiences. However, some may not like a book that is too personal (they want some distance from the author and desire that the contents to speak for themselves). So choosing what voice to use must be done with care. And if there are several authors, how best can you blend their voices? A related issue is if whether the author should address the reader directly, or keep the form more generic. Either way, the book needs to apply the same approach consistently throughout.

#### Style of notes and references

Some academic works include references to such an extent that it affects readability. When writing a book for practitioners it may be better to keep references more low-key and not have too many footnotes.

#### Patterns

Since our planned book has patterns, we need to think specifically about the above topics in the context of a patterns book. Unless we create a patterns book that is more of a reference type book, there are challenges with getting the appropriate balance between the flow and the collection of patterns, and to deal with the broader context and specific details without being repetitive within the individual patterns. One book that does this well is Allan Kelly’s *Business Patterns*.

Another challenge is how to include examples in a natural way. The practice of “3 known uses” in every pattern is not the most user-friendly solution. A running example and a good discussion of the application of the patterns may be better.

A patterns book, in contrast to a conference paper, has the benefit of dealing with a collection of patterns in one piece of work. That enables the authors to not only deal with the shared context outside any individual pattern, but also to introduce patterns sequences and to tell stories that weave the collection together into the whole that “patterns people “ long for.

There is no one solution for addressing these topics in a book design. The appropriate way to deal with a particular topic depends on the context of the book, the needs of its intended readers, as well as the preference of the authors. Probably, many books are writen without an explicit decision about each design topic. Be we firmly believe that to make an effort into understanding and deciding on how to deal with these topics will benefit our final work. In table 1 we look at the characteristics of each type of book in relation to design topics.

|  |  |  |
| --- | --- | --- |
| Book type | Book examples | Characteristics |
| Story-telling (or elements of) | *The Mythical Man Month*  *The Deadline*  *Agile Testing*  *The Phoenix project*  *Fearless Change*  *The Goal* |  |
| Mindset/philosophical | *The Mythical Man Month*  *Retrospectives*  *Extreme Programming*  *Code Complete*  *The Pragmatic Programmer*  *Refactoring*  *Agile Testing*  *Lessons Learned in Software Testing*  *Accelerate*  *Fearless Change*  *Business Patterns*  *The Goal*  *The Lean Startup* |  |
| Practical Guides | *Retrospectives*  *Extreme Programming*  *The Scrum Guide*  *The Nexus Framework*  *Large-Scale Scrum*  *Writing Effective Use Cases*  *Code Complete*  *Refactoring*  *How to Break Software*  *Lessons Learned in Software Testing*  *The DevOps Handbook*  *The Site Reliability Workbook*  *Fearless Change*  *Org Patterns*  *Business Patterns* |  |
| References | *Software Requirements*  *How to Break Software*  *Design Patterns*  *POSA*  *Org Patterns* |  |
| Empirical | *Accelerate*  *Org Patterns* |  |

Table 1: Book types and characteristics

# What we learned from the Readers

We started off considering what we like about technical books, being consumers as well as authors (Rebecca is an experienced book author, Lisa is an aspiring book author). By interviewing each other about books we have read, and discussing what we appreciated with the these books and what we would have changed, we came up with a set of questions to send to people we know in the software community. These are the questions we asked:

### *Survey of technical books (e.g. books on software process or software technology)*

* *What makes you choose a technical book?*
* *Is there a technical book that you would recommend to your colleagues and if so why?*
* *Do you have personal favorite technical book, and if so what do you like about it?*
* *Do you have a technical book that you read more than 2 years ago that still has an impact on your work?*
* *If you think of a technical book that you did not particularly like or find useful, can you explain why?*
* *If you have written books yourself, do you have any advice to share?*
* *How much of your technical knowledge is from reading books versus information from other sources?*
* *What medium do you prefer for technical knowledge (books, articles, blogs, other online sources)?*

The primary readers of our patterns are roles that work extensively on managing requirements using digital tooling (e.g. ALM systems like JIRA, Doors NG, VSTS etc.). Because we felt that our thinking around writing technical books was not particularly specific to this topic, we also decided to reach out to people that we know are reading books regularly and have a broader reference frame. Targeting only people in a Product Owner or Business Analyst role would limit the people we could ask for feedback and the books that we would analyze for design ideas without good reason. We also ended up asking a few friends who are electrical and mechanical engineers to see if they had additional or different feedback than software engineers.

We sent out the questionnaire to colleagues and friends within the software community and ended up with xx responses from a variety of roles and affiliations (see table 2). In a few cases we followed up with questions to learn more about comments that we found intriguing.

(insert table 2 showing background of responders)

Table 2: Background of responders

Going through the various answers we created a table extracting the essence of the responses. This is shown in table 3 below.

|  |  |
| --- | --- |
|  | Extracted common threads |
| What makes you choose a software process book? | Recommendation from colleague or friend (someone trusted) From conference or talk (either speaker is an author or the speaker recommends a book) Knowing the author Popularity Required for work |
| Is there a book that you would recommend to your colleagues and if so why? | Recommended books are typically classics or very new, they are mostly very known, and connected to the role of the person recommending (QA, architect, etc.) |
| Do you have a personal favorite of this type of book, and if so what do you like about it? | Books that are:  - Conceptual - Helpful (constructive) - Books as novels |
| Do you have a software process book that you read more than 2 years ago that still has an impact on your work? | Most have personal favorites that they return to, for example Fowler’s Analysis Patterns, or The Mythical Man-month |
| If you think of a book that you did not particularly like or find useful, can you explain why? | Books that bring nothing new and have too much fluff Books written for the wrong reasons (marketing of company, or joining the hype bandwagon without proper experience in the topic) Hard reads (technical, good content but hard to access) |
| If you have written books yourself, do you have any advice to share? | Use stories. Write less. Publish more. Create several small books rather than one big one. Use LeanPub to allow the book to grow. Write most of the book before you go to a publisher. |
| How much of your technical knowledge gathering is from reading books versus information from other sources? | Varies from 5% to 70%. |
| What medium do you prefer for gathering knowledge about software process/practices/tools (books, articles, blogs, other online sources)? | Many different ways to gain insights, but books are still important to most. Conferences or online resources may bring initial ideas, but the fundamentals in a well-written book last longer. Audio books are a good alternative to some as you can listen while doing other things. |

Table 3: Summarizing the responses

(but lots more to harvest here, need a good discussion)

# Creating personas

Instead of doing user-centered design of a software product, we are doing reader-centered design of a book for software professionals. When designing for users (readers), a persona is a way of representing user roles or user goals in a way that both make the user more personal to deal with, and that can represent more than just the role itself. A persona can be a user of a certain type with a defined experience level and preference, and this helps the designer think about all aspects of users.

The following dimensions create the user space we want to explore for our readers:

* User roles: Product Owner, Business Analyst, Senior software developer/architect
* Experience: Not familiar with Application Lifecycle Management (ALM) tools, used ALM tools before but did not create the structures, expert user of ALM tools (admin level privileges)
* Personality: reads to get the general understanding (impatient), reads software process book to gain all knowledge possible (thorough)

If we were to cover all combinations of these characteristics we would have to create 18 personas. But to represent typical users, we only need to make sure all characteristics are included in at least one persona. In this case we decided to create three personas: Allison, Caroline, and Taylor.

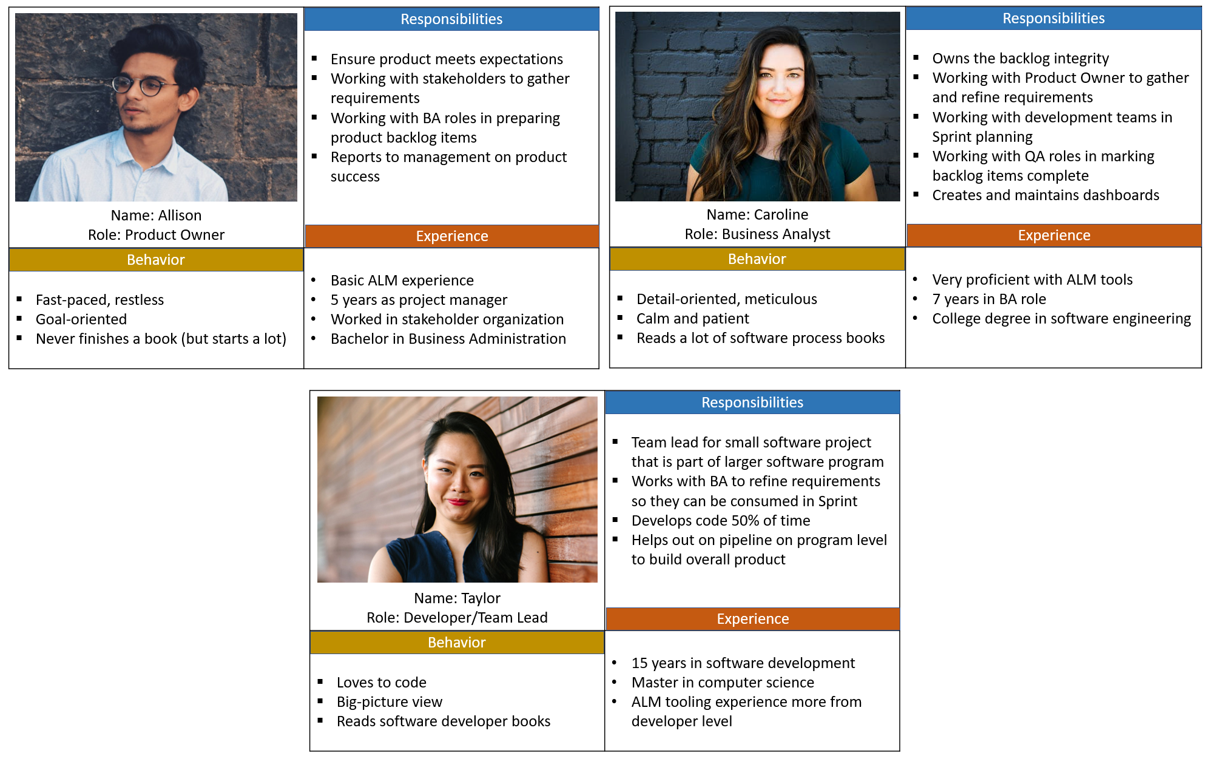


Figure 2 Persona cards

# Creating a story

(TBD: write the stories (e.g. informal use cases) after we get the feedback from at least some of the people we asked)

# Designing a book for our personas

(TBD: building on the above stories)

# Conclusion

(TBD: when paper is about done)

About the selection of books – not excluding other books. Show me your bookshelf and I will tell you who you are.

We initially planned to include books about software tools, since our work is about using digital tooling to manage requirements. We eventually decided against this for two reasons: we are staying focused on methodology and deliberately avoid particular tooling, and books on tooling tend to be as short lived as the technology version itself so it quickly gets outdated.

# Acknowledgements

Thank authors and people who were interviewed. Thank the shepherd.

# References

(TBD, including all books and writings we are listing under Books)

# Appendix A: The books

This chapter presents an overview of the books that we selected to analyze, each with a short description. In the next appendix on book design we discuss various design practices and styles, as well as a table that summarizes the characteristics found in the books. We grouped the books into the following categories: fundamental books, books on specific processes, books for specific professions within the software industry, books on DevOps (representing a hot topic), patterns books, and business books.

## Fundamental books

These books are classics in the software domain, books that many software professionals know of and look upon as a fundamental part of a proper software education that include some understanding of the history of software development.

***The Mythical Man Month*** by Fred Brooks is a collection of essays on software engineering and was first published in 1975, with a second edition in 1982 and a third edition in 1995. The lessons learned and the insights provided in these essays have mostly stood the test of time and most of this book is as valid today as it was when first published. The topics are about software development processes on a conceptual level that do not depend on a particular software process, and about human nature.

***Software Requirements*** by Karl Wiegers and Joy Beatty provides a fundamental understanding of software requirements and their characteristics. The third edition was published in 2013, while the first edition dates back to 1999. The tables with quality characteristics for requirements and requirements collections are just as valid and valuable today as they were in 1999.

***The Deadline: A Novel about Project Management*** by Tom DeMarco tells the story of a software manager who is downsized but then is kidnapped to an imaginary country and gets the opportunity to test out project management principles in a large scale experiment. By choosing the novel format, the author makes this an entertaining read, and since this is a type of storytelling it is easier to remember the advice given on a number of topics. Some of the process thinking has evolved since the book was published in 2011, without that making the book less of a recommended read.

***Retrospectives*** by Norm Kerth is the first book about doing retrospectives, and although some of the thinking is a bit out of touch with today’s business reality (like an off-site 3 day event), and it is missing the idea of heartbeat retrospectives, it is still the primary retrospective book that lays the foundation of the practice not least from the perspective of the philosophy of reflection and trust.

## Books on specific processes and methods

Every software methodology has its core documentation, and here we have selected a few representative items.

***Extreme Programming*** by Kent Beck  
(Rebecca, do you have this book??)

***The Scrum Guide*** by Ken Schwaber and Jeff Sutherland is according to the authors the definitive guide to Scrum, and they make it freely available on its own web site independent of any commercial interests. This writing is a short and very specific description of how Scrum works that should be a must read to any member of a Scrum team. First published in 2010, it would also classify as a fundamental work.

***The Nexus Framework*** by Kurt Bittner, Patricia Kong and Dave West tackles the scaling of Scrum to handle multiple Scrum teams working from one product backlog (e.g. program level methodology). It is a short book that follows the crisp and less is more style of the Scrum Guide.

***Large-Scale Scrum*** by Craig Larman and Bas Vodde is their version of a multi-team Scrum methodology. This book provides more insights in the goals and “why’s” than the short Nexus documentation.

***Writing Effective Use Cases*** by Alistair Cockburn was the book about use cases and for those that feel the limitation of user stories is not quite enough to really understand users interaction with their product this is still a very useful book to read.

## Books for specific professions (architects, developers, and testers)

The processes and methods in these books are mostly on the level of the individual developer.

***Code Complete*** by Steve McConnell is a highly recognized guide to developers. The second edition was published in 2004, (TBD)

***The Pragmatic Programmer*** by Andrew Hunt and David Thomas was first published in 1999, and it has tips and practices for software engineering (not a consistent methodology). A second edition is coming this year.

***Refactoring*** by Martin Fowler addresses ways to deal with legacy code.

***Agile Testing*** by Lisa Crispin and Janet Gregory is an essential guide for how the role of testing is integrated in agile teams.

***How to Break Software*** by James Whittaker has a number of testing techniques to attack software products in a systematic way. It is short and very practical, and written in a fun way following the philosophy of the exploratory testing community seeing testing as an intellectual activity.

***Lessons Learned in Software Testing*** by Cem Kaner, James Bach, and Bret Pettichord is another book from authors that are thought leaders within exploratory testing. It deals not just with practical testing techniques but with the overall way of thinking as a tester.

**Developing and Communicating Software Architecture** by Nick Rozanski and Eoin Woods is a book that provides a framework for designing and documenting relevant views of an architecture.

## DevOps books

Since DevOps is the buzzword these days and we have read some of the books on the topic we include this to represent current writings.

***The Phoenix Project*** by Gene Kim, Kevin Behr, and George Spafford is written as a novel following the story of an IT manager who is given 90 days to turn around his company’s work processes to improve product delivery (e.g. DevOps).

***The DevOps Handbook*** by Gene Kim, Jez Humble, Patrick Debois, and John Willis is the companion to the Phoenix Project that covers the workflows and methodology that emerges in the novel in a systematic fashion.

***Site Reliability Engineering*** edited by Betsy Beyer et. al. is a collection of essays and articles by multiple authors that describe the DevOps approach within the Google organization.

***The Site Reliability Workbook*** edited by Betsy Beyer et. al. is describing how the Google SRE is done in practice within Google and other organizations that are implementing SRE based on the Google model.

***Accelerate*** by Nicole Forsgren, Jez Humble, and Gene Kim is based on their research when doing the State of DevOps reports from 2014 to 2017. It provides an empirical study of the effectiveness of DevOps practices.

## Patterns books

***Design Patterns*** by Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides is the fundamental classic of software design patterns containing 23 documented patterns.

***The POSA Series*** by Frank Bushman et. al. are six books that focus on pattern-oriented software architecture.

***Analysis Patterns*** by Martin Fowler focuses on patterns for business processes and the architecture of domain specific systems.

***Domain-Driven Design Domain-Driven Design*** by Eric Evans focuses on patterns for identifying the core business process and developing a common language among business experts and developers who create software that supports the business.

***Organizational Patterns*** by Jim Coplien and Neil Harrison

***Fearless Change*** by Mary Lynn Manns and Linda Rising is a patterns book but also a business book, presenting patterns to introducing change in an organization from a grassroot level.

***Business Patterns*** by Allan Kelly is also a patterns book within a business context. It is aimed at roles that are considering starting businesses in the software industry whether doing consulting or selling software products.

## Business books

There are business books that are written by software professionals and/or targeting software organizations, and also general business books that have gained popularity in the software community:

***Agile and Lean Program Management*** by Johanna Rothman is a thorough Guide for creating and managing software programs (e.g. multiple projects working together to deliver a product).

***The Goal*** by Eli Goldratt is now available in its fourth edition. It was first published in 1984, and is a novel following a manufacturing manager as he is demanded to turn around productivity at the factory site he is managing. The Phoenix project is written following the same recipe but focusing on software product deliver instead of the delivery of good. In both books the manager is helped by a role working as a hand-off coach.

***The Lean Startup*** by Eric Ries is inspired by lean manufacturing ideas and is focused on entrepreneurial management based on short product development cycles and rapid validation by customers.

# Appendix B: Design principles for software process books

We processed the survey feedback by grouping similar responses and then extracting out the essence of the answers. Our explorations followed the categories we defined when designing the questionnaire.

Here are the key principles we found for designing software process books, with some thinking around how they can be fulfilled. They are not listed in a particular order, but we tried to group them and have some kind of natural flow.

Don’t make me read the whole book

Some books have an introductory part that gives an overview and general understanding of the contents. Following chapters then go deeper into the various concepts, and there is a guide in the overview section that clarifies what chapters are covering what content. This means that readers who want only a general understanding can get away with reading maybe the first 40-50 pages. Readers who are familiar with, or particularly interested in, parts of the material can choose what later chapters to read. In some cases each chapter is also structured in an overview section and then goes deeper.

Share the excitement

Some may prefer the author to be formal, but in most cases the book turn to be more attractive to read if there is a genuine feeling that the author cares deeply about the contents. This can be done by sharing some personally experienced situation, or by sidebars with additional insights.

Avoid being negative

In general, focusing on what is working and how to get to the desired outcome is a better use of the reader’s time than to spend paragraph upon paragraph on what are bad practices. It is mostly more helpful to know what to do than to know what to avoid, and an author that keeps tarping on negative topics will most likely lose the reader quickly.

Limit repetition

This should be an evident thing to remove when a draft reaches an editor, but we have encountered several technical books of 300 pages or more that would have done better staying below 200 pages. Note that some repetition can be done by design, as it is known that it can help the reader remember. But then it should be accomplished through chapter summaries and not by what seems to be verbose sections with little content.

Shorter is better

This is related to the point of limiting repetition, but goes further in not just trying to avoid repetition but in keeping sentences short and specific, and mercilessly cutting out content that is not required to assist the reader. The Scrum Guide is a good example where you get the feeling that every word and every sentence is carefully selected and created, and there is no fluff or part that is not deliberately included.

Focus on goals not on tasks

This is not only good advice when writing user manuals, but also when dealing with software processes and practices. Good pattern names typically follow this advice, it is not so much about specifically what to do but on what you want to achieve (e.g. focus on why and how to generate a desired result). Failing applications of agile processes fall into the trap of focusing on tasks, that is doing described practices but failing to understand and achieve the purpose.

Have a consistent voice

As an author one should not be afraid of making the writing personal while still showing respect for the reader. Maybe the right saying is professionally personal – not including stories involving family and friends but rather focusing on personal experience as a software professional, and always considering whether or not a specific story is working towards the goal of making the contents more accessible to the user. A personal style tends to be less dry and more entertaining, and it lends more credibility to the contents.

Simpler is better

Deal with complex stuff but splitting it up.

Do not complicate to show off. This may be more frequent in academic writings, but sometimes one can get the impression that the author is wanting to …

No spaghetti

Topics are self-contained.

Navigation

Headings and a well-organized TOC is super important to help the reader. Names should be from the user’s domain. Focus on goals.

Leave dry for dry martini

So this one is difficult. Technical stuff is by nature rather boring, so how to not cause your reader to doze off?

Illustrations that make sense

Illustrations are important for several reasons. They give a different way to document and so complement the text. They are an alternative to text that lightens up the reading and provide provide information visually. Illustrations must make sense and fit in the flow. Place then where the illustration is discussed and always discuss the illustration in the text. Make sure each illustration is easy to understand and that it complements the text.

Visual

Bb

Flat structure

Bb

Marketing

Bb

Storytelling

Bb

(TBD: complete current topics, expand with feedback from the survey, and get in a natural order where topics build on one another).

# More work to do

Lok at this:

<http://www.ambysoft.com/books/bookWriting.html>

<https://paulcunningham.me/writing-technical-books/>

<https://www.thecreativepenn.com/2015/12/11/how-to-write-non-fiction/>

1. Although this ordering from basic or fundamental topics to more advanced concepts may cause the reader to skip important concepts that are stuck in the back of a long book. Eric Evans has remarked that the most important part of his *Domain-Driven Design* book was the second half and unfortunately, because of this book’s length, not many readers got that far. [↑](#footnote-ref-1)
2. Assuming that the readers read the introduction, of course. [↑](#footnote-ref-2)
3. During the writing process of my second book, I, Rebecca, felt I really had to include sections on documenting designs and handling exceptions. While these were great topics, they were not core to responsibility-driven design. Good chapters, but not essential ones. [↑](#footnote-ref-3)