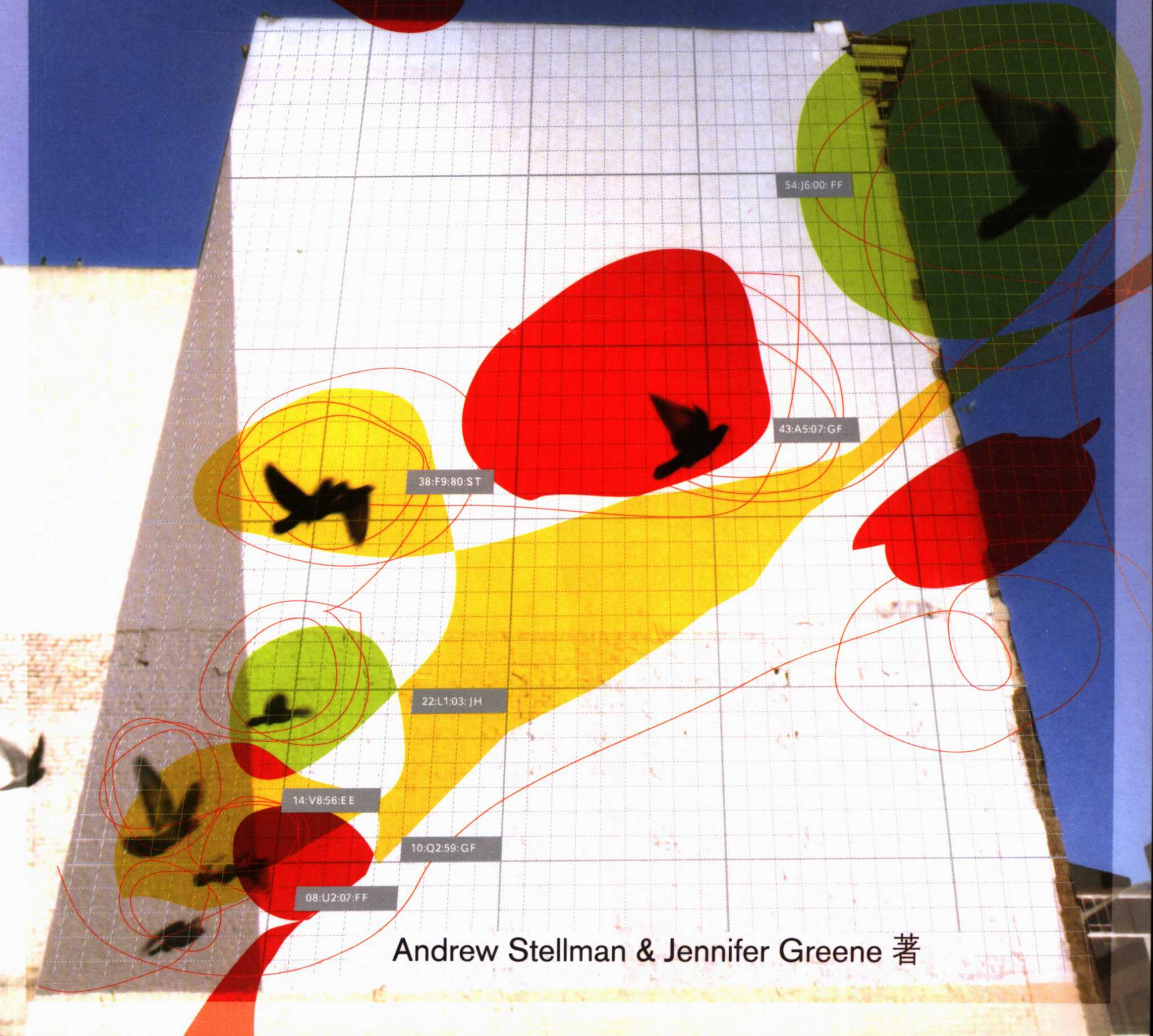
/THEORY/IN/PRACTICE

O'REILLY®
東南大學出版社

Applied Software Project Management
实用软件项目管理（影印版）



Andrew Stellman & Jennifer Greene 著

Applied Software Project Management
实用软件项目管理（影印版）

图书在版编目（CIP）数据

实用软件项目管理 /（美）斯特尔曼（Stellman, A.），（美）格林（Greene, J.）著 .— 影印本 .— 南京：东南大学出版社，2006.11

书名原文：Applied Software Project Management

ISBN 7-5641-0578-X

Ⅰ. 实… Ⅱ. ①斯… ②格… Ⅲ. 软件开发－项目管理－英文 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2006）第 121606 号

江苏省版权局著作权合同登记

图字：10-2006-255 号

书　　名 /	实用软件项目管理（影印版）
书　　号 /	ISBN 7-5641-0578-X
责任编辑 /	张烨
封面设计 /	MendeDesign，张健
出版发行 /	东南大学出版社（press.seu.edu.cn）
地　　址 /	南京四牌楼 2 号（邮政编码 210096）
印　　刷 /	扬中市印刷有限公司
开　　本 /	787 毫米 × 980 毫米　16 开本　20.25 印张
版　　次 /	2006 年 11 月第 1 版　2006 年 11 月第 1 次印刷
印　　数 /	0001-2500 册
定　　价 /	42.00 元（册）

O'Reilly Media, Inc.介绍

O'Reilly Media, Inc.是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到 GNN（最早的 Internet 门户和商业网站），再到 WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为 O'Reilly Media, Inc.紧密地与计算机业界联系着，所以 O'Reilly Media, Inc.知道市场上真正需要什么图书。

出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的一批影印版图书，包括：

- 《深入浅出 Ajax》(影印版)
- 《Ajax Hacks》(影印版)
- 《深入理解 Linux 网络内幕》(影印版)
- 《Web 设计技术手册 第三版》(影印版)
- 《软件预构艺术》(影印版)
- 《Ruby on Rails: Up and Running》(影印版)
- 《Ruby Cookbook》(影印版)
- 《Python 编程 第三版》(影印版)
- 《Python 技术手册 第二版》(影印版)
- 《Ajax 设计模式》(影印版)
- 《实用软件项目管理》(影印版)
- 《用户界面设计模式》(影印版)

Preface

SOFTWARE PROJECT MANAGEMENT is the art and science of planning and leading software projects. It requires knowledge of the entire software development lifecycle: defining the vision, planning the tasks, gathering the people who will do the work, estimating the effort, creating the schedule, overseeing the work, gathering the requirements, designing and programming the software, and testing the end product. Throughout the process, there are many team members who are responsible for these tasks; the project manager needs to have enough knowledge of their work to make sure the project is staying on track.

To be effective, a project manager must have a wide range of expertise. In this book, we provide an introduction to all of these areas so that you can guide the rest of your team on their tasks. We help you run successful software projects, and we help you diagnose and fix the ones that have gone off track.

Goals of the Book

This is a practical book. This book describes the specific tools, techniques, and practices that a project manager needs to put in place in order to run a software project or fix an ailing one. A project manager can use this book to diagnose and fix the most serious problems that plague software projects. It contains essential project management tools, techniques, and practices, which have been optimized to be as straightforward and easy to implement as possible. It also contains advice for avoiding the problems that a project manager will typically encounter when bringing these tools into an organization.

By the time you have read this book, you should be able to:

- Define the scope of your project
- Estimate the effort required to do the work and schedule your project
- Conduct thorough reviews of documents and code
- Gather software requirements and create specifications
- Effectively manage the design, programming, and testing of the software
- Provide guidance if your project runs into quality problems
- Manage an outsourced project
- Make effective changes to the way projects are run in your organization

We have been researching and implementing these tools, techniques, and practices throughout our combined careers. Each of them is the culmination of years of trial and error in many different organizations across multiple industries. Every one of these practices is the solution to a specific, chronic problem. Many people opt to live with the problem, because the solution seems too complicated. Our ultimate goal in writing this book is to help you build better software.

Often, the original idea (before we optimized it) came from a book that we read to solve a specific problem that we encountered on a software project (just like you are doing right now!). References to some of the books that we found most helpful appear in the text in order to help you learn more.

Who Should Read This Book

Many software organizations have trouble delivering high-quality software on time. Most of them have talented team members; almost all of them realize that there is a problem. People in these organizations may have already read books about management, quality, and programming. What all of these people have in common is a need to change the way a software organization works. They may not always recognize the nature of the problem, or what is causing delays or bugs. What they do know is that something needs to change.

We wrote this book for anyone in a software organization where there are chronic problems producing software on schedule and without defects. The intended readers of this book include:

- A project manager responsible for a software development project and/or team
- A programmer, designer, business analyst, architect, tester, or other member of a software team looking to improve the product he is working on

- A quality assurance manager or team member who is attempting to implement defect prevention, or is responsible for establishing or improving an organization's software process
- A consultant hired to improve project management practices, software process, or overall software quality in an organization
- A project manager who has been put in charge of a project that has been outsourced

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/appliedprojectmgmt

We also have a companion site featuring additional information for project managers, trainers, educators, and practitioners here:

http://www.stellman-greene.com

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Safari Enabled

Acknowledgments

This book would not have been possible without the efforts of many people. First and foremost, we would like to thank our friends at O'Reilly Media: Andrew Odewahn, Mike Hendrickson, and Andy Oram. They have provided us with invaluable guidance and support throughout the entire process. And special thanks to Mary O'Brien, for guiding us from the first draft to the completed version that you are holding.

We are indebted to our reviewers for the time and effort they put into helping this book be successful. Without them, this book would contain many factual and conceptual errors. So, thanks to:

- Scott Berkun
- Gnaneshwar Dayanand
- Oleg Fishel
- Tarun Ganguly
- Sam Kass
- Dave Murdock
- Neil Potter
- Eric Renkey
- Virginia Smith
- Chris Winters
- Heather Day
- Matt Doar
- Karl Fogel
- Faisal Jawdat
- Marc Kellner
- Jayne Oh
- DVS Raju
- Rob Rothman
- Lana Sze

We would also like to thank our friends at Carnegie Mellon University: Sandra Slaughter at the Tepper School of Business and Anthony Lattanza at the School of Computer Science, who helped us with our initial proposal.

And special thanks go to Lisa Kellner, without whom this would be a much more difficult book to read, and to Nisha Sondhe for taking the photographs that capture each chapter's main concepts (except for Chapter 7, which was taken by Steven Stellman. Thanks Dad!).

Andrew would like to thank Mark Stehlik, Catherine Copetas, Klaus Sutner, and Robert Harper at the CMU School of Computer Science. Jenny would like to thank Robert Horton, who exposed her to many of the ideas that later became the practices in this book. And we would both like to thank Jim Over, Tim Olson, and the rest of the folks from the Software Engineering Institute who have answered our questions over the years.

And finally, we would like to thank our families and friends for putting up with our incessant babble about all of the things you're about to read.

Good luck!

—Andrew Stellman and Jennifer Greene
November, 2005

TABLE OF CONTENTS

CHAPTER ONE

Introduction

SAY A PROJECT THAT STARTED OUT AS A SMALL, STOPGAP UTILITY has turned into a raging behemoth, sucking seemingly unlimited time from your programmers. Or the president of your company announced that your project will be done this week, even though you know that it still has an enormous number of bugs. Or your team delivered the software, only to have users complain that an entire feature is missing. Or every time the team fixes a bug, they seem to uncover a dozen more—including ones that you *know* were fixed six months ago. If you are a software project manager, you may recognize these problems (or similar ones) from your own career.

Many software organizations have problems delivering quality software that is finished on time and meets the users' needs. Luckily, most software project problems have surprisingly few root causes, and these causes are well understood. Solutions to these problems have been discovered, explained, and tested in thousands of software organizations around the world. These solutions are generally straightforward and easy to implement. However, they are not always intuitive to people who do not understand project management, and that makes them difficult to introduce. The goal of this book is to teach you about these solutions and help you integrate them into your own organization.

But this book is about more than just solutions to typical project problems. Every single technique, practice, and tool also helps establish an environment of trust, openness, and honesty among the project team, the management of the organization, and the people who will use or benefit from the software. By sharing all of your project information, both your team and your managers can understand your decisions, and they can see exactly why you made them.

It's easy to forget that project management is more than just a technical engineering skill. Good project management really boils down to a few basic principles that, if you keep them in mind, will help guide you through any software project:

- Make sure all decisions are based on openly shared information.
- Don't second-guess your team members' expertise.
- Introduce software quality from the very beginning of the project.
- Don't impose an artificial hierarchy on the project team.
- Remember that the fastest way through the project is to use good engineering practices.

A project manager needs to understand every facet of software development in order to make good judgements. You don't need to be a programmer, software tester, requirements analyst, or architect in order to be a good project manager. But you do need to know what these people do, why they are on your team, the common pitfalls they succumb to, and how to fix them. You need to be able to read and understand the documents that they create and provide intelligent feedback. And by relying on objective analysis (rather than gut feelings, personal preferences, or a perceived hierarchy within your team), you can use this knowledge in order to make decisions based on the best interests of the project.

Tell Everyone the Truth All the Time

The most important principle in this book is transparency. A project manager constantly makes decisions about the project. If those decisions are based on real information that's gathered by the team and trusted by management, that's the most likely way to make sure the project succeeds. Creating a transparent environment means making all of that information public and explaining the rationale behind your decisions. No software project goes exactly as planned; the only way to deal with obstacles is by sharing the true nature of each problem with everyone involved in the project and by allowing the best solution to come from the most qualified people.

But while anyone would agree with this in principle, it's much harder to keep yourself and your project honest in practice. Say you're a project manager, and your project is running late. What do you do if your boss—much to your surprise—announces to the world that your project will be done on time? Unfortunately, when faced with this situation, most project managers try to change reality rather than deal with the truth. It's not hard to see why that approach is appealing. Most people in software engineering are very positive, and it's not hard to convince them that an unrealistic deadline is just another technical challenge to be met. But the passage of time is not a technical challenge, and if the expectations are unrealistic, then even the most talented team will fail to meet them. The only real solution to this problem is to be open and honest about the real status of the project—and that's going make your boss unhappy.

And so, instead of telling the truth, many project managers faced with a deadline that's clearly unrealistic will put pressure on their team to work late and make up the time. They silently trim the scope, gut quality tasks, start eliminating reviews, inspections, and pretty much any documentation, and just stop updating the schedule entirely. And, above all, they wait until the very last minute to tell everyone that the project is late.., hoping against hope that luck, long hours, and a whole lot of coffee will correct the situation.

And sometimes it works... sort of, until the users have to work around bugs or missing features, until programmers have to start patching the software, and until managers have to go on a charm offensive in order to smooth over rough relations among everyone involved. Even if the deadline was met, the software was clearly not *really* ready for release. (And that's assuming the team even managed to squeeze it out on time!)

That's why the most important part of building better software is establishing transparency. It's about making sure that, from the very beginning of the project, everyone agrees on what needs to be built, how long it will take to build it, what steps will be taken in order to complete the project, and how they will know that it's been done properly. Every tool, technique, and practice in this book is based on the principles of freely sharing information and keeping the entire project team "in the loop" on every important decision.

Trust Your Team

If you are a project manager, it's your job to be responsible for the project. That means that you have to do whatever it takes to get the project out the door. But it does not necessarily mean that you know more about the project than everyone else on the team. Yet many project managers act in exactly this way. They arbitrarily cut down or inflate any estimates that they don't understand, or that give them a bad gut feeling. They base their schedules on numbers that they simply made up. And, most importantly, they make every project decision based on how it will affect the schedule, instead of considering how it will affect the software.

Managing a project is all about forming a team and making sure that it is productive. The best way to do that is to rely on the expertise of the team members. Any project manager who tries to micromanage his team will immediately get overwhelmed, and the project will come screeching to a halt.

Every single role in a software project requires expertise, skill, training, and experience. There is no way that one person can fill all of those different roles—she would need several lifetimes just to gain enough knowledge of each discipline! Luckily, nobody has to do it alone: that's why you have a team full of qualified people. It's up to them to recommend the best course of action at each stage of the project; it's up to you to make informed decisions based on their recommendations.

If you don't have a good reason to veto an idea, don't do it. Usually, the people on your team know best what it takes to do their job. The most important thing that you can do to support them is to trust them and listen to them.

However, you cannot blindly trust your team. You need to evaluate their ideas in relation to solid engineering principles. This is why the first part of this book goes beyond "traditional" project management (creating project plans, developing estimates, building schedules, etc.) to cover *all* parts of a software project. Every project manager needs to understand at least the basic principles of software requirements engineering, design and architecture, programming, and software testing in order to guide a software project through all of the phases of development.

Review Everything, Test Everything

Reviews get a bad rap. Many people see them as frivolous. "In a perfect world," many project managers say, "we would review all of our documents. But we live in the real world, and we just don't have time." Others feel that the only reason for a review is to force various people to sign off on a document—as if a signature on a page somehow guarantees that they agree with everything that it's stapled to.

The purpose of a review is not to make a perfect document or to generate a page of signatures. Rather, a review does two things: it prevents defects in the software and it helps the project manager gain a real, informed commitment from the team. What's more, it's important to recognize that no review is perfect—and that's just fine. It may not be possible to catch 100% of the defects before coding has started, but a good review will catch enough defects to more than pay for the time it took to hold the review.

It is *always* faster and cheaper to hold a review meeting than it is to skip it, simply because it's much easier to fix something on paper than it is to build it first and fix it later. When a review turns up an error that takes a few minutes to fix in a document, it saves the team the hours, days, or weeks that it would take to fix the error once it has been built into the software. But even more importantly, reviews frequently uncover errors in documents whose resolution requires a lot of discussion and decision-making. Errors like this can completely destroy a software project if they make it all the way into the code.

Many project managers try to schedule reviews, only to meet with an enormous amount of resistance from everyone around them. Peers, project team members, and senior managers all seem to resist the idea of "yet another meeting." Oddly, the same project managers who are unable to scrape together an hour and a half to review a scope document at the beginning of the project generally have no difficulty whatsoever scheduling lengthy, tedious weekly status meetings with no agenda, goal, or purpose. (Of course, not all project status meetings have to be run that way!)

The truth is that there is no such thing as wasted time in a review. On average, every hour spent reviewing and repairing documents saves well over an hour later in the project. This is true because it catches costly errors early on in the project, when they are cheap to fix. But reviews have other valuable benefits as well. By bringing team members into a room to evaluate each others' work, reviews foster respect among the team members for everyone's contributions. And, most importantly, a review results in a real commitment to the work that is produced by the team, not just a signature.

Testing—whether it is unit testing, functional testing, performance testing or regression testing—is just as important, and just as likely to be dismissed as unaffordable or "idealistic." Yet software testing activities are just as cost-effective as reviews. The team can't "test in" quality at the end of a project just by tacking on some testing tasks. Testing must be planned from the beginning and then supported throughout the entire project. To come up with a quality product, the entire team needs to actively look for defects at every stage, in every document, and in the software. It's the project manager's responsibility to make sure that this happens.

All Software Engineers Are Created Equal

A software project requires much more than just writing code. There are all sorts of work products that are produced along the way: documents, schedules, plans, source code, bug reports, and builds are all created by many different team members. No single work product is more important than any other; if any one of them has a serious error, that error will have an impact on the end product. That means each team member responsible for any of these work products plays an important role in the project, and all of those people can make or break the final build that is delivered to the users.

There are many project managers who, when faced with a disagreement between a programmer and a tester, will always trust the programmer. This same project manager might always trust a requirements analyst or a business analyst over a programmer, if and when they disagree. Many people have some sort of a hierarchy in their heads in which certain engineering team members are more valuable or more skilled than others. This is a dangerous idea, and it is one that has no place on an effective project team.

One key to building better software is treating each idea objectively, no matter who suggests it or whether or not it's immediately intuitive to you. That's another reason the practices, techniques, and tools in this book cover all areas of the software project. Every one of these practices is based on an objective evaluation of all of the important activities in software development. Every discipline is equally important, and everyone on the team contributes equally to the project. A software requirements specification (SRS), for example, is no more important than the code: the code could not be created without the SRS, and the SRS would have no purpose if it weren't the basis of the software. It is in the best interest of everyone on the team to make sure that both of them have as few defects as possible, and that the authors of both work products have equal say in project decisions.

The project manager must respect all team members, and should not second-guess their expertise. This is an important principle because it is the basis of real commitments. It's easy for a senior manager to simply issue an edict that everyone must build software without defects and do a good job; however, this rarely works well in practice. The best way to make sure that the software is built well is to make sure that everyone on the team feels respected and valued, and to gain a true commitment from each person to make the software the best it can be.

Doing the Project Right Is Most Efficient

The first part of this book is a presentation of techniques, tools, and practices for every phase of a software project. They are designed to be implemented one at a time and in any order (with a few restrictions). This means that you have a lot of freedom to choose the approach that is best for your project.

But no matter which one you choose to implement first, you can be sure that your project will be better off with the practice than it would be without it. This is because building the software correctly the first time is always preferable to doing it wrong and having to go back and fix it.

Every practice in this book is designed to help you build software efficiently and accurately. What's more, there are many ways to implement every single one of these practices. We put a great deal of effort into developing the most efficient version of each tool, technique, or practice we present in this book. We did this by stripping out as many of the "bells and whistles" as possible from each practice without compromising its basic integrity.

There are more complex and involved ways to implement every single idea in this book. Wherever possible, there are references that will point you to more in-depth reading that