Book Translation

Original Language: English

Target Language: Persian

Total Sections: 159

Translation Engine: Avalai Service Provider

Model: gpt-4o

Endpoint: https://api.avalai.ir/v1

Page Overlap: Enabled (for better continuity)

Generated on: 2025-08-21 11:59:00

# Section 1

## Original Text:

Early praise for The Nature of Software Development  
This book should be “The CTO’s Guide to Professional Software Develop-  
ment. ” This is a book every CTO, every VP of engineering, every director  
of software, and every software team leader should read. In this book  
they’ll find answers to questions that have plagued their peers for decades. The book is simple and direct, and yet it tackles one of the most complicat-  
ed tasks that humans have ever attempted: managing teams that build  
high-quality software systems. ➤Robert “Uncle Bob” Martin, founder, Object Mentor  
Ditch the buzzword-laden books and read this instead. Ron takes us back  
to development basics with a great summary of a simple development  
process that works. Ron shows you just what’s important in software de-  
velopment. If you’re doing more than this, you’re trying too hard. ➤Jeff Langr, author, Pragmatic Unit Testing in Java 8 with JUnit and  
Modern C++ Programming With Test-Driven Development  
The Nature of Software Development is just like spending a morning with  
Ron, only you don’t have to. ➤Chet Hendrickson  
Agile Teacher and Consultant, HendricksonXP  
I love this book. Every page has a sketch and a clear explanation of  
something you can try out right away. It’s like sitting down with Ron over  
a cup of coffee. ➤Daniel H Steinberg  
Dim Sum Thinking  
In straightforward prose and sketches, Ron explores the deep question of  
how to best deliver software. This book is accessible not just to software  
team members, but to customers and users as well. ➤Bill Wake  
Industrial Logic, Inc. [CONTEXT FROM PREVIOUS PAGE: Early praise for The Nature of Software Development  
This book should be “The CTO’s Guide to Professional Software Develop-  
ment. ” This is a book every CTO, every VP of engineering, every director  
of software, and every software team leader should read. In this book  
they’ll find answers to questions that have plagued their peers for decades.

## Translated Text:

تحسین‌های اولیه برای کتاب The Nature of Software Development   
این کتاب باید به عنوان "راهنمای CTO برای توسعه حرفه‌ای نرم‌افزار" شناخته شود. این کتابی است که هر CTO، هر VP مهندسی، هر مدیر نرم‌افزار و هر رهبر تیم نرم‌افزاری باید آن را بخواند. در این کتاب، پاسخ‌هایی به سوالاتی پیدا خواهند کرد که دهه‌ها ذهن همتایانشان را مشغول کرده است. این کتاب ساده و مستقیم است، اما یکی از پیچیده‌ترین وظایفی که انسان‌ها تاکنون تلاش کرده‌اند را مورد بررسی قرار می‌دهد: مدیریت تیم‌هایی که سیستم‌های نرم‌افزاری با کیفیت بالا می‌سازند. ➤ رابرت "عمو باب" مارتین، بنیان‌گذار Object Mentor   
  
کتاب‌های پر از اصطلاحات بی‌معنی را کنار بگذارید و به جای آن این کتاب را بخوانید. ران ما را به اصول اولیه توسعه بازمی‌گرداند و خلاصه‌ای عالی از یک فرآیند توسعه ساده که کار می‌کند ارائه می‌دهد. ران به شما نشان می‌دهد که چه چیزی در توسعه نرم‌افزار واقعاً مهم است. اگر بیش از این انجام می‌دهید، بیش از حد تلاش می‌کنید. ➤ جف لنگر، نویسنده Pragmatic Unit Testing in Java 8 with JUnit و Modern C++ Programming With Test-Driven Development   
  
کتاب The Nature of Software Development درست مثل گذراندن یک صبح با ران است، فقط نیازی نیست که این کار را انجام دهید. ➤ چت هندریکسون   
مدرس و مشاور Agile، HendricksonXP   
  
من عاشق این کتاب هستم. هر صفحه یک طرح و توضیح واضح از چیزی دارد که می‌توانید بلافاصله امتحان کنید. این کتاب مثل نشستن با ران در کنار یک فنجان قهوه است. ➤ دانیل اچ استاینبرگ   
Dim Sum Thinking   
  
با نثر ساده و طرح‌ها، ران سوال عمیقی را بررسی می‌کند: چگونه بهترین نرم‌افزار را ارائه دهیم. این کتاب نه تنها برای اعضای تیم‌های نرم‌افزاری، بلکه برای مشتریان و کاربران نیز قابل دسترسی است. ➤ بیل ویک   
Industrial Logic, Inc.   
  
[متن از صفحه قبلی: تحسین‌های اولیه برای کتاب The Nature of Software Development   
این کتاب باید به عنوان "راهنمای CTO برای توسعه حرفه‌ای نرم‌افزار" شناخته شود. این کتابی است که هر CTO، هر VP مهندسی، هر مدیر نرم‌افزار و هر رهبر تیم نرم‌افزاری باید آن را بخواند. در این کتاب، پاسخ‌هایی به سوالاتی پیدا خواهند کرد که دهه‌ها ذهن همتایانشان را مشغول کرده است.]

================================================================================

# Section 2

## Original Text:

The book is simple and direct, and yet it tackles one of the most complicat-  
ed tasks that humans have ever attempted: managing teams that build  
high-quality software systems. ➤Robert “Uncle Bob” Martin, founder, Object Mentor  
Ditch the buzzword-laden books and read this instead. Ron takes us back  
to development basics with a great summary of a simple development  
process that works. Ron shows you just what’s important in software de-  
velopment. If you’re doing more than this, you’re trying too hard. ➤Jeff Langr, author, Pragmatic Unit Testing in Java 8 with JUnit and  
Modern C++ Programming With Test-Driven Development  
The Nature of Software Development is just like spending a morning with  
Ron, only you don’t have to. ➤Chet Hendrickson  
Agile Teacher and Consultant, HendricksonXP  
I love this book. Every page has a sketch and a clear explanation of  
something you can try out right away. It’s like sitting down with Ron over  
a cup of coffee. ➤Daniel H Steinberg  
Dim Sum Thinking  
In straightforward prose and sketches, Ron explores the deep question of  
how to best deliver software. This book is accessible not just to software  
team members, but to customers and users as well. ➤Bill Wake  
Industrial Logic, Inc. ]  
  
We've left this page blank to  
make the page numbers the  
same in the electronic and  
paper books. We tried just leaving it out,  
but then people wrote us to  
ask about the missing pages. Anyway, Eddy the Gerbil  
wanted to say “hello. ”  
  
[CONTEXT FROM PREVIOUS PAGE: We've left this page blank to  
make the page numbers the  
same in the electronic and  
paper books. We tried just leaving it out,  
but then people wrote us to  
ask about the missing pages. Anyway, Eddy the Gerbil  
wanted to say “hello.

## Translated Text:

کتاب ساده و مستقیم است، و با این حال یکی از پیچیده‌ترین وظایفی که انسان‌ها تاکنون تلاش کرده‌اند را مورد بررسی قرار می‌دهد: مدیریت تیم‌هایی که سیستم‌های نرم‌افزاری با کیفیت بالا می‌سازند. ➤رابرت "عمو باب" مارتین، بنیان‌گذار، Object Mentor   
کتاب‌های پر از اصطلاحات بی‌معنی را کنار بگذارید و به جای آن این کتاب را بخوانید. ران ما را به اصول اولیه توسعه بازمی‌گرداند با یک خلاصه عالی از یک فرآیند توسعه ساده که کار می‌کند. ران به شما نشان می‌دهد که چه چیزی در توسعه نرم‌افزار مهم است. اگر بیشتر از این انجام می‌دهید، بیش از حد تلاش می‌کنید. ➤جف لنگر، نویسنده Pragmatic Unit Testing in Java 8 with JUnit و Modern C++ Programming With Test-Driven Development   
کتاب The Nature of Software Development درست مانند گذراندن یک صبح با ران است، فقط نیازی نیست که این کار را انجام دهید. ➤چت هندریکسون   
مدرس و مشاور Agile، HendricksonXP   
من عاشق این کتاب هستم. هر صفحه یک طرح و یک توضیح واضح از چیزی دارد که می‌توانید بلافاصله امتحان کنید. این شبیه نشستن با ران بر سر یک فنجان قهوه است. ➤دنیل اچ استاینبرگ   
Dim Sum Thinking   
با نثری ساده و طرح‌هایی روشن، ران به پرسش عمیق درباره بهترین روش ارائه نرم‌افزار می‌پردازد. این کتاب نه تنها برای اعضای تیم نرم‌افزاری، بلکه برای مشتریان و کاربران نیز قابل دسترسی است. ➤بیل ویک   
Industrial Logic, Inc.   
  
ما این صفحه را خالی گذاشته‌ایم تا شماره صفحات در نسخه‌های الکترونیکی و چاپی یکسان باشد. سعی کردیم آن را حذف کنیم،   
اما بعد مردم برای ما نوشتند و درباره صفحات گم‌شده پرسیدند. به هر حال، ادی خرگوش صحرایی می‌خواست بگوید "سلام".   
  
[متن از صفحه قبلی: ما این صفحه را خالی گذاشته‌ایم تا شماره صفحات در نسخه‌های الکترونیکی و چاپی یکسان باشد. سعی کردیم آن را حذف کنیم،   
اما بعد مردم برای ما نوشتند و درباره صفحات گم‌شده پرسیدند. به هر حال، ادی خرگوش صحرایی می‌خواست بگوید "سلام".]

================================================================================

# Section 3

## Original Text:

”]  
  
The Nature of  
Software Development  
Keep It Simple,  
Make It Valuable,  
Build It Piece by Piece  
Ron Jeffries  
The Pragmatic Bookshelf  
Dallas, Texas • Raleigh, North Carolina  
  
[CONTEXT FROM PREVIOUS PAGE: The Nature of  
Software Development  
Keep It Simple,  
Make It Valuable,  
Build It Piece by Piece  
Ron Jeffries  
The Pragmatic Bookshelf  
Dallas, Texas • Raleigh, North Carolina]  
  
Many of the designations used by manufacturers and sellers to distinguish their  
products are claimed as trademarks. Where those designations appear in this book,  
and The Pragmatic Programmers, LLC was aware of a trademark claim, the desig-  
nations have been printed in initial capital letters or in all capitals. The Pragmatic  
Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic  
Bookshelf, PragProg and the linking gdevice are trademarks of The Pragmatic  
Programmers, LLC. Every precaution was taken in the preparation of this book. However, the publisher  
assumes no responsibility for errors or omissions, or for damages that may result  
from the use of information (including program listings) contained herein. Our Pragmatic courses, workshops, and other products can help you and your  
team create better software and have more fun. For more information, as well as  
the latest Pragmatic titles, please visit us at https://pragprog. com. The team that produced this book includes:  
Michael Swaine (editor)  
Potomac Indexing (indexer)  
Liz Welch (copyeditor)  
Dave Thomas (typesetter)  
Janet Furlow (producer)  
Ellie Callahan (support)  
For international rights, please contact rights@pragprog. com. Copyright © 2015 The Pragmatic Programmers, LLC. All rights reserved. No part of this publication may be reproduced, stored  
in a retrieval system, or transmitted, in any form, or by  
any means, electronic, mechanical, photocopying,  
recording, or otherwise, without the prior consent of  
the publisher. Printed in the United States of America.

## Translated Text:

ماهیت   
توسعه نرم‌افزار   
ساده نگه دارید،   
ارزشمندش کنید،   
قطعه به قطعه بسازید   
ران جفریس   
The Pragmatic Bookshelf   
دالاس، تگزاس • رالی، کارولینای شمالی   
  
[متن از صفحه قبلی:   
ماهیت   
توسعه نرم‌افزار   
ساده نگه دارید،   
ارزشمندش کنید،   
قطعه به قطعه بسازید   
ران جفریس   
The Pragmatic Bookshelf   
دالاس، تگزاس • رالی، کارولینای شمالی]   
  
بسیاری از عناوینی که توسط تولیدکنندگان و فروشندگان برای تمایز محصولاتشان استفاده می‌شوند، به‌عنوان علائم تجاری ادعا شده‌اند. هر جا که این عناوین در این کتاب ظاهر شده‌اند و The Pragmatic Programmers, LLC از ادعای علامت تجاری آگاه بوده است، این عناوین با حروف بزرگ یا تمام حروف بزرگ چاپ شده‌اند. The Pragmatic Starter Kit، The Pragmatic Programmer، Pragmatic Programming، Pragmatic Bookshelf، PragProg و دستگاه پیونددهنده علائم تجاری The Pragmatic Programmers, LLC هستند.   
  
تمامی احتیاط‌های لازم در تهیه این کتاب رعایت شده است. با این حال، ناشر هیچ مسئولیتی در قبال خطاها یا حذف‌ها، یا خساراتی که ممکن است از استفاده از اطلاعات (از جمله لیست‌های برنامه) موجود در اینجا ناشی شود، نمی‌پذیرد.   
  
دوره‌ها، کارگاه‌ها و سایر محصولات Pragmatic ما می‌توانند به شما و تیم‌تان کمک کنند تا نرم‌افزار بهتری ایجاد کنید و بیشتر لذت ببرید. برای اطلاعات بیشتر، و همچنین جدیدترین عناوین Pragmatic، لطفاً به https://pragprog.com مراجعه کنید.   
  
تیمی که این کتاب را تولید کرده است شامل:   
مایکل سوین (ویراستار)   
Potomac Indexing (نمایه‌ساز)   
لیز ولچ (ویراستار متنی)   
دیو توماس (حروف‌چین)   
جنت فرلو (تولیدکننده)   
الی کالاهان (پشتیبانی)   
  
برای حقوق بین‌المللی، لطفاً با rights@pragprog.com تماس بگیرید.   
  
کپی‌رایت © 2015 The Pragmatic Programmers, LLC. تمامی حقوق محفوظ است. هیچ بخشی از این انتشار نمی‌تواند بازتولید، در یک سیستم بازیابی ذخیره، یا به هر شکلی، یا با هر وسیله‌ای، الکترونیکی، مکانیکی، فتوکپی، ضبط یا به هر روش دیگری، بدون رضایت قبلی ناشر منتقل شود.   
  
چاپ شده در ایالات متحده آمریکا.

================================================================================

# Section 4

## Original Text:

ISBN-13: 978-1-941222-37-9  
Encoded using the finest acid-free high-entropy binary digits. Book version: P1. 1—March 2015  
  
[CONTEXT FROM PREVIOUS PAGE: Many of the designations used by manufacturers and sellers to distinguish their  
products are claimed as trademarks. Where those designations appear in this book,  
and The Pragmatic Programmers, LLC was aware of a trademark claim, the desig-  
nations have been printed in initial capital letters or in all capitals. The Pragmatic  
Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic  
Bookshelf, PragProg and the linking gdevice are trademarks of The Pragmatic  
Programmers, LLC. Every precaution was taken in the preparation of this book. However, the publisher  
assumes no responsibility for errors or omissions, or for damages that may result  
from the use of information (including program listings) contained herein. Our Pragmatic courses, workshops, and other products can help you and your  
team create better software and have more fun. For more information, as well as  
the latest Pragmatic titles, please visit us at https://pragprog. com. The team that produced this book includes:  
Michael Swaine (editor)  
Potomac Indexing (indexer)  
Liz Welch (copyeditor)  
Dave Thomas (typesetter)  
Janet Furlow (producer)  
Ellie Callahan (support)  
For international rights, please contact rights@pragprog. com. Copyright © 2015 The Pragmatic Programmers, LLC. All rights reserved. No part of this publication may be reproduced, stored  
in a retrieval system, or transmitted, in any form, or by  
any means, electronic, mechanical, photocopying,  
recording, or otherwise, without the prior consent of  
the publisher. Printed in the United States of America. ISBN-13: 978-1-941222-37-9  
Encoded using the finest acid-free high-entropy binary digits. Book version: P1. 1—March 2015]  
  
Preface. v i i i  
Acknowledgments. x  
Introduction. x i i  
Part I — The Circle of Value  
1. The Search for Value. 4  
2. Value Is What We Want. 6  
3.

## Translated Text:

شابک-13: 978-1-941222-37-9   
رمزگذاری شده با استفاده از بهترین ارقام باینری با آنتروپی بالا و بدون اسید. نسخه کتاب: P1. 1—مارس 2015   
  
[زمینه از صفحه قبلی: بسیاری از نام‌هایی که توسط تولیدکنندگان و فروشندگان برای تمایز محصولاتشان استفاده می‌شود، به عنوان علائم تجاری شناخته می‌شوند. در مواردی که این نام‌ها در این کتاب ظاهر شده‌اند و The Pragmatic Programmers, LLC از ادعای علامت تجاری آگاه بوده است، این نام‌ها با حروف بزرگ اولیه یا به طور کامل با حروف بزرگ چاپ شده‌اند. The Pragmatic Starter Kit، The Pragmatic Programmer، Pragmatic Programming، Pragmatic Bookshelf، PragProg و دستگاه لینک‌کننده علائم تجاری The Pragmatic Programmers, LLC هستند. تمام اقدامات احتیاطی در تهیه این کتاب انجام شده است. با این حال، ناشر هیچ مسئولیتی در قبال خطاها یا حذف‌ها، یا خساراتی که ممکن است از استفاده از اطلاعات (از جمله لیست‌های برنامه) موجود در این کتاب ناشی شود، نمی‌پذیرد. دوره‌ها، کارگاه‌ها و سایر محصولات Pragmatic ما می‌توانند به شما و تیم شما کمک کنند تا نرم‌افزار بهتری ایجاد کنید و بیشتر لذت ببرید. برای اطلاعات بیشتر، و همچنین جدیدترین عناوین Pragmatic، لطفاً به https://pragprog.com مراجعه کنید. تیمی که این کتاب را تولید کرده است شامل:   
Michael Swaine (ویراستار)   
Potomac Indexing (نمایه‌ساز)   
Liz Welch (ویراستار نسخه)   
Dave Thomas (حروف‌چین)   
Janet Furlow (تولیدکننده)   
Ellie Callahan (پشتیبانی)   
برای حقوق بین‌المللی، لطفاً با rights@pragprog.com تماس بگیرید.   
کپی‌رایت © 2015 The Pragmatic Programmers, LLC. تمامی حقوق محفوظ است. هیچ بخشی از این انتشار نمی‌تواند بدون رضایت قبلی ناشر بازتولید، در یک سیستم بازیابی ذخیره یا به هر شکلی، یا به هر وسیله‌ای، الکترونیکی، مکانیکی، فتوکپی، ضبط یا غیره منتقل شود. چاپ شده در ایالات متحده آمریکا.   
شابک-13: 978-1-941222-37-9   
رمزگذاری شده با استفاده از بهترین ارقام باینری با آنتروپی بالا و بدون اسید. نسخه کتاب: P1. 1—مارس 2015]   
  
پیشگفتار. viii   
تشکر و قدردانی. x   
مقدمه. xii   
قسمت اول — چرخه ارزش   
1. جستجو برای ارزش. 4   
2. ارزش همان چیزی است که ما می‌خواهیم. 6   
3.

================================================================================

# Section 5

## Original Text:

Guiding Goes Better “Feature by Feature”. 18  
4. Organizing by Feature. 2 6  
5. Planning Feature by Feature. 3 2  
6. Building the Product, Feature by Feature. 42  
7. Build Features and Foundation in Parallel. 50  
8. Bug-Free and Well Designed. 6 0  
9. Full Circle. 7 6  
Part II — Notes and Essays  
10. Value—What Is It. 8 0  
11. Value—How Can We Measure It. 8 4  
12. Of Course It’s Hard. 8 8  
13. Not That Simple. 9 2  
14. Creating Teams That Thrive. 9 4  
15. The “Five-Card Method”for Initial Forecasting. 9 8  
16. Managing Natural Software Development. 100  
17. Whip the Ponies Harder. 1 1 2  
18. To Speed Up, Build with Skill. 1 1 6  
19. Refactoring. 1 2 0  
20. Agile Methods. 1 2 6  
21. Scaling Agile. 1 3 0  
22. Conclusion. 1 4 8  
Bibliography. 1 5 0  
Index. 1 5 3  
  
[CONTEXT FROM PREVIOUS PAGE: Preface. v i i i  
Acknowledgments. x  
Introduction. x i i  
Part I — The Circle of Value  
1. The Search for Value. 4  
2. Value Is What We Want. 6  
3. Guiding Goes Better “Feature by Feature”. 18  
4. Organizing by Feature. 2 6  
5. Planning Feature by Feature. 3 2  
6. Building the Product, Feature by Feature. 42  
7. Build Features and Foundation in Parallel. 50  
8. Bug-Free and Well Designed. 6 0  
9. Full Circle. 7 6  
Part II — Notes and Essays  
10. Value—What Is It. 8 0  
11. Value—How Can We Measure It. 8 4  
12. Of Course It’s Hard. 8 8  
13. Not That Simple. 9 2  
14. Creating Teams That Thrive. 9 4  
15. The “Five-Card Method”for Initial Forecasting. 9 8  
16. Managing Natural Software Development. 100  
17. Whip the Ponies Harder. 1 1 2  
18. To Speed Up, Build with Skill. 1 1 6  
19. Refactoring. 1 2 0  
20. Agile Methods. 1 2 6  
21. Scaling Agile. 1 3 0  
22. Conclusion. 1 4 8  
Bibliography. 1 5 0  
Index. 1 5 3]  
  
Preface  
I’ve been doing software for over a half century. I’ve had  
some great successes and some truly colossal failures. For all that time, I’ve been talking with people, coaching,  
and teaching about software development. And mostly, I’ve  
been thinking.

## Translated Text:

راهنمایی بهتر انجام می‌شود "Feature by Feature". ۱۸   
۴. سازمان‌دهی بر اساس Feature. ۲۶   
۵. برنامه‌ریزی Feature by Feature. ۳۲   
۶. ساخت محصول، Feature by Feature. ۴۲   
۷. ساخت Features و Foundation به صورت موازی. ۵۰   
۸. بدون Bug و طراحی خوب. ۶۰   
۹. چرخه کامل. ۷۶   
  
بخش دوم — یادداشت‌ها و مقالات   
۱۰. ارزش — چیست. ۸۰   
۱۱. ارزش — چگونه می‌توان آن را اندازه‌گیری کرد. ۸۴   
۱۲. البته سخت است. ۸۸   
۱۳. به این سادگی نیست. ۹۲   
۱۴. ایجاد تیم‌هایی که موفق شوند. ۹۴   
۱۵. روش "Five-Card" برای پیش‌بینی اولیه. ۹۸   
۱۶. مدیریت توسعه نرم‌افزار طبیعی. ۱۰۰   
۱۷. سخت‌تر به اسب‌ها فشار بیاورید. ۱۱۲   
۱۸. برای سرعت بخشیدن، با مهارت Build کنید. ۱۱۶   
۱۹. Refactoring. ۱۲۰   
۲۰. Agile Methods. ۱۲۶   
۲۱. Scaling Agile. ۱۳۰   
۲۲. نتیجه‌گیری. ۱۴۸   
کتاب‌شناسی. ۱۵۰   
نمایه. ۱۵۳   
  
[زمینه از صفحه قبلی:   
پیش‌گفتار. viii   
تشکر و قدردانی. x   
مقدمه. xii   
بخش اول — چرخه ارزش   
۱. جستجوی ارزش. ۴   
۲. ارزش همان چیزی است که می‌خواهیم. ۶   
۳. راهنمایی بهتر انجام می‌شود "Feature by Feature". ۱۸   
۴. سازمان‌دهی بر اساس Feature. ۲۶   
۵. برنامه‌ریزی Feature by Feature. ۳۲   
۶. ساخت محصول، Feature by Feature. ۴۲   
۷. ساخت Features و Foundation به صورت موازی. ۵۰   
۸. بدون Bug و طراحی خوب. ۶۰   
۹. چرخه کامل. ۷۶   
  
بخش دوم — یادداشت‌ها و مقالات   
۱۰. ارزش — چیست. ۸۰   
۱۱. ارزش — چگونه می‌توان آن را اندازه‌گیری کرد. ۸۴   
۱۲. البته سخت است. ۸۸   
۱۳. به این سادگی نیست. ۹۲   
۱۴. ایجاد تیم‌هایی که موفق شوند. ۹۴   
۱۵. روش "Five-Card" برای پیش‌بینی اولیه. ۹۸   
۱۶. مدیریت توسعه نرم‌افزار طبیعی. ۱۰۰   
۱۷. سخت‌تر به اسب‌ها فشار بیاورید. ۱۱۲   
۱۸. برای سرعت بخشیدن، با مهارت Build کنید. ۱۱۶   
۱۹. Refactoring. ۱۲۰   
۲۰. Agile Methods. ۱۲۶   
۲۱. Scaling Agile. ۱۳۰   
۲۲. نتیجه‌گیری. ۱۴۸   
کتاب‌شناسی. ۱۵۰   
نمایه. ۱۵۳]   
  
پیش‌گفتار   
من بیش از نیم قرن است که در حوزه نرم‌افزار فعالیت می‌کنم. موفقیت‌های بزرگی داشته‌ام و همچنین شکست‌های واقعاً عظیمی را تجربه کرده‌ام. در تمام این مدت، با مردم صحبت کرده‌ام، مربی‌گری کرده‌ام و در مورد توسعه نرم‌افزار آموزش داده‌ام. و بیشتر از همه، فکر کرده‌ام.

================================================================================

# Section 6

## Original Text:

I’ve been trying to figure out how this can all  
seem so simple and yet be so complex. If you’ve been  
involved in software development, you too have probably  
often felt that all this should be simple, but somehow it gets  
all complicated. Thanks to being in the right place at the right time, I’ve been  
part of the Agile movement since the very beginning. That  
has drawn me back toward simplicity. Like many of the best ideas in software development, mod-  
ern “ Agile” software development offers to make software  
development more productive and better controlled by  
making it simpler. Agile is simple. Four values, a dozen  
principles. How complex could it be. Well, it still seems to  
get pretty darn complex. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Preface  
I’ve been doing software for over a half century. I’ve had  
some great successes and some truly colossal failures. For all that time, I’ve been talking with people, coaching,  
and teaching about software development. And mostly, I’ve  
been thinking. I’ve been trying to figure out how this can all  
seem so simple and yet be so complex. If you’ve been  
involved in software development, you too have probably  
often felt that all this should be simple, but somehow it gets  
all complicated. Thanks to being in the right place at the right time, I’ve been  
part of the Agile movement since the very beginning. That  
has drawn me back toward simplicity. Like many of the best ideas in software development, mod-  
ern “ Agile” software development offers to make software  
development more productive and better controlled by  
making it simpler. Agile is simple. Four values, a dozen  
principles. How complex could it be. Well, it still seems to  
get pretty darn complex. report erratum • discuss]  
  
Agile methods like Scrum and XP are also simple. Again a  
few values, a couple of meetings, a handful of artifacts, how  
complex could they be. And still it gets so complicated so  
quickly. What’s up with that.

## Translated Text:

من سعی کرده‌ام بفهمم چگونه ممکن است همه این‌ها این‌قدر ساده به نظر برسند و در عین حال این‌قدر پیچیده باشند. اگر شما هم درگیر توسعه نرم‌افزار بوده‌اید، احتمالاً بارها احساس کرده‌اید که همه این‌ها باید ساده باشند، اما به نوعی همه چیز پیچیده می‌شود. به لطف حضور در مکان و زمان درست، من از همان ابتدای حرکت Agile بخشی از آن بوده‌ام. این موضوع مرا به سمت سادگی بازگردانده است. مانند بسیاری از بهترین ایده‌ها در توسعه نرم‌افزار، توسعه نرم‌افزار مدرن "Agile" پیشنهاد می‌دهد که توسعه نرم‌افزار را با ساده‌تر کردن آن، پربازده‌تر و بهتر کنترل‌شده کند. Agile ساده است. چهار ارزش، دوازده اصل. چقدر می‌تواند پیچیده باشد؟ خب، به نظر می‌رسد که هنوز هم خیلی پیچیده می‌شود. گزارش خطا • بحث کنید  
  
[متن از صفحه قبلی: مقدمه   
من بیش از نیم قرن است که در زمینه نرم‌افزار فعالیت می‌کنم. موفقیت‌های بزرگی داشته‌ام و شکست‌های واقعاً عظیمی را تجربه کرده‌ام. در تمام این مدت، با مردم صحبت کرده‌ام، مربی‌گری کرده‌ام و درباره توسعه نرم‌افزار آموزش داده‌ام. و بیشتر از همه، فکر کرده‌ام. من سعی کرده‌ام بفهمم چگونه ممکن است همه این‌ها این‌قدر ساده به نظر برسند و در عین حال این‌قدر پیچیده باشند. اگر شما هم درگیر توسعه نرم‌افزار بوده‌اید، احتمالاً بارها احساس کرده‌اید که همه این‌ها باید ساده باشند، اما به نوعی همه چیز پیچیده می‌شود. به لطف حضور در مکان و زمان درست، من از همان ابتدای حرکت Agile بخشی از آن بوده‌ام. این موضوع مرا به سمت سادگی بازگردانده است. مانند بسیاری از بهترین ایده‌ها در توسعه نرم‌افزار، توسعه نرم‌افزار مدرن "Agile" پیشنهاد می‌دهد که توسعه نرم‌افزار را با ساده‌تر کردن آن، پربازده‌تر و بهتر کنترل‌شده کند. Agile ساده است. چهار ارزش، دوازده اصل. چقدر می‌تواند پیچیده باشد؟ خب، به نظر می‌رسد که هنوز هم خیلی پیچیده می‌شود. گزارش خطا • بحث کنید]  
  
روش‌های Agile مانند Scrum و XP نیز ساده هستند. باز هم چند ارزش، چند جلسه، و تعداد کمی artifacts. چقدر می‌توانند پیچیده باشند؟ و با این حال، خیلی سریع خیلی پیچیده می‌شوند. موضوع از چه قرار است؟

================================================================================

# Section 7

## Original Text:

I have begun to see a way of looking at the whole process  
of software development. I’m starting to see a general  
overview that might help us keep things simple. Inside, there  
will still be plenty of complexity, but I hope this high-level  
map will help us pull back and find the simplicity when we  
find ourselves in the weeds. Software development has many facets: determining value,  
managing value flow, organizing around the work, planning,  
building, and so on. Each of these facets needs to focus on  
producing value. Value needs to be visible so that it can be  
guided and managed. For this, we need to step back from  
the details and find the essential simplicity in this very  
complex activity. When I think about things, I draw pictures that focus on  
some aspect of the topic. I try to think of a few words that  
will quickly focus my thinking when next I think about the  
topic. I use pictures to give me a different perspective. Since  
my drawings are perforce simple—I’m not very skilled—I  
use them to cut away complexity and look at what’s left. I’m  
giving you a look at that thinking. This book is an attempt at finding some essential simplicity  
inside the complex activity of building software products. I  
believe I have a handle on some good ideas. At best, this is  
a bit of a clearing along a tangled trail. Please take these  
thoughts and use them to find your own sense of simplicity  
amid all the chaos. Good luck. report erratum • discussPreface • ix  
  
[CONTEXT FROM PREVIOUS PAGE: Agile methods like Scrum and XP are also simple. Again a  
few values, a couple of meetings, a handful of artifacts, how  
complex could they be. And still it gets so complicated so  
quickly. What’s up with that. I have begun to see a way of looking at the whole process  
of software development. I’m starting to see a general  
overview that might help us keep things simple.

## Translated Text:

من شروع به دیدن یک روش برای نگاه کردن به کل فرآیند توسعه نرم‌افزار کرده‌ام. دارم یک دید کلی پیدا می‌کنم که ممکن است به ما کمک کند تا کارها را ساده نگه داریم. درون این فرآیند، همچنان پیچیدگی‌های زیادی وجود خواهد داشت، اما امیدوارم این نقشه در سطح بالا به ما کمک کند تا از جزئیات فاصله بگیریم و سادگی را پیدا کنیم، زمانی که در میان پیچیدگی‌ها گیر می‌افتیم.   
  
توسعه نرم‌افزار جنبه‌های متعددی دارد: تعیین ارزش، مدیریت جریان ارزش، سازماندهی حول کار، برنامه‌ریزی، ساخت و غیره. هر یک از این جنبه‌ها باید بر تولید ارزش تمرکز کنند. ارزش باید قابل مشاهده باشد تا بتوان آن را هدایت و مدیریت کرد. برای این منظور، باید از جزئیات فاصله بگیریم و سادگی اساسی را در این فعالیت بسیار پیچیده پیدا کنیم.   
  
وقتی به موضوعی فکر می‌کنم، تصاویری می‌کشم که بر جنبه‌ای از آن موضوع تمرکز دارند. سعی می‌کنم چند کلمه پیدا کنم که به سرعت تمرکز فکرم را در دفعه بعدی که به آن موضوع فکر می‌کنم، جلب کنند. از تصاویر استفاده می‌کنم تا دیدگاه متفاوتی به دست آورم. از آنجا که نقاشی‌هایم به ناچار ساده هستند—چندان مهارت ندارم—از آن‌ها برای حذف پیچیدگی و نگاه کردن به آنچه باقی می‌ماند استفاده می‌کنم. دارم نگاهی به این نوع تفکر به شما می‌دهم.   
  
این کتاب تلاشی است برای یافتن سادگی اساسی درون فعالیت پیچیده ساخت محصولات نرم‌افزاری. باور دارم که به چند ایده خوب دست یافته‌ام. در بهترین حالت، این کتاب مانند یک پاکسازی کوچک در طول یک مسیر پیچیده است. لطفاً این افکار را بگیرید و از آن‌ها برای یافتن حس سادگی خودتان در میان تمام این هرج و مرج استفاده کنید. موفق باشید.   
  
[متن از صفحه قبلی: روش‌های Agile مانند Scrum و XP نیز ساده هستند. باز هم چند ارزش، چند جلسه، تعداد کمی artifact، چقدر می‌توانند پیچیده باشند؟ و با این حال، خیلی سریع پیچیده می‌شوند. چه چیزی باعث این اتفاق می‌شود؟ من شروع به دیدن یک روش برای نگاه کردن به کل فرآیند توسعه نرم‌افزار کرده‌ام. دارم یک دید کلی پیدا می‌کنم که ممکن است به ما کمک کند تا کارها را ساده نگه داریم.]

================================================================================

# Section 8

## Original Text:

Inside, there  
will still be plenty of complexity, but I hope this high-level  
map will help us pull back and find the simplicity when we  
find ourselves in the weeds. Software development has many facets: determining value,  
managing value flow, organizing around the work, planning,  
building, and so on. Each of these facets needs to focus on  
producing value. Value needs to be visible so that it can be  
guided and managed. For this, we need to step back from  
the details and find the essential simplicity in this very  
complex activity. When I think about things, I draw pictures that focus on  
some aspect of the topic. I try to think of a few words that  
will quickly focus my thinking when next I think about the  
topic. I use pictures to give me a different perspective. Since  
my drawings are perforce simple—I’m not very skilled—I  
use them to cut away complexity and look at what’s left. I’m  
giving you a look at that thinking. This book is an attempt at finding some essential simplicity  
inside the complex activity of building software products. I  
believe I have a handle on some good ideas. At best, this is  
a bit of a clearing along a tangled trail. Please take these  
thoughts and use them to find your own sense of simplicity  
amid all the chaos. Good luck. report erratum • discussPreface • ix]  
  
Acknowledgments  
…where to begin…where to end…  
My parents, for freedom, trust, and a great library…  
Sister Mary Marjorie, for a first taste of science; Mr. Dansky,  
for a first taste of love for a subject; the Jesuits, for showing  
the value of thinking and of course for my fashion sense. Rick Camp, for inviting the kid up the street to be an intern  
at Strategic Air Command; Bill Rogers, for tossing me into  
programming and then helping me learn to swim. Colleagues over the years: Charles Bair, Karen Dueweke,  
Steve Weiss, Gene Somdahl, Rick Evarts, Mike McConnell,  
Jean Musinski, Jeanne Hernandez, Dorothy Lieffers, Don  
Devine…it would take pages to mention everyone who has  
touched me.

## Translated Text:

در درون، همچنان پیچیدگی‌های زیادی وجود خواهد داشت، اما امیدوارم این نقشه در سطح بالا به ما کمک کند تا عقب بکشیم و سادگی را پیدا کنیم وقتی که خود را در میان جزئیات گم کرده‌ایم. توسعه نرم‌افزار جنبه‌های متعددی دارد: تعیین ارزش، مدیریت جریان ارزش، سازماندهی کار، برنامه‌ریزی، Build کردن، و غیره. هر یک از این جنبه‌ها باید بر تولید ارزش تمرکز کنند. ارزش باید قابل مشاهده باشد تا بتوان آن را هدایت و مدیریت کرد. برای این منظور، باید از جزئیات فاصله بگیریم و سادگی اساسی را در این فعالیت بسیار پیچیده پیدا کنیم.  
  
وقتی به مسائل فکر می‌کنم، تصاویری می‌کشم که بر جنبه‌ای از موضوع تمرکز دارند. سعی می‌کنم چند کلمه پیدا کنم که وقتی دوباره به موضوع فکر می‌کنم، به سرعت تمرکزم را جلب کنند. از تصاویر استفاده می‌کنم تا دیدگاه متفاوتی به دست آورم. از آنجا که نقاشی‌هایم به ناچار ساده هستند—چون مهارت زیادی ندارم—از آنها برای حذف پیچیدگی و نگاه کردن به آنچه باقی می‌ماند استفاده می‌کنم. دارم به شما نگاهی به این نوع تفکر می‌دهم.  
  
این کتاب تلاشی است برای یافتن سادگی اساسی درون فعالیت پیچیده ساخت محصولات نرم‌افزاری. باور دارم که به برخی ایده‌های خوب دست یافته‌ام. در بهترین حالت، این کتاب مانند یک پاکسازی در مسیر پر پیچ و خم است. لطفاً این افکار را بگیرید و از آنها برای یافتن حس خود از سادگی در میان تمام این آشفتگی استفاده کنید. موفق باشید.  
  
\*\*Acknowledgments\*\*   
...از کجا شروع کنم...کجا پایان دهم...   
پدر و مادرم، برای آزادی، اعتماد، و یک کتابخانه عالی...   
خواهر مری مارژوری، برای اولین طعم از علم؛ آقای دانسکی، برای اولین طعم از عشق به یک موضوع؛ یسوعیان، برای نشان دادن ارزش تفکر و البته برای حس مد من.   
ریک کمپ، برای دعوت از پسری از خیابان برای کارآموزی در Strategic Air Command؛ بیل راجرز، برای انداختنم به دنیای برنامه‌نویسی و سپس کمک به من برای یادگیری شنا کردن.   
  
همکاران در طول سال‌ها: چارلز بیر، کارن دوویکه، استیو وایس، جین سامدال، ریک اوارتس، مایک مک‌کانل، جین موسینسکی، جین هرناندز، دوروتی لیفرز، دان دوین... ذکر نام همه کسانی که بر من تأثیر گذاشته‌اند، صفحات زیادی می‌طلبد.

================================================================================

# Section 9

## Original Text:

Partners, mentors, colleagues in Agile: Ward Cunningham,  
Kent Beck, Chet Hendrickson, Ann Anderson, Bob Martin,  
Alistair Cockburn, Martin Fowler, Michael Feathers, Bob  
Koss, Brian Button, Brian Marick, Ken Schwaber, Jeff  
Sutherland, Ken Auer…I can’t begin to list all those to whom  
I’m grateful. The Internet and Twittersphere, who are surely tired of  
seeing me try to explain these thoughts so often. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Acknowledgments  
…where to begin…where to end…  
My parents, for freedom, trust, and a great library…  
Sister Mary Marjorie, for a first taste of science; Mr. Dansky,  
for a first taste of love for a subject; the Jesuits, for showing  
the value of thinking and of course for my fashion sense. Rick Camp, for inviting the kid up the street to be an intern  
at Strategic Air Command; Bill Rogers, for tossing me into  
programming and then helping me learn to swim. Colleagues over the years: Charles Bair, Karen Dueweke,  
Steve Weiss, Gene Somdahl, Rick Evarts, Mike McConnell,  
Jean Musinski, Jeanne Hernandez, Dorothy Lieffers, Don  
Devine…it would take pages to mention everyone who has  
touched me. Partners, mentors, colleagues in Agile: Ward Cunningham,  
Kent Beck, Chet Hendrickson, Ann Anderson, Bob Martin,  
Alistair Cockburn, Martin Fowler, Michael Feathers, Bob  
Koss, Brian Button, Brian Marick, Ken Schwaber, Jeff  
Sutherland, Ken Auer…I can’t begin to list all those to whom  
I’m grateful. The Internet and Twittersphere, who are surely tired of  
seeing me try to explain these thoughts so often. report erratum • discuss]  
  
Helpers with this book: Bill Tozier, Laura Fisher, and of  
course Chet Hendrickson, who has listened to and shaped  
every word. Any remaining errors are of course his fault.

## Translated Text:

شرکا، مربیان، همکاران در Agile: وارد کانینگهام،   
کنت بک، چت هندریکسون، آن اندرسون، باب مارتین،   
الیستر کاکبرن، مارتین فاولر، مایکل فدرز، باب   
کاس، برایان باتن، برایان ماریک، کن شوابر، جف   
ساترلند، کن آئور... نمی‌توانم شروع به فهرست کردن همه کسانی کنم که به آنها مدیونم.   
اینترنت و توییتر، که مطمئناً از دیدن تلاش‌های مکرر من برای توضیح این افکار خسته شده‌اند.   
report erratum • discuss   
  
[متن از صفحه قبلی: تقدیر و تشکر   
...از کجا شروع کنم...کجا پایان دهم...   
پدر و مادرم، برای آزادی، اعتماد، و یک کتابخانه عالی...   
خواهر مری مارژوری، برای اولین طعم علم؛ آقای دانسکی،   
برای اولین طعم عشق به یک موضوع؛ یسوعیان، برای نشان دادن   
ارزش تفکر و البته برای حس مد من. ریک کمپ، برای دعوت از بچه‌ای که در خیابان زندگی می‌کرد تا کارآموز   
در فرماندهی استراتژیک هوایی شود؛ بیل راجرز، برای انداختن من در   
برنامه‌نویسی و سپس کمک به یادگیری شنا. همکاران در طول سال‌ها: چارلز بیر، کارن دوئک،   
استیو وایس، جین سامدال، ریک اوارتس، مایک مک‌کانل،   
جین موسینسکی، جین هرناندز، دوروتی لیفرز، دان   
دیواین... صفحات زیادی لازم است تا همه کسانی را که بر من تأثیر گذاشته‌اند ذکر کنم.   
شرکا، مربیان، همکاران در Agile: وارد کانینگهام،   
کنت بک، چت هندریکسون، آن اندرسون، باب مارتین،   
الیستر کاکبرن، مارتین فاولر، مایکل فدرز، باب   
کاس، برایان باتن، برایان ماریک، کن شوابر، جف   
ساترلند، کن آئور... نمی‌توانم شروع به فهرست کردن همه کسانی کنم که به آنها مدیونم.   
اینترنت و توییتر، که مطمئناً از دیدن تلاش‌های مکرر من برای توضیح این افکار خسته شده‌اند.   
report erratum • discuss]   
  
کمک‌کنندگان به این کتاب: بیل توزیر، لورا فیشر، و البته چت هندریکسون، که هر کلمه را شنیده و شکل داده است.   
هر خطای باقی‌مانده البته تقصیر اوست.

================================================================================

# Section 10

## Original Text:

The wonderful folks at The Pragmatic Programmers: Andy  
Hunt and Dave Thomas; Susannah Pfalzer, who knew when  
to manage me and when to stand back; Janet Furlow, who  
pushed the book through production; my patient and long-  
suffering editor, Mike Swaine—without these people there’d  
be no book. My “boys,” Ron and Mike, of whom I am most proud and  
who have filled my life with joy and events of interest. And more than all of these, Ricia, my wife: without her  
nothing would seem worth doing. Thank you for taking care  
of me. Thanks. report erratum • discussAcknowledgments • xi  
  
[CONTEXT FROM PREVIOUS PAGE: Helpers with this book: Bill Tozier, Laura Fisher, and of  
course Chet Hendrickson, who has listened to and shaped  
every word. Any remaining errors are of course his fault. The wonderful folks at The Pragmatic Programmers: Andy  
Hunt and Dave Thomas; Susannah Pfalzer, who knew when  
to manage me and when to stand back; Janet Furlow, who  
pushed the book through production; my patient and long-  
suffering editor, Mike Swaine—without these people there’d  
be no book. My “boys,” Ron and Mike, of whom I am most proud and  
who have filled my life with joy and events of interest. And more than all of these, Ricia, my wife: without her  
nothing would seem worth doing. Thank you for taking care  
of me. Thanks. report erratum • discussAcknowledgments • xi]  
  
Software is Lava  
Introduction  
Kids often play a game: The floor is lava. In this game, you  
have to get from one place to another without touching the  
floor. Because the floor is lava. If you step in lava, you die,  
horribly, screaming. Don’t step in lava. So, in the game, you  
must jump from the couch to the chair, crawl across the table,  
and leap to safety in the kitchen, where the floor is not lava. Software is lava. Often it seems that there’s no safe place to  
step. Worse yet, we’re not allowed to jump on the furniture. Mom said. Sorry. So what are we to do. As we build software, it seems that  
we’re stepping in lava every day.

## Translated Text:

افراد فوق‌العاده در The Pragmatic Programmers: اندی هانت و دیو توماس؛ سوزانا فالزر، که می‌دانست چه زمانی من را مدیریت کند و چه زمانی کنار بایستد؛ جنت فرلو، که کتاب را از مراحل تولید عبور داد؛ ویرایشگر صبور و رنج‌کشیده من، مایک سوین—بدون این افراد، این کتاب وجود نداشت. "پسران" من، ران و مایک، که به آن‌ها بسیار افتخار می‌کنم و زندگی‌ام را با شادی و اتفاقات جالب پر کرده‌اند. و بیشتر از همه این‌ها، ریشیا، همسرم: بدون او هیچ چیزی ارزش انجام دادن نداشت. ممنون که از من مراقبت کردی. متشکرم.   
\*\*report erratum • discussAcknowledgments • xi\*\*  
  
[متن از صفحه قبلی: کمک‌کنندگان به این کتاب: بیل توزیه، لورا فیشر، و البته چت هندریکسون، که به هر کلمه گوش داد و آن را شکل داد. هر اشتباهی که باقی مانده، البته تقصیر اوست. افراد فوق‌العاده در The Pragmatic Programmers: اندی هانت و دیو توماس؛ سوزانا فالزر، که می‌دانست چه زمانی من را مدیریت کند و چه زمانی کنار بایستد؛ جنت فرلو، که کتاب را از مراحل تولید عبور داد؛ ویرایشگر صبور و رنج‌کشیده من، مایک سوین—بدون این افراد، این کتاب وجود نداشت. "پسران" من، ران و مایک، که به آن‌ها بسیار افتخار می‌کنم و زندگی‌ام را با شادی و اتفاقات جالب پر کرده‌اند. و بیشتر از همه این‌ها، ریشیا، همسرم: بدون او هیچ چیزی ارزش انجام دادن نداشت. ممنون که از من مراقبت کردی. متشکرم.   
\*\*report erratum • discussAcknowledgments • xi\*\*]  
  
\*\*نرم‌افزار مانند گدازه است\*\*   
\*\*مقدمه\*\*   
بچه‌ها اغلب یک بازی انجام می‌دهند: زمین گدازه است. در این بازی، شما باید از یک مکان به مکان دیگر بروید بدون اینکه زمین را لمس کنید. چون زمین گدازه است. اگر روی گدازه پا بگذارید، به طرز وحشتناکی می‌میرید، با فریاد. روی گدازه پا نگذارید. بنابراین، در این بازی، شما باید از روی مبل به صندلی بپرید، روی میز بخزید و به آشپزخانه که زمین آن گدازه نیست، به سلامت برسید.   
  
نرم‌افزار مانند گدازه است. اغلب به نظر می‌رسد هیچ جای امنی برای قدم گذاشتن وجود ندارد. بدتر از آن، ما اجازه نداریم روی مبلمان بپریم. مامان گفت. متأسفم. پس چه باید بکنیم؟ وقتی نرم‌افزار می‌سازیم، به نظر می‌رسد که هر روز روی گدازه قدم می‌گذاریم.

================================================================================

# Section 11

## Original Text:

It’s complicated, it gets  
more complicated, and often it seems that we’re just doomed. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Software is Lava  
Introduction  
Kids often play a game: The floor is lava. In this game, you  
have to get from one place to another without touching the  
floor. Because the floor is lava. If you step in lava, you die,  
horribly, screaming. Don’t step in lava. So, in the game, you  
must jump from the couch to the chair, crawl across the table,  
and leap to safety in the kitchen, where the floor is not lava. Software is lava. Often it seems that there’s no safe place to  
step. Worse yet, we’re not allowed to jump on the furniture. Mom said. Sorry. So what are we to do. As we build software, it seems that  
we’re stepping in lava every day. It’s complicated, it gets  
more complicated, and often it seems that we’re just doomed. report erratum • discuss]  
  
There has to be a better way. We all feel it. We’re all sure that there must be a way to build  
software that isn’t lava. We didn’t get there last time, but  
next time… next time …we’ll get it right. And, sure enough, next time, more lava. Ow. Die screaming. Yet most of us have had moments when our feet weren’t  
burning. There seem to be cool, grassy patches amid the  
lava. Sometimes we find them. It feels so good to be there. The premise of this book is that there aren’t just patches of  
grass—there is a cool, green, grassy path. Maybe we can’t  
be on that path every moment, but understanding the path  
better is the way to a happier project. I call that path “the Natural Way,” because I believe that the  
path is built into a simple notion, a focus on delivering value  
early and often. report erratum • discussIntroduction • xiii  
  
[CONTEXT FROM PREVIOUS PAGE: There has to be a better way. We all feel it. We’re all sure that there must be a way to build  
software that isn’t lava. We didn’t get there last time, but  
next time… next time …we’ll get it right. And, sure enough, next time, more lava.

## Translated Text:

این پیچیده است، پیچیده‌تر می‌شود، و اغلب به نظر می‌رسد که ما محکوم به شکست هستیم. گزارش اشتباه • بحث کنید  
  
[متن قبلی: نرم‌افزار مثل گدازه است   
مقدمه   
بچه‌ها اغلب یک بازی انجام می‌دهند: زمین گدازه است. در این بازی، شما باید از یک نقطه به نقطه دیگر بروید بدون اینکه زمین را لمس کنید. چون زمین گدازه است. اگر روی گدازه پا بگذارید، به طرز وحشتناکی می‌میرید، فریادزنان. روی گدازه پا نگذارید. بنابراین، در این بازی، باید از روی مبل به صندلی بپرید، روی میز بخزید و به آشپزخانه که زمین آن گدازه نیست، بپرید. نرم‌افزار مثل گدازه است. اغلب به نظر می‌رسد که هیچ جای امنی برای پا گذاشتن وجود ندارد. بدتر از آن، ما اجازه نداریم روی مبلمان بپریم. مامان گفت. متأسفم. پس چه باید بکنیم؟ وقتی نرم‌افزار می‌سازیم، به نظر می‌رسد که هر روز روی گدازه پا می‌گذاریم. این پیچیده است، پیچیده‌تر می‌شود، و اغلب به نظر می‌رسد که ما محکوم به شکست هستیم. گزارش اشتباه • بحث کنید]  
  
باید راه بهتری وجود داشته باشد. همه ما این را حس می‌کنیم. همه ما مطمئن هستیم که باید راهی برای ساخت نرم‌افزاری وجود داشته باشد که مثل گدازه نباشد. دفعه قبل به آن نرسیدیم، اما دفعه بعد... دفعه بعد... درستش می‌کنیم. و البته، دفعه بعد، باز هم گدازه. آخ. فریادزنان بمیر. با این حال، بیشتر ما لحظاتی داشته‌ایم که پاهایمان نمی‌سوخت. به نظر می‌رسد که در میان گدازه‌ها، تکه‌های خنک و سبز چمن وجود دارد. گاهی آن‌ها را پیدا می‌کنیم. بودن در آنجا حس خیلی خوبی دارد. فرضیه این کتاب این است که فقط تکه‌های چمن نیستند—یک مسیر خنک، سبز و چمنی وجود دارد. شاید نتوانیم هر لحظه روی آن مسیر باشیم، اما درک بهتر آن مسیر راهی به سوی پروژه‌ای شادتر است. من آن مسیر را «راه طبیعی» می‌نامم، چون معتقدم که آن مسیر بر اساس یک مفهوم ساده ساخته شده است، تمرکز بر ارائه ارزش زودهنگام و مکرر. گزارش اشتباه • بحث کنید مقدمه • xiii

================================================================================

# Section 12

## Original Text:

Ow. Die screaming. Yet most of us have had moments when our feet weren’t  
burning. There seem to be cool, grassy patches amid the  
lava. Sometimes we find them. It feels so good to be there. The premise of this book is that there aren’t just patches of  
grass—there is a cool, green, grassy path. Maybe we can’t  
be on that path every moment, but understanding the path  
better is the way to a happier project. I call that path “the Natural Way,” because I believe that the  
path is built into a simple notion, a focus on delivering value  
early and often. report erratum • discussIntroduction • xiii]  
  
We will wander off the path. Even though we much prefer to be on the grass than in the  
lava, it seems that we always get in the lava. (Sometimes  
lava is spelled differently. Anyway, we’re in it. )  
If there is a path—and I hope to show you that there is—we  
will wander off of it. Yes, we will. So as I describe the path  
to you, don’t imagine that I believe we’ll all be on the path  
and live happily ever after with no problems, with our  
grateful feet caressing the happy grasses of the path. We  
couldn’t be that good, or that lucky. What we can do is remain aware that there is a path. When  
we’re not on the path, we’ll think about value. We’ll think  
about the Natural Way. And quite likely we’ll be able to find  
our way back, if not to the grass, at least to a place where  
the lava isn’t quite so hot. Introduction • xiv  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: We will wander off the path. Even though we much prefer to be on the grass than in the  
lava, it seems that we always get in the lava. (Sometimes  
lava is spelled differently. Anyway, we’re in it. )  
If there is a path—and I hope to show you that there is—we  
will wander off of it. Yes, we will. So as I describe the path  
to you, don’t imagine that I believe we’ll all be on the path  
and live happily ever after with no problems, with our  
grateful feet caressing the happy grasses of the path.

## Translated Text:

آخ. با فریاد بمیر. با این حال، اکثر ما لحظاتی را تجربه کرده‌ایم که پاهایمان نمی‌سوخت. به نظر می‌رسد که در میان گدازه‌ها، تکه‌های خنک و پوشیده از چمن وجود دارد. گاهی اوقات آن‌ها را پیدا می‌کنیم. بودن در آنجا حس فوق‌العاده‌ای دارد. فرضیه این کتاب این است که فقط تکه‌های چمن وجود ندارد—بلکه یک مسیر خنک، سبز و پوشیده از چمن وجود دارد. شاید نتوانیم همیشه روی آن مسیر باشیم، اما درک بهتر آن مسیر راهی به سوی یک پروژه شادتر است. من آن مسیر را «راه طبیعی» می‌نامم، زیرا معتقدم که این مسیر بر اساس یک مفهوم ساده ساخته شده است: تمرکز بر ارائه ارزش زودهنگام و مکرر.   
report erratum • discuss   
Introduction • xiii   
  
ما از مسیر خارج خواهیم شد. حتی اگر ترجیح دهیم روی چمن باشیم تا در گدازه، به نظر می‌رسد که همیشه وارد گدازه می‌شویم. (گاهی اوقات گدازه به شکل دیگری نوشته می‌شود. به هر حال، ما در آن هستیم.)   
اگر مسیری وجود داشته باشد—و امیدوارم نشان دهم که وجود دارد—ما از آن مسیر خارج خواهیم شد. بله، خواهیم شد. بنابراین وقتی مسیر را برای شما توصیف می‌کنم، تصور نکنید که من باور دارم همه ما همیشه روی مسیر خواهیم بود و تا ابد بدون مشکل زندگی خواهیم کرد، با پاهای سپاسگزارمان که چمن‌های شاد مسیر را نوازش می‌کنند. ما نمی‌توانیم آنقدر خوب یا آنقدر خوش‌شانس باشیم.   
کاری که می‌توانیم انجام دهیم این است که آگاه باشیم که مسیری وجود دارد. وقتی روی مسیر نیستیم، به ارزش فکر خواهیم کرد. به راه طبیعی فکر خواهیم کرد. و به احتمال زیاد می‌توانیم راه خود را پیدا کنیم، اگر نه به چمن، حداقل به جایی که گدازه آنقدر داغ نباشد.   
Introduction • xiv   
report erratum • discuss

================================================================================

# Section 13

## Original Text:

We  
couldn’t be that good, or that lucky. What we can do is remain aware that there is a path. When  
we’re not on the path, we’ll think about value. We’ll think  
about the Natural Way. And quite likely we’ll be able to find  
our way back, if not to the grass, at least to a place where  
the lava isn’t quite so hot. Introduction • xiv  
report erratum • discuss]  
  
The Natural Way  
The story in this book is a simple one: there is a Natural Way  
to build software, and it serves everyone well. The Natural Way serves end users well because it delivers  
value to them sooner. The Natural Way serves the business well because it provides  
a return on investment sooner, because it provides important  
information quickly, and because it provides the ability to  
adjust direction as needed. The Natural Way serves management well too. It lets man-  
agement see what’s really going on inside the project so that  
when action is needed, there will be time to act. And it  
reduces management’s problems by making information  
visible so that we don’t have to dig for it. The Natural Way even makes the job easier for developers. It provides them with clear direction and allows them free-  
dom to use their skills to build what the organization needs,  
when it’s needed. report erratum • discussIntroduction • xv  
  
[CONTEXT FROM PREVIOUS PAGE: The Natural Way  
The story in this book is a simple one: there is a Natural Way  
to build software, and it serves everyone well. The Natural Way serves end users well because it delivers  
value to them sooner. The Natural Way serves the business well because it provides  
a return on investment sooner, because it provides important  
information quickly, and because it provides the ability to  
adjust direction as needed. The Natural Way serves management well too. It lets man-  
agement see what’s really going on inside the project so that  
when action is needed, there will be time to act.

## Translated Text:

ما نمی‌توانیم آن‌قدر خوب یا آن‌قدر خوش‌شانس باشیم. کاری که می‌توانیم انجام دهیم این است که آگاه باشیم که مسیری وجود دارد. وقتی که در مسیر نیستیم، به ارزش فکر خواهیم کرد. به Natural Way فکر خواهیم کرد. و به احتمال زیاد می‌توانیم راه خود را پیدا کنیم، اگر نه به سمت چمنزار، حداقل به جایی که گدازه‌ها آن‌قدر داغ نباشند.   
مقدمه • xiv   
report erratum • discuss   
  
### Natural Way   
داستان این کتاب ساده است: یک Natural Way برای ساخت نرم‌افزار وجود دارد که به نفع همه است. Natural Way به نفع کاربران نهایی است زیرا ارزش را زودتر به آن‌ها ارائه می‌دهد. Natural Way به نفع کسب‌وکار است زیرا بازگشت سرمایه را زودتر فراهم می‌کند، اطلاعات مهم را سریع‌تر ارائه می‌دهد و امکان تنظیم مسیر را در صورت نیاز فراهم می‌سازد. Natural Way به نفع مدیریت نیز هست. این روش به مدیریت اجازه می‌دهد که ببیند واقعاً چه چیزی در پروژه در حال وقوع است تا وقتی که اقدامی لازم باشد، زمان کافی برای عمل وجود داشته باشد. همچنین مشکلات مدیریت را با قابل مشاهده کردن اطلاعات کاهش می‌دهد تا نیازی به جستجوی عمیق نباشد.   
  
حتی Natural Way کار را برای توسعه‌دهندگان آسان‌تر می‌کند. این روش به آن‌ها جهت‌گیری واضحی ارائه می‌دهد و به آن‌ها آزادی می‌دهد تا از مهارت‌های خود برای ساخت آنچه سازمان نیاز دارد، در زمان مورد نیاز استفاده کنند.   
report erratum • discuss   
مقدمه • xv   
  
[CONTEXT FROM PREVIOUS PAGE: Natural Way   
داستان این کتاب ساده است: یک Natural Way برای ساخت نرم‌افزار وجود دارد که به نفع همه است. Natural Way به نفع کاربران نهایی است زیرا ارزش را زودتر به آن‌ها ارائه می‌دهد. Natural Way به نفع کسب‌وکار است زیرا بازگشت سرمایه را زودتر فراهم می‌کند، اطلاعات مهم را سریع‌تر ارائه می‌دهد و امکان تنظیم مسیر را در صورت نیاز فراهم می‌سازد. Natural Way به نفع مدیریت نیز هست. این روش به مدیریت اجازه می‌دهد که ببیند واقعاً چه چیزی در پروژه در حال وقوع است تا وقتی که اقدامی لازم باشد، زمان کافی برای عمل وجود داشته باشد.]

================================================================================

# Section 14

## Original Text:

And it  
reduces management’s problems by making information  
visible so that we don’t have to dig for it. The Natural Way even makes the job easier for developers. It provides them with clear direction and allows them free-  
dom to use their skills to build what the organization needs,  
when it’s needed. report erratum • discussIntroduction • xv]  
  
What is described here is simple—but it’s not easy. You’ll  
need to think about these ideas, to figure out how they’ll be  
valuable to you, and to learn to do the things we explore  
here. Keep moving toward simplicity. You’ll be glad you  
did. The Natural Way does require us to think, to learn, and to  
change a bit. I think you’ll see here that moving toward the  
Natural Way need not be traumatic. It can actually be quite  
a bit of fun. Come along with me, and explore how we can make software  
development simpler by focusing on frequent delivery of  
visible value. We’ll not talk about how things are, but how  
they might be, if we try. Introduction • xvi  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: What is described here is simple—but it’s not easy. You’ll  
need to think about these ideas, to figure out how they’ll be  
valuable to you, and to learn to do the things we explore  
here. Keep moving toward simplicity. You’ll be glad you  
did. The Natural Way does require us to think, to learn, and to  
change a bit. I think you’ll see here that moving toward the  
Natural Way need not be traumatic. It can actually be quite  
a bit of fun. Come along with me, and explore how we can make software  
development simpler by focusing on frequent delivery of  
visible value. We’ll not talk about how things are, but how  
they might be, if we try. Introduction • xvi  
report erratum • discuss]  
  
A final warning before you jump in:  
Channeling comedian Eddie Izzard’s NSFW “Death Star  
Canteen” bit:  
This is not a book of what the heck to do. It’s not a book of recipes. It’s not about one way to do  
something. That’s not our purpose here.

## Translated Text:

و این کار مشکلات مدیریت را با قابل مشاهده کردن اطلاعات کاهش می‌دهد، به طوری که نیازی به جستجو برای آن‌ها نیست. روش طبیعی حتی کار را برای توسعه‌دهندگان نیز آسان‌تر می‌کند. این روش به آن‌ها جهت‌گیری واضحی می‌دهد و آزادی استفاده از مهارت‌هایشان برای Build کردن آنچه سازمان نیاز دارد، در زمان مورد نیاز، فراهم می‌کند.   
report erratum • discuss   
Introduction • xv   
  
آنچه در اینجا توصیف شده ساده است—اما آسان نیست. شما باید درباره این ایده‌ها فکر کنید، بفهمید که چگونه برای شما ارزشمند خواهند بود، و یاد بگیرید که کارهایی که در اینجا بررسی می‌کنیم را انجام دهید. به حرکت به سمت سادگی ادامه دهید. خوشحال خواهید شد که این کار را کردید. روش طبیعی از ما می‌خواهد که فکر کنیم، یاد بگیریم، و کمی تغییر کنیم. فکر می‌کنم خواهید دید که حرکت به سمت روش طبیعی نباید ترسناک باشد. در واقع می‌تواند بسیار سرگرم‌کننده باشد. با من همراه شوید و بررسی کنیم که چگونه می‌توانیم توسعه نرم‌افزار را با تمرکز بر تحویل مکرر ارزش قابل مشاهده ساده‌تر کنیم. ما درباره اینکه اوضاع چگونه است صحبت نمی‌کنیم، بلکه درباره اینکه چگونه می‌تواند باشد، اگر تلاش کنیم، صحبت خواهیم کرد.   
Introduction • xvi   
report erratum • discuss   
  
[متن از صفحه قبلی: آنچه در اینجا توصیف شده ساده است—اما آسان نیست. شما باید درباره این ایده‌ها فکر کنید، بفهمید که چگونه برای شما ارزشمند خواهند بود، و یاد بگیرید که کارهایی که در اینجا بررسی می‌کنیم را انجام دهید. به حرکت به سمت سادگی ادامه دهید. خوشحال خواهید شد که این کار را کردید. روش طبیعی از ما می‌خواهد که فکر کنیم، یاد بگیریم، و کمی تغییر کنیم. فکر می‌کنم خواهید دید که حرکت به سمت روش طبیعی نباید ترسناک باشد. در واقع می‌تواند بسیار سرگرم‌کننده باشد. با من همراه شوید و بررسی کنیم که چگونه می‌توانیم توسعه نرم‌افزار را با تمرکز بر تحویل مکرر ارزش قابل مشاهده ساده‌تر کنیم. ما درباره اینکه اوضاع چگونه است صحبت نمی‌کنیم، بلکه درباره اینکه چگونه می‌تواند باشد، اگر تلاش کنیم، صحبت خواهیم کرد.   
Introduction • xvi   
report erratum • discuss]   
  
یک هشدار نهایی قبل از اینکه شروع کنید:   
با الهام از بخش NSFW "Death Star Canteen" از کمدین Eddie Izzard:   
این کتاب درباره اینکه چه کاری باید انجام دهید نیست. این کتاب دستورالعمل‌ها نیست. این درباره یک روش خاص برای انجام کاری نیست. این هدف ما در اینجا نیست.

================================================================================

# Section 15

## Original Text:

We’re here to think  
about how things work, to ready ourselves for whatever  
may happen. There are many ways to accomplish what you  
need. I trust you to find ways, think of ways, and select  
among them. report erratum • discussIntroduction • xvii  
  
[CONTEXT FROM PREVIOUS PAGE: A final warning before you jump in:  
Channeling comedian Eddie Izzard’s NSFW “Death Star  
Canteen” bit:  
This is not a book of what the heck to do. It’s not a book of recipes. It’s not about one way to do  
something. That’s not our purpose here. We’re here to think  
about how things work, to ready ourselves for whatever  
may happen. There are many ways to accomplish what you  
need. I trust you to find ways, think of ways, and select  
among them. report erratum • discussIntroduction • xvii]  
  
Part I  
The Circle of Value  
Sometimes you just have to stop holding on with both hands, both feet, and  
your tail, to get someplace better. Of course you might plummet to the earth  
and die, but probably not: you were made for this. [CONTEXT FROM PREVIOUS PAGE: Part I  
The Circle of Value  
Sometimes you just have to stop holding on with both hands, both feet, and  
your tail, to get someplace better. Of course you might plummet to the earth  
and die, but probably not: you were made for this. ]  
  
Successful software development is hard. It will always be hard. However, doing it smoothly and gracefully has a very real simplicity. Let’s talk about that essential simplicity. As we do, your job is to  
think a lot, while I write very little. • 3  
  
[CONTEXT FROM PREVIOUS PAGE: Successful software development is hard. It will always be hard. However, doing it smoothly and gracefully has a very real simplicity. Let’s talk about that essential simplicity. As we do, your job is to  
think a lot, while I write very little. • 3]  
  
CHAPTER 1  
The Search for Value  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 1  
The Search for Value  
report erratum • discuss]  
  
This picture shows the flow of our argument here.

## Translated Text:

ما اینجا هستیم تا درباره‌ی نحوه‌ی کارکرد چیزها فکر کنیم و خودمان را برای هر چیزی که ممکن است اتفاق بیفتد آماده کنیم. راه‌های زیادی برای رسیدن به آنچه نیاز دارید وجود دارد. من به شما اعتماد دارم که راه‌هایی پیدا کنید، به آن‌ها فکر کنید و از میانشان انتخاب کنید.   
گزارش خطا • بحث   
مقدمه • xvii   
  
[متن از صفحه‌ی قبلی: یک هشدار نهایی پیش از شروع:   
با الهام از بخش NSFW "Death Star Canteen" از کمدین Eddie Izzard:   
این کتابی نیست که بگوید چه کاری باید انجام دهید. این کتاب دستورالعمل‌ها نیست. درباره‌ی یک روش خاص برای انجام کاری نیست. هدف ما اینجا این نیست. ما اینجا هستیم تا درباره‌ی نحوه‌ی کارکرد چیزها فکر کنیم و خودمان را برای هر چیزی که ممکن است اتفاق بیفتد آماده کنیم. راه‌های زیادی برای رسیدن به آنچه نیاز دارید وجود دارد. من به شما اعتماد دارم که راه‌هایی پیدا کنید، به آن‌ها فکر کنید و از میانشان انتخاب کنید.   
گزارش خطا • بحث   
مقدمه • xvii]   
  
بخش اول   
دایره‌ی ارزش   
گاهی لازم است که دست‌ها، پاها و حتی دم خود را رها کنید تا به جایی بهتر برسید. البته ممکن است به زمین سقوط کنید و بمیرید، اما احتمالاً این‌طور نیست: شما برای این ساخته شده‌اید.   
[متن از صفحه‌ی قبلی: بخش اول   
دایره‌ی ارزش   
گاهی لازم است که دست‌ها، پاها و حتی دم خود را رها کنید تا به جایی بهتر برسید. البته ممکن است به زمین سقوط کنید و بمیرید، اما احتمالاً این‌طور نیست: شما برای این ساخته شده‌اید.]   
  
توسعه‌ی موفق نرم‌افزار سخت است. همیشه سخت خواهد بود. با این حال، انجام آن به صورت روان و زیبا، یک سادگی واقعی دارد. بیایید درباره‌ی آن سادگی اساسی صحبت کنیم. در این مسیر، وظیفه‌ی شما این است که زیاد فکر کنید، در حالی که من خیلی کم می‌نویسم. • 3   
  
[متن از صفحه‌ی قبلی: توسعه‌ی موفق نرم‌افزار سخت است. همیشه سخت خواهد بود. با این حال، انجام آن به صورت روان و زیبا، یک سادگی واقعی دارد. بیایید درباره‌ی آن سادگی اساسی صحبت کنیم. در این مسیر، وظیفه‌ی شما این است که زیاد فکر کنید، در حالی که من خیلی کم می‌نویسم. • 3]   
  
فصل ۱   
جستجوی ارزش   
گزارش خطا • بحث   
  
[متن از صفحه‌ی قبلی: فصل ۱   
جستجوی ارزش   
گزارش خطا • بحث]   
  
این تصویر جریان استدلال ما را در اینجا نشان می‌دهد.

================================================================================

# Section 16

## Original Text:

Our story  
begins with value, and value is the point of our work:  
Value. Value, we’ll see, is “what you want. ” It can be any  
kind of value, from money to laughs or lives. We’ll explore  
a bit about what value is. We’ll tell the story by building up from the bottom of the  
pyramid, describing how to guide, organize, plan, and build  
our product, in small slices, with a focus on quality. The  
value we produce is based on these. Guiding. We produce value by creating teams with responsi-  
bility for creating value. We make sure they understand  
what is needed, and understand the time available. We guide  
them by observing what they actually build. Organizing. We organize teams with the ability to get the job  
done. We organize around features, because features give  
us the ability to plan and build value most rapidly. We apply  
good people and help them build their skills. Planning. We steer our projects by selecting the features we  
need, in the order we need them. We produce value in a  
timely fashion. Building. We build up our product feature by feature. This  
provides frequent delivery of value. We can see how things  
are progressing early and often. Slicing. We slice features down to the smallest possible value-  
bearing size. We build a capable product as early as possible,  
and then enhance and grow it as the deadline approaches. We’re always ready to ship. Quality. We apply the necessary practices to ensure that our  
product always has a good design and that it is as nearly  
defect-free as possible. We’re able to build value continuous-  
ly, sustainably, indefinitely. report erratum • discussChapter 1. The Search for Value • 5  
  
[CONTEXT FROM PREVIOUS PAGE: This picture shows the flow of our argument here. Our story  
begins with value, and value is the point of our work:  
Value. Value, we’ll see, is “what you want. ” It can be any  
kind of value, from money to laughs or lives. We’ll explore  
a bit about what value is.

## Translated Text:

داستان ما   
با ارزش آغاز می‌شود، و ارزش نقطه‌ی تمرکز کار ما است:   
ارزش. ارزش، همانطور که خواهیم دید، «چیزی است که شما می‌خواهید.» این می‌تواند هر نوع ارزشی باشد، از پول گرفته تا خنده یا زندگی‌ها. ما کمی درباره‌ی اینکه ارزش چیست بررسی خواهیم کرد.   
ما داستان را با ساختن از پایین هرم آغاز می‌کنیم، و توضیح می‌دهیم که چگونه محصول خود را راهنمایی، سازماندهی، برنامه‌ریزی و Build کنیم، در قطعات کوچک، با تمرکز بر کیفیت. ارزشی که تولید می‌کنیم بر اساس این اصول است.   
  
\*\*راهنمایی.\*\* ما ارزش را با ایجاد تیم‌هایی که مسئولیت تولید ارزش را دارند، تولید می‌کنیم. ما مطمئن می‌شویم که آنها نیازها را درک کرده و زمان موجود را بفهمند. ما آنها را با مشاهده‌ی چیزی که واقعاً Build می‌کنند، راهنمایی می‌کنیم.   
  
\*\*سازماندهی.\*\* ما تیم‌هایی را سازماندهی می‌کنیم که توانایی انجام کار را دارند. ما حول Features سازماندهی می‌کنیم، زیرا Features به ما امکان می‌دهند که سریع‌تر برنامه‌ریزی و ارزش ایجاد کنیم. ما افراد توانمند را به کار می‌گیریم و به آنها کمک می‌کنیم مهارت‌های خود را توسعه دهند.   
  
\*\*برنامه‌ریزی.\*\* ما پروژه‌های خود را با انتخاب Features مورد نیاز، به ترتیب مورد نیاز، هدایت می‌کنیم. ما ارزش را به موقع تولید می‌کنیم.   
  
\*\*ساختن.\*\* ما محصول خود را Feature به Feature Build می‌کنیم. این کار تحویل مکرر ارزش را فراهم می‌کند. ما می‌توانیم پیشرفت کار را زود و به طور مکرر ببینیم.   
  
\*\*قطعه‌بندی.\*\* ما Features را به کوچک‌ترین اندازه‌ی ممکن که ارزش ایجاد می‌کند، تقسیم می‌کنیم. ما یک محصول توانمند را در اسرع وقت Build می‌کنیم و سپس آن را با نزدیک شدن به Deadline بهبود و گسترش می‌دهیم. ما همیشه آماده‌ی Ship کردن هستیم.   
  
\*\*کیفیت.\*\* ما Practices لازم را اعمال می‌کنیم تا مطمئن شویم محصول ما همیشه طراحی خوبی دارد و تا حد ممکن بدون نقص است. ما قادر به تولید ارزش به صورت پیوسته، پایدار و نامحدود هستیم.   
  
[CONTEXT FROM PREVIOUS PAGE: این تصویر جریان استدلال ما را نشان می‌دهد. داستان ما   
با ارزش آغاز می‌شود، و ارزش نقطه‌ی تمرکز کار ما است:   
ارزش. ارزش، همانطور که خواهیم دید، «چیزی است که شما می‌خواهید.» این می‌تواند هر نوع ارزشی باشد، از پول گرفته تا خنده یا زندگی‌ها. ما کمی درباره‌ی اینکه ارزش چیست بررسی خواهیم کرد.]

================================================================================

# Section 17

## Original Text:

We’ll tell the story by building up from the bottom of the  
pyramid, describing how to guide, organize, plan, and build  
our product, in small slices, with a focus on quality. The  
value we produce is based on these. Guiding. We produce value by creating teams with responsi-  
bility for creating value. We make sure they understand  
what is needed, and understand the time available. We guide  
them by observing what they actually build. Organizing. We organize teams with the ability to get the job  
done. We organize around features, because features give  
us the ability to plan and build value most rapidly. We apply  
good people and help them build their skills. Planning. We steer our projects by selecting the features we  
need, in the order we need them. We produce value in a  
timely fashion. Building. We build up our product feature by feature. This  
provides frequent delivery of value. We can see how things  
are progressing early and often. Slicing. We slice features down to the smallest possible value-  
bearing size. We build a capable product as early as possible,  
and then enhance and grow it as the deadline approaches. We’re always ready to ship. Quality. We apply the necessary practices to ensure that our  
product always has a good design and that it is as nearly  
defect-free as possible. We’re able to build value continuous-  
ly, sustainably, indefinitely. report erratum • discussChapter 1. The Search for Value • 5]  
  
What is software value. CHAPTER 2  
Value Is What We Want  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: What is software value. CHAPTER 2  
Value Is What We Want  
report erratum • discuss]  
  
We all want value. Value is what we want. Value is— what  
we want. In software, we generally get value by delivering  
features. Features that have value. Features that we want. Often it’s about money, because software can save time or  
money. Software can help us earn money. There are other  
kinds of value: software can make lives more convenient.

## Translated Text:

ما داستان را با ساختن از پایین هرم شروع می‌کنیم، توضیح می‌دهیم که چگونه محصول خود را راهنمایی، سازمان‌دهی، برنامه‌ریزی و Build کنیم، در بخش‌های کوچک، با تمرکز بر کیفیت. ارزشی که تولید می‌کنیم بر اساس این اصول است.   
  
راهنمایی. ما با ایجاد تیم‌هایی که مسئولیت تولید ارزش را دارند، ارزش تولید می‌کنیم. ما اطمینان حاصل می‌کنیم که آن‌ها نیازها را درک کنند و زمان موجود را بفهمند. ما آن‌ها را با مشاهده آنچه واقعاً Build می‌کنند، راهنمایی می‌کنیم.   
  
سازمان‌دهی. ما تیم‌هایی را سازمان‌دهی می‌کنیم که توانایی انجام کار را داشته باشند. ما حول Features سازمان‌دهی می‌کنیم، زیرا Features به ما این امکان را می‌دهند که سریع‌تر برنامه‌ریزی و ارزش تولید کنیم. ما افراد خوب را به کار می‌گیریم و به آن‌ها کمک می‌کنیم مهارت‌هایشان را توسعه دهند.   
  
برنامه‌ریزی. ما پروژه‌های خود را با انتخاب Features مورد نیاز، به ترتیب اولویت آن‌ها، هدایت می‌کنیم. ما ارزش را به‌موقع تولید می‌کنیم.   
  
ساختن. ما محصول خود را Feature به Feature Build می‌کنیم. این کار تحویل مکرر ارزش را فراهم می‌کند. ما می‌توانیم پیشرفت را زود و به‌طور مداوم مشاهده کنیم.   
  
بخش‌بندی. ما Features را به کوچک‌ترین اندازه ممکن که ارزش ایجاد می‌کند، تقسیم می‌کنیم. ما یک محصول قابل‌قبول را در سریع‌ترین زمان ممکن Build می‌کنیم و سپس آن را با نزدیک شدن به مهلت زمانی بهبود و گسترش می‌دهیم. ما همیشه آماده Ship کردن هستیم.   
  
کیفیت. ما Practices لازم را اعمال می‌کنیم تا اطمینان حاصل کنیم که محصول ما همیشه طراحی خوبی دارد و تا حد ممکن بدون نقص است. ما قادر هستیم به‌طور مداوم، پایدار و بی‌پایان ارزش تولید کنیم.   
  
report erratum • discuss   
فصل ۱. جستجوی ارزش • ۵   
  
ارزش نرم‌افزار چیست. فصل ۲   
ارزش همان چیزی است که ما می‌خواهیم   
report erratum • discuss   
  
[متن از صفحه قبلی: ارزش نرم‌افزار چیست. فصل ۲   
ارزش همان چیزی است که ما می‌خواهیم   
report erratum • discuss]   
  
ما همه ارزش می‌خواهیم. ارزش همان چیزی است که ما می‌خواهیم. ارزش همان چیزی است که ما می‌خواهیم. در نرم‌افزار، ما معمولاً با تحویل Features ارزش به دست می‌آوریم. Features که ارزش دارند. Features که ما می‌خواهیم. اغلب این موضوع به پول مربوط می‌شود، زیرا نرم‌افزار می‌تواند زمان یا پول را صرفه‌جویی کند. نرم‌افزار می‌تواند به ما کمک کند پول به دست آوریم. انواع دیگری از ارزش نیز وجود دارد: نرم‌افزار می‌تواند زندگی‌ها را راحت‌تر کند.

================================================================================

# Section 18

## Original Text:

Software can even save lives. In the end, I think of value as simply what we want. We might  
like to put a number on value, but it’s not necessary. As we  
build the software, we’ll make choices. Each choice gives us  
something we value. We’ll choose information, happy users,  
or saved lives. We’ll choose what makes sense. We’ll choose  
what we want. Working incrementally, we’ll choose the next thing we want. We’ll have our team put it into the software, as quickly and  
solidly as they can. When they’re done, we’ll look to be sure  
we got what we want: we’ll check for the value. We’ll say, “show us the software,” to see the value. What kinds of value does your project deliver to its users. To your organization. To the team. What value does it  
deliver to you. report erratum • discussChapter 2. Value Is What We Want • 7  
  
[CONTEXT FROM PREVIOUS PAGE: We all want value. Value is what we want. Value is— what  
we want. In software, we generally get value by delivering  
features. Features that have value. Features that we want. Often it’s about money, because software can save time or  
money. Software can help us earn money. There are other  
kinds of value: software can make lives more convenient. Software can even save lives. In the end, I think of value as simply what we want. We might  
like to put a number on value, but it’s not necessary. As we  
build the software, we’ll make choices. Each choice gives us  
something we value. We’ll choose information, happy users,  
or saved lives. We’ll choose what makes sense. We’ll choose  
what we want. Working incrementally, we’ll choose the next thing we want. We’ll have our team put it into the software, as quickly and  
solidly as they can. When they’re done, we’ll look to be sure  
we got what we want: we’ll check for the value. We’ll say, “show us the software,” to see the value. What kinds of value does your project deliver to its users. To your organization. To the team. What value does it  
deliver to you. report erratum • discussChapter 2.

## Translated Text:

نرم‌افزار حتی می‌تواند جان‌ها را نجات دهد. در نهایت، من ارزش را به‌سادگی چیزی که می‌خواهیم تعریف می‌کنم. ممکن است بخواهیم عددی برای ارزش تعیین کنیم، اما این ضروری نیست. وقتی نرم‌افزار را Build می‌کنیم، انتخاب‌هایی خواهیم داشت. هر انتخاب چیزی که برای ما ارزشمند است به ما می‌دهد. ما اطلاعات، کاربران خوشحال، یا جان‌های نجات‌یافته را انتخاب خواهیم کرد. ما چیزی که منطقی است را انتخاب می‌کنیم. ما چیزی که می‌خواهیم را انتخاب می‌کنیم.   
  
با کار کردن به‌صورت تدریجی، ما چیزی که بعدی می‌خواهیم را انتخاب خواهیم کرد. تیم ما آن را به نرم‌افزار اضافه خواهد کرد، به سریع‌ترین و مطمئن‌ترین شکلی که می‌توانند. وقتی کارشان تمام شد، بررسی خواهیم کرد تا مطمئن شویم چیزی که می‌خواستیم را به دست آورده‌ایم: ما ارزش را بررسی خواهیم کرد. خواهیم گفت، "نرم‌افزار را به ما نشان دهید"، تا ارزش را ببینیم. پروژه شما چه نوع ارزشی به کاربرانش ارائه می‌دهد؟ به سازمان شما؟ به تیم؟ چه ارزشی به شما ارائه می‌دهد؟   
  
report erratum • discuss   
فصل ۲. ارزش همان چیزی است که ما می‌خواهیم • ۷   
  
[متن از صفحه قبلی: همه ما ارزش می‌خواهیم. ارزش همان چیزی است که ما می‌خواهیم. ارزش همان چیزی است— که ما می‌خواهیم. در نرم‌افزار، ما معمولاً با ارائه Features ارزش به دست می‌آوریم. Features که ارزشمند هستند. Features که ما می‌خواهیم. اغلب این موضوع به پول مربوط می‌شود، زیرا نرم‌افزار می‌تواند زمان یا پول صرفه‌جویی کند. نرم‌افزار می‌تواند به ما کمک کند پول به دست آوریم. انواع دیگری از ارزش نیز وجود دارد: نرم‌افزار می‌تواند زندگی‌ها را راحت‌تر کند. نرم‌افزار حتی می‌تواند جان‌ها را نجات دهد...]

================================================================================

# Section 19

## Original Text:

Value Is What We Want • 7]  
  
Value starts when we ship the software. A project delivers value only when we ship the software  
and put it to use. If we wait until we finish everything, it  
will be a long time before we get any value. Let’s find a way  
to deliver value early. We’d rather have the pony now, not later, but we can’t create  
everything right now. We have many features in mind, and  
they’ll take time to build. The more we want, the longer it  
takes. What benefits could there be if we could deliver sooner. How might the organization benefit. What about the team. What about you and me. Chapter 2. Value Is What We Want • 8  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Value starts when we ship the software. A project delivers value only when we ship the software  
and put it to use. If we wait until we finish everything, it  
will be a long time before we get any value. Let’s find a way  
to deliver value early. We’d rather have the pony now, not later, but we can’t create  
everything right now. We have many features in mind, and  
they’ll take time to build. The more we want, the longer it  
takes. What benefits could there be if we could deliver sooner. How might the organization benefit. What about the team. What about you and me. Chapter 2. Value Is What We Want • 8  
report erratum • discuss]  
  
What if we shipped some valuable part sooner than the rest. Every product is made up of pieces. Call them features, or  
minimum marketable features. Call them aspects, functions,  
or capabilities. Each big piece has smaller pieces. Each is full  
of details that make that piece more complete, more useful,  
or just nicer. Remember, most users of a product don’t use every feature. There’s some kind of 80/20 rule going on. Everyone may  
want something different, but no one wants everything. Even in the products you know best and use most, you  
probably use only a fraction of the features. report erratum • discussChapter 2.

## Translated Text:

ارزش همان چیزی است که ما می‌خواهیم • ۷  
  
ارزش زمانی شروع می‌شود که نرم‌افزار را Ship کنیم. یک پروژه تنها زمانی ارزش ایجاد می‌کند که نرم‌افزار را Ship کنیم و از آن استفاده کنیم. اگر صبر کنیم تا همه چیز را تمام کنیم، مدت زیادی طول می‌کشد تا به هرگونه ارزشی برسیم. بیایید راهی پیدا کنیم تا زودتر ارزش را ارائه دهیم. ما ترجیح می‌دهیم حالا چیزی داشته باشیم، نه بعداً، اما نمی‌توانیم همه چیز را همین حالا ایجاد کنیم. ما ویژگی‌های زیادی در ذهن داریم و ساخت آن‌ها زمان می‌برد. هرچه بیشتر بخواهیم، زمان بیشتری طول می‌کشد. چه مزایایی می‌تواند وجود داشته باشد اگر بتوانیم زودتر ارائه دهیم؟ سازمان چگونه می‌تواند سود ببرد؟ تیم چطور؟ من و شما چطور؟   
فصل ۲. ارزش همان چیزی است که ما می‌خواهیم • ۸   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: ارزش زمانی شروع می‌شود که نرم‌افزار را Ship کنیم. یک پروژه تنها زمانی ارزش ایجاد می‌کند که نرم‌افزار را Ship کنیم و از آن استفاده کنیم. اگر صبر کنیم تا همه چیز را تمام کنیم، مدت زیادی طول می‌کشد تا به هرگونه ارزشی برسیم. بیایید راهی پیدا کنیم تا زودتر ارزش را ارائه دهیم. ما ترجیح می‌دهیم حالا چیزی داشته باشیم، نه بعداً، اما نمی‌توانیم همه چیز را همین حالا ایجاد کنیم. ما ویژگی‌های زیادی در ذهن داریم و ساخت آن‌ها زمان می‌برد. هرچه بیشتر بخواهیم، زمان بیشتری طول می‌کشد. چه مزایایی می‌تواند وجود داشته باشد اگر بتوانیم زودتر ارائه دهیم؟ سازمان چگونه می‌تواند سود ببرد؟ تیم چطور؟ من و شما چطور؟   
فصل ۲. ارزش همان چیزی است که ما می‌خواهیم • ۸   
گزارش خطا • بحث کنید]   
  
اگر بخشی ارزشمند را زودتر از بقیه Ship کنیم چه می‌شود؟ هر محصول از قطعاتی تشکیل شده است. آن‌ها را Features یا Minimum Marketable Features بنامید. آن‌ها را Aspects، Functions، یا Capabilities بنامید. هر قطعه بزرگ شامل قطعات کوچکتری است. هر کدام پر از جزئیاتی است که آن قطعه را کامل‌تر، مفیدتر یا فقط زیباتر می‌کند. به یاد داشته باشید، اکثر کاربران یک محصول از همه Features آن استفاده نمی‌کنند. چیزی شبیه به قانون ۸۰/۲۰ در جریان است. ممکن است هر کسی چیزی متفاوت بخواهد، اما هیچ‌کس همه چیز را نمی‌خواهد. حتی در محصولاتی که بهتر می‌شناسید و بیشتر استفاده می‌کنید، احتمالاً فقط از بخشی از Features استفاده می‌کنید.   
گزارش خطا • بحث کنید   
فصل ۲.

================================================================================

# Section 20

## Original Text:

Value Is What We Want • 9  
  
[CONTEXT FROM PREVIOUS PAGE: What if we shipped some valuable part sooner than the rest. Every product is made up of pieces. Call them features, or  
minimum marketable features. Call them aspects, functions,  
or capabilities. Each big piece has smaller pieces. Each is full  
of details that make that piece more complete, more useful,  
or just nicer. Remember, most users of a product don’t use every feature. There’s some kind of 80/20 rule going on. Everyone may  
want something different, but no one wants everything. Even in the products you know best and use most, you  
probably use only a fraction of the features. report erratum • discussChapter 2. Value Is What We Want • 9]  
  
Does shipping something small make sense. Since most users don’t use all the features, a smaller set of  
features can provide real value, and provide it sooner. Sometimes we think we have to have it all. Let’s face it,  
though: if you’ve done very many software projects, you  
probably didn’t get everything you wanted by the date you  
wanted it. We never get it all. We can stamp our feet and demand a pony, or we can act  
like managers and steer our software projects to the best  
possible result. Very likely, there’s a subset of capability that  
can start providing value sooner than the whole package. Let’s find those features and ship them first. That way, we’ll  
prosper. After that first release, we may need to follow up with the  
rest of the product; otherwise, the final product may be  
worth less over its lifetime. So we’ll usually plan multiple  
releases. But there are times when we might just ship the first bit and  
then stop. When could that be the best thing to do. How  
many different reasons can you think of. Chapter 2. Value Is What We Want • 10  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Does shipping something small make sense. Since most users don’t use all the features, a smaller set of  
features can provide real value, and provide it sooner.

## Translated Text:

ارزش چیزی است که می‌خواهیم • 9  
  
[متن از صفحه قبلی: اگر بخشی ارزشمند را زودتر از بقیه ارائه دهیم چه می‌شود؟ هر محصول از قطعاتی تشکیل شده است. آنها را Features یا حداقل Features قابل بازاریابی بنامید. آنها را جنبه‌ها، Functions یا قابلیت‌ها بنامید. هر قطعه بزرگ شامل قطعات کوچکتری است. هر کدام پر از جزئیاتی هستند که آن قطعه را کامل‌تر، مفیدتر یا صرفاً جذاب‌تر می‌کنند. به یاد داشته باشید، اکثر کاربران یک محصول از همه Features آن استفاده نمی‌کنند. نوعی قانون 80/20 در اینجا وجود دارد. ممکن است هر کسی چیزی متفاوت بخواهد، اما هیچ‌کس همه چیز را نمی‌خواهد. حتی در محصولاتی که بهترین شناخت را از آنها دارید و بیشتر استفاده می‌کنید، احتمالاً فقط از بخشی از Features استفاده می‌کنید. گزارش خطا • بحث فصل 2. ارزش چیزی است که می‌خواهیم • 9]  
  
آیا ارائه چیزی کوچک منطقی است؟ از آنجا که اکثر کاربران از همه Features استفاده نمی‌کنند، یک مجموعه کوچکتر از Features می‌تواند ارزش واقعی ارائه دهد و آن را زودتر ارائه کند. گاهی اوقات فکر می‌کنیم باید همه چیز را داشته باشیم. اما بیایید صادق باشیم: اگر پروژه‌های نرم‌افزاری زیادی انجام داده باشید، احتمالاً همه چیزهایی که می‌خواستید را در تاریخی که می‌خواستید به دست نیاورده‌اید. ما هرگز همه چیز را به دست نمی‌آوریم. می‌توانیم پای خود را بکوبیم و یک اسب کوچک بخواهیم، یا می‌توانیم مانند مدیران عمل کنیم و پروژه‌های نرم‌افزاری خود را به بهترین نتیجه ممکن هدایت کنیم. به احتمال زیاد، یک زیرمجموعه از قابلیت‌ها وجود دارد که می‌تواند زودتر از کل بسته ارزش ارائه دهد. بیایید آن Features را پیدا کنیم و ابتدا آنها را ارائه دهیم. به این ترتیب، موفق خواهیم شد. پس از اولین Release، ممکن است نیاز داشته باشیم که با بقیه محصول ادامه دهیم؛ در غیر این صورت، محصول نهایی ممکن است در طول عمر خود ارزش کمتری داشته باشد. بنابراین معمولاً چندین Release را برنامه‌ریزی می‌کنیم. اما مواقعی وجود دارد که ممکن است فقط اولین بخش را ارائه دهیم و سپس متوقف شویم. چه زمانی این می‌تواند بهترین کار باشد؟ چند دلیل مختلف می‌توانید برای آن تصور کنید؟ فصل 2. ارزش چیزی است که می‌خواهیم • 10   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: آیا ارائه چیزی کوچک منطقی است؟ از آنجا که اکثر کاربران از همه Features استفاده نمی‌کنند، یک مجموعه کوچکتر از Features می‌تواند ارزش واقعی ارائه دهد و آن را زودتر ارائه کند.]

================================================================================

# Section 21

## Original Text:

Sometimes we think we have to have it all. Let’s face it,  
though: if you’ve done very many software projects, you  
probably didn’t get everything you wanted by the date you  
wanted it. We never get it all. We can stamp our feet and demand a pony, or we can act  
like managers and steer our software projects to the best  
possible result. Very likely, there’s a subset of capability that  
can start providing value sooner than the whole package. Let’s find those features and ship them first. That way, we’ll  
prosper. After that first release, we may need to follow up with the  
rest of the product; otherwise, the final product may be  
worth less over its lifetime. So we’ll usually plan multiple  
releases. But there are times when we might just ship the first bit and  
then stop. When could that be the best thing to do. How  
many different reasons can you think of. Chapter 2. Value Is What We Want • 10  
report erratum • discuss]  
  
Deliver just one part and then stop. If we ship just once, we’ll get an earlier return, but it will  
probably be less than if we shipped the whole product, even  
later on. Or will it. Information has value as well. Sometimes the most important  
information we can get is that we’re doing the wrong thing. One very good way to find out if we’re going in the right  
direction is to ship a small version of the product early. If it  
flops, we can change direction at low cost. Usually, though, we do have a good idea. With a good idea,  
what kind of pieces should we work on. How would our  
product best be delivered a bit at a time. What should the  
pieces look like. report erratum • discussChapter 2. Value Is What We Want • 11  
  
[CONTEXT FROM PREVIOUS PAGE: Deliver just one part and then stop. If we ship just once, we’ll get an earlier return, but it will  
probably be less than if we shipped the whole product, even  
later on. Or will it. Information has value as well. Sometimes the most important  
information we can get is that we’re doing the wrong thing.

## Translated Text:

گاهی فکر می‌کنیم که باید همه چیز را داشته باشیم. اما بیایید صادق باشیم: اگر پروژه‌های نرم‌افزاری زیادی انجام داده باشید، احتمالاً همه چیزهایی که می‌خواستید را تا تاریخی که می‌خواستید به دست نیاورده‌اید. ما هرگز همه چیز را به دست نمی‌آوریم. می‌توانیم پا به زمین بکوبیم و یک اسب بخواهیم، یا می‌توانیم مانند مدیران عمل کنیم و پروژه‌های نرم‌افزاری خود را به بهترین نتیجه ممکن هدایت کنیم. به احتمال زیاد، یک زیرمجموعه از قابلیت‌ها وجود دارد که می‌تواند زودتر از کل بسته، ارزش ایجاد کند. بیایید آن قابلیت‌ها را پیدا کنیم و ابتدا آن‌ها را Ship کنیم. به این ترتیب، موفق خواهیم شد.   
  
بعد از آن اولین Release، ممکن است نیاز داشته باشیم که با بقیه محصول ادامه دهیم؛ در غیر این صورت، محصول نهایی ممکن است در طول عمر خود ارزش کمتری داشته باشد. بنابراین معمولاً چندین Release را برنامه‌ریزی می‌کنیم. اما مواقعی وجود دارد که ممکن است فقط اولین بخش را Ship کنیم و سپس متوقف شویم. چه زمانی این می‌تواند بهترین کار باشد؟ چند دلیل مختلف می‌توانید برای آن تصور کنید؟   
  
فصل ۲. ارزش چیزی است که می‌خواهیم • ۱۰   
report erratum • discuss   
  
فقط یک بخش را تحویل دهید و سپس متوقف شوید. اگر فقط یک بار Ship کنیم، زودتر بازگشت سرمایه خواهیم داشت، اما احتمالاً کمتر از زمانی خواهد بود که کل محصول را Ship کنیم، حتی اگر دیرتر باشد. یا شاید نه؟ اطلاعات نیز ارزش دارد. گاهی اوقات مهم‌ترین اطلاعاتی که می‌توانیم به دست آوریم این است که داریم کار اشتباهی انجام می‌دهیم.   
  
یکی از بهترین راه‌ها برای فهمیدن اینکه آیا در مسیر درست هستیم یا نه، این است که یک نسخه کوچک از محصول را زودتر Ship کنیم. اگر شکست بخورد، می‌توانیم با هزینه کم مسیر را تغییر دهیم. با این حال، معمولاً ایده خوبی داریم. با یک ایده خوب، باید روی چه نوع بخش‌هایی کار کنیم؟ محصول ما چگونه بهتر است به صورت تدریجی تحویل داده شود؟ این بخش‌ها باید چگونه باشند؟   
  
report erratum • discuss   
فصل ۲. ارزش چیزی است که می‌خواهیم • ۱۱

================================================================================

# Section 22

## Original Text:

One very good way to find out if we’re going in the right  
direction is to ship a small version of the product early. If it  
flops, we can change direction at low cost. Usually, though, we do have a good idea. With a good idea,  
what kind of pieces should we work on. How would our  
product best be delivered a bit at a time. What should the  
pieces look like. report erratum • discussChapter 2. Value Is What We Want • 11]  
  
We must see and understand the pieces. It’s not enough for our teams to work on mysterious technical  
bits that make sense only to them. We need to guide our teams to build pieces that make sense  
to us, and to our users. These are often called minimal mar-  
ketable features (MMFs). In fact, we’ll often benefit from pro-  
viding business direction at an even finer grain than the  
usual MMF. Here, we’ll call those pieces features. When we say “show us  
the software,” we want to see features that we want and  
understand. Looking back at some previous projects, what are some  
features you wish you could have shipped sooner, and why. What are some features that should have been different. Are there some that shouldn’t have been done at all. Chapter 2. Value Is What We Want • 12  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: We must see and understand the pieces. It’s not enough for our teams to work on mysterious technical  
bits that make sense only to them. We need to guide our teams to build pieces that make sense  
to us, and to our users. These are often called minimal mar-  
ketable features (MMFs). In fact, we’ll often benefit from pro-  
viding business direction at an even finer grain than the  
usual MMF. Here, we’ll call those pieces features. When we say “show us  
the software,” we want to see features that we want and  
understand. Looking back at some previous projects, what are some  
features you wish you could have shipped sooner, and why. What are some features that should have been different. Are there some that shouldn’t have been done at all.

## Translated Text:

یک روش بسیار خوب برای اینکه بفهمیم آیا در مسیر درست حرکت می‌کنیم یا نه، این است که یک نسخه کوچک از محصول را زودتر ارائه دهیم. اگر شکست بخورد، می‌توانیم با هزینه کم مسیر را تغییر دهیم. معمولاً، با این حال، ما یک ایده خوب داریم. با داشتن یک ایده خوب، چه نوع بخش‌هایی باید روی آن کار کنیم؟ محصول ما چگونه می‌تواند به بهترین شکل، مرحله به مرحله ارائه شود؟ این بخش‌ها باید چگونه باشند؟   
report erratum • discuss   
Chapter 2. Value Is What We Want • 11   
  
ما باید بخش‌ها را ببینیم و درک کنیم. کافی نیست که تیم‌های ما روی بخش‌های فنی مرموزی کار کنند که فقط برای خودشان معنا دارد. ما باید تیم‌های خود را راهنمایی کنیم تا بخش‌هایی بسازند که برای ما و کاربرانمان معنا داشته باشند. این بخش‌ها اغلب به عنوان minimal marketable features (MMFs) شناخته می‌شوند. در واقع، ما اغلب از ارائه جهت‌گیری کسب‌وکار در سطحی حتی جزئی‌تر از MMF معمولی بهره‌مند می‌شویم. در اینجا، ما این بخش‌ها را features می‌نامیم. وقتی می‌گوییم "نرم‌افزار را به ما نشان دهید"، می‌خواهیم featuresی را ببینیم که می‌خواهیم و درک می‌کنیم.   
  
با نگاه به برخی پروژه‌های قبلی، چه featuresی وجود دارند که آرزو می‌کردید زودتر ارائه می‌شدند و چرا؟ چه featuresی باید متفاوت می‌بودند؟ آیا برخی از آنها نباید اصلاً انجام می‌شدند؟   
Chapter 2. Value Is What We Want • 12   
report erratum • discuss

================================================================================

# Section 23

## Original Text:

Chapter 2. Value Is What We Want • 12  
report erratum • discuss]  
  
Value, by feature  
Each feature that we might build adds some value to the  
product. And each one takes some amount of time. We can’t  
know exactly how valuable or exactly how much time. But  
we can still get an excellent sense of what to do. Suppose the height of the features is their value, and the  
width is their cost. Which ones should we build first, and  
which ones should we defer until later. Pretty clear, isn’t it. report erratum • discussChapter 2. Value Is What We Want • 13  
  
[CONTEXT FROM PREVIOUS PAGE: Value, by feature  
Each feature that we might build adds some value to the  
product. And each one takes some amount of time. We can’t  
know exactly how valuable or exactly how much time. But  
we can still get an excellent sense of what to do. Suppose the height of the features is their value, and the  
width is their cost. Which ones should we build first, and  
which ones should we defer until later. Pretty clear, isn’t it. report erratum • discussChapter 2. Value Is What We Want • 13]  
  
Value growth depends on what we choose to do. Look at the difference in the growth of value if we choose  
the higher-value, inexpensive features first and defer lower-  
value, costly features until later. And these features only  
vary by about a factor of three to one. In most products the  
best ideas are tens of times better than the worst—or more. The results would hardly fit on the page. Some of those later features look pretty boring. What would  
happen if we did different, more valuable features, even for  
some other product. Chapter 2. Value Is What We Want • 14  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Value growth depends on what we choose to do. Look at the difference in the growth of value if we choose  
the higher-value, inexpensive features first and defer lower-  
value, costly features until later. And these features only  
vary by about a factor of three to one.

## Translated Text:

فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۲   
گزارش خطا • بحث کنید   
  
ارزش، بر اساس قابلیت   
هر قابلیت که ممکن است Build کنیم، مقداری ارزش به محصول اضافه می‌کند. و هر کدام مقداری زمان می‌برد. ما نمی‌توانیم دقیقاً بدانیم چقدر ارزشمند است یا چقدر زمان می‌برد. اما هنوز می‌توانیم حس خوبی از اینکه چه کاری انجام دهیم داشته باشیم. فرض کنید ارتفاع قابلیت‌ها ارزش آن‌ها است و عرض هزینه آن‌ها. کدام‌ها را باید ابتدا Build کنیم و کدام‌ها را باید به بعد موکول کنیم؟ کاملاً واضح است، اینطور نیست؟   
گزارش خطا • بحث کنید   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۳   
  
[متن قبلی: ارزش، بر اساس قابلیت   
هر قابلیت که ممکن است Build کنیم، مقداری ارزش به محصول اضافه می‌کند. و هر کدام مقداری زمان می‌برد. ما نمی‌توانیم دقیقاً بدانیم چقدر ارزشمند است یا چقدر زمان می‌برد. اما هنوز می‌توانیم حس خوبی از اینکه چه کاری انجام دهیم داشته باشیم. فرض کنید ارتفاع قابلیت‌ها ارزش آن‌ها است و عرض هزینه آن‌ها. کدام‌ها را باید ابتدا Build کنیم و کدام‌ها را باید به بعد موکول کنیم؟ کاملاً واضح است، اینطور نیست؟   
گزارش خطا • بحث کنید   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۳]   
  
رشد ارزش بستگی به چیزی دارد که انتخاب می‌کنیم انجام دهیم. به تفاوت رشد ارزش نگاه کنید اگر ابتدا قابلیت‌های با ارزش بالا و ارزان را انتخاب کنیم و قابلیت‌های با ارزش پایین و هزینه بالا را به بعد موکول کنیم. و این قابلیت‌ها فقط حدوداً با ضریب سه به یک متفاوت هستند. در بیشتر محصولات، بهترین ایده‌ها ده‌ها برابر بهتر از بدترین‌ها—یا حتی بیشتر—هستند. نتایج به سختی روی صفحه جا می‌شوند. برخی از آن قابلیت‌های بعدی خیلی کسل‌کننده به نظر می‌رسند. چه اتفاقی می‌افتد اگر قابلیت‌های متفاوت و ارزشمندتری انجام دهیم، حتی برای یک محصول دیگر.   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۴   
گزارش خطا • بحث کنید   
  
[متن قبلی: رشد ارزش بستگی به چیزی دارد که انتخاب می‌کنیم انجام دهیم. به تفاوت رشد ارزش نگاه کنید اگر ابتدا قابلیت‌های با ارزش بالا و ارزان را انتخاب کنیم و قابلیت‌های با ارزش پایین و هزینه بالا را به بعد موکول کنیم. و این قابلیت‌ها فقط حدوداً با ضریب سه به یک متفاوت هستند.]

================================================================================

# Section 24

## Original Text:

In most products the  
best ideas are tens of times better than the worst—or more. The results would hardly fit on the page. Some of those later features look pretty boring. What would  
happen if we did different, more valuable features, even for  
some other product. Chapter 2. Value Is What We Want • 14  
report erratum • discuss]  
  
We might even switch our investment to a new product. When we begin to ship frequently, with highest value first,  
the time soon comes when the next features aren’t worth the  
time and money to create them. This is a good thing. We can  
often do far better by investing in a new product. What’s the next product we’d like to do. Who might feel  
negatively impacted by a product shift. How might we make  
that shift a good thing for everyone. Can we focus on a  
portfolio rather than separate products with diminishing  
returns. Can we show more software, with more value. report erratum • discussChapter 2. Value Is What We Want • 15  
  
[CONTEXT FROM PREVIOUS PAGE: We might even switch our investment to a new product. When we begin to ship frequently, with highest value first,  
the time soon comes when the next features aren’t worth the  
time and money to create them. This is a good thing. We can  
often do far better by investing in a new product. What’s the next product we’d like to do. Who might feel  
negatively impacted by a product shift. How might we make  
that shift a good thing for everyone. Can we focus on a  
portfolio rather than separate products with diminishing  
returns. Can we show more software, with more value. report erratum • discussChapter 2. Value Is What We Want • 15]  
  
Best value comes from small, value-focused features, delivered  
frequently. OK, we can see that small features could deliver value  
sooner if we can do them. Let’s think next about managing  
our project. Will smaller visible results help us manage. How might they get in the way. What about our teams. Are they organized to work this  
way.

## Translated Text:

در بیشتر محصولات، بهترین ایده‌ها ده‌ها برابر بهتر از بدترین ایده‌ها هستند—یا حتی بیشتر. نتایج به سختی در یک صفحه جا می‌شوند. برخی از آن ویژگی‌های بعدی کاملاً کسل‌کننده به نظر می‌رسند. چه اتفاقی می‌افتد اگر ویژگی‌های متفاوت و ارزشمندتری ایجاد کنیم، حتی برای محصولی دیگر؟   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۴   
report erratum • discuss   
  
ما حتی ممکن است سرمایه‌گذاری خود را به یک محصول جدید منتقل کنیم. وقتی شروع به Ship کردن مکرر با بالاترین ارزش می‌کنیم، به زودی زمانی می‌رسد که ویژگی‌های بعدی ارزش زمان و هزینه برای ایجاد را ندارند. این یک چیز خوب است. ما اغلب می‌توانیم با سرمایه‌گذاری در یک محصول جدید عملکرد بهتری داشته باشیم. محصول بعدی که می‌خواهیم انجام دهیم چیست؟ چه کسی ممکن است از تغییر محصول احساس منفی داشته باشد؟ چگونه می‌توانیم این تغییر را برای همه به یک چیز خوب تبدیل کنیم؟ آیا می‌توانیم به جای محصولات جداگانه با بازده کاهشی، روی یک پورتفولیو تمرکز کنیم؟ آیا می‌توانیم نرم‌افزار بیشتری با ارزش بیشتر نشان دهیم؟   
report erratum • discuss   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۵   
  
[متن از صفحه قبلی: ما حتی ممکن است سرمایه‌گذاری خود را به یک محصول جدید منتقل کنیم. وقتی شروع به Ship کردن مکرر با بالاترین ارزش می‌کنیم، به زودی زمانی می‌رسد که ویژگی‌های بعدی ارزش زمان و هزینه برای ایجاد را ندارند. این یک چیز خوب است. ما اغلب می‌توانیم با سرمایه‌گذاری در یک محصول جدید عملکرد بهتری داشته باشیم. محصول بعدی که می‌خواهیم انجام دهیم چیست؟ چه کسی ممکن است از تغییر محصول احساس منفی داشته باشد؟ چگونه می‌توانیم این تغییر را برای همه به یک چیز خوب تبدیل کنیم؟ آیا می‌توانیم به جای محصولات جداگانه با بازده کاهشی، روی یک پورتفولیو تمرکز کنیم؟ آیا می‌توانیم نرم‌افزار بیشتری با ارزش بیشتر نشان دهیم؟   
report erratum • discuss   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۵]   
  
بهترین ارزش از ویژگی‌های کوچک و متمرکز بر ارزش که به طور مکرر Deliver می‌شوند به دست می‌آید. خوب، می‌توانیم ببینیم که ویژگی‌های کوچک می‌توانند زودتر ارزش ایجاد کنند اگر بتوانیم آن‌ها را انجام دهیم. بیایید بعدی درباره مدیریت پروژه خود فکر کنیم. آیا نتایج کوچک و قابل مشاهده به ما در مدیریت کمک می‌کنند؟ چگونه ممکن است مانع شوند؟ تیم‌های ما چطور؟ آیا آن‌ها به گونه‌ای سازماندهی شده‌اند که این‌گونه کار کنند؟

================================================================================

# Section 25

## Original Text:

Do they have the people they need, the skills they  
need, and the help they need. Read on—we’ll talk about all  
those things. The main thing to remember is that we get the best results  
from delivering the software, feature by feature. Chapter 2. Value Is What We Want • 16  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Best value comes from small, value-focused features, delivered  
frequently. OK, we can see that small features could deliver value  
sooner if we can do them. Let’s think next about managing  
our project. Will smaller visible results help us manage. How might they get in the way. What about our teams. Are they organized to work this  
way. Do they have the people they need, the skills they  
need, and the help they need. Read on—we’ll talk about all  
those things. The main thing to remember is that we get the best results  
from delivering the software, feature by feature. Chapter 2. Value Is What We Want • 16  
report erratum • discuss]  
  
Further reading:  
•Chapter 10, Value—What Is It. , on page 80  
•Chapter 11, Value—How Can We Measure It. , on page  
84  
report erratum • discussChapter 2. Value Is What We Want • 17  
  
[CONTEXT FROM PREVIOUS PAGE: Further reading:  
•Chapter 10, Value—What Is It. , on page 80  
•Chapter 11, Value—How Can We Measure It. , on page  
84  
report erratum • discussChapter 2. Value Is What We Want • 17]  
  
CHAPTER 3  
Guiding Goes Better  
“Feature by Feature”  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 3  
Guiding Goes Better  
“Feature by Feature”  
report erratum • discuss]  
  
The first thing we know about any project is the deadline—at  
least it always seems that way. That’s the vertical blue line  
with the triangle at the bottom. And what do we want by the deadline. Why, everything,  
of course. That’s the horizontal line. The star is our plan:  
have everything by the deadline. No problem. Somehow it doesn’t turn out that way. We usually wind up  
shipping less, or later, or both: the red lines with the question  
marks.

## Translated Text:

آیا افراد مورد نیاز، مهارت‌های مورد نیاز و کمک‌های مورد نیاز را دارند؟ ادامه دهید—ما درباره همه این موارد صحبت خواهیم کرد. نکته اصلی که باید به خاطر داشته باشید این است که بهترین نتایج از Deliver کردن نرم‌افزار، Feature به Feature به دست می‌آید. فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۶   
report erratum • discuss   
  
[متن از صفحه قبلی: بهترین ارزش از Featureهای کوچک و متمرکز بر ارزش که به صورت مکرر Deliver می‌شوند، به دست می‌آید. خوب، می‌توانیم ببینیم که Featureهای کوچک می‌توانند ارزش را زودتر Deliver کنند اگر بتوانیم آن‌ها را انجام دهیم. بیایید درباره مدیریت پروژه‌مان فکر کنیم. آیا نتایج کوچک و قابل مشاهده به مدیریت ما کمک می‌کنند؟ چگونه ممکن است مانع شوند؟ تیم‌های ما چطور؟ آیا آن‌ها سازماندهی شده‌اند تا به این روش کار کنند؟ آیا افراد مورد نیاز، مهارت‌های مورد نیاز و کمک‌های مورد نیاز را دارند؟ ادامه دهید—ما درباره همه این موارد صحبت خواهیم کرد. نکته اصلی که باید به خاطر داشته باشید این است که بهترین نتایج از Deliver کردن نرم‌افزار، Feature به Feature به دست می‌آید. فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۶   
report erratum • discuss]   
  
مطالعه بیشتر:   
• فصل ۱۰، ارزش—چیست؟، در صفحه ۸۰   
• فصل ۱۱، ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم؟، در صفحه ۸۴   
report erratum • discuss   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۷   
  
[متن از صفحه قبلی: مطالعه بیشتر:   
• فصل ۱۰، ارزش—چیست؟، در صفحه ۸۰   
• فصل ۱۱، ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم؟، در صفحه ۸۴   
report erratum • discuss   
فصل ۲. ارزش چیزی است که ما می‌خواهیم • ۱۷]   
  
فصل ۳   
راهنمایی بهتر پیش می‌رود   
“Feature به Feature”   
report erratum • discuss   
  
[متن از صفحه قبلی: فصل ۳   
راهنمایی بهتر پیش می‌رود   
“Feature به Feature”   
report erratum • discuss]   
  
اولین چیزی که درباره هر پروژه می‌دانیم Deadline است—حداقل همیشه اینطور به نظر می‌رسد. این خط عمودی آبی با مثلثی در پایین است. و چه چیزی را تا Deadline می‌خواهیم؟ خب، همه چیز، البته. این خط افقی است. ستاره برنامه ما است: داشتن همه چیز تا Deadline. مشکلی نیست. به نوعی اینطور پیش نمی‌رود. معمولاً با Deliver کردن کمتر، یا دیرتر، یا هر دو مواجه می‌شویم: خطوط قرمز با علامت‌های سوال.

================================================================================

# Section 26

## Original Text:

Heck, we’re sure to get less than we want. After all,  
we asked for everything. We really can’t have it all. Let’s manage that reality, not just  
let things happen. Let’s steer our project, not just ride it  
wherever it takes us. In a recent project of yours, what important things didn’t  
get done. What got done that turned out to be wasted. What  
did you find out about too late, or nearly so. report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 19  
  
[CONTEXT FROM PREVIOUS PAGE: The first thing we know about any project is the deadline—at  
least it always seems that way. That’s the vertical blue line  
with the triangle at the bottom. And what do we want by the deadline. Why, everything,  
of course. That’s the horizontal line. The star is our plan:  
have everything by the deadline. No problem. Somehow it doesn’t turn out that way. We usually wind up  
shipping less, or later, or both: the red lines with the question  
marks. Heck, we’re sure to get less than we want. After all,  
we asked for everything. We really can’t have it all. Let’s manage that reality, not just  
let things happen. Let’s steer our project, not just ride it  
wherever it takes us. In a recent project of yours, what important things didn’t  
get done. What got done that turned out to be wasted. What  
did you find out about too late, or nearly so. report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 19]  
  
Conventional software projects proceed in phases. Many projects plan with activity-based phases: Analysis,  
Design, Coding, and finally Testing. The green line is our  
plan for such a project, and it may look good. But even if we  
get Analysis done on time, that doesn’t tell us how well we’ll  
do on Design or Coding. Until we begin to see the software, we can’t really tell how  
well we’re doing. And when we start getting and testing  
that code, what happens. Generally nothing good. Have any of your projects given you too little time to react  
when trouble arose.

## Translated Text:

خب، مطمئن هستیم که کمتر از چیزی که می‌خواهیم به دست خواهیم آورد. به هر حال، ما همه چیز را خواستیم. واقعاً نمی‌توانیم همه چیز را داشته باشیم. بیایید این واقعیت را مدیریت کنیم، نه اینکه فقط اجازه دهیم اتفاقات بیفتند. بیایید پروژه خود را هدایت کنیم، نه اینکه فقط سوار آن شویم و ببینیم به کجا می‌رود. در یکی از پروژه‌های اخیر شما، چه چیزهای مهمی انجام نشدند؟ چه کارهایی انجام شدند که در نهایت بیهوده بودند؟ چه چیزی را خیلی دیر، یا تقریباً دیر، متوجه شدید؟   
گزارش خطا • بحث   
فصل ۳. هدایت بهتر پیش می‌رود "Feature by Feature" • ۱۹   
  
[متن از صفحه قبلی: اولین چیزی که درباره هر پروژه می‌دانیم، مهلت آن است—حداقل همیشه این‌طور به نظر می‌رسد. این همان خط آبی عمودی با مثلث در پایین است. و ما چه چیزی را تا مهلت می‌خواهیم؟ خب، همه چیز، البته. این همان خط افقی است. ستاره برنامه ما است: داشتن همه چیز تا مهلت. مشکلی نیست. به نوعی این‌طور پیش نمی‌رود. معمولاً ما با کمتر، یا دیرتر، یا هر دو مواجه می‌شویم: خطوط قرمز با علامت سؤال. خب، مطمئن هستیم که کمتر از چیزی که می‌خواهیم به دست خواهیم آورد. به هر حال، ما همه چیز را خواستیم. واقعاً نمی‌توانیم همه چیز را داشته باشیم. بیایید این واقعیت را مدیریت کنیم، نه اینکه فقط اجازه دهیم اتفاقات بیفتند. بیایید پروژه خود را هدایت کنیم، نه اینکه فقط سوار آن شویم و ببینیم به کجا می‌رود. در یکی از پروژه‌های اخیر شما، چه چیزهای مهمی انجام نشدند؟ چه کارهایی انجام شدند که در نهایت بیهوده بودند؟ چه چیزی را خیلی دیر، یا تقریباً دیر، متوجه شدید؟   
گزارش خطا • بحث   
فصل ۳. هدایت بهتر پیش می‌رود "Feature by Feature" • ۱۹]   
  
پروژه‌های نرم‌افزاری متعارف به صورت فازبندی پیش می‌روند. بسیاری از پروژه‌ها با فازهای مبتنی بر فعالیت برنامه‌ریزی می‌شوند: Analysis، Design، Coding، و در نهایت Testing. خط سبز برنامه ما برای چنین پروژه‌ای است و ممکن است خوب به نظر برسد. اما حتی اگر Analysis را به موقع انجام دهیم، این به ما نمی‌گوید که چقدر در Design یا Coding موفق خواهیم بود. تا زمانی که شروع به دیدن نرم‌افزار نکنیم، واقعاً نمی‌توانیم بگوییم چقدر خوب پیش می‌رویم. و وقتی شروع به دریافت و Testing آن کد می‌کنیم، چه اتفاقی می‌افتد؟ معمولاً هیچ چیز خوبی. آیا هیچ‌کدام از پروژه‌های شما زمان کافی برای واکنش نشان دادن هنگام بروز مشکل به شما داده‌اند؟

================================================================================

# Section 27

## Original Text:

Would there be value to knowing  
sooner what’s really going on. Did you ever wish you could  
get at least some value out of all that effort. Chapter 3. Guiding Goes Better “Feature by Feature” • 20  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Conventional software projects proceed in phases. Many projects plan with activity-based phases: Analysis,  
Design, Coding, and finally Testing. The green line is our  
plan for such a project, and it may look good. But even if we  
get Analysis done on time, that doesn’t tell us how well we’ll  
do on Design or Coding. Until we begin to see the software, we can’t really tell how  
well we’re doing. And when we start getting and testing  
that code, what happens. Generally nothing good. Have any of your projects given you too little time to react  
when trouble arose. Would there be value to knowing  
sooner what’s really going on. Did you ever wish you could  
get at least some value out of all that effort. Chapter 3. Guiding Goes Better “Feature by Feature” • 20  
report erratum • discuss]  
  
Worse yet, things rarely go according to plan. Finally, we begin to see and test the code. And the facts aren’t  
good. Inevitably we’re later than we thought. We have less  
done than we thought. What we have done doesn’t work  
very well. We knew we had asked for more than we could do: that’s  
the nature of goal setting. But by the time we find out where  
we are, it’s too late to do much about it. With more warning, maybe we could have shipped a subset  
on time. Now we have few choices. We could write the  
project off, but that would be career suicide. Or we can  
trudge gamely on, hoping to ship something before they  
give up on us. Either way, we look bad. Either way, it isbad. Have you ever had to ship in bad condition. Were there too  
many defects still in the software. Was the software too hard  
to change. Were important features missing. Important new  
ideas that it was too late to add. report erratum • discussChapter 3.

## Translated Text:

آیا دانستن زودتر از آنچه واقعاً در حال رخ دادن است، ارزشمند خواهد بود؟ آیا تا به حال آرزو کرده‌اید که بتوانید حداقل مقداری ارزش از تمام آن تلاش‌ها به دست آورید؟   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۰   
report erratum • discuss   
  
[متن از صفحه قبلی: پروژه‌های نرم‌افزاری متعارف به صورت مرحله‌ای پیش می‌روند. بسیاری از پروژه‌ها با مراحل مبتنی بر فعالیت برنامه‌ریزی می‌شوند: Analysis، Design، Coding و در نهایت Testing. خط سبز برنامه ما برای چنین پروژه‌ای است و ممکن است خوب به نظر برسد. اما حتی اگر Analysis را به موقع انجام دهیم، این به ما نمی‌گوید که در Design یا Coding چقدر خوب عمل خواهیم کرد. تا زمانی که شروع به دیدن نرم‌افزار نکنیم، واقعاً نمی‌توانیم بگوییم چقدر خوب پیش می‌رویم. و وقتی شروع به دریافت و Testing آن کد می‌کنیم، چه اتفاقی می‌افتد؟ معمولاً هیچ چیز خوبی نیست. آیا هیچ‌یک از پروژه‌های شما زمان کافی برای واکنش در هنگام بروز مشکل به شما نداده است؟ آیا دانستن زودتر از آنچه واقعاً در حال رخ دادن است، ارزشمند خواهد بود؟ آیا تا به حال آرزو کرده‌اید که بتوانید حداقل مقداری ارزش از تمام آن تلاش‌ها به دست آورید؟   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۰   
report erratum • discuss]   
  
بدتر از آن، اوضاع به ندرت طبق برنامه پیش می‌رود. در نهایت، ما شروع به دیدن و Testing کد می‌کنیم. و واقعیت‌ها خوب نیستند. به طور اجتناب‌ناپذیری، ما دیرتر از آنچه فکر می‌کردیم هستیم. کمتر از آنچه فکر می‌کردیم انجام داده‌ایم. آنچه انجام داده‌ایم خیلی خوب کار نمی‌کند. ما می‌دانستیم که بیش از آنچه می‌توانیم انجام دهیم درخواست کرده‌ایم: این ماهیت تعیین هدف است. اما زمانی که متوجه می‌شویم کجا هستیم، خیلی دیر شده است که کار زیادی انجام دهیم. با هشدار بیشتر، شاید می‌توانستیم یک زیرمجموعه را به موقع ارسال کنیم. اکنون گزینه‌های کمی داریم. می‌توانیم پروژه را کنار بگذاریم، اما این به معنای خودکشی شغلی است. یا می‌توانیم با شجاعت ادامه دهیم، به این امید که چیزی را ارسال کنیم قبل از اینکه دیگران از ما ناامید شوند. در هر صورت، ما بد به نظر می‌رسیم. در هر صورت، اوضاع بد است. آیا تا به حال مجبور شده‌اید که در شرایط بد چیزی را ارسال کنید؟ آیا نقص‌های زیادی هنوز در نرم‌افزار وجود داشت؟ آیا نرم‌افزار بیش از حد سخت برای تغییر بود؟ آیا ویژگی‌های مهمی وجود داشتند که از دست رفته بودند؟ ایده‌های جدید مهمی که برای اضافه کردن آن‌ها خیلی دیر شده بود؟   
report erratum • discuss   
فصل ۳.

================================================================================

# Section 28

## Original Text:

Guiding Goes Better “Feature by Feature” • 21  
  
[CONTEXT FROM PREVIOUS PAGE: Worse yet, things rarely go according to plan. Finally, we begin to see and test the code. And the facts aren’t  
good. Inevitably we’re later than we thought. We have less  
done than we thought. What we have done doesn’t work  
very well. We knew we had asked for more than we could do: that’s  
the nature of goal setting. But by the time we find out where  
we are, it’s too late to do much about it. With more warning, maybe we could have shipped a subset  
on time. Now we have few choices. We could write the  
project off, but that would be career suicide. Or we can  
trudge gamely on, hoping to ship something before they  
give up on us. Either way, we look bad. Either way, it isbad. Have you ever had to ship in bad condition. Were there too  
many defects still in the software. Was the software too hard  
to change. Were important features missing. Important new  
ideas that it was too late to add. report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 21]  
  
The activity-based product is a monolith. With a monolithic project, late in the game we can’t do much  
to cut costs. We have already written requirements for things  
we’ll never get. We’ve designed and even written code for  
things that we’ll never complete. All that work is wasted. If only we had known the truth. We could have deferred  
some of that work. We laid out this project with an all-or-nothing mentality. We analyzed it all. We designed it all. We tried to code it all. We discovered, too late, that we can’t have it all. Trying to plan and build it all has hurt us. We have no time  
to change, and even if we had time, we’d never untangle all  
the things we shouldn’t have done from the things we should  
have. Instead, let’s plan for multiple releases from the very  
beginning. Multiple releases are easier to manage and  
deliver value sooner. It’s even easier to build the software  
that way. Everyone wins.

## Translated Text:

راهنمایی بهتر می‌شود "ویژگی به ویژگی" • ۲۱  
  
[متن از صفحه قبلی: بدتر از آن، اوضاع به ندرت طبق برنامه پیش می‌رود. در نهایت، ما شروع به دیدن و Test کردن کد می‌کنیم. و حقایق خوب نیستند. به‌طور اجتناب‌ناپذیری، ما دیرتر از آنچه فکر می‌کردیم هستیم. کمتر از آنچه فکر می‌کردیم انجام داده‌ایم. آنچه انجام داده‌ایم خیلی خوب کار نمی‌کند. ما می‌دانستیم که بیشتر از توانمان درخواست کرده‌ایم: این ذات تعیین اهداف است. اما زمانی که متوجه می‌شویم کجا هستیم، خیلی دیر شده که کاری درباره آن انجام دهیم. اگر هشدار بیشتری داشتیم، شاید می‌توانستیم یک زیرمجموعه را به‌موقع ارسال کنیم. حالا انتخاب‌های کمی داریم. می‌توانیم پروژه را کنار بگذاریم، اما این کار خودکشی شغلی خواهد بود. یا می‌توانیم با شجاعت ادامه دهیم، به این امید که چیزی ارسال کنیم قبل از اینکه از ما ناامید شوند. در هر صورت، ما بد به نظر می‌رسیم. در هر صورت، اوضاع بد است. آیا تا به حال مجبور شده‌اید چیزی را در شرایط بد ارسال کنید؟ آیا نقص‌های زیادی هنوز در نرم‌افزار وجود داشت؟ آیا نرم‌افزار خیلی سخت برای تغییر بود؟ آیا ویژگی‌های مهمی از دست رفته بودند؟ ایده‌های جدید مهمی که خیلی دیر شده بود تا اضافه شوند؟]  
  
محصول مبتنی بر فعالیت یک Monolith است. با یک پروژه Monolithic، در مراحل پایانی نمی‌توانیم کار زیادی برای کاهش هزینه‌ها انجام دهیم. ما قبلاً برای چیزهایی که هرگز به دست نخواهیم آورد، نیازمندی نوشته‌ایم. ما طراحی کرده‌ایم و حتی کدی برای چیزهایی نوشته‌ایم که هرگز کامل نخواهند شد. تمام آن کارها هدر رفته است. ای کاش حقیقت را زودتر می‌دانستیم. می‌توانستیم بخشی از آن کارها را به تعویق بیندازیم. ما این پروژه را با یک ذهنیت همه یا هیچ برنامه‌ریزی کردیم. ما همه چیز را تحلیل کردیم. همه چیز را طراحی کردیم. تلاش کردیم همه چیز را کدنویسی کنیم. خیلی دیر متوجه شدیم که نمی‌توانیم همه چیز را داشته باشیم. تلاش برای برنامه‌ریزی و ساخت همه چیز به ما آسیب زده است. ما زمانی برای تغییر نداریم، و حتی اگر زمان داشتیم، هرگز نمی‌توانستیم همه چیزهایی که نباید انجام می‌دادیم را از چیزهایی که باید انجام می‌دادیم جدا کنیم. در عوض، بیایید از همان ابتدا برای چندین Release برنامه‌ریزی کنیم. چندین Release مدیریت را آسان‌تر می‌کند و زودتر ارزش ارائه می‌دهد. حتی ساخت نرم‌افزار به این روش آسان‌تر است. همه برنده می‌شوند.

================================================================================

# Section 29

## Original Text:

Do those plan, analyze, design-code-test phases really help  
you manage your project. Wouldn’t it be easier to manage  
things if you could just get the features, a few at a time, in  
the order you wanted them, starting right at the beginning. Let’s look at that. Chapter 3. Guiding Goes Better “Feature by Feature” • 22  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: The activity-based product is a monolith. With a monolithic project, late in the game we can’t do much  
to cut costs. We have already written requirements for things  
we’ll never get. We’ve designed and even written code for  
things that we’ll never complete. All that work is wasted. If only we had known the truth. We could have deferred  
some of that work. We laid out this project with an all-or-nothing mentality. We analyzed it all. We designed it all. We tried to code it all. We discovered, too late, that we can’t have it all. Trying to plan and build it all has hurt us. We have no time  
to change, and even if we had time, we’d never untangle all  
the things we shouldn’t have done from the things we should  
have. Instead, let’s plan for multiple releases from the very  
beginning. Multiple releases are easier to manage and  
deliver value sooner. It’s even easier to build the software  
that way. Everyone wins. Do those plan, analyze, design-code-test phases really help  
you manage your project. Wouldn’t it be easier to manage  
things if you could just get the features, a few at a time, in  
the order you wanted them, starting right at the beginning. Let’s look at that. Chapter 3. Guiding Goes Better “Feature by Feature” • 22  
report erratum • discuss]  
  
A project that delivers feature by feature is more predictable. We’ve seen that delivering release by release, feature by  
feature, lets us ship value sooner. What about our ability to  
manage and guide the effort. Our old red conventional project drones on and on, deliver-  
ing too little information, too late.

## Translated Text:

آیا این مراحل برنامه‌ریزی، تحلیل، طراحی-کدنویسی-تست واقعاً به شما در مدیریت پروژه کمک می‌کنند؟ آیا مدیریت کارها آسان‌تر نمی‌شد اگر می‌توانستید ویژگی‌ها را، یکی یکی و به ترتیبی که می‌خواهید، از همان ابتدا دریافت کنید؟ بیایید به این موضوع نگاه کنیم.   
فصل ۳. هدایت بهتر می‌شود "ویژگی به ویژگی" • ۲۲   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: محصول مبتنی بر فعالیت یک monolith است. در یک پروژه monolithic، در مراحل پایانی نمی‌توانیم کار زیادی برای کاهش هزینه‌ها انجام دهیم. ما قبلاً نیازمندی‌هایی را نوشته‌ایم که هرگز به دست نمی‌آوریم. ما طراحی کرده‌ایم و حتی کدی نوشته‌ایم برای چیزهایی که هرگز کامل نخواهند شد. تمام آن کارها هدر رفته است. اگر فقط حقیقت را می‌دانستیم، می‌توانستیم بخشی از آن کارها را به تعویق بیندازیم. ما این پروژه را با ذهنیتی "همه یا هیچ" برنامه‌ریزی کردیم. همه چیز را تحلیل کردیم. همه چیز را طراحی کردیم. سعی کردیم همه چیز را کدنویسی کنیم. خیلی دیر متوجه شدیم که نمی‌توانیم همه چیز را داشته باشیم. تلاش برای برنامه‌ریزی و ساخت همه چیز به ما آسیب زده است. زمانی برای تغییر نداریم، و حتی اگر زمان داشتیم، نمی‌توانستیم تمام چیزهایی که نباید انجام می‌دادیم را از چیزهایی که باید انجام می‌دادیم جدا کنیم. در عوض، بیایید از همان ابتدا برای چندین Release برنامه‌ریزی کنیم. چندین Release مدیریت را آسان‌تر می‌کند و ارزش را زودتر ارائه می‌دهد. حتی ساخت نرم‌افزار به این روش آسان‌تر است. همه برنده می‌شوند. آیا این مراحل برنامه‌ریزی، تحلیل، طراحی-کدنویسی-تست واقعاً به شما در مدیریت پروژه کمک می‌کنند؟ آیا مدیریت کارها آسان‌تر نمی‌شد اگر می‌توانستید ویژگی‌ها را، یکی یکی و به ترتیبی که می‌خواهید، از همان ابتدا دریافت کنید؟ بیایید به این موضوع نگاه کنیم.   
فصل ۳. هدایت بهتر می‌شود "ویژگی به ویژگی" • ۲۲   
گزارش خطا • بحث کنید]   
  
یک پروژه که ویژگی به ویژگی ارائه می‌دهد، قابل پیش‌بینی‌تر است. ما دیده‌ایم که ارائه Release به Release، ویژگی به ویژگی، به ما اجازه می‌دهد ارزش را زودتر ارائه دهیم. توانایی ما برای مدیریت و هدایت تلاش‌ها چطور؟ پروژه قدیمی و سنتی ما همچنان به کندی پیش می‌رود و اطلاعات بسیار کمی را، خیلی دیر ارائه می‌دهد.

================================================================================

# Section 30

## Original Text:

But the green project  
shows us real, valuable features at frequent intervals. We  
can see what is happening. We can see the software. Can you see how a flow of visible features would be easier  
to manage. Can you see how you could maximize project  
value as you go. What about risk. Can you see how to evaluate or reduce a  
project risk by building something visible. Can you see how  
to deal with a marketing risk with a small test feature. report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 23  
  
[CONTEXT FROM PREVIOUS PAGE: A project that delivers feature by feature is more predictable. We’ve seen that delivering release by release, feature by  
feature, lets us ship value sooner. What about our ability to  
manage and guide the effort. Our old red conventional project drones on and on, deliver-  
ing too little information, too late. But the green project  
shows us real, valuable features at frequent intervals. We  
can see what is happening. We can see the software. Can you see how a flow of visible features would be easier  
to manage. Can you see how you could maximize project  
value as you go. What about risk. Can you see how to evaluate or reduce a  
project risk by building something visible. Can you see how  
to deal with a marketing risk with a small test feature. report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 23]  
  
Feature by feature gives better information, better guidance, better  
results. When we build our software projects feature by feature,  
things go better. We can see how much is done and how  
rapidly the project is progressing. We get a good sense of  
how much will be done by any given date. We choose the most important features to do next. We build  
the best possible combination of features for any desired  
shipment date—even one earlier than our original desired  
date. We can even change features, adding new ones in  
response to better ideas or changing user needs.

## Translated Text:

اما پروژه سبز   
به ما ویژگی‌های واقعی و ارزشمند را در فواصل زمانی مکرر نشان می‌دهد.   
ما می‌توانیم ببینیم چه اتفاقی در حال رخ دادن است.   
ما می‌توانیم نرم‌افزار را ببینیم.   
آیا می‌توانید ببینید که چگونه یک جریان از ویژگی‌های قابل مشاهده مدیریت را آسان‌تر می‌کند؟   
آیا می‌توانید ببینید که چگونه می‌توانید ارزش پروژه را در حین پیشرفت به حداکثر برسانید؟   
در مورد ریسک چطور؟   
آیا می‌توانید ببینید که چگونه می‌توان یک ریسک پروژه را با ساختن چیزی قابل مشاهده ارزیابی یا کاهش داد؟   
آیا می‌توانید ببینید که چگونه می‌توان با یک ویژگی تست کوچک، یک ریسک بازاریابی را مدیریت کرد؟   
  
گزارش خطا • بحث   
فصل ۳. راهنمایی بهتر می‌شود "ویژگی به ویژگی" • ۲۳   
  
[متن از صفحه قبلی: پروژه‌ای که ویژگی به ویژگی تحویل می‌دهد قابل پیش‌بینی‌تر است.   
ما دیده‌ایم که تحویل Release به Release و ویژگی به ویژگی، به ما اجازه می‌دهد زودتر ارزش را ارائه دهیم.   
در مورد توانایی ما برای مدیریت و هدایت تلاش‌ها چطور؟   
پروژه قدیمی قرمز و متعارف ما همچنان ادامه دارد و اطلاعات بسیار کمی را با تأخیر زیاد ارائه می‌دهد.   
اما پروژه سبز به ما ویژگی‌های واقعی و ارزشمند را در فواصل زمانی مکرر نشان می‌دهد.   
ما می‌توانیم ببینیم چه اتفاقی در حال رخ دادن است.   
ما می‌توانیم نرم‌افزار را ببینیم.   
آیا می‌توانید ببینید که چگونه یک جریان از ویژگی‌های قابل مشاهده مدیریت را آسان‌تر می‌کند؟   
آیا می‌توانید ببینید که چگونه می‌توانید ارزش پروژه را در حین پیشرفت به حداکثر برسانید؟   
در مورد ریسک چطور؟   
آیا می‌توانید ببینید که چگونه می‌توان یک ریسک پروژه را با ساختن چیزی قابل مشاهده ارزیابی یا کاهش داد؟   
آیا می‌توانید ببینید که چگونه می‌توان با یک ویژگی تست کوچک، یک ریسک بازاریابی را مدیریت کرد؟   
  
گزارش خطا • بحث   
فصل ۳. راهنمایی بهتر می‌شود "ویژگی به ویژگی" • ۲۳]   
  
ویژگی به ویژگی اطلاعات بهتر، راهنمایی بهتر و نتایج بهتری ارائه می‌دهد.   
وقتی پروژه‌های نرم‌افزاری خود را ویژگی به ویژگی می‌سازیم، اوضاع بهتر پیش می‌رود.   
ما می‌توانیم ببینیم چه مقدار انجام شده و پروژه با چه سرعتی در حال پیشرفت است.   
ما درک خوبی از این داریم که چه مقدار تا هر تاریخ مشخصی انجام خواهد شد.   
ما مهم‌ترین ویژگی‌ها را برای انجام بعدی انتخاب می‌کنیم.   
ما بهترین ترکیب ممکن از ویژگی‌ها را برای هر تاریخ ارسال مورد نظر می‌سازیم—حتی تاریخی زودتر از تاریخ مورد نظر اولیه ما.   
ما حتی می‌توانیم ویژگی‌ها را تغییر دهیم و ویژگی‌های جدیدی را در پاسخ به ایده‌های بهتر یا نیازهای در حال تغییر کاربران اضافه کنیم.

================================================================================

# Section 31

## Original Text:

When our projects grow feature by feature, we can respond  
to what’s really happening. We can respond to the changing  
needs and inputs of the business and of management. What would it take to make this way of working possible. How can we plan a project when we don’t even know what  
we’ll wind up wanting to do. Chapter 3. Guiding Goes Better “Feature by Feature” • 24  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Feature by feature gives better information, better guidance, better  
results. When we build our software projects feature by feature,  
things go better. We can see how much is done and how  
rapidly the project is progressing. We get a good sense of  
how much will be done by any given date. We choose the most important features to do next. We build  
the best possible combination of features for any desired  
shipment date—even one earlier than our original desired  
date. We can even change features, adding new ones in  
response to better ideas or changing user needs. When our projects grow feature by feature, we can respond  
to what’s really happening. We can respond to the changing  
needs and inputs of the business and of management. What would it take to make this way of working possible. How can we plan a project when we don’t even know what  
we’ll wind up wanting to do. Chapter 3. Guiding Goes Better “Feature by Feature” • 24  
report erratum • discuss]  
  
Further reading:  
•Chapter 14, Creating Teams That Thrive , on page 94  
•Chapter 16, Managing Natural Software Development , on  
page 100  
report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 25  
  
[CONTEXT FROM PREVIOUS PAGE: Further reading:  
•Chapter 14, Creating Teams That Thrive , on page 94  
•Chapter 16, Managing Natural Software Development , on  
page 100  
report erratum • discussChapter 3. Guiding Goes Better “Feature by Feature” • 25]  
  
Building features requires multiple skills. CHAPTER 4  
Organizing by Feature  
We want to get value in small bites: features.

## Translated Text:

وقتی پروژه‌های ما به صورت feature به feature رشد می‌کنند، می‌توانیم به آنچه واقعاً اتفاق می‌افتد پاسخ دهیم. ما می‌توانیم به نیازها و ورودی‌های در حال تغییر کسب‌وکار و مدیریت پاسخ دهیم. چه چیزی لازم است تا این روش کاری ممکن شود؟ چگونه می‌توانیم یک پروژه را برنامه‌ریزی کنیم وقتی حتی نمی‌دانیم در نهایت چه چیزی می‌خواهیم انجام دهیم؟   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۴   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: انجام کار به صورت feature به feature اطلاعات بهتر، راهنمایی بهتر و نتایج بهتری ارائه می‌دهد. وقتی پروژه‌های نرم‌افزاری خود را به صورت feature به feature می‌سازیم، همه چیز بهتر پیش می‌رود. ما می‌توانیم ببینیم چه مقدار انجام شده و پروژه با چه سرعتی در حال پیشرفت است. ما درک خوبی از این داریم که چه مقدار تا یک تاریخ مشخص انجام خواهد شد. ما مهم‌ترین ویژگی‌ها را برای انجام بعدی انتخاب می‌کنیم. ما بهترین ترکیب ممکن از ویژگی‌ها را برای هر تاریخ ارسال دلخواه می‌سازیم—حتی تاریخی زودتر از تاریخ دلخواه اولیه ما. ما حتی می‌توانیم ویژگی‌ها را تغییر دهیم، ویژگی‌های جدیدی را در پاسخ به ایده‌های بهتر یا نیازهای در حال تغییر کاربران اضافه کنیم. وقتی پروژه‌های ما به صورت feature به feature رشد می‌کنند، می‌توانیم به آنچه واقعاً اتفاق می‌افتد پاسخ دهیم. ما می‌توانیم به نیازها و ورودی‌های در حال تغییر کسب‌وکار و مدیریت پاسخ دهیم. چه چیزی لازم است تا این روش کاری ممکن شود؟ چگونه می‌توانیم یک پروژه را برنامه‌ریزی کنیم وقتی حتی نمی‌دانیم در نهایت چه چیزی می‌خواهیم انجام دهیم؟   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۴   
گزارش خطا • بحث کنید]   
  
مطالعه بیشتر:   
• فصل ۱۴، ایجاد تیم‌هایی که رشد می‌کنند، در صفحه ۹۴   
• فصل ۱۶، مدیریت توسعه نرم‌افزار طبیعی، در صفحه ۱۰۰   
گزارش خطا • بحث کنید   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۵   
  
[متن از صفحه قبلی: مطالعه بیشتر:   
• فصل ۱۴، ایجاد تیم‌هایی که رشد می‌کنند، در صفحه ۹۴   
• فصل ۱۶، مدیریت توسعه نرم‌افزار طبیعی، در صفحه ۱۰۰   
گزارش خطا • بحث کنید   
فصل ۳. هدایت بهتر می‌شود "Feature by Feature" • ۲۵]   
  
ساختن ویژگی‌ها نیازمند مهارت‌های متعددی است.   
فصل ۴   
سازماندهی بر اساس Feature   
ما می‌خواهیم ارزش را در بخش‌های کوچک به دست آوریم: ویژگی‌ها.

================================================================================

# Section 32

## Original Text:

We prosper  
when we manage in terms of value, in terms of features. How can we organize our work, and ourselves, for the best  
and most rapid flow of value. To get the work done, different parts require different skills. The work won’t be done—or at least not done well—until  
it has had the attention of people with each needed skill. If we organize teams by skill-set, each piece of work will  
need to be passed around among teams. Each handoff will  
require scheduling and cause delays. Quite likely, problems  
will arise from each handoff. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Building features requires multiple skills. CHAPTER 4  
Organizing by Feature  
We want to get value in small bites: features. We prosper  
when we manage in terms of value, in terms of features. How can we organize our work, and ourselves, for the best  
and most rapid flow of value. To get the work done, different parts require different skills. The work won’t be done—or at least not done well—until  
it has had the attention of people with each needed skill. If we organize teams by skill-set, each piece of work will  
need to be passed around among teams. Each handoff will  
require scheduling and cause delays. Quite likely, problems  
will arise from each handoff. report erratum • discuss]  
  
Teams build features. The answer is simple: organize into small teams, each of  
which builds features that the Product Champions can  
understand. Make sure that each team has all the people  
and all the skills necessary to build the entire feature, not  
just part of it. The advantages of this should be clear: we can allocate work  
across teams easily. We can see where everything is. Each  
feature gets dedicated attention. Responsibility and author-  
ity are aligned. It’s simple and works well. But it’s not that easy, is it. report erratum • discussChapter 4. Organizing by Feature • 27  
  
[CONTEXT FROM PREVIOUS PAGE: Teams build features.

## Translated Text:

ما پیشرفت می‌کنیم   
زمانی که مدیریت را بر اساس ارزش و ویژگی‌ها انجام دهیم. چگونه می‌توانیم کار خود و خودمان را به گونه‌ای سازماندهی کنیم که بهترین و سریع‌ترین جریان ارزش را داشته باشیم؟ برای انجام کار، بخش‌های مختلف به مهارت‌های متفاوتی نیاز دارند. کار انجام نخواهد شد—یا حداقل به خوبی انجام نخواهد شد—تا زمانی که توجه افرادی با هر مهارت مورد نیاز به آن جلب شود. اگر تیم‌ها را بر اساس مجموعه مهارت‌ها سازماندهی کنیم، هر بخش از کار باید بین تیم‌ها منتقل شود. هر انتقال نیاز به زمان‌بندی دارد و باعث تأخیر می‌شود. به احتمال زیاد، مشکلاتی از هر انتقال به وجود خواهد آمد. گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: ایجاد ویژگی‌ها به مهارت‌های متعددی نیاز دارد. فصل ۴   
سازماندهی بر اساس ویژگی   
ما می‌خواهیم ارزش را در بخش‌های کوچک دریافت کنیم: ویژگی‌ها. ما پیشرفت می‌کنیم   
زمانی که مدیریت را بر اساس ارزش و ویژگی‌ها انجام دهیم. چگونه می‌توانیم کار خود و خودمان را به گونه‌ای سازماندهی کنیم که بهترین و سریع‌ترین جریان ارزش را داشته باشیم؟ برای انجام کار، بخش‌های مختلف به مهارت‌های متفاوتی نیاز دارند. کار انجام نخواهد شد—یا حداقل به خوبی انجام نخواهد شد—تا زمانی که توجه افرادی با هر مهارت مورد نیاز به آن جلب شود. اگر تیم‌ها را بر اساس مجموعه مهارت‌ها سازماندهی کنیم، هر بخش از کار باید بین تیم‌ها منتقل شود. هر انتقال نیاز به زمان‌بندی دارد و باعث تأخیر می‌شود. به احتمال زیاد، مشکلاتی از هر انتقال به وجود خواهد آمد. گزارش خطا • بحث کنید]   
  
تیم‌ها ویژگی‌ها را می‌سازند. پاسخ ساده است: به تیم‌های کوچک سازماندهی شوید، که هر یک ویژگی‌هایی را بسازند که Product Champions بتوانند آن را درک کنند. اطمینان حاصل کنید که هر تیم تمام افراد و تمام مهارت‌های لازم برای ساخت کل ویژگی را دارد، نه فقط بخشی از آن. مزایای این کار باید واضح باشد: ما می‌توانیم کار را به راحتی بین تیم‌ها تخصیص دهیم. می‌توانیم ببینیم هر چیزی کجاست. هر ویژگی توجه اختصاصی دریافت می‌کند. مسئولیت و اختیار هماهنگ هستند. این ساده است و خوب کار می‌کند. اما این به آن آسانی نیست، این‌طور نیست؟ گزارش خطا • بحث کنید فصل ۴. سازماندهی بر اساس ویژگی • ۲۷   
  
[متن از صفحه قبلی: تیم‌ها ویژگی‌ها را می‌سازند.]

================================================================================

# Section 33

## Original Text:

The answer is simple: organize into small teams, each of  
which builds features that the Product Champions can  
understand. Make sure that each team has all the people  
and all the skills necessary to build the entire feature, not  
just part of it. The advantages of this should be clear: we can allocate work  
across teams easily. We can see where everything is. Each  
feature gets dedicated attention. Responsibility and author-  
ity are aligned. It’s simple and works well. But it’s not that easy, is it. report erratum • discussChapter 4. Organizing by Feature • 27]  
  
But…but…but we’re not organized that way. I know. And I’m here to say that you probably should be. If each feature that you want must be passed through multi-  
ple teams, it takes longer and results in lower quality. Why. Because the teams need to be coordinated somehow, and  
each item has to be passed from one team to the next. After  
it’s passed, it needs to sit in the queue for the next team to  
wait its turn. And often—very often—the feature needs to  
go back to the first team to fix something they didn’t  
understand. Often it will go back and forth several times. This slows you down. Yes, feature teams may be a change for you. But if you want  
to go rapidly and smoothly, you’ll very likely benefit from  
moving in that direction. Take your time, don’t panic, but  
give it a try. Create a Feature Team, a darn good one. See  
whether fewer handoffs speed delivery and improve quality. I’m betting things will go better. If so, rinse, repeat. Chapter 4. Organizing by Feature • 28  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: But…but…but we’re not organized that way. I know. And I’m here to say that you probably should be. If each feature that you want must be passed through multi-  
ple teams, it takes longer and results in lower quality. Why. Because the teams need to be coordinated somehow, and  
each item has to be passed from one team to the next.

## Translated Text:

پاسخ ساده است: تیم‌های کوچک تشکیل دهید که هر کدام قابلیت‌هایی را بسازند که Product Champions بتوانند آن‌ها را درک کنند. مطمئن شوید که هر تیم تمام افراد و مهارت‌های لازم برای ساخت کل قابلیت را دارد، نه فقط بخشی از آن. مزایای این کار باید واضح باشد: ما می‌توانیم کارها را به‌راحتی بین تیم‌ها تخصیص دهیم. می‌توانیم ببینیم هر چیزی کجاست. هر قابلیت توجه اختصاصی دریافت می‌کند. مسئولیت و اختیار همسو می‌شوند. این ساده است و به‌خوبی کار می‌کند. اما این کار چندان آسان نیست، این‌طور نیست؟   
   
[report erratum • discuss]   
[Chapter 4. Organizing by Feature • 27]   
  
اما... اما... اما ما این‌طور سازماندهی نشده‌ایم. می‌دانم. و اینجا هستم تا بگویم که احتمالاً باید این‌طور باشید. اگر هر قابلیتی که می‌خواهید باید از چندین تیم عبور کند، زمان بیشتری می‌برد و کیفیت پایین‌تری خواهد داشت. چرا؟ چون تیم‌ها باید به نوعی هماهنگ شوند و هر آیتم باید از یک تیم به تیم دیگر منتقل شود. بعد از انتقال، باید در صف تیم بعدی منتظر نوبت خود بماند. و اغلب—خیلی اغلب—قابلیت باید به تیم اول بازگردد تا چیزی را که متوجه نشده‌اند اصلاح کنند. اغلب این فرآیند چندین بار رفت و برگشت خواهد داشت. این کار شما را کند می‌کند.   
  
بله، تیم‌های قابلیت ممکن است تغییری برای شما باشند. اما اگر می‌خواهید سریع و روان پیش بروید، به احتمال زیاد از حرکت در این مسیر سود خواهید برد. وقت بگذارید، نگران نباشید، اما آن را امتحان کنید. یک Feature Team ایجاد کنید، یک تیم واقعاً خوب. ببینید آیا کاهش انتقال‌ها باعث تسریع در تحویل و بهبود کیفیت می‌شود یا خیر. شرط می‌بندم که اوضاع بهتر خواهد شد. اگر چنین بود، این کار را تکرار کنید.   
  
[Chapter 4. Organizing by Feature • 28]   
[report erratum • discuss]

================================================================================

# Section 34

## Original Text:

After  
it’s passed, it needs to sit in the queue for the next team to  
wait its turn. And often—very often—the feature needs to  
go back to the first team to fix something they didn’t  
understand. Often it will go back and forth several times. This slows you down. Yes, feature teams may be a change for you. But if you want  
to go rapidly and smoothly, you’ll very likely benefit from  
moving in that direction. Take your time, don’t panic, but  
give it a try. Create a Feature Team, a darn good one. See  
whether fewer handoffs speed delivery and improve quality. I’m betting things will go better. If so, rinse, repeat. Chapter 4. Organizing by Feature • 28  
report erratum • discuss]  
  
We don’t have enough specialists. Maybe you don’t have enough database experts or user  
experience (UX) experts. You couldn’t possibly have one on  
every feature team. You might wind up with more things  
to do than teams to do them. It seems you can’t create that  
next feature team that you need. Well, maybe. I’ll bet you have people who understand  
databases rather well even though you wouldn’t call them  
“experts. ” You might even have some people who are experts  
and don’t have the badge. You probably have plenty of  
people who can build an interface and get close to what your  
best UX people can do. They might even do better. So here’s a thought experiment: what if you create teams to  
build features, in order of most important features first. Since the team you’re working on now has the most impor-  
tant features to do that aren’t already being done, they  
clearly deserve the best database person and UX person you  
have next. Put them in there. Rinse, repeat. Soon, you might find yourself creating a team whose  
database person, or UX person, really isn’t quite good  
enough. Super. You’ve identified a training opportunity. A  
learning opportunity. An opportunity to form a guild, a  
Community of Practice. report erratum • discussChapter 4.

## Translated Text:

پس از اینکه عبور کرد، باید در صف برای تیم بعدی منتظر نوبت خود بماند. و اغلب—خیلی اوقات—Feature باید به تیم اول بازگردد تا چیزی را که متوجه نشده‌اند، اصلاح کنند. معمولاً این فرآیند چندین بار بین تیم‌ها رفت و برگشت خواهد داشت. این موضوع شما را کند می‌کند. بله، Feature Teams ممکن است تغییری برای شما باشد. اما اگر می‌خواهید سریع و روان پیش بروید، احتمالاً از حرکت در این مسیر سود خواهید برد. زمان بگذارید، وحشت نکنید، اما امتحانش کنید. یک Feature Team ایجاد کنید، یک تیم واقعاً خوب. ببینید آیا کاهش handoffها باعث تسریع در تحویل و بهبود کیفیت می‌شود یا خیر. من شرط می‌بندم که اوضاع بهتر خواهد شد. اگر چنین بود، تکرار کنید.   
  
فصل ۴. سازماندهی بر اساس Feature • ۲۸   
report erratum • discuss  
  
ما به اندازه کافی متخصص نداریم. شاید شما به اندازه کافی متخصص Database یا User Experience (UX) نداشته باشید. شما نمی‌توانید در هر Feature Team یک متخصص داشته باشید. ممکن است کارهای بیشتری نسبت به تعداد تیم‌ها برای انجام آن‌ها داشته باشید. به نظر می‌رسد نمی‌توانید آن Feature Team بعدی که نیاز دارید را ایجاد کنید. خب، شاید. من شرط می‌بندم که افرادی دارید که Database را به‌خوبی درک می‌کنند حتی اگر آن‌ها را "متخصص" ننامید. شاید حتی افرادی داشته باشید که متخصص هستند اما نشان تخصص ندارند. احتمالاً افراد زیادی دارید که می‌توانند یک Interface بسازند و به چیزی نزدیک شوند که بهترین افراد UX شما می‌توانند انجام دهند. شاید حتی بهتر عمل کنند.   
  
پس این یک آزمایش فکری است: چه می‌شود اگر تیم‌هایی برای ساخت Featureها ایجاد کنید، به ترتیب از مهم‌ترین Featureها شروع کنید. از آنجایی که تیمی که اکنون روی آن کار می‌کنید مهم‌ترین Featureهایی را دارد که هنوز انجام نشده‌اند، آن‌ها به وضوح شایسته بهترین فرد Database و UX شما هستند. آن‌ها را در آنجا قرار دهید. تکرار کنید. به زودی ممکن است خود را در حال ایجاد تیمی بیابید که فرد Database یا UX آن واقعاً به اندازه کافی خوب نیست. عالی. شما یک فرصت آموزشی شناسایی کرده‌اید. یک فرصت یادگیری. یک فرصت برای تشکیل یک Guild، یک Community of Practice.   
  
report erratum • discuss   
Chapter 4.

================================================================================

# Section 35

## Original Text:

Organizing by Feature • 29  
  
[CONTEXT FROM PREVIOUS PAGE: We don’t have enough specialists. Maybe you don’t have enough database experts or user  
experience (UX) experts. You couldn’t possibly have one on  
every feature team. You might wind up with more things  
to do than teams to do them. It seems you can’t create that  
next feature team that you need. Well, maybe. I’ll bet you have people who understand  
databases rather well even though you wouldn’t call them  
“experts. ” You might even have some people who are experts  
and don’t have the badge. You probably have plenty of  
people who can build an interface and get close to what your  
best UX people can do. They might even do better. So here’s a thought experiment: what if you create teams to  
build features, in order of most important features first. Since the team you’re working on now has the most impor-  
tant features to do that aren’t already being done, they  
clearly deserve the best database person and UX person you  
have next. Put them in there. Rinse, repeat. Soon, you might find yourself creating a team whose  
database person, or UX person, really isn’t quite good  
enough. Super. You’ve identified a training opportunity. A  
learning opportunity. An opportunity to form a guild, a  
Community of Practice. report erratum • discussChapter 4. Organizing by Feature • 29]  
  
Create Communities of Practice. Now create a Database Community of Practice, or UX  
Community of Practice. Build it around the people who used  
to be in the database or UX group, and include all the people  
who are now charged with doing those things on your fea-  
ture teams. This isn’t a new department: the people belong  
to the feature teams. They are also members of the Database or UX Community  
of Practice. You belong to your family; you’re a member of  
the golf club. Your senior people—not always the ones you thought were  
the senior people—now have an additional responsibility:  
bringing the less experienced people up to full speed.

## Translated Text:

سازماندهی بر اساس ویژگی • ۲۹  
  
[متن از صفحه قبلی: ما به اندازه کافی متخصص نداریم. شاید شما به اندازه کافی متخصص Database یا متخصصین تجربه کاربری (UX) نداشته باشید. شما نمی‌توانید در هر تیم ویژگی یک متخصص داشته باشید. ممکن است کارهای بیشتری برای انجام داشته باشید تا تیم‌هایی که بتوانند آنها را انجام دهند. به نظر می‌رسد نمی‌توانید تیم ویژگی بعدی که نیاز دارید را ایجاد کنید. خب، شاید. من شرط می‌بندم که افرادی دارید که Database را به‌خوبی درک می‌کنند، حتی اگر آنها را "متخصص" ننامید. شاید حتی برخی افراد داشته باشید که متخصص هستند اما نشان تخصص ندارند. احتمالاً افراد زیادی دارید که می‌توانند یک رابط کاربری بسازند و به چیزی نزدیک به بهترین افراد UX شما برسند. شاید حتی بهتر عمل کنند. بنابراین، این یک آزمایش فکری است: چه می‌شود اگر تیم‌هایی برای ساخت ویژگی‌ها ایجاد کنید، به ترتیب از مهم‌ترین ویژگی‌ها شروع کنید. از آنجایی که تیمی که اکنون روی آن کار می‌کنید مهم‌ترین ویژگی‌هایی را دارد که هنوز انجام نشده‌اند، آنها به وضوح شایسته بهترین فرد Database و بهترین فرد UX شما هستند. آنها را به آنجا بفرستید. این کار را تکرار کنید. به زودی ممکن است خود را در حال ایجاد تیمی بیابید که فرد Database یا فرد UX آن واقعاً به اندازه کافی خوب نیست. عالی است. شما یک فرصت آموزشی شناسایی کرده‌اید. یک فرصت یادگیری. یک فرصت برای تشکیل یک انجمن، یک Community of Practice. گزارش خطا • بحث فصل ۴. سازماندهی بر اساس ویژگی • ۲۹]  
  
ایجاد Communities of Practice. اکنون یک Database Community of Practice یا یک UX Community of Practice ایجاد کنید. آن را حول افرادی که قبلاً در گروه Database یا UX بودند بسازید و همه افرادی را که اکنون مسئول انجام این کارها در تیم‌های ویژگی شما هستند، شامل کنید. این یک بخش جدید نیست: افراد به تیم‌های ویژگی تعلق دارند. آنها همچنین اعضای Database یا UX Community of Practice هستند. شما به خانواده خود تعلق دارید؛ شما عضو باشگاه گلف هستید. افراد ارشد شما—که همیشه همان افرادی نیستند که فکر می‌کردید ارشد هستند—اکنون یک مسئولیت اضافی دارند: رساندن افراد کم‌تجربه به سطح کامل.

================================================================================

# Section 36

## Original Text:

A  
highly paid expert shouldn’t be highly paid just because  
she’s an expert. She should be highly paid because she is  
helping other people become experts. Your top people shepherd the less experienced ones. They  
drop into other teams and help them, making sure that the  
real team member retains responsibility and learns what  
needs to be done and how to do it. Soon you’ll have all the  
experts you need—and happier people as well. Chapter 4. Organizing by Feature • 30  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Create Communities of Practice. Now create a Database Community of Practice, or UX  
Community of Practice. Build it around the people who used  
to be in the database or UX group, and include all the people  
who are now charged with doing those things on your fea-  
ture teams. This isn’t a new department: the people belong  
to the feature teams. They are also members of the Database or UX Community  
of Practice. You belong to your family; you’re a member of  
the golf club. Your senior people—not always the ones you thought were  
the senior people—now have an additional responsibility:  
bringing the less experienced people up to full speed. A  
highly paid expert shouldn’t be highly paid just because  
she’s an expert. She should be highly paid because she is  
helping other people become experts. Your top people shepherd the less experienced ones. They  
drop into other teams and help them, making sure that the  
real team member retains responsibility and learns what  
needs to be done and how to do it. Soon you’ll have all the  
experts you need—and happier people as well. Chapter 4. Organizing by Feature • 30  
report erratum • discuss]  
  
Feature teams make “scaling” easy. We have a chapter on “scaling” coming up in Part II. The  
short message is this: much of your work can be done by a  
single cross-functional team. Most of the rest can be done  
by a number of feature teams working in parallel on features  
that you understand and desire.

## Translated Text:

یک کارشناس با حقوق بالا نباید فقط به این دلیل که کارشناس است، حقوق بالایی دریافت کند. او باید به این دلیل حقوق بالایی دریافت کند که به دیگران کمک می‌کند تا کارشناس شوند. افراد برتر شما افراد کم‌تجربه‌تر را هدایت می‌کنند. آنها به تیم‌های دیگر سر می‌زنند و به آنها کمک می‌کنند، در حالی که اطمینان حاصل می‌کنند که عضو واقعی تیم مسئولیت را حفظ کرده و یاد می‌گیرد چه کاری باید انجام شود و چگونه آن را انجام دهد. به زودی تمام کارشناسانی که نیاز دارید را خواهید داشت—و افراد شادتری نیز خواهید داشت.  
  
فصل ۴. سازماندهی بر اساس قابلیت • ۳۰   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: جوامع عملی را ایجاد کنید. حالا یک Database Community of Practice یا UX Community of Practice ایجاد کنید. آن را حول افرادی که قبلاً در گروه Database یا UX بودند بسازید و تمام افرادی که اکنون مسئول انجام این کارها در تیم‌های قابلیت شما هستند را شامل کنید. این یک بخش جدید نیست: افراد به تیم‌های قابلیت تعلق دارند. آنها همچنین اعضای Database یا UX Community of Practice هستند. شما به خانواده خود تعلق دارید؛ شما عضو باشگاه گلف هستید. افراد ارشد شما—که همیشه همان افرادی نیستند که فکر می‌کردید افراد ارشد هستند—اکنون یک مسئولیت اضافی دارند: رساندن افراد کم‌تجربه به سطح کامل. یک کارشناس با حقوق بالا نباید فقط به این دلیل که کارشناس است، حقوق بالایی دریافت کند. او باید به این دلیل حقوق بالایی دریافت کند که به دیگران کمک می‌کند تا کارشناس شوند. افراد برتر شما افراد کم‌تجربه‌تر را هدایت می‌کنند. آنها به تیم‌های دیگر سر می‌زنند و به آنها کمک می‌کنند، در حالی که اطمینان حاصل می‌کنند که عضو واقعی تیم مسئولیت را حفظ کرده و یاد می‌گیرد چه کاری باید انجام شود و چگونه آن را انجام دهد. به زودی تمام کارشناسانی که نیاز دارید را خواهید داشت—و افراد شادتری نیز خواهید داشت. فصل ۴. سازماندهی بر اساس قابلیت • ۳۰   
گزارش خطا • بحث کنید]   
  
تیم‌های قابلیت "scaling" را آسان می‌کنند. ما یک فصل درباره "scaling" در بخش دوم داریم. پیام کوتاه این است: بخش زیادی از کار شما می‌تواند توسط یک تیم cross-functional انجام شود. بیشتر باقی‌مانده کارها را می‌توان توسط تعدادی تیم قابلیت که به صورت موازی روی قابلیت‌هایی که شما آنها را درک می‌کنید و می‌خواهید کار می‌کنند، انجام داد.

================================================================================

# Section 37

## Original Text:

There may be very little in your organization that really  
requires multiple teams working in lockstep. If you’ll break  
out your work feature by feature, you’ll quite likely discover  
that what’s left is simple. This will save you from spending  
a fortune “scaling” into a structure and organization that  
will not serve you. Organize by feature. You’ll be glad you did. report erratum • discussChapter 4. Organizing by Feature • 31  
  
[CONTEXT FROM PREVIOUS PAGE: Feature teams make “scaling” easy. We have a chapter on “scaling” coming up in Part II. The  
short message is this: much of your work can be done by a  
single cross-functional team. Most of the rest can be done  
by a number of feature teams working in parallel on features  
that you understand and desire. There may be very little in your organization that really  
requires multiple teams working in lockstep. If you’ll break  
out your work feature by feature, you’ll quite likely discover  
that what’s left is simple. This will save you from spending  
a fortune “scaling” into a structure and organization that  
will not serve you. Organize by feature. You’ll be glad you did. report erratum • discussChapter 4. Organizing by Feature • 31]  
  
Moving from vision to details  
CHAPTER 5  
Planning Feature by Feature  
Things go best with frequent releases of software. Value  
grows faster and better. Management gets visible progress  
at short intervals. Development works best with small, clear  
objectives. The product “vision,” however, starts with big, grand ideas,  
vague yet enticing. Vision is about big ideas, not tiny bites. How can we move from our grand product ideas down to  
those detailed features we need for the best visibility and  
control. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Moving from vision to details  
CHAPTER 5  
Planning Feature by Feature  
Things go best with frequent releases of software. Value  
grows faster and better. Management gets visible progress  
at short intervals.

## Translated Text:

ممکن است در سازمان شما موارد بسیار کمی وجود داشته باشد که واقعاً نیاز به کار چندین تیم به صورت هماهنگ داشته باشد. اگر کار خود را به صورت feature به feature تقسیم کنید، به احتمال زیاد متوجه خواهید شد که آنچه باقی می‌ماند ساده است. این کار شما را از صرف هزینه زیاد برای “scaling” به یک ساختار و سازمانی که به شما خدمت نخواهد کرد، نجات می‌دهد. بر اساس feature سازماندهی کنید. خوشحال خواهید شد که این کار را انجام داده‌اید.   
report erratum • discuss   
Chapter 4. Organizing by Feature • 31   
  
[CONTEXT FROM PREVIOUS PAGE: تیم‌های feature “scaling” را آسان می‌کنند. ما یک فصل درباره “scaling” در بخش دوم داریم. پیام کوتاه این است: بخش زیادی از کار شما می‌تواند توسط یک تیم cross-functional انجام شود. بیشتر باقی‌مانده کارها می‌تواند توسط تعدادی تیم feature که به صورت موازی روی featureهایی که شما درک می‌کنید و می‌خواهید، کار می‌کنند، انجام شود. ممکن است در سازمان شما موارد بسیار کمی وجود داشته باشد که واقعاً نیاز به کار چندین تیم به صورت هماهنگ داشته باشد. اگر کار خود را به صورت feature به feature تقسیم کنید، به احتمال زیاد متوجه خواهید شد که آنچه باقی می‌ماند ساده است. این کار شما را از صرف هزینه زیاد برای “scaling” به یک ساختار و سازمانی که به شما خدمت نخواهد کرد، نجات می‌دهد. بر اساس feature سازماندهی کنید. خوشحال خواهید شد که این کار را انجام داده‌اید.   
report erratum • discuss   
Chapter 4. Organizing by Feature • 31]   
  
حرکت از چشم‌انداز به جزئیات   
CHAPTER 5   
Planning Feature by Feature   
کارها با Releaseهای مکرر نرم‌افزار بهترین نتیجه را می‌دهند. ارزش سریع‌تر و بهتر رشد می‌کند. مدیریت در بازه‌های زمانی کوتاه پیشرفت قابل مشاهده‌ای دریافت می‌کند. توسعه با اهداف کوچک و واضح بهترین عملکرد را دارد. با این حال، “vision” محصول با ایده‌های بزرگ و عظیم شروع می‌شود، مبهم اما جذاب. Vision درباره ایده‌های بزرگ است، نه تکه‌های کوچک. چگونه می‌توانیم از ایده‌های بزرگ محصول خود به سمت آن featureهای جزئی که برای بهترین دید و کنترل نیاز داریم، حرکت کنیم؟   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: حرکت از چشم‌انداز به جزئیات   
CHAPTER 5   
Planning Feature by Feature   
کارها با Releaseهای مکرر نرم‌افزار بهترین نتیجه را می‌دهند. ارزش سریع‌تر و بهتر رشد می‌کند. مدیریت در بازه‌های زمانی کوتاه پیشرفت قابل مشاهده‌ای دریافت می‌کند.]

================================================================================

# Section 38

## Original Text:

Development works best with small, clear  
objectives. The product “vision,” however, starts with big, grand ideas,  
vague yet enticing. Vision is about big ideas, not tiny bites. How can we move from our grand product ideas down to  
those detailed features we need for the best visibility and  
control. report erratum • discuss]  
  
Planning is indispensable. General Eisenhower said, “Plans are useless, but planning  
is indispensable. ” We do really need to think deeply about  
our product, not just at the beginning but all the time. We do need to plan. We don’t need a detailed list of what  
will happen and when. When the time comes, let’s decide  
then what to do next. A too-detailed plan will just waste  
time and create confusion. It’s important to identify key features that we’ll need to have  
early, as well as features we can’t live without. Let’s identify  
and record those. We need to defer implementing low-value ideas indefinitely. Let’s not waste time thinking about and keeping track of  
them. Still, planning is important. We probably have to consider  
a lot of bad ideas to get a few good ones. So let’s do plan but  
at the same time stay loose and ready for change. Reflect on some past projects where there was a huge feature  
list. How many of those ideas came to complete fruition. How many of those ideas were really great. Were any of  
them really duds. I know a lot of mine were. report erratum • discussChapter 5. Planning Feature by Feature • 33  
  
[CONTEXT FROM PREVIOUS PAGE: Planning is indispensable. General Eisenhower said, “Plans are useless, but planning  
is indispensable. ” We do really need to think deeply about  
our product, not just at the beginning but all the time. We do need to plan. We don’t need a detailed list of what  
will happen and when. When the time comes, let’s decide  
then what to do next. A too-detailed plan will just waste  
time and create confusion.

## Translated Text:

توسعه زمانی بهترین عملکرد را دارد که اهداف کوچک و واضح داشته باشد. با این حال، "چشم‌انداز" محصول با ایده‌های بزرگ و عظیم شروع می‌شود، ایده‌هایی مبهم اما جذاب. چشم‌انداز درباره ایده‌های بزرگ است، نه جزئیات کوچک. چگونه می‌توانیم از ایده‌های بزرگ محصول خود به آن ویژگی‌های دقیق که برای بهترین دید و کنترل نیاز داریم، برسیم؟  
  
برنامه‌ریزی ضروری است. ژنرال آیزنهاور گفت: «برنامه‌ها بی‌فایده هستند، اما برنامه‌ریزی ضروری است.» ما واقعاً باید عمیقاً درباره محصول خود فکر کنیم، نه فقط در ابتدا، بلکه در تمام مدت. ما نیاز به برنامه‌ریزی داریم. نیازی به یک لیست دقیق از اینکه چه اتفاقی خواهد افتاد و چه زمانی، نداریم. وقتی زمانش رسید، همان موقع تصمیم بگیریم که قدم بعدی چیست. یک برنامه خیلی دقیق فقط زمان را هدر می‌دهد و باعث سردرگمی می‌شود.   
  
مهم است که ویژگی‌های کلیدی که در ابتدا به آن‌ها نیاز داریم و همچنین ویژگی‌هایی که نمی‌توانیم بدون آن‌ها زندگی کنیم را شناسایی کنیم. بیایید آن‌ها را شناسایی و ثبت کنیم. ما باید اجرای ایده‌های کم‌ارزش را به طور نامحدود به تعویق بیندازیم. بیایید زمان خود را صرف فکر کردن به آن‌ها و پیگیری‌شان نکنیم. با این حال، برنامه‌ریزی مهم است. احتمالاً باید ایده‌های بد زیادی را در نظر بگیریم تا به چند ایده خوب برسیم. بنابراین بیایید برنامه‌ریزی کنیم، اما در عین حال انعطاف‌پذیر و آماده تغییر باشیم.  
  
به برخی از پروژه‌های گذشته فکر کنید که لیست ویژگی‌های عظیمی داشتند. چه تعداد از آن ایده‌ها به طور کامل به ثمر رسیدند؟ چه تعداد از آن ایده‌ها واقعاً عالی بودند؟ آیا هیچ‌کدام از آن‌ها واقعاً بی‌ارزش بودند؟ من می‌دانم که بسیاری از ایده‌های من این‌گونه بودند.

================================================================================

# Section 39

## Original Text:

It’s important to identify key features that we’ll need to have  
early, as well as features we can’t live without. Let’s identify  
and record those. We need to defer implementing low-value ideas indefinitely. Let’s not waste time thinking about and keeping track of  
them. Still, planning is important. We probably have to consider  
a lot of bad ideas to get a few good ones. So let’s do plan but  
at the same time stay loose and ready for change. Reflect on some past projects where there was a huge feature  
list. How many of those ideas came to complete fruition. How many of those ideas were really great. Were any of  
them really duds. I know a lot of mine were. report erratum • discussChapter 5. Planning Feature by Feature • 33]  
  
Detailed plans are useless. If planning is good, isn’t a bigger plan better. There we are,  
with some vague ideas in mind, which may or may not be  
things anyone would ever want. But for some reason, we  
feel that we need to figure out how long each one will take  
so that we can add them up and decide what will be done  
a hundred days from Tuesday. We’ve all read the horrible statistics about how many soft-  
ware projects go far beyond their budget. Maybe software  
people are terrible at estimating, and they should try harder. Well, yes, software people are terrible at estimating, because  
humans are terrible at estimating. Let’s not just try harder. Let’s find a better way. Here’s a better way: set a time and money budget; produce  
the most valuable features first; keep the product ready to  
ship at any time—and stop when the clock runs out. Quite  
likely we’ll even stop before the deadline, because we’ve  
already got the important stuff done. We deliver the bulk  
of the value in far less time, for far less money. How much long-term detail do you need to set a budget for  
your project. Is it better to plan for everything, trying to fit  
it inside the budget. Might it be better to set a tight deadline  
and build the best possible thing in that time. Chapter 5.

## Translated Text:

مهم است که ویژگی‌های کلیدی را که در ابتدا به آن‌ها نیاز داریم و همچنین ویژگی‌هایی که بدون آن‌ها نمی‌توانیم کار کنیم، شناسایی کنیم. بیایید آن‌ها را شناسایی و ثبت کنیم. باید پیاده‌سازی ایده‌های کم‌ارزش را به طور نامحدود به تعویق بیندازیم. بیایید وقت خود را صرف فکر کردن به آن‌ها و پیگیری‌شان نکنیم. با این حال، برنامه‌ریزی مهم است. احتمالاً باید ایده‌های بد زیادی را بررسی کنیم تا به چند ایده خوب برسیم. بنابراین بیایید برنامه‌ریزی کنیم اما در عین حال انعطاف‌پذیر باشیم و آماده تغییر. به برخی از پروژه‌های گذشته فکر کنید که لیست ویژگی‌های بزرگی داشتند. چند تا از آن ایده‌ها به طور کامل به نتیجه رسیدند؟ چند تا از آن ایده‌ها واقعاً عالی بودند؟ آیا هیچ‌کدام از آن‌ها کاملاً بی‌ارزش بودند؟ می‌دانم که بسیاری از ایده‌های من این‌گونه بودند.  
  
برنامه‌های دقیق بی‌فایده هستند. اگر برنامه‌ریزی خوب است، آیا یک برنامه بزرگ‌تر بهتر نیست؟ ما آنجا هستیم، با چند ایده مبهم در ذهن که ممکن است چیزهایی باشند که هیچ‌کس هرگز نخواهد. اما به دلایلی احساس می‌کنیم که باید بفهمیم هر کدام چقدر زمان می‌برند تا بتوانیم آن‌ها را جمع بزنیم و تصمیم بگیریم که صد روز بعد از سه‌شنبه چه چیزی انجام خواهد شد. همه ما آمارهای وحشتناک در مورد اینکه چند پروژه نرم‌افزاری بسیار فراتر از بودجه خود می‌روند را خوانده‌ایم. شاید افراد نرم‌افزار در تخمین زدن بد باشند و باید بیشتر تلاش کنند. خب، بله، افراد نرم‌افزار در تخمین زدن بد هستند، زیرا انسان‌ها در تخمین زدن بد هستند. بیایید فقط بیشتر تلاش نکنیم. بیایید راه بهتری پیدا کنیم.   
  
اینجا یک راه بهتر است: یک بودجه زمانی و مالی تعیین کنید؛ ابتدا باارزش‌ترین ویژگی‌ها را تولید کنید؛ محصول را در هر لحظه آماده ارسال نگه دارید—و زمانی که زمان تمام شد، متوقف شوید. احتمالاً حتی قبل از مهلت متوقف خواهیم شد، زیرا کارهای مهم را قبلاً انجام داده‌ایم. ما بخش عمده‌ای از ارزش را در زمان بسیار کمتر و با هزینه بسیار کمتر ارائه می‌دهیم. چقدر جزئیات بلندمدت نیاز دارید تا برای پروژه خود بودجه تعیین کنید؟ آیا بهتر است برای همه چیز برنامه‌ریزی کنید و سعی کنید آن را در بودجه جا دهید؟ یا شاید بهتر باشد یک مهلت زمانی محدود تعیین کنید و بهترین چیزی که می‌توانید در آن زمان بسازید؟ فصل ۵.

================================================================================

# Section 40

## Original Text:

Planning Feature by Feature • 34  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Detailed plans are useless. If planning is good, isn’t a bigger plan better. There we are,  
with some vague ideas in mind, which may or may not be  
things anyone would ever want. But for some reason, we  
feel that we need to figure out how long each one will take  
so that we can add them up and decide what will be done  
a hundred days from Tuesday. We’ve all read the horrible statistics about how many soft-  
ware projects go far beyond their budget. Maybe software  
people are terrible at estimating, and they should try harder. Well, yes, software people are terrible at estimating, because  
humans are terrible at estimating. Let’s not just try harder. Let’s find a better way. Here’s a better way: set a time and money budget; produce  
the most valuable features first; keep the product ready to  
ship at any time—and stop when the clock runs out. Quite  
likely we’ll even stop before the deadline, because we’ve  
already got the important stuff done. We deliver the bulk  
of the value in far less time, for far less money. How much long-term detail do you need to set a budget for  
your project. Is it better to plan for everything, trying to fit  
it inside the budget. Might it be better to set a tight deadline  
and build the best possible thing in that time. Chapter 5. Planning Feature by Feature • 34  
report erratum • discuss]  
  
Getting started  
Ideally, we would begin just by beginning. Get an idea, think  
about it a bit, put together a little team, and start building. That will tell us quickly whether we can produce something  
of value, and about how long it will take. Then we can decide  
to cut our investment, to keep going, or to invest more. Sometimes our organization just doesn’t work that way. They insist on knowing whether this proposed project will  
take weeks, months, or years, long before they’ll decide to  
invest in it.

## Translated Text:

برنامه‌ریزی ویژگی به ویژگی • ۳۴   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: برنامه‌های دقیق بی‌فایده هستند. اگر برنامه‌ریزی خوب است، آیا یک برنامه بزرگ‌تر بهتر نیست؟ ما اینجا هستیم، با چند ایده مبهم در ذهن که ممکن است یا ممکن است چیزهایی باشند که کسی واقعاً بخواهد. اما به دلایلی، احساس می‌کنیم که باید بفهمیم هر کدام چقدر زمان می‌برند تا بتوانیم آن‌ها را جمع کنیم و تصمیم بگیریم که صد روز بعد از سه‌شنبه چه چیزی انجام خواهد شد. همه ما آمار وحشتناک درباره اینکه چند پروژه نرم‌افزاری بسیار فراتر از بودجه خود می‌روند را خوانده‌ایم. شاید افراد نرم‌افزاری در تخمین زدن افتضاح هستند و باید بیشتر تلاش کنند. خب، بله، افراد نرم‌افزاری در تخمین زدن افتضاح هستند، چون انسان‌ها در تخمین زدن افتضاح هستند. بیایید فقط بیشتر تلاش نکنیم. بیایید یک روش بهتر پیدا کنیم. اینجا یک روش بهتر است: یک بودجه زمانی و مالی تعیین کنید؛ ابتدا ارزشمندترین ویژگی‌ها را تولید کنید؛ محصول را در هر لحظه آماده ارسال نگه دارید—و وقتی زمان تمام شد، متوقف شوید. احتمالاً حتی قبل از ضرب‌الاجل متوقف خواهیم شد، چون قبلاً کارهای مهم را انجام داده‌ایم. ما بخش عمده‌ای از ارزش را در زمان بسیار کمتر و با هزینه بسیار کمتر ارائه می‌دهیم. چقدر جزئیات بلندمدت نیاز دارید تا یک بودجه برای پروژه خود تعیین کنید؟ آیا بهتر است برای همه چیز برنامه‌ریزی کنید و سعی کنید آن را در بودجه جا دهید؟ آیا ممکن است بهتر باشد یک ضرب‌الاجل محدود تعیین کنید و بهترین چیز ممکن را در آن زمان بسازید؟ فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۳۴   
گزارش خطا • بحث کنید]   
  
شروع کار   
در حالت ایده‌آل، ما فقط با شروع کردن، شروع می‌کردیم. یک ایده بگیرید، کمی درباره آن فکر کنید، یک تیم کوچک جمع کنید و شروع به ساختن کنید. این به ما سریعاً نشان خواهد داد که آیا می‌توانیم چیزی ارزشمند تولید کنیم و حدوداً چقدر زمان خواهد برد. سپس می‌توانیم تصمیم بگیریم که سرمایه‌گذاری خود را کاهش دهیم، ادامه دهیم، یا بیشتر سرمایه‌گذاری کنیم. گاهی اوقات سازمان ما فقط به این شکل کار نمی‌کند. آن‌ها اصرار دارند که بدانند آیا این پروژه پیشنهادی هفته‌ها، ماه‌ها، یا سال‌ها طول خواهد کشید، مدت‌ها قبل از اینکه تصمیم بگیرند در آن سرمایه‌گذاری کنند.

================================================================================

# Section 41

## Original Text:

We might be able to respond by asking to form  
a team and build for a while. If we can, that’s probably the  
way to go: we can learn a lot quickly. Sometimes that  
approach, sensible though it is, just won’t fly. We need a  
way to get an initial cut at project size. A lot has been written about how to do estimation. Yet  
projects continue to deliver too little, too late. Search for  
better estimates if you must: I’d prefer to work from a bud-  
get, a deadline, a Product Champion who decides what to  
do next, and a team who can ship software at will. How much do “they” need to know about a proposed  
project. How close do we need to be to correct when we bid  
on it. Can we steer to success within a rough estimate. report erratum • discussChapter 5. Planning Feature by Feature • 35  
  
[CONTEXT FROM PREVIOUS PAGE: Getting started  
Ideally, we would begin just by beginning. Get an idea, think  
about it a bit, put together a little team, and start building. That will tell us quickly whether we can produce something  
of value, and about how long it will take. Then we can decide  
to cut our investment, to keep going, or to invest more. Sometimes our organization just doesn’t work that way. They insist on knowing whether this proposed project will  
take weeks, months, or years, long before they’ll decide to  
invest in it. We might be able to respond by asking to form  
a team and build for a while. If we can, that’s probably the  
way to go: we can learn a lot quickly. Sometimes that  
approach, sensible though it is, just won’t fly. We need a  
way to get an initial cut at project size. A lot has been written about how to do estimation. Yet  
projects continue to deliver too little, too late. Search for  
better estimates if you must: I’d prefer to work from a bud-  
get, a deadline, a Product Champion who decides what to  
do next, and a team who can ship software at will. How much do “they” need to know about a proposed  
project. How close do we need to be to correct when we bid  
on it.

## Translated Text:

ممکن است بتوانیم پاسخ دهیم با درخواست تشکیل یک تیم و Build کردن برای مدتی. اگر بتوانیم، احتمالاً این بهترین راه است: می‌توانیم سریعاً چیزهای زیادی یاد بگیریم. گاهی اوقات این رویکرد، با وجود منطقی بودن، کارساز نیست. ما به روشی نیاز داریم تا یک برآورد اولیه از اندازه پروژه داشته باشیم. مطالب زیادی درباره چگونگی انجام Estimation نوشته شده است. با این حال، پروژه‌ها همچنان کمتر از حد انتظار و دیرتر از موعد تحویل داده می‌شوند. اگر لازم است به دنبال Estimation بهتر بگردید: من ترجیح می‌دهم از یک Budget، یک Deadline، یک Product Champion که تصمیم می‌گیرد چه کاری بعداً انجام شود، و یک تیم که بتواند هر زمان که بخواهد نرم‌افزار را Ship کند، کار کنم.   
  
چقدر نیاز است که "آن‌ها" درباره یک پروژه پیشنهادی بدانند؟ چقدر باید هنگام ارائه پیشنهاد به آن نزدیک باشیم؟ آیا می‌توانیم با یک تخمین تقریبی به موفقیت برسیم؟  
  
[متن از صفحه قبلی: شروع کار   
در حالت ایده‌آل، ما فقط با شروع کردن کار را آغاز می‌کردیم. یک ایده بگیرید، کمی درباره آن فکر کنید، یک تیم کوچک تشکیل دهید، و شروع به Build کردن کنید. این کار به ما سریعاً نشان می‌دهد که آیا می‌توانیم چیزی ارزشمند تولید کنیم و چقدر طول خواهد کشید. سپس می‌توانیم تصمیم بگیریم که سرمایه‌گذاری خود را کاهش دهیم، ادامه دهیم یا بیشتر سرمایه‌گذاری کنیم. گاهی اوقات سازمان ما به این شکل کار نمی‌کند. آن‌ها اصرار دارند که بدانند آیا این پروژه پیشنهادی هفته‌ها، ماه‌ها یا سال‌ها طول خواهد کشید، خیلی قبل از اینکه تصمیم بگیرند روی آن سرمایه‌گذاری کنند. ممکن است بتوانیم پاسخ دهیم با درخواست تشکیل یک تیم و Build کردن برای مدتی. اگر بتوانیم، احتمالاً این بهترین راه است: می‌توانیم سریعاً چیزهای زیادی یاد بگیریم. گاهی اوقات این رویکرد، با وجود منطقی بودن، کارساز نیست. ما به روشی نیاز داریم تا یک برآورد اولیه از اندازه پروژه داشته باشیم. مطالب زیادی درباره چگونگی انجام Estimation نوشته شده است. با این حال، پروژه‌ها همچنان کمتر از حد انتظار و دیرتر از موعد تحویل داده می‌شوند. اگر لازم است به دنبال Estimation بهتر بگردید: من ترجیح می‌دهم از یک Budget، یک Deadline، یک Product Champion که تصمیم می‌گیرد چه کاری بعداً انجام شود، و یک تیم که بتواند هر زمان که بخواهد نرم‌افزار را Ship کند، کار کنم. چقدر نیاز است که "آن‌ها" درباره یک پروژه پیشنهادی بدانند؟ چقدر باید هنگام ارائه پیشنهاد به آن نزدیک باشیم؟]

================================================================================

# Section 42

## Original Text:

Can we steer to success within a rough estimate. report erratum • discussChapter 5. Planning Feature by Feature • 35]  
  
Continuous planning: feature splitting  
It’s not good enough to plan just at the beginning. Because  
we’re focused on value, we need to plan all the time. The  
team should be working to a fixed cadence, often called  
iterations, or sprints , a couple of weeks long. Things go best  
if each feature, often called a story , takes only two or three  
days to do. I don’t recommend working with larger stories and breaking  
them down into technical items, often called tasks. If we use  
tasks, the business-side people do not have a clear look at  
how things are going, and they often do not get a good sense  
of how to help until the end of the two-week sprint interval. Stick with stories: they work better all around. It’s better to break down stories into smaller stories , each  
making sense to the business-side people. As far as I can see,  
this is always possible. It’s a bit tricky at first, but with only  
a few hours of practice, teams quickly learn how to split  
features into smaller features, rather than into technical steps. Think of some big feature that your product needs. How  
might it be broken down into smaller things. Are some of  
those things substantially more valuable than others. What  
does that suggest to you. Chapter 5. Planning Feature by Feature • 36  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Continuous planning: feature splitting  
It’s not good enough to plan just at the beginning. Because  
we’re focused on value, we need to plan all the time. The  
team should be working to a fixed cadence, often called  
iterations, or sprints , a couple of weeks long. Things go best  
if each feature, often called a story , takes only two or three  
days to do. I don’t recommend working with larger stories and breaking  
them down into technical items, often called tasks.

## Translated Text:

آیا می‌توانیم با یک تخمین تقریبی به موفقیت برسیم؟   
گزارش خطا • بحث   
فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۳۵   
  
برنامه‌ریزی مستمر: تقسیم‌بندی ویژگی‌ها   
برنامه‌ریزی فقط در ابتدای کار کافی نیست. چون ما بر ارزش تمرکز داریم، باید همیشه برنامه‌ریزی کنیم. تیم باید با یک ریتم ثابت کار کند که اغلب به آن iterations یا sprints گفته می‌شود و معمولاً چند هفته طول می‌کشد. اوضاع زمانی بهتر پیش می‌رود که هر ویژگی، که اغلب به آن story گفته می‌شود، تنها دو یا سه روز زمان ببرد.   
  
من توصیه نمی‌کنم با storyهای بزرگ کار کنید و آن‌ها را به آیتم‌های فنی که اغلب tasks نامیده می‌شوند، تقسیم کنید. اگر از tasks استفاده کنیم، افراد سمت کسب‌وکار دید واضحی از روند کار ندارند و اغلب تا پایان بازه دو هفته‌ای sprint درک خوبی از نحوه کمک کردن پیدا نمی‌کنند.   
  
به stories پایبند باشید: آن‌ها در همه جنبه‌ها بهتر عمل می‌کنند. بهتر است stories را به stories کوچک‌تر تقسیم کنید که هر کدام برای افراد سمت کسب‌وکار معنا داشته باشند. تا جایی که من می‌بینم، این همیشه ممکن است. در ابتدا کمی دشوار است، اما تنها با چند ساعت تمرین، تیم‌ها به سرعت یاد می‌گیرند که چگونه ویژگی‌ها را به ویژگی‌های کوچک‌تر تقسیم کنند، به جای اینکه آن‌ها را به مراحل فنی تبدیل کنند.   
  
به یک ویژگی بزرگ فکر کنید که محصول شما به آن نیاز دارد. چگونه می‌توان آن را به بخش‌های کوچک‌تر تقسیم کرد؟ آیا برخی از آن بخش‌ها به طور قابل توجهی ارزشمندتر از دیگران هستند؟ این موضوع چه چیزی را به شما نشان می‌دهد؟   
فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۳۶   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: برنامه‌ریزی مستمر: تقسیم‌بندی ویژگی‌ها   
برنامه‌ریزی فقط در ابتدای کار کافی نیست. چون ما بر ارزش تمرکز داریم، باید همیشه برنامه‌ریزی کنیم. تیم باید با یک ریتم ثابت کار کند که اغلب به آن iterations یا sprints گفته می‌شود و معمولاً چند هفته طول می‌کشد. اوضاع زمانی بهتر پیش می‌رود که هر ویژگی، که اغلب به آن story گفته می‌شود، تنها دو یا سه روز زمان ببرد.   
  
من توصیه نمی‌کنم با storyهای بزرگ کار کنید و آن‌ها را به آیتم‌های فنی که اغلب tasks نامیده می‌شوند، تقسیم کنید.]

================================================================================

# Section 43

## Original Text:

If we use  
tasks, the business-side people do not have a clear look at  
how things are going, and they often do not get a good sense  
of how to help until the end of the two-week sprint interval. Stick with stories: they work better all around. It’s better to break down stories into smaller stories , each  
making sense to the business-side people. As far as I can see,  
this is always possible. It’s a bit tricky at first, but with only  
a few hours of practice, teams quickly learn how to split  
features into smaller features, rather than into technical steps. Think of some big feature that your product needs. How  
might it be broken down into smaller things. Are some of  
those things substantially more valuable than others. What  
does that suggest to you. Chapter 5. Planning Feature by Feature • 36  
report erratum • discuss]  
  
How much work should the team take on. The answer to this one is clear: the team itself should decide  
how much work it can accomplish in the next two-week  
interval. They know better than anyone else, and they will  
feel more commitment if they make the determination on  
their own. There are many practices to help the team choose  
how much to do. They are all based on an idea that I first  
learned from Kent Beck and Martin Fowler: “Yesterday’s  
Weather. ”  
You’ll probably get as much done today as you did yester-  
day. In iterative projects, plan to do as much this iteration  
as you did last iteration. We plan each iteration right before it begins. To decide how  
much work to take on, we need to understand the work. As  
a team, we discuss the work. The team’s Product Champion  
presents one feature at a time, followed by a brief team dis-  
cussion about what it’ll take to accomplish the feature. Everyone stays engaged, and the team understands the fea-  
ture before committing to do it. I don’t recommend estimating the individual work pieces  
at all. Instead, understand them, and then look at the  
aggregate and decide how much of it the team can do.

## Translated Text:

اگر از tasks استفاده کنیم، افراد سمت کسب‌وکار دید واضحی نسبت به اینکه اوضاع چگونه پیش می‌رود ندارند و اغلب تا پایان دوره دو هفته‌ای sprint حس خوبی از نحوه کمک کردن پیدا نمی‌کنند. با stories کار کنید: آنها در همه‌جا بهتر عمل می‌کنند. بهتر است stories را به stories کوچک‌تر تقسیم کنیم، به‌طوری‌که هر کدام برای افراد سمت کسب‌وکار معنادار باشد. تا جایی که من می‌بینم، این همیشه ممکن است. در ابتدا کمی دشوار است، اما با تنها چند ساعت تمرین، تیم‌ها به‌سرعت یاد می‌گیرند که چگونه features را به features کوچک‌تر تقسیم کنند، به‌جای اینکه آنها را به مراحل فنی تقسیم کنند. به یک feature بزرگ فکر کنید که محصول شما به آن نیاز دارد. چگونه می‌توان آن را به اجزای کوچک‌تر تقسیم کرد؟ آیا برخی از آن اجزا به‌طور قابل‌توجهی ارزشمندتر از بقیه هستند؟ این موضوع چه چیزی را به شما نشان می‌دهد؟  
  
فصل ۵. برنامه‌ریزی Feature به Feature • ۳۶   
گزارش خطا • بحث کنید  
  
تیم باید چه مقدار کار را بر عهده بگیرد؟ پاسخ این سؤال واضح است: خود تیم باید تصمیم بگیرد که چه مقدار کار می‌تواند در دوره دو هفته‌ای بعدی انجام دهد. آنها بهتر از هر کس دیگری این موضوع را می‌دانند و اگر خودشان این تصمیم را بگیرند، احساس تعهد بیشتری خواهند داشت. روش‌های زیادی وجود دارد که به تیم کمک می‌کند تصمیم بگیرد چه مقدار کار را انجام دهد. همه این روش‌ها بر اساس ایده‌ای است که من اولین بار از Kent Beck و Martin Fowler یاد گرفتم: "Yesterday’s Weather."  
  
شما احتمالاً امروز به همان اندازه کار انجام می‌دهید که دیروز انجام دادید. در پروژه‌های تکرارشونده، برنامه‌ریزی کنید که در این iteration به همان اندازه کار انجام دهید که در iteration قبلی انجام دادید. ما هر iteration را درست قبل از شروع آن برنامه‌ریزی می‌کنیم. برای تصمیم‌گیری درباره اینکه چه مقدار کار را بر عهده بگیریم، باید کار را درک کنیم. به‌عنوان یک تیم، درباره کار بحث می‌کنیم. Product Champion تیم یک feature را در هر بار ارائه می‌دهد و سپس یک بحث کوتاه تیمی درباره اینکه برای انجام آن feature چه چیزی لازم است، انجام می‌شود. همه درگیر می‌شوند و تیم قبل از تعهد به انجام آن، feature را درک می‌کند. من اصلاً توصیه نمی‌کنم که قطعات کاری فردی را تخمین بزنید. در عوض، آنها را درک کنید و سپس به مجموعه کلی نگاه کنید و تصمیم بگیرید که تیم چه مقدار از آن را می‌تواند انجام دهد.

================================================================================

# Section 44

## Original Text:

If  
estimates really help the team, go for it. But be careful. The  
point isn’t to make good estimates—the point is to do good  
work at a consistent pace. report erratum • discussChapter 5. Planning Feature by Feature • 37  
  
[CONTEXT FROM PREVIOUS PAGE: How much work should the team take on. The answer to this one is clear: the team itself should decide  
how much work it can accomplish in the next two-week  
interval. They know better than anyone else, and they will  
feel more commitment if they make the determination on  
their own. There are many practices to help the team choose  
how much to do. They are all based on an idea that I first  
learned from Kent Beck and Martin Fowler: “Yesterday’s  
Weather. ”  
You’ll probably get as much done today as you did yester-  
day. In iterative projects, plan to do as much this iteration  
as you did last iteration. We plan each iteration right before it begins. To decide how  
much work to take on, we need to understand the work. As  
a team, we discuss the work. The team’s Product Champion  
presents one feature at a time, followed by a brief team dis-  
cussion about what it’ll take to accomplish the feature. Everyone stays engaged, and the team understands the fea-  
ture before committing to do it. I don’t recommend estimating the individual work pieces  
at all. Instead, understand them, and then look at the  
aggregate and decide how much of it the team can do. If  
estimates really help the team, go for it. But be careful. The  
point isn’t to make good estimates—the point is to do good  
work at a consistent pace. report erratum • discussChapter 5. Planning Feature by Feature • 37]  
  
Estimation is risky. There are some serious risks to estimates: we have an almost  
irresistible desire to “improve” them, or to compare them. Both of these practices are pernicious. Keep in mind that  
business and management get the best results by selecting  
the work to be done and the work to be deferred.

## Translated Text:

اگر تخمین‌ها واقعاً به تیم کمک می‌کنند، از آن‌ها استفاده کنید. اما مراقب باشید. هدف این نیست که تخمین‌های خوبی بزنید—هدف این است که کار خوب را با یک سرعت ثابت انجام دهید.   
گزارش خطا • بحث   
فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۳۷   
  
[متن از صفحه قبلی: چه مقدار کار باید تیم انجام دهد؟ پاسخ به این سؤال واضح است: خود تیم باید تصمیم بگیرد که چه مقدار کار می‌تواند در بازه دو هفته‌ای بعدی انجام دهد. آن‌ها بهتر از هر کس دیگری می‌دانند و اگر خودشان این تصمیم را بگیرند، احساس تعهد بیشتری خواهند داشت. روش‌های زیادی وجود دارد که به تیم کمک می‌کند تصمیم بگیرد چه مقدار کار انجام دهد. همه این روش‌ها بر پایه ایده‌ای است که من اولین بار از Kent Beck و Martin Fowler یاد گرفتم: "Yesterday’s Weather."   
احتمالاً امروز به همان اندازه که دیروز کار انجام داده‌اید، کار خواهید کرد. در پروژه‌های تکرارشونده، برنامه‌ریزی کنید که به اندازه همان مقدار کاری که در تکرار قبلی انجام داده‌اید، در این تکرار نیز انجام دهید. ما هر تکرار را درست قبل از شروع آن برنامه‌ریزی می‌کنیم. برای تصمیم‌گیری در مورد اینکه چه مقدار کار انجام دهیم، باید کار را درک کنیم. به عنوان یک تیم، درباره کار بحث می‌کنیم. Product Champion تیم هر بار یک Feature را ارائه می‌دهد و سپس یک بحث کوتاه تیمی درباره آنچه برای انجام آن Feature لازم است، انجام می‌شود. همه درگیر می‌مانند و تیم Feature را قبل از تعهد به انجام آن درک می‌کند. من اصلاً توصیه نمی‌کنم که قطعات کاری فردی را تخمین بزنید. در عوض، آن‌ها را درک کنید و سپس به صورت کلی نگاه کنید و تصمیم بگیرید که تیم چه مقدار از آن را می‌تواند انجام دهد. اگر تخمین‌ها واقعاً به تیم کمک می‌کنند، از آن‌ها استفاده کنید. اما مراقب باشید. هدف این نیست که تخمین‌های خوبی بزنید—هدف این است که کار خوب را با یک سرعت ثابت انجام دهید.   
گزارش خطا • بحث   
فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۳۷]   
  
تخمین زدن ریسک دارد. تخمین‌ها دارای برخی ریسک‌های جدی هستند: ما تقریباً یک میل غیرقابل مقاومت برای "بهبود" آن‌ها یا مقایسه آن‌ها داریم. هر دوی این روش‌ها مضر هستند. به خاطر داشته باشید که کسب‌وکار و مدیریت بهترین نتایج را با انتخاب کارهایی که باید انجام شوند و کارهایی که باید به تعویق بیفتند، به دست می‌آورند.

================================================================================

# Section 45

## Original Text:

Focusing  
on estimates detracts from this responsibility and is almost  
certain to create conservative estimates in the hope of creat-  
ing accurate ones. Many teams today are working quite successfully using no  
detailed estimates at all. They think about the work. They  
break it down, often all the way down to a single test’s worth  
of story. Then they just get to work. When they need to  
predict how long things will take, they just count things  
done. Examine your desire to know estimates and actuals. What  
would better estimates give you. Which of those things  
might get in the way of managing scope. Which of those  
things might be accomplished better by managing scope  
rather than improving estimates. Would you prefer to have better estimates if it meant they  
would be more conservative. Is prediction better than  
steering. Chapter 5. Planning Feature by Feature • 38  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Estimation is risky. There are some serious risks to estimates: we have an almost  
irresistible desire to “improve” them, or to compare them. Both of these practices are pernicious. Keep in mind that  
business and management get the best results by selecting  
the work to be done and the work to be deferred. Focusing  
on estimates detracts from this responsibility and is almost  
certain to create conservative estimates in the hope of creat-  
ing accurate ones. Many teams today are working quite successfully using no  
detailed estimates at all. They think about the work. They  
break it down, often all the way down to a single test’s worth  
of story. Then they just get to work. When they need to  
predict how long things will take, they just count things  
done. Examine your desire to know estimates and actuals. What  
would better estimates give you. Which of those things  
might get in the way of managing scope. Which of those  
things might be accomplished better by managing scope  
rather than improving estimates.

## Translated Text:

تمرکز بر روی estimates از این مسئولیت منحرف می‌کند و تقریباً به طور قطع منجر به ایجاد estimates محافظه‌کارانه می‌شود به امید ایجاد estimates دقیق. بسیاری از تیم‌ها امروزه بدون استفاده از estimates دقیق، بسیار موفق عمل می‌کنند. آنها درباره کار فکر می‌کنند. آن را تجزیه می‌کنند، اغلب تا حد یک test story کوچک. سپس فقط شروع به کار می‌کنند. وقتی نیاز دارند پیش‌بینی کنند که کارها چقدر طول می‌کشد، فقط تعداد کارهای انجام‌شده را می‌شمارند. تمایل خود را برای دانستن estimates و actuals بررسی کنید. estimates بهتر چه چیزی به شما می‌دهد؟ کدام یک از این موارد ممکن است در مدیریت scope اختلال ایجاد کند؟ کدام یک از این موارد ممکن است با مدیریت بهتر scope به جای بهبود estimates بهتر انجام شود؟ آیا ترجیح می‌دهید estimates بهتری داشته باشید اگر به معنای محافظه‌کارانه‌تر بودن آنها باشد؟ آیا پیش‌بینی بهتر از steering است؟   
  
فصل ۵. Planning Feature by Feature • ۳۸   
report erratum • discuss   
  
[متن از صفحه قبلی: Estimation خطرناک است. برخی از خطرات جدی در مورد estimates وجود دارد: ما تقریباً یک تمایل غیرقابل مقاومت برای "بهبود" آنها یا مقایسه آنها داریم. هر دوی این شیوه‌ها مضر هستند. به خاطر داشته باشید که کسب‌وکار و مدیریت بهترین نتایج را با انتخاب کارهایی که باید انجام شوند و کارهایی که باید به تعویق بیفتند، به دست می‌آورند. تمرکز بر روی estimates از این مسئولیت منحرف می‌کند و تقریباً به طور قطع منجر به ایجاد estimates محافظه‌کارانه می‌شود به امید ایجاد estimates دقیق. بسیاری از تیم‌ها امروزه بدون استفاده از estimates دقیق، بسیار موفق عمل می‌کنند. آنها درباره کار فکر می‌کنند. آن را تجزیه می‌کنند، اغلب تا حد یک test story کوچک. سپس فقط شروع به کار می‌کنند. وقتی نیاز دارند پیش‌بینی کنند که کارها چقدر طول می‌کشد، فقط تعداد کارهای انجام‌شده را می‌شمارند. تمایل خود را برای دانستن estimates و actuals بررسی کنید. estimates بهتر چه چیزی به شما می‌دهد؟ کدام یک از این موارد ممکن است در مدیریت scope اختلال ایجاد کند؟ کدام یک از این موارد ممکن است با مدیریت بهتر scope به جای بهبود estimates بهتر انجام شود؟]

================================================================================

# Section 46

## Original Text:

Would you prefer to have better estimates if it meant they  
would be more conservative. Is prediction better than  
steering. Chapter 5. Planning Feature by Feature • 38  
report erratum • discuss]  
  
Planning with “stretch goals” is destructive. During planning, especially short-term planning, it’s  
tempting to set up “stretch goals” or to “encourage” the  
team to do “just one more feature. ” Please do not do this. It’s devastatingly destructive. The reason is that the team  
will in fact try. Eager to please, they’ll unconsciously hurry. They’ll leave out just a few tests. They’ll leave the code not  
quite as clean as they might have, just to squeeze in one more  
feature. Hurrying, they’ll inject more defects. Since defects take  
longer to fix than they do to prevent, hurrying will slow you  
down. Worse yet, it will slow you down when you least  
need it, the closer you get to the end of the project. Dirty code slows you down as well. If the code is clean, the  
next features go in smoothly. If it gets dirty, everything takes  
just a bit longer. Pressure is destructive. Avoid it. What are some bad things that pressure has caused in your  
projects. Has it ever increased your defect count. Cost you  
time. Cost you valuable people. When has pressure really  
been helpful. Can you think of other ways to get those  
benefits. report erratum • discussChapter 5. Planning Feature by Feature • 39  
  
[CONTEXT FROM PREVIOUS PAGE: Planning with “stretch goals” is destructive. During planning, especially short-term planning, it’s  
tempting to set up “stretch goals” or to “encourage” the  
team to do “just one more feature. ” Please do not do this. It’s devastatingly destructive. The reason is that the team  
will in fact try. Eager to please, they’ll unconsciously hurry. They’ll leave out just a few tests. They’ll leave the code not  
quite as clean as they might have, just to squeeze in one more  
feature. Hurrying, they’ll inject more defects.

## Translated Text:

آیا ترجیح می‌دهید تخمین‌های بهتری داشته باشید اگر به این معنا باشد که آن‌ها محافظه‌کارانه‌تر باشند؟ آیا پیش‌بینی بهتر از هدایت است؟   
فصل ۵. برنامه‌ریزی Feature به Feature • ۳۸   
report erratum • discuss   
  
برنامه‌ریزی با "اهداف کششی" مخرب است. در طول برنامه‌ریزی، به‌ویژه برنامه‌ریزی کوتاه‌مدت، وسوسه‌انگیز است که "اهداف کششی" تعیین کنید یا تیم را تشویق کنید که "فقط یک Feature دیگر" انجام دهند. لطفاً این کار را نکنید. این کار به‌شدت مخرب است. دلیلش این است که تیم واقعاً تلاش خواهد کرد. آن‌ها مشتاق به رضایت دادن، ناخودآگاه عجله می‌کنند. آن‌ها فقط چند Test را کنار می‌گذارند. کد را به اندازه‌ای که می‌توانستند تمیز نگه نمی‌دارند، فقط برای اینکه یک Feature دیگر را جا بدهند. با عجله، آن‌ها Defectهای بیشتری تزریق می‌کنند. از آنجایی که رفع Defectها بیشتر از پیشگیری از آن‌ها زمان می‌برد، عجله شما را کند می‌کند. بدتر از آن، زمانی شما را کند می‌کند که کمترین نیاز به آن دارید، یعنی هرچه به پایان پروژه نزدیک‌تر شوید.   
  
کد کثیف نیز شما را کند می‌کند. اگر کد تمیز باشد، Featureهای بعدی به‌راحتی اضافه می‌شوند. اگر کثیف شود، همه چیز کمی بیشتر زمان می‌برد. فشار مخرب است. از آن اجتناب کنید. فشار چه چیزهای بدی در پروژه‌های شما ایجاد کرده است؟ آیا تاکنون تعداد Defectهای شما را افزایش داده است؟ آیا برای شما زمان هزینه کرده است؟ آیا افراد ارزشمند شما را از دست داده است؟ فشار چه زمانی واقعاً مفید بوده است؟ آیا می‌توانید به روش‌های دیگری برای دستیابی به آن مزایا فکر کنید؟   
report erratum • discuss   
فصل ۵. برنامه‌ریزی Feature به Feature • ۳۹   
  
[متن از صفحه قبلی: برنامه‌ریزی با "اهداف کششی" مخرب است. در طول برنامه‌ریزی، به‌ویژه برنامه‌ریزی کوتاه‌مدت، وسوسه‌انگیز است که "اهداف کششی" تعیین کنید یا تیم را تشویق کنید که "فقط یک Feature دیگر" انجام دهند. لطفاً این کار را نکنید. این کار به‌شدت مخرب است. دلیلش این است که تیم واقعاً تلاش خواهد کرد. آن‌ها مشتاق به رضایت دادن، ناخودآگاه عجله می‌کنند. آن‌ها فقط چند Test را کنار می‌گذارند. کد را به اندازه‌ای که می‌توانستند تمیز نگه نمی‌دارند، فقط برای اینکه یک Feature دیگر را جا بدهند. با عجله، آن‌ها Defectهای بیشتری تزریق می‌کنند.]

================================================================================

# Section 47

## Original Text:

Since defects take  
longer to fix than they do to prevent, hurrying will slow you  
down. Worse yet, it will slow you down when you least  
need it, the closer you get to the end of the project. Dirty code slows you down as well. If the code is clean, the  
next features go in smoothly. If it gets dirty, everything takes  
just a bit longer. Pressure is destructive. Avoid it. What are some bad things that pressure has caused in your  
projects. Has it ever increased your defect count. Cost you  
time. Cost you valuable people. When has pressure really  
been helpful. Can you think of other ways to get those  
benefits. report erratum • discussChapter 5. Planning Feature by Feature • 39]  
  
Working without estimates  
Once we become proficient at breaking all features down to  
approximately the same size, we can manage the project  
very nicely, because we have a good sense of how long things  
will take. Since our job is to select the work to do versus the  
work to defer, and to select the most valuable work first, we  
can steer the project to success without the overhead of  
estimation. Estimates remain a controversial topic in software develop-  
ment today. Many people think that estimates and detailed  
planning are important. Certainly some companies want  
these things badly, and that may be enough to cause us to  
do them. Unfortunately, “badly” is almost always how we  
do with estimates. For teams in flight, estimation is at most  
an internal matter and may well be almost entirely waste. Generally speaking, estimates are likely to be wrong, and  
they focus our attention on the cost of things rather than on  
value. Consider de-emphasizing or eliminating cost estimates  
and steering to success by a focus on value. Chapter 5.

## Translated Text:

از آنجا که رفع defects بیشتر از جلوگیری از وقوع آن‌ها زمان می‌برد، عجله کردن شما را کندتر می‌کند. بدتر از آن، این کندی زمانی اتفاق می‌افتد که کمترین نیاز را به آن دارید، یعنی هرچه به پایان پروژه نزدیک‌تر می‌شوید. کد کثیف نیز شما را کند می‌کند. اگر کد تمیز باشد، اضافه کردن features بعدی به‌راحتی انجام می‌شود. اما اگر کد کثیف شود، همه چیز کمی بیشتر زمان می‌برد. فشار مخرب است. از آن اجتناب کنید. چه چیزهای بدی فشار در پروژه‌های شما ایجاد کرده است؟ آیا تاکنون باعث افزایش تعداد defects شده است؟ آیا زمان شما را هدر داده است؟ آیا افراد ارزشمند شما را از دست داده‌اید؟ چه زمانی فشار واقعاً مفید بوده است؟ آیا می‌توانید به روش‌های دیگری برای دستیابی به آن مزایا فکر کنید؟  
  
\*\*گزارش خطا • بحث\*\*  
\*\*فصل ۵. برنامه‌ریزی Feature به Feature • صفحه ۳۹\*\*  
  
\*\*کار کردن بدون تخمین‌ها\*\*   
وقتی ما در شکستن تمام features به اندازه‌های تقریباً یکسان مهارت پیدا کنیم، می‌توانیم پروژه را به‌خوبی مدیریت کنیم، زیرا درک خوبی از مدت زمان انجام کارها خواهیم داشت. از آنجا که کار ما انتخاب کارهایی است که باید انجام شوند در مقابل کارهایی که باید به تعویق بیفتند، و انتخاب باارزش‌ترین کارها در اولویت است، می‌توانیم پروژه را بدون سربار تخمین‌ها به موفقیت هدایت کنیم.   
  
تخمین‌ها همچنان یک موضوع بحث‌برانگیز در توسعه نرم‌افزار امروز هستند. بسیاری از افراد فکر می‌کنند که تخمین‌ها و برنامه‌ریزی دقیق مهم هستند. قطعاً برخی شرکت‌ها به‌شدت این موارد را می‌خواهند و این ممکن است به‌اندازه کافی باعث شود که ما آن‌ها را انجام دهیم. متأسفانه، «به‌شدت» تقریباً همیشه همان چیزی است که ما با تخمین‌ها انجام می‌دهیم. برای تیم‌هایی که در حال کار هستند، تخمین‌ها در بهترین حالت یک موضوع داخلی هستند و ممکن است تقریباً به‌طور کامل اتلاف باشند.   
  
به‌طور کلی، تخمین‌ها احتمالاً اشتباه هستند و توجه ما را به هزینه کارها معطوف می‌کنند تا به ارزش آن‌ها. در نظر بگیرید که تأکید بر تخمین هزینه‌ها را کاهش دهید یا آن‌ها را حذف کنید و با تمرکز بر ارزش، پروژه را به موفقیت هدایت کنید.   
\*\*فصل ۵.\*\*

================================================================================

# Section 48

## Original Text:

Planning Feature by Feature • 40  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Working without estimates  
Once we become proficient at breaking all features down to  
approximately the same size, we can manage the project  
very nicely, because we have a good sense of how long things  
will take. Since our job is to select the work to do versus the  
work to defer, and to select the most valuable work first, we  
can steer the project to success without the overhead of  
estimation. Estimates remain a controversial topic in software develop-  
ment today. Many people think that estimates and detailed  
planning are important. Certainly some companies want  
these things badly, and that may be enough to cause us to  
do them. Unfortunately, “badly” is almost always how we  
do with estimates. For teams in flight, estimation is at most  
an internal matter and may well be almost entirely waste. Generally speaking, estimates are likely to be wrong, and  
they focus our attention on the cost of things rather than on  
value. Consider de-emphasizing or eliminating cost estimates  
and steering to success by a focus on value. Chapter 5. Planning Feature by Feature • 40  
report erratum • discuss]  
  
Plan often, select what’s next, don’t overeat. As the project proceeds, we plan every couple of weeks, we  
decide what the next most important things are to do, and  
the team decides how many of them to take on. Choose the  
most valuable ideas first—that’s how we grow value most  
rapidly. Even over the two-week period of the typical cadence, expect  
to learn. Sometimes we’ll have been too conservative and  
can pick up a bit more work toward the end. Often, we’ll  
have been too optimistic, or under a bit of pressure, and  
we’ll have taken on too much. When there’s too much food on your plate, don’t eat it. That  
way lies obesity and lethargy. We can’t work well with fat,  
tired code. It’s far better to do eight things well than ten  
things poorly.

## Translated Text:

برنامه‌ریزی ویژگی به ویژگی • ۴۰   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: کار کردن بدون تخمین   
وقتی ما مهارت کافی در شکستن تمام ویژگی‌ها به اندازه‌های تقریباً یکسان پیدا کنیم، می‌توانیم پروژه را به خوبی مدیریت کنیم، زیرا درک خوبی از مدت زمان لازم برای انجام کارها خواهیم داشت. از آنجا که وظیفه ما انتخاب کارهایی است که باید انجام شود در مقابل کارهایی که باید به تعویق بیفتد، و انتخاب کارهای با ارزش‌تر در اولویت است، می‌توانیم پروژه را بدون سربار تخمین‌زنی به موفقیت هدایت کنیم. تخمین‌ها همچنان موضوعی بحث‌برانگیز در توسعه نرم‌افزار هستند. بسیاری از افراد معتقدند که تخمین‌ها و برنامه‌ریزی دقیق مهم هستند. قطعاً برخی شرکت‌ها به شدت خواهان این موارد هستند و این ممکن است باعث شود ما آن‌ها را انجام دهیم. متأسفانه، «به شدت» تقریباً همیشه نحوه برخورد ما با تخمین‌ها است. برای تیم‌هایی که در حال حرکت هستند، تخمین‌زنی در بهترین حالت یک موضوع داخلی است و ممکن است تقریباً به طور کامل اتلاف باشد. به طور کلی، تخمین‌ها احتمالاً اشتباه هستند و توجه ما را به جای ارزش، به هزینه‌ها معطوف می‌کنند. کاهش تأکید یا حذف تخمین هزینه‌ها را در نظر بگیرید و با تمرکز بر ارزش، پروژه را به موفقیت هدایت کنید.   
فصل ۵. برنامه‌ریزی ویژگی به ویژگی • ۴۰   
گزارش خطا • بحث کنید]   
  
اغلب برنامه‌ریزی کنید، انتخاب کنید که چه چیزی بعدی است، زیاده‌روی نکنید.   
همان‌طور که پروژه پیش می‌رود، هر چند هفته یک بار برنامه‌ریزی می‌کنیم، تصمیم می‌گیریم که مهم‌ترین کارهای بعدی چیست، و تیم تصمیم می‌گیرد که چه تعداد از آن‌ها را انجام دهد. ابتدا ایده‌های با ارزش‌تر را انتخاب کنید—این همان روشی است که سریع‌ترین رشد ارزش را به همراه دارد. حتی در طول دوره دو هفته‌ای ریتم معمول، انتظار یادگیری داشته باشید. گاهی اوقات ممکن است بیش از حد محافظه‌کار بوده باشیم و بتوانیم کمی کار بیشتر را در انتهای دوره انجام دهیم. اغلب، ممکن است بیش از حد خوش‌بین بوده باشیم یا تحت فشار کمی قرار گرفته باشیم و کارهای زیادی را بر عهده گرفته باشیم. وقتی غذای زیادی در بشقاب شماست، آن را نخورید. این راه منجر به چاقی و کسالت می‌شود. ما نمی‌توانیم با کدهای سنگین و خسته به خوبی کار کنیم. انجام هشت کار به خوبی بسیار بهتر از انجام ده کار به صورت ضعیف است.

================================================================================

# Section 49

## Original Text:

As soon as you realize the team has taken on  
too much, remove something from their plate. We’re in this  
for the whole project, and staying healthy is critical. Further reading:  
•Chapter 15, The “Five-Card Method”for Initial Forecasting ,  
on page 98. report erratum • discussChapter 5. Planning Feature by Feature • 41  
  
[CONTEXT FROM PREVIOUS PAGE: Plan often, select what’s next, don’t overeat. As the project proceeds, we plan every couple of weeks, we  
decide what the next most important things are to do, and  
the team decides how many of them to take on. Choose the  
most valuable ideas first—that’s how we grow value most  
rapidly. Even over the two-week period of the typical cadence, expect  
to learn. Sometimes we’ll have been too conservative and  
can pick up a bit more work toward the end. Often, we’ll  
have been too optimistic, or under a bit of pressure, and  
we’ll have taken on too much. When there’s too much food on your plate, don’t eat it. That  
way lies obesity and lethargy. We can’t work well with fat,  
tired code. It’s far better to do eight things well than ten  
things poorly. As soon as you realize the team has taken on  
too much, remove something from their plate. We’re in this  
for the whole project, and staying healthy is critical. Further reading:  
•Chapter 15, The “Five-Card Method”for Initial Forecasting ,  
on page 98. report erratum • discussChapter 5. Planning Feature by Feature • 41]  
  
CHAPTER 6  
Building the Product,  
Feature by Feature  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 6  
Building the Product,  
Feature by Feature  
report erratum • discuss]  
  
Working feature by feature delivers better value. Because  
the team can show us the software, it’s easy to manage and  
plan. But is this a practical approach. Can our developers  
really build software feature by feature. Absolutely. Feature-by-feature development is quite possible. Teams have been doing it successfully for decades. Hundreds  
of organizations have learned how.

## Translated Text:

به محض اینکه متوجه شدید تیم بیش از حد کار به عهده گرفته است، بخشی از وظایف آن‌ها را حذف کنید. ما برای کل پروژه اینجا هستیم و حفظ سلامت بسیار مهم است. مطالعه بیشتر:   
• فصل ۱۵، روش "Five-Card" برای پیش‌بینی اولیه،   
در صفحه ۹۸. گزارش خطا • بحث فصل ۵. برنامه‌ریزی بر اساس Feature • ۴۱   
  
[متن از صفحه قبلی: برنامه‌ریزی مکرر، انتخاب کار بعدی، پرهیز از زیاده‌روی. همان‌طور که پروژه پیش می‌رود، هر چند هفته یک بار برنامه‌ریزی می‌کنیم، تصمیم می‌گیریم که مهم‌ترین کارهای بعدی چیست، و تیم تصمیم می‌گیرد که چه تعداد از آن‌ها را انجام دهد. ابتدا ایده‌های با ارزش‌تر را انتخاب کنید—این بهترین روش برای رشد سریع‌تر ارزش است. حتی در دوره دو هفته‌ای معمول، انتظار یادگیری داشته باشید. گاهی ممکن است بیش از حد محافظه‌کار بوده باشیم و بتوانیم کمی کار بیشتر در انتها انجام دهیم. اغلب، ممکن است بیش از حد خوش‌بین بوده باشیم یا تحت فشار کمی قرار گرفته باشیم و بیش از حد کار به عهده گرفته باشیم. وقتی غذا زیادی روی بشقاب دارید، آن را نخورید. این مسیر به چاقی و رخوت منجر می‌شود. ما نمی‌توانیم با Code چاق و خسته خوب کار کنیم. انجام هشت کار به خوبی بسیار بهتر از انجام ده کار به صورت ضعیف است. به محض اینکه متوجه شدید تیم بیش از حد کار به عهده گرفته است، بخشی از وظایف آن‌ها را حذف کنید. ما برای کل پروژه اینجا هستیم و حفظ سلامت بسیار مهم است. مطالعه بیشتر:   
• فصل ۱۵، روش "Five-Card" برای پیش‌بینی اولیه،   
در صفحه ۹۸. گزارش خطا • بحث فصل ۵. برنامه‌ریزی بر اساس Feature • ۴۱]   
  
فصل ۶   
ساخت محصول،   
بر اساس Feature   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: فصل ۶   
ساخت محصول،   
بر اساس Feature   
گزارش خطا • بحث]   
  
کار کردن بر اساس Feature ارزش بهتری ارائه می‌دهد. زیرا تیم می‌تواند نرم‌افزار را به ما نشان دهد، مدیریت و برنامه‌ریزی آسان می‌شود. اما آیا این رویکرد عملی است؟ آیا توسعه‌دهندگان ما واقعاً می‌توانند نرم‌افزار را بر اساس Feature بسازند؟ کاملاً. توسعه بر اساس Feature کاملاً ممکن است. تیم‌ها دهه‌هاست که این کار را با موفقیت انجام داده‌اند. صدها سازمان این روش را یاد گرفته‌اند.

================================================================================

# Section 50

## Original Text:

Every kind of software  
that you can imagine is being built feature by feature right  
now—yes, even the kind of software that you build. You and your organization can benefit from the feature-by-  
feature approach. Everyone needs to learn a bit, and change  
a bit. Improved management and faster return makes it  
worthwhile. Let’s take a quick look at what’s required to work feature  
by feature, and then drill into some details. If we want small features frequently, what, in general, will  
all of us need to do. What specific pitfalls do you see. report erratum • discussChapter 6. Building the Product, Feature by Feature • 43  
  
[CONTEXT FROM PREVIOUS PAGE: Working feature by feature delivers better value. Because  
the team can show us the software, it’s easy to manage and  
plan. But is this a practical approach. Can our developers  
really build software feature by feature. Absolutely. Feature-by-feature development is quite possible. Teams have been doing it successfully for decades. Hundreds  
of organizations have learned how. Every kind of software  
that you can imagine is being built feature by feature right  
now—yes, even the kind of software that you build. You and your organization can benefit from the feature-by-  
feature approach. Everyone needs to learn a bit, and change  
a bit. Improved management and faster return makes it  
worthwhile. Let’s take a quick look at what’s required to work feature  
by feature, and then drill into some details. If we want small features frequently, what, in general, will  
all of us need to do. What specific pitfalls do you see. report erratum • discussChapter 6. Building the Product, Feature by Feature • 43]  
  
Build a tiny product, completely, in each small cycle. We plan and manage our project in short one- or two-week  
cycles. In each cycle, we define the next few features to build  
and we say how they’ll be tested. The team then builds the  
features and we all verify whether they pass the tests.

## Translated Text:

هر نوع نرم‌افزاری که تصور کنید، در حال حاضر به صورت feature by feature ساخته می‌شود—بله، حتی نرم‌افزاری که شما می‌سازید. شما و سازمانتان می‌توانید از رویکرد feature-by-feature بهره‌مند شوید. همه باید کمی یاد بگیرند و کمی تغییر کنند. مدیریت بهبود یافته و بازگشت سریع‌تر این کار را ارزشمند می‌کند. بیایید نگاهی سریع به آنچه برای کار به صورت feature by feature لازم است بیندازیم و سپس به جزئیات بپردازیم. اگر بخواهیم ویژگی‌های کوچک را به طور مکرر داشته باشیم، به طور کلی، همه ما باید چه کاری انجام دهیم؟ چه مشکلات خاصی را می‌بینید؟  
  
[متن از صفحه قبلی: کار به صورت feature by feature ارزش بهتری ارائه می‌دهد. چون تیم می‌تواند نرم‌افزار را به ما نشان دهد، مدیریت و برنامه‌ریزی آسان‌تر می‌شود. اما آیا این یک رویکرد عملی است؟ آیا توسعه‌دهندگان ما واقعاً می‌توانند نرم‌افزار را feature by feature بسازند؟ قطعاً. توسعه به صورت feature-by-feature کاملاً ممکن است. تیم‌ها دهه‌هاست که این کار را با موفقیت انجام داده‌اند. صدها سازمان یاد گرفته‌اند چگونه این کار را انجام دهند. هر نوع نرم‌افزاری که تصور کنید، در حال حاضر به صورت feature by feature ساخته می‌شود—بله، حتی نرم‌افزاری که شما می‌سازید. شما و سازمانتان می‌توانید از رویکرد feature-by-feature بهره‌مند شوید. همه باید کمی یاد بگیرند و کمی تغییر کنند. مدیریت بهبود یافته و بازگشت سریع‌تر این کار را ارزشمند می‌کند. بیایید نگاهی سریع به آنچه برای کار به صورت feature by feature لازم است بیندازیم و سپس به جزئیات بپردازیم. اگر بخواهیم ویژگی‌های کوچک را به طور مکرر داشته باشیم، به طور کلی، همه ما باید چه کاری انجام دهیم؟ چه مشکلات خاصی را می‌بینید؟]  
  
یک محصول کوچک را به طور کامل در هر چرخه کوچک بسازید. ما پروژه خود را در چرخه‌های کوتاه یک یا دو هفته‌ای برنامه‌ریزی و مدیریت می‌کنیم. در هر چرخه، چند ویژگی بعدی که باید ساخته شوند را تعریف می‌کنیم و می‌گوییم چگونه تست خواهند شد. سپس تیم ویژگی‌ها را می‌سازد و همه ما بررسی می‌کنیم که آیا تست‌ها را پاس می‌کنند یا خیر.

================================================================================

# Section 51

## Original Text:

In each of these one- or two-week iterations, we go through  
a complete product development cycle, from concept to  
ready to ship. At first we may not be very good at it, but in  
a few iterations we’ll begin to get the process shaken down. At that point we’ll have an increasingly predictable series  
of subsets of the project, each one cleaner, and more ready  
to go, than the last. In each cycle, we learn. We learn how much we can do in a  
couple of weeks. We learn how to test whether we have done  
what we set out to do. We learn the most effective way to  
define features without excess overhead. We learn how to  
build feature by feature while keeping the code and design  
clear and alive. What difficulties can you foresee if you begin to build com-  
plete versions of the product in small cycles. What could go  
wrong. What will you have to learn. Chapter 6. Building the Product, Feature by Feature • 44  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Build a tiny product, completely, in each small cycle. We plan and manage our project in short one- or two-week  
cycles. In each cycle, we define the next few features to build  
and we say how they’ll be tested. The team then builds the  
features and we all verify whether they pass the tests. In each of these one- or two-week iterations, we go through  
a complete product development cycle, from concept to  
ready to ship. At first we may not be very good at it, but in  
a few iterations we’ll begin to get the process shaken down. At that point we’ll have an increasingly predictable series  
of subsets of the project, each one cleaner, and more ready  
to go, than the last. In each cycle, we learn. We learn how much we can do in a  
couple of weeks. We learn how to test whether we have done  
what we set out to do. We learn the most effective way to  
define features without excess overhead. We learn how to  
build feature by feature while keeping the code and design  
clear and alive.

## Translated Text:

در هر یک از این تکرارهای یک یا دو هفته‌ای، ما یک چرخه کامل توسعه محصول را از مفهوم تا آماده برای عرضه طی می‌کنیم. در ابتدا ممکن است در این کار خیلی خوب نباشیم، اما پس از چند تکرار، فرآیند را بهتر درک خواهیم کرد. در آن مرحله، ما یک سری زیرمجموعه‌های قابل پیش‌بینی از پروژه خواهیم داشت که هر کدام تمیزتر و آماده‌تر از قبلی هستند.   
  
در هر چرخه، ما یاد می‌گیریم. یاد می‌گیریم که در طول یک یا دو هفته چه مقدار کار می‌توانیم انجام دهیم. یاد می‌گیریم که چگونه تست کنیم تا مطمئن شویم آنچه را که قصد انجامش را داشتیم، انجام داده‌ایم. یاد می‌گیریم که چگونه ویژگی‌ها را بدون ایجاد سربار اضافی تعریف کنیم. یاد می‌گیریم که چگونه ویژگی به ویژگی بسازیم، در حالی که کد و طراحی را شفاف و پویا نگه داریم.  
  
چه مشکلاتی را می‌توانید پیش‌بینی کنید اگر شروع به ساخت نسخه‌های کامل محصول در چرخه‌های کوچک کنید؟ چه چیزی ممکن است اشتباه پیش برود؟ چه چیزهایی را باید یاد بگیرید؟  
  
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۴   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: یک محصول کوچک را به طور کامل در هر چرخه کوچک بسازید. ما پروژه خود را در چرخه‌های کوتاه یک یا دو هفته‌ای برنامه‌ریزی و مدیریت می‌کنیم. در هر چرخه، چند ویژگی بعدی که باید ساخته شوند را تعریف می‌کنیم و مشخص می‌کنیم که چگونه تست خواهند شد. سپس تیم ویژگی‌ها را می‌سازد و همه ما بررسی می‌کنیم که آیا آنها تست‌ها را پاس کرده‌اند یا خیر.   
  
در هر یک از این تکرارهای یک یا دو هفته‌ای، ما یک چرخه کامل توسعه محصول را از مفهوم تا آماده برای عرضه طی می‌کنیم. در ابتدا ممکن است در این کار خیلی خوب نباشیم، اما پس از چند تکرار، فرآیند را بهتر درک خواهیم کرد. در آن مرحله، ما یک سری زیرمجموعه‌های قابل پیش‌بینی از پروژه خواهیم داشت که هر کدام تمیزتر و آماده‌تر از قبلی هستند.   
  
در هر چرخه، ما یاد می‌گیریم. یاد می‌گیریم که در طول یک یا دو هفته چه مقدار کار می‌توانیم انجام دهیم. یاد می‌گیریم که چگونه تست کنیم تا مطمئن شویم آنچه را که قصد انجامش را داشتیم، انجام داده‌ایم. یاد می‌گیریم که چگونه ویژگی‌ها را بدون ایجاد سربار اضافی تعریف کنیم. یاد می‌گیریم که چگونه ویژگی به ویژگی بسازیم، در حالی که کد و طراحی را شفاف و پویا نگه داریم.]

================================================================================

# Section 52

## Original Text:

What difficulties can you foresee if you begin to build com-  
plete versions of the product in small cycles. What could go  
wrong. What will you have to learn. Chapter 6. Building the Product, Feature by Feature • 44  
report erratum • discuss]  
  
Refine the product vision. In a feature-by-feature approach, we ask the development  
team to build features every couple of weeks. We want to  
see things coming out that we actually want, not some  
technical whatnot that we don’t understand. This means that our business-side people need to build up  
skill in breaking down large, vague, sweeping requirements  
into small, practical next steps that deliver maximum value  
for minimum effort. They’ll need help from around the  
organization, and perhaps from outside, to get good at this. Mostly, they need to work at it: trust them and their teams  
to figure out what they need. This is no small task, but it’s critical to success. We need to  
sharpen our vision of what the product must do—and what’s  
just “nice to have. ” The result is a faster return on our soft-  
ware investment. What big features have you in mind for your product. What  
small pieces can you identify that would help you see how  
you’re doing and what to do next. What software should  
we show you, right now. report erratum • discussChapter 6. Building the Product, Feature by Feature • 45  
  
[CONTEXT FROM PREVIOUS PAGE: Refine the product vision. In a feature-by-feature approach, we ask the development  
team to build features every couple of weeks. We want to  
see things coming out that we actually want, not some  
technical whatnot that we don’t understand. This means that our business-side people need to build up  
skill in breaking down large, vague, sweeping requirements  
into small, practical next steps that deliver maximum value  
for minimum effort. They’ll need help from around the  
organization, and perhaps from outside, to get good at this. Mostly, they need to work at it: trust them and their teams  
to figure out what they need.

## Translated Text:

چه مشکلاتی را می‌توانید پیش‌بینی کنید اگر شروع به Build کردن نسخه‌های کامل محصول در چرخه‌های کوچک کنید؟ چه چیزی ممکن است اشتباه پیش برود؟ چه چیزی باید یاد بگیرید؟   
فصل ۶. ساخت محصول، Feature به Feature • ۴۴   
[گزارش خطا • بحث کنید]  
  
بهبود چشم‌انداز محصول. در رویکرد Feature به Feature، ما از تیم توسعه می‌خواهیم هر چند هفته یک بار Features را Build کنند. ما می‌خواهیم چیزهایی را ببینیم که واقعاً می‌خواهیم، نه یک سری موارد فنی که نمی‌فهمیم. این به این معناست که افراد بخش کسب‌وکار ما باید مهارت خود را در تجزیه و تحلیل نیازهای بزرگ، مبهم و کلی به گام‌های کوچک و عملی که بیشترین ارزش را با کمترین تلاش ارائه می‌دهند، افزایش دهند. آن‌ها برای بهتر شدن در این کار به کمک از سراسر سازمان و شاید از خارج نیاز خواهند داشت. بیشتر از همه، آن‌ها باید روی این موضوع کار کنند: به آن‌ها و تیم‌هایشان اعتماد کنید تا بفهمند چه چیزی نیاز دارند.   
  
این کار آسان نیست، اما برای موفقیت حیاتی است. ما باید چشم‌انداز خود را در مورد اینکه محصول چه کاری باید انجام دهد—و چه چیزی فقط "خوب است که داشته باشیم"—شفاف‌تر کنیم. نتیجه این کار بازگشت سریع‌تر سرمایه‌گذاری ما در نرم‌افزار خواهد بود. چه Features بزرگی برای محصول خود در نظر دارید؟ چه قطعات کوچکی را می‌توانید شناسایی کنید که به شما کمک کند بفهمید چگونه پیش می‌روید و چه کاری باید انجام دهید؟ چه نرم‌افزاری را باید همین حالا به شما نشان دهیم؟   
[گزارش خطا • بحث کنید]   
فصل ۶. ساخت محصول، Feature به Feature • ۴۵   
  
[متن از صفحه قبلی: بهبود چشم‌انداز محصول. در رویکرد Feature به Feature، ما از تیم توسعه می‌خواهیم هر چند هفته یک بار Features را Build کنند. ما می‌خواهیم چیزهایی را ببینیم که واقعاً می‌خواهیم، نه یک سری موارد فنی که نمی‌فهمیم. این به این معناست که افراد بخش کسب‌وکار ما باید مهارت خود را در تجزیه و تحلیل نیازهای بزرگ، مبهم و کلی به گام‌های کوچک و عملی که بیشترین ارزش را با کمترین تلاش ارائه می‌دهند، افزایش دهند. آن‌ها برای بهتر شدن در این کار به کمک از سراسر سازمان و شاید از خارج نیاز خواهند داشت. بیشتر از همه، آن‌ها باید روی این موضوع کار کنند: به آن‌ها و تیم‌هایشان اعتماد کنید تا بفهمند چه چیزی نیاز دارند.]

================================================================================

# Section 53

## Original Text:

This is no small task, but it’s critical to success. We need to  
sharpen our vision of what the product must do—and what’s  
just “nice to have. ” The result is a faster return on our soft-  
ware investment. What big features have you in mind for your product. What  
small pieces can you identify that would help you see how  
you’re doing and what to do next. What software should  
we show you, right now. report erratum • discussChapter 6. Building the Product, Feature by Feature • 45]  
  
Always work on the highest possible value next. As we progress through the iterations, it becomes increasing-  
ly clear how much work we can finish by “the date. ” It’s  
critical to do the most valuable features first. The team works  
together to identify the high-value, low-cost features we can  
do. The team learns to build the best possible product within  
the time and money available. We must do all this in the open. Everyone needs to see the  
actual progress. We can’t accept “90 percent done. ” Features  
need to be “done” or “not done”: there is no middle ground. We have to see what’s going on inside the project so that we  
can guide it to the best possible conclusion. What do we need to know about each feature as it gets  
“done”. What could interfere with our ability to guide the  
project and grow the product. Chapter 6. Building the Product, Feature by Feature • 46  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Always work on the highest possible value next. As we progress through the iterations, it becomes increasing-  
ly clear how much work we can finish by “the date. ” It’s  
critical to do the most valuable features first. The team works  
together to identify the high-value, low-cost features we can  
do. The team learns to build the best possible product within  
the time and money available. We must do all this in the open. Everyone needs to see the  
actual progress. We can’t accept “90 percent done. ” Features  
need to be “done” or “not done”: there is no middle ground.

## Translated Text:

این کار کوچکی نیست، اما برای موفقیت حیاتی است. ما باید دیدگاه خود را در مورد اینکه محصول باید چه کاری انجام دهد و چه چیزی فقط "خوب است که داشته باشیم" واضح‌تر کنیم. نتیجه این کار بازگشت سریع‌تر سرمایه‌گذاری نرم‌افزاری ما خواهد بود. چه ویژگی‌های بزرگی برای محصول خود در نظر دارید؟ چه قطعات کوچکی را می‌توانید شناسایی کنید که به شما کمک کند ببینید چگونه پیش می‌روید و چه کاری باید بعد انجام دهید؟ چه نرم‌افزاری باید همین حالا به شما نشان دهیم؟   
report erratum • discuss   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۵   
  
همیشه روی بالاترین ارزش ممکن بعدی کار کنید. همان‌طور که از طریق Iterationها پیشرفت می‌کنیم، به‌طور فزاینده‌ای مشخص می‌شود که چقدر کار می‌توانیم تا "تاریخ" انجام دهیم. انجام مهم‌ترین ویژگی‌ها در ابتدا حیاتی است. تیم با همکاری یکدیگر ویژگی‌های با ارزش بالا و هزینه پایین را که می‌توانیم انجام دهیم شناسایی می‌کند. تیم یاد می‌گیرد که بهترین محصول ممکن را در زمان و بودجه موجود بسازد. ما باید همه این کارها را به‌صورت شفاف انجام دهیم. همه باید پیشرفت واقعی را ببینند. ما نمی‌توانیم "۹۰ درصد انجام شده" را بپذیریم. ویژگی‌ها باید یا "انجام شده" باشند یا "انجام نشده": هیچ حد وسطی وجود ندارد. ما باید ببینیم داخل پروژه چه اتفاقی می‌افتد تا بتوانیم آن را به بهترین نتیجه ممکن هدایت کنیم.   
چه چیزی باید درباره هر ویژگی بدانیم وقتی که "انجام شده" است؟ چه چیزی ممکن است توانایی ما را برای هدایت پروژه و رشد محصول مختل کند؟   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۶   
report erratum • discuss   
  
[زمینه از صفحه قبلی: همیشه روی بالاترین ارزش ممکن بعدی کار کنید. همان‌طور که از طریق Iterationها پیشرفت می‌کنیم، به‌طور فزاینده‌ای مشخص می‌شود که چقدر کار می‌توانیم تا "تاریخ" انجام دهیم. انجام مهم‌ترین ویژگی‌ها در ابتدا حیاتی است. تیم با همکاری یکدیگر ویژگی‌های با ارزش بالا و هزینه پایین را که می‌توانیم انجام دهیم شناسایی می‌کند. تیم یاد می‌گیرد که بهترین محصول ممکن را در زمان و بودجه موجود بسازد. ما باید همه این کارها را به‌صورت شفاف انجام دهیم. همه باید پیشرفت واقعی را ببینند. ما نمی‌توانیم "۹۰ درصد انجام شده" را بپذیریم. ویژگی‌ها باید یا "انجام شده" باشند یا "انجام نشده": هیچ حد وسطی وجود ندارد.]

================================================================================

# Section 54

## Original Text:

We have to see what’s going on inside the project so that we  
can guide it to the best possible conclusion. What do we need to know about each feature as it gets  
“done”. What could interfere with our ability to guide the  
project and grow the product. Chapter 6. Building the Product, Feature by Feature • 46  
report erratum • discuss]  
  
Identify real progress. The feature-by-feature style includes a complete develop-  
ment cycle in every iteration: requirements, design, coding,  
and testing. They are all present all the time so that they’re  
all visible all the time. Feature-by-feature development is  
safe, practical, and effective. It will work for you. As our project goes on, we’ll be learning how to tell the dif-  
ference between apparent progress and real progress. We’ll  
be building up an understanding of when “done” really  
means done in our unique situation. When we can see real, running features, we have clear and  
solid information about our project’s condition. When we’re  
not able to look at features, or they aren’t done, we won’t  
know what’s going on. When your team starts building features and telling you  
they’re done, what might make those glowing reports mis-  
leadingly optimistic. How can they be made more accurate. report erratum • discussChapter 6. Building the Product, Feature by Feature • 47  
  
[CONTEXT FROM PREVIOUS PAGE: Identify real progress. The feature-by-feature style includes a complete develop-  
ment cycle in every iteration: requirements, design, coding,  
and testing. They are all present all the time so that they’re  
all visible all the time. Feature-by-feature development is  
safe, practical, and effective. It will work for you. As our project goes on, we’ll be learning how to tell the dif-  
ference between apparent progress and real progress. We’ll  
be building up an understanding of when “done” really  
means done in our unique situation. When we can see real, running features, we have clear and  
solid information about our project’s condition.

## Translated Text:

باید ببینیم درون پروژه چه می‌گذرد تا بتوانیم آن را به بهترین نتیجه ممکن هدایت کنیم. چه چیزهایی باید درباره هر Feature بدانیم وقتی که "done" می‌شود؟ چه چیزی می‌تواند مانع توانایی ما در هدایت پروژه و رشد Product شود؟   
فصل ۶. ساخت Product، Feature به Feature • ۴۶   
report erratum • discuss   
  
تشخیص پیشرفت واقعی. سبک Feature-by-Feature شامل یک چرخه کامل توسعه در هر Iteration است: Requirements، Design، Coding، و Testing. همه این‌ها همیشه حاضر هستند تا همیشه قابل مشاهده باشند. توسعه به سبک Feature-by-Feature امن، عملی و مؤثر است. این روش برای شما کار خواهد کرد.   
همان‌طور که پروژه ما پیش می‌رود، یاد خواهیم گرفت که تفاوت بین پیشرفت ظاهری و پیشرفت واقعی را تشخیص دهیم. ما درک خواهیم کرد که در شرایط منحصربه‌فرد ما، "done" واقعاً چه زمانی به معنای تمام شدن است.   
وقتی می‌توانیم Features واقعی و در حال اجرا را ببینیم، اطلاعات شفاف و محکمی درباره وضعیت پروژه خود داریم. وقتی نمی‌توانیم به Features نگاه کنیم یا آن‌ها تمام نشده‌اند، نمی‌دانیم چه می‌گذرد.   
وقتی تیم شما شروع به ساخت Features می‌کند و به شما می‌گوید که آن‌ها "done" هستند، چه چیزی ممکن است این گزارش‌های امیدوارکننده را به‌طور گمراه‌کننده‌ای خوش‌بینانه کند؟ چگونه می‌توان آن‌ها را دقیق‌تر کرد؟   
report erratum • discuss   
فصل ۶. ساخت Product، Feature به Feature • ۴۷   
  
[متن از صفحه قبلی: تشخیص پیشرفت واقعی. سبک Feature-by-Feature شامل یک چرخه کامل توسعه در هر Iteration است: Requirements، Design، Coding، و Testing. همه این‌ها همیشه حاضر هستند تا همیشه قابل مشاهده باشند. توسعه به سبک Feature-by-Feature امن، عملی و مؤثر است. این روش برای شما کار خواهد کرد.   
همان‌طور که پروژه ما پیش می‌رود، یاد خواهیم گرفت که تفاوت بین پیشرفت ظاهری و پیشرفت واقعی را تشخیص دهیم. ما درک خواهیم کرد که در شرایط منحصربه‌فرد ما، "done" واقعاً چه زمانی به معنای تمام شدن است.   
وقتی می‌توانیم Features واقعی و در حال اجرا را ببینیم، اطلاعات شفاف و محکمی درباره وضعیت پروژه خود داریم.]

================================================================================

# Section 55

## Original Text:

When we’re  
not able to look at features, or they aren’t done, we won’t  
know what’s going on. When your team starts building features and telling you  
they’re done, what might make those glowing reports mis-  
leadingly optimistic. How can they be made more accurate. report erratum • discussChapter 6. Building the Product, Feature by Feature • 47]  
  
Eliminate the test-and-fix finish. Many projects end with a test-and-fix interval that seems to  
drag on forever, furrowing into our soul. This can even  
happen when we work feature by feature—if the features  
aren’t really done. Because we’re learning to complete each one- or two-week  
iteration with finished, potentially shippable software, we  
are learning how to reduce or eliminate that long unknown  
block of test and fix time that shows up at the end of so many  
projects. For feature-by-feature development to work, the software  
needs to be nearly free of defects at the end of every two-  
week iteration. It needs to be nearly free of defects all the  
time. To be sure we’re free of defects, we need to check everything,  
all the time. That’s not as hard as you think; we’ll talk more  
about it later. But first, there’s more. Chapter 6. Building the Product, Feature by Feature • 48  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Eliminate the test-and-fix finish. Many projects end with a test-and-fix interval that seems to  
drag on forever, furrowing into our soul. This can even  
happen when we work feature by feature—if the features  
aren’t really done. Because we’re learning to complete each one- or two-week  
iteration with finished, potentially shippable software, we  
are learning how to reduce or eliminate that long unknown  
block of test and fix time that shows up at the end of so many  
projects. For feature-by-feature development to work, the software  
needs to be nearly free of defects at the end of every two-  
week iteration. It needs to be nearly free of defects all the  
time.

## Translated Text:

وقتی نمی‌توانیم به ویژگی‌ها نگاه کنیم، یا آنها کامل نشده‌اند، نمی‌توانیم بفهمیم چه اتفاقی در حال رخ دادن است. وقتی تیم شما شروع به ساختن ویژگی‌ها می‌کند و به شما می‌گوید که آنها کامل شده‌اند، چه چیزی ممکن است باعث شود این گزارش‌های درخشان به طرز گمراه‌کننده‌ای خوش‌بینانه باشند؟ چگونه می‌توان آنها را دقیق‌تر کرد؟   
report erratum • discuss   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۷   
  
حذف پایان تست-و-اصلاح. بسیاری از پروژه‌ها با یک دوره تست-و-اصلاح به پایان می‌رسند که به نظر می‌رسد تا ابد ادامه دارد و روح ما را خسته می‌کند. این حتی زمانی که ما به صورت ویژگی به ویژگی کار می‌کنیم نیز ممکن است اتفاق بیفتد—اگر ویژگی‌ها واقعاً کامل نشده باشند. چون ما در حال یادگیری تکمیل هر iteration یک یا دو هفته‌ای با نرم‌افزار کامل و بالقوه قابل انتشار هستیم، در حال یادگیری کاهش یا حذف آن بلوک طولانی و ناشناخته زمان تست و اصلاح هستیم که در انتهای بسیاری از پروژه‌ها ظاهر می‌شود.   
  
برای اینکه توسعه ویژگی به ویژگی کار کند، نرم‌افزار باید در پایان هر iteration دو هفته‌ای تقریباً بدون نقص باشد. نرم‌افزار باید همیشه تقریباً بدون نقص باشد. برای اطمینان از اینکه ما بدون نقص هستیم، باید همه چیز را همیشه بررسی کنیم. این به آن سختی که فکر می‌کنید نیست؛ بعداً بیشتر در مورد آن صحبت خواهیم کرد. اما اول، چیزهای بیشتری وجود دارد.   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۸   
report erratum • discuss   
  
[متن از صفحه قبلی: حذف پایان تست-و-اصلاح. بسیاری از پروژه‌ها با یک دوره تست-و-اصلاح به پایان می‌رسند که به نظر می‌رسد تا ابد ادامه دارد و روح ما را خسته می‌کند. این حتی زمانی که ما به صورت ویژگی به ویژگی کار می‌کنیم نیز ممکن است اتفاق بیفتد—اگر ویژگی‌ها واقعاً کامل نشده باشند. چون ما در حال یادگیری تکمیل هر iteration یک یا دو هفته‌ای با نرم‌افزار کامل و بالقوه قابل انتشار هستیم، در حال یادگیری کاهش یا حذف آن بلوک طولانی و ناشناخته زمان تست و اصلاح هستیم که در انتهای بسیاری از پروژه‌ها ظاهر می‌شود. برای اینکه توسعه ویژگی به ویژگی کار کند، نرم‌افزار باید در پایان هر iteration دو هفته‌ای تقریباً بدون نقص باشد. نرم‌افزار باید همیشه تقریباً بدون نقص باشد.]

================================================================================

# Section 56

## Original Text:

To be sure we’re free of defects, we need to check everything,  
all the time. That’s not as hard as you think; we’ll talk more  
about it later. But first, there’s more. Chapter 6. Building the Product, Feature by Feature • 48  
report erratum • discuss]  
  
Grow and refine the design as we go. As we build feature by feature, being free of defects isn’t  
enough. We also need to grow the design as we go. If we  
design too much, we won’t get as many features, and that  
will show up. If we design too little, features will be hard to  
do, we’ll slow down, and again that will become visible. By observing how our velocity—the pace at which we ship  
features—changes, we’ll learn how to “right-size” our design  
effort. Too much design, we slow down. Too little, we slow  
down. Tweak, observe, tweak again. This is a tricky area. Most of us have seen software whose  
design got so bad that progress became almost impossible. The techniques for keeping the design good enough are easy  
to learn. They are also easy to forget, especially if we put  
teams under enough pressure. How can we help our developers keep the design clean. Do  
they have the needed skills. What should we avoid that  
would discourage clean design. Further reading:  
•Chapter 18, To Speed Up, Build with Skill , on page 116  
•Chapter 19, Refactoring , on page 120  
report erratum • discussChapter 6. Building the Product, Feature by Feature • 49  
  
[CONTEXT FROM PREVIOUS PAGE: Grow and refine the design as we go. As we build feature by feature, being free of defects isn’t  
enough. We also need to grow the design as we go. If we  
design too much, we won’t get as many features, and that  
will show up. If we design too little, features will be hard to  
do, we’ll slow down, and again that will become visible. By observing how our velocity—the pace at which we ship  
features—changes, we’ll learn how to “right-size” our design  
effort. Too much design, we slow down. Too little, we slow  
down. Tweak, observe, tweak again. This is a tricky area.

## Translated Text:

برای اطمینان از اینکه هیچ نقصی وجود ندارد، باید همه چیز را همیشه بررسی کنیم. این کار به آن سختی که فکر می‌کنید نیست؛ بعداً بیشتر در مورد آن صحبت خواهیم کرد. اما ابتدا، موارد بیشتری وجود دارد.   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۸   
report erratum • discuss   
  
طراحی را در حین پیشرفت رشد داده و بهبود دهید. همانطور که ویژگی به ویژگی پیش می‌رویم، فقط عاری از نقص بودن کافی نیست. ما همچنین باید طراحی را در حین پیشرفت رشد دهیم. اگر بیش از حد طراحی کنیم، نمی‌توانیم به تعداد زیادی Feature برسیم و این موضوع آشکار خواهد شد. اگر خیلی کم طراحی کنیم، پیاده‌سازی Features دشوار خواهد شد، سرعت ما کاهش می‌یابد و باز هم این موضوع آشکار خواهد شد.   
با مشاهده تغییرات Velocity—سرعتی که Features را ارائه می‌دهیم—یاد می‌گیریم که چگونه تلاش طراحی خود را به اندازه مناسب تنظیم کنیم. طراحی بیش از حد، ما را کند می‌کند. طراحی کم، ما را کند می‌کند. تنظیم کنید، مشاهده کنید، دوباره تنظیم کنید. این یک حوزه چالش‌برانگیز است.   
  
بیشتر ما نرم‌افزاری را دیده‌ایم که طراحی آن به قدری بد شده که پیشرفت تقریباً غیرممکن شده است. تکنیک‌های حفظ طراحی در حد کافی خوب، آسان برای یادگیری هستند. اما همچنین به راحتی فراموش می‌شوند، به‌ویژه اگر تیم‌ها را تحت فشار زیادی قرار دهیم.   
چگونه می‌توانیم به توسعه‌دهندگان خود کمک کنیم تا طراحی را تمیز نگه دارند؟ آیا مهارت‌های لازم را دارند؟ از چه چیزهایی باید اجتناب کنیم که ممکن است طراحی تمیز را دلسرد کند؟   
  
مطالعه بیشتر:   
• فصل ۱۸، برای سرعت بخشیدن، با مهارت بسازید، در صفحه ۱۱۶   
• فصل ۱۹، Refactoring، در صفحه ۱۲۰   
report erratum • discuss   
فصل ۶. ساخت محصول، ویژگی به ویژگی • ۴۹

================================================================================

# Section 57

## Original Text:

Most of us have seen software whose  
design got so bad that progress became almost impossible. The techniques for keeping the design good enough are easy  
to learn. They are also easy to forget, especially if we put  
teams under enough pressure. How can we help our developers keep the design clean. Do  
they have the needed skills. What should we avoid that  
would discourage clean design. Further reading:  
•Chapter 18, To Speed Up, Build with Skill , on page 116  
•Chapter 19, Refactoring , on page 120  
report erratum • discussChapter 6. Building the Product, Feature by Feature • 49]  
  
Our product needs a solid foundation. CHAPTER 7  
Build Features and  
Foundation in Parallel  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Our product needs a solid foundation. CHAPTER 7  
Build Features and  
Foundation in Parallel  
report erratum • discuss]  
  
Every product has a set of key features needed to deliver  
value. We plan to build those and X out the rest. Everything we build must rest on a solid foundation. We  
often use words like architecture ,design , orinfrastructure when  
talking about the foundation. According to the guidance in this book, we’re going to build  
those features incrementally, starting with the most valuable,  
down to the least valuable. To keep the features coming  
smoothly, we need to keep the system’s design solid from  
the first day to the last. How can we best balance design and features as we go  
along. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 51  
  
[CONTEXT FROM PREVIOUS PAGE: Every product has a set of key features needed to deliver  
value. We plan to build those and X out the rest. Everything we build must rest on a solid foundation. We  
often use words like architecture ,design , orinfrastructure when  
talking about the foundation. According to the guidance in this book, we’re going to build  
those features incrementally, starting with the most valuable,  
down to the least valuable.

## Translated Text:

بیشتر ما نرم‌افزارهایی را دیده‌ایم که طراحی آن‌ها آن‌قدر بد شده که پیشرفت تقریباً غیرممکن شده است. تکنیک‌های حفظ طراحی در حد کافی خوب، آسان برای یادگیری هستند. این تکنیک‌ها همچنین به راحتی فراموش می‌شوند، به‌ویژه اگر تیم‌ها را تحت فشار زیادی قرار دهیم. چگونه می‌توانیم به توسعه‌دهندگان خود کمک کنیم تا طراحی را تمیز نگه دارند؟ آیا آن‌ها مهارت‌های لازم را دارند؟ از چه چیزی باید اجتناب کنیم که ممکن است طراحی تمیز را دلسرد کند؟   
مطالعه بیشتر:   
• فصل ۱۸، To Speed Up, Build with Skill، صفحه ۱۱۶   
• فصل ۱۹، Refactoring، صفحه ۱۲۰   
report erratum • discuss   
فصل ۶. Building the Product, Feature by Feature • 49   
  
محصول ما به یک پایه محکم نیاز دارد.   
فصل ۷   
Build Features and Foundation in Parallel   
report erratum • discuss   
  
[متن از صفحه قبلی: محصول ما به یک پایه محکم نیاز دارد.   
فصل ۷   
Build Features and Foundation in Parallel   
report erratum • discuss]   
  
هر محصول مجموعه‌ای از ویژگی‌های کلیدی دارد که برای ارائه ارزش مورد نیاز است. ما برنامه داریم که آن‌ها را بسازیم و بقیه را کنار بگذاریم. هر چیزی که می‌سازیم باید بر یک پایه محکم استوار باشد. ما اغلب از کلماتی مانند architecture، design یا infrastructure هنگام صحبت درباره پایه استفاده می‌کنیم. طبق راهنمایی‌های این کتاب، ما قصد داریم این ویژگی‌ها را به‌صورت تدریجی بسازیم، از باارزش‌ترین تا کم‌ارزش‌ترین. برای اینکه ویژگی‌ها به‌صورت روان ارائه شوند، باید طراحی سیستم را از روز اول تا روز آخر محکم نگه داریم. چگونه می‌توانیم بهترین تعادل را بین طراحی و ویژگی‌ها در طول مسیر حفظ کنیم؟   
report erratum • discuss   
فصل ۷. Build Features and Foundation in Parallel • 51   
  
[متن از صفحه قبلی: هر محصول مجموعه‌ای از ویژگی‌های کلیدی دارد که برای ارائه ارزش مورد نیاز است. ما برنامه داریم که آن‌ها را بسازیم و بقیه را کنار بگذاریم. هر چیزی که می‌سازیم باید بر یک پایه محکم استوار باشد. ما اغلب از کلماتی مانند architecture، design یا infrastructure هنگام صحبت درباره پایه استفاده می‌کنیم. طبق راهنمایی‌های این کتاب، ما قصد داریم این ویژگی‌ها را به‌صورت تدریجی بسازیم، از باارزش‌ترین تا کم‌ارزش‌ترین.]

================================================================================

# Section 58

## Original Text:

To keep the features coming  
smoothly, we need to keep the system’s design solid from  
the first day to the last. How can we best balance design and features as we go  
along. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 51]  
  
Each feature needs a solid foundation of design, a solid  
“infrastructure. ”  
Without a good design foundation, the product will be full  
of defects and hard to work with. We will slow down and  
quite possibly fail. There’s no disputing this: features do  
need to be supported by a foundation, and the foundation  
needs to be well designed. We can’t build features without a foundation. What’s the  
best way to approach building features, given that we require  
a solid foundation. Chapter 7. Build Features and Foundation in Parallel • 52  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Each feature needs a solid foundation of design, a solid  
“infrastructure. ”  
Without a good design foundation, the product will be full  
of defects and hard to work with. We will slow down and  
quite possibly fail. There’s no disputing this: features do  
need to be supported by a foundation, and the foundation  
needs to be well designed. We can’t build features without a foundation. What’s the  
best way to approach building features, given that we require  
a solid foundation. Chapter 7. Build Features and Foundation in Parallel • 52  
report erratum • discuss]  
  
Ideally we would deliver all the features, complete, by the deadline. Remember that when we envisioned all these features, we  
imagined everything we might want if we could have it all. It’s very unlikely that we’ll get everything we want. Even after prioritizing, we surely have too much to do. We  
need to do as little work as possible to deliver the best pos-  
sible product by our delivery date…and we need to do that  
work as soundly as we can afford.

## Translated Text:

برای اینکه ویژگی‌ها به‌طور روان ارائه شوند، باید طراحی سیستم را از روز اول تا روز آخر محکم نگه داریم. چگونه می‌توانیم طراحی و ویژگی‌ها را به بهترین شکل ممکن در طول مسیر متعادل کنیم؟   
گزارش خطا • بحث   
فصل ۷. Build Features and Foundation in Parallel • 51   
  
هر ویژگی به یک پایه طراحی محکم، یک "زیرساخت" قوی نیاز دارد.   
بدون یک پایه طراحی خوب، محصول پر از نقص خواهد بود و کار با آن دشوار می‌شود. ما کند خواهیم شد و احتمالاً شکست خواهیم خورد. این موضوع قابل انکار نیست: ویژگی‌ها باید توسط یک پایه پشتیبانی شوند و این پایه باید به‌خوبی طراحی شده باشد. ما نمی‌توانیم ویژگی‌ها را بدون یک پایه بسازیم. بهترین روش برای ساخت ویژگی‌ها با توجه به نیاز به یک پایه محکم چیست؟   
فصل ۷. Build Features and Foundation in Parallel • 52   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: هر ویژگی به یک پایه طراحی محکم، یک "زیرساخت" قوی نیاز دارد.   
بدون یک پایه طراحی خوب، محصول پر از نقص خواهد بود و کار با آن دشوار می‌شود. ما کند خواهیم شد و احتمالاً شکست خواهیم خورد. این موضوع قابل انکار نیست: ویژگی‌ها باید توسط یک پایه پشتیبانی شوند و این پایه باید به‌خوبی طراحی شده باشد. ما نمی‌توانیم ویژگی‌ها را بدون یک پایه بسازیم. بهترین روش برای ساخت ویژگی‌ها با توجه به نیاز به یک پایه محکم چیست؟   
فصل ۷. Build Features and Foundation in Parallel • 52   
گزارش خطا • بحث]   
  
به‌صورت ایده‌آل، ما تمام ویژگی‌ها را به‌طور کامل تا موعد مقرر تحویل می‌دهیم. به یاد داشته باشید که وقتی همه این ویژگی‌ها را تصور کردیم، هر چیزی را که ممکن بود بخواهیم، در نظر گرفتیم. بسیار بعید است که همه چیزهایی را که می‌خواهیم به دست آوریم. حتی پس از اولویت‌بندی، مطمئناً کارهای زیادی برای انجام داریم.   
ما باید تا حد ممکن کمترین کار را انجام دهیم تا بهترین محصول ممکن را تا تاریخ تحویل ارائه دهیم... و باید این کار را تا حدی که توانایی مالی داریم، به‌درستی انجام دهیم.

================================================================================

# Section 59

## Original Text:

We need to build features and foundation in a way that will  
keep us safe, keep us moving rapidly, and waste as little  
time and effort as possible. Let’s consider some options: We  
could build out the whole design first, or we could build  
each full feature one at a time, each with its foundation. Both  
of these approaches raise concerns. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 53  
  
[CONTEXT FROM PREVIOUS PAGE: Ideally we would deliver all the features, complete, by the deadline. Remember that when we envisioned all these features, we  
imagined everything we might want if we could have it all. It’s very unlikely that we’ll get everything we want. Even after prioritizing, we surely have too much to do. We  
need to do as little work as possible to deliver the best pos-  
sible product by our delivery date…and we need to do that  
work as soundly as we can afford. We need to build features and foundation in a way that will  
keep us safe, keep us moving rapidly, and waste as little  
time and effort as possible. Let’s consider some options: We  
could build out the whole design first, or we could build  
each full feature one at a time, each with its foundation. Both  
of these approaches raise concerns. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 53]  
  
Foundation first means too few features get to market. If we build out the design first, we almost always wind up  
with too few shippable features by the date. We’re not sure how fast we can go: we never are. If we build  
foundation first, we will surely build too much of it. We’ll  
consume too much time and we’ll wind up with fewer fea-  
tures than we could have had. Each feature represents value to our users, and revenue or  
other benefits to us. We need as many as we can reasonably  
get. Even if we do have time to do all the foundation and then  
build the features on top, we can guide our project only  
when we can see the actual working features created.

## Translated Text:

ما باید Features و Foundation را به گونه‌ای Build کنیم که ما را ایمن نگه دارد، به ما امکان حرکت سریع بدهد و کمترین زمان و تلاش ممکن را هدر دهد. بیایید برخی گزینه‌ها را بررسی کنیم: می‌توانیم ابتدا کل Design را Build کنیم، یا می‌توانیم هر Feature کامل را یکی یکی، همراه با Foundation آن Build کنیم. هر دو این رویکردها نگرانی‌هایی را به همراه دارند.   
[CONTEXT FROM PREVIOUS PAGE: ایده‌آل این است که تمام Features را به طور کامل تا موعد مقرر Deliver کنیم. به یاد داشته باشید که وقتی تمام این Features را تصور کردیم، همه چیزهایی را که ممکن بود بخواهیم، در نظر گرفتیم. بسیار بعید است که بتوانیم همه چیزهایی که می‌خواهیم را به دست آوریم. حتی پس از اولویت‌بندی، مطمئناً کارهای زیادی برای انجام دادن داریم. ما باید تا حد امکان کمترین کار را انجام دهیم تا بهترین محصول ممکن را تا تاریخ Delivery ارائه دهیم... و باید این کار را تا حدی که می‌توانیم با دقت انجام دهیم. ما باید Features و Foundation را به گونه‌ای Build کنیم که ما را ایمن نگه دارد، به ما امکان حرکت سریع بدهد و کمترین زمان و تلاش ممکن را هدر دهد. بیایید برخی گزینه‌ها را بررسی کنیم: می‌توانیم ابتدا کل Design را Build کنیم، یا می‌توانیم هر Feature کامل را یکی یکی، همراه با Foundation آن Build کنیم. هر دو این رویکردها نگرانی‌هایی را به همراه دارند.]  
  
Foundation اول به این معنی است که تعداد کمی از Features به بازار می‌رسند. اگر ابتدا Design را Build کنیم، تقریباً همیشه با تعداد کمی از Features قابل Ship تا تاریخ مقرر مواجه می‌شویم. ما نمی‌دانیم که چقدر سریع می‌توانیم پیش برویم: هیچ‌وقت نمی‌دانیم. اگر Foundation را اول Build کنیم، مطمئناً بیش از حد آن را Build خواهیم کرد. زمان زیادی مصرف خواهیم کرد و در نهایت تعداد کمتری Features نسبت به آنچه می‌توانستیم داشته باشیم، خواهیم داشت. هر Feature نشان‌دهنده ارزش برای کاربران ما و درآمد یا مزایای دیگر برای ما است. ما به تعداد بیشتری از آنها نیاز داریم که به طور معقولی بتوانیم به دست آوریم. حتی اگر زمان کافی برای انجام تمام Foundation و سپس Build کردن Features بر روی آن داشته باشیم، تنها زمانی می‌توانیم پروژه خود را هدایت کنیم که بتوانیم Features واقعی و عملیاتی ایجاد شده را ببینیم.

================================================================================

# Section 60

## Original Text:

Building foundation first inhibits our ability to manage our  
work. We dare not build the foundation first: it will defer and  
inevitably reduce the product’s value. Well then, what about building up each feature, each with  
its foundation, one at a time. Chapter 7. Build Features and Foundation in Parallel • 54  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Foundation first means too few features get to market. If we build out the design first, we almost always wind up  
with too few shippable features by the date. We’re not sure how fast we can go: we never are. If we build  
foundation first, we will surely build too much of it. We’ll  
consume too much time and we’ll wind up with fewer fea-  
tures than we could have had. Each feature represents value to our users, and revenue or  
other benefits to us. We need as many as we can reasonably  
get. Even if we do have time to do all the foundation and then  
build the features on top, we can guide our project only  
when we can see the actual working features created. Building foundation first inhibits our ability to manage our  
work. We dare not build the foundation first: it will defer and  
inevitably reduce the product’s value. Well then, what about building up each feature, each with  
its foundation, one at a time. Chapter 7. Build Features and Foundation in Parallel • 54  
report erratum • discuss]  
  
Building complete features one at a time still means too few features  
get to market. If we build complete features one after another, we’re likely  
to have key capabilities missing when time runs out. When we envision a major feature, we see it in its fullest  
possible glory, just as we see our product with every possible  
feature. Just as it would be wrong to build every possible feature, it  
is wrong to start by building each feature out to its fullest  
glory. The desirability of our product depends on a set of  
features that is complete enough—and glorious enough—to  
attract and satisfy our users. What’s left.

## Translated Text:

ساختن Foundation در ابتدا توانایی ما را در مدیریت کارمان محدود می‌کند. ما جرأت نمی‌کنیم Foundation را ابتدا بسازیم: این کار باعث تأخیر و به‌طور اجتناب‌ناپذیری کاهش ارزش محصول خواهد شد. خب، پس چه می‌شود اگر هر Feature را، هر کدام با Foundation خودش، یکی یکی بسازیم؟   
فصل ۷. Build Features and Foundation in Parallel • ۵۴   
report erratum • discuss   
  
[متن از صفحه قبلی: ساختن Foundation در ابتدا به این معناست که تعداد کمی از Features به بازار می‌رسند. اگر ابتدا Design را بسازیم، تقریباً همیشه با تعداد کمی از Features قابل انتشار تا تاریخ مورد نظر مواجه خواهیم شد. ما نمی‌دانیم چقدر سریع می‌توانیم پیش برویم: هیچ‌وقت نمی‌دانیم. اگر Foundation را ابتدا بسازیم، مطمئناً بیش از حد از آن خواهیم ساخت. زمان زیادی مصرف خواهیم کرد و در نهایت تعداد Features کمتری نسبت به آنچه می‌توانستیم داشته باشیم خواهیم داشت. هر Feature نمایانگر ارزش برای کاربران ما و درآمد یا مزایای دیگر برای ما است. ما به تعداد بیشتری از آن‌ها نیاز داریم که به‌طور منطقی بتوانیم به دست آوریم. حتی اگر زمان کافی برای ساختن تمام Foundation و سپس ساختن Features روی آن داشته باشیم، تنها زمانی می‌توانیم پروژه خود را هدایت کنیم که بتوانیم Features واقعی و عملیاتی ایجاد شده را ببینیم. ساختن Foundation در ابتدا توانایی ما را در مدیریت کارمان محدود می‌کند. ما جرأت نمی‌کنیم Foundation را ابتدا بسازیم: این کار باعث تأخیر و به‌طور اجتناب‌ناپذیری کاهش ارزش محصول خواهد شد. خب، پس چه می‌شود اگر هر Feature را، هر کدام با Foundation خودش، یکی یکی بسازیم؟   
فصل ۷. Build Features and Foundation in Parallel • ۵۴   
report erratum • discuss]   
  
ساختن Features کامل یکی یکی همچنان به این معناست که تعداد کمی از Features به بازار می‌رسند. اگر Features کامل را یکی پس از دیگری بسازیم، احتمالاً قابلیت‌های کلیدی زمانی که زمان تمام شود، از دست خواهند رفت. وقتی یک Feature بزرگ را تصور می‌کنیم، آن را در کامل‌ترین حالت ممکنش می‌بینیم، همان‌طور که محصولمان را با تمام Features ممکن می‌بینیم. همان‌طور که ساختن تمام Features ممکن اشتباه است، شروع با ساختن هر Feature به کامل‌ترین حالت آن نیز اشتباه است. جذابیت محصول ما به مجموعه‌ای از Features بستگی دارد که به اندازه کافی کامل—و به اندازه کافی جذاب—هستند تا کاربران ما را جذب و راضی کنند. چه چیزی باقی می‌ماند؟

================================================================================

# Section 61

## Original Text:

We can’t build all the foundation first, and we  
can’t build all the features first. Are we doomed. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 55  
  
[CONTEXT FROM PREVIOUS PAGE: Building complete features one at a time still means too few features  
get to market. If we build complete features one after another, we’re likely  
to have key capabilities missing when time runs out. When we envision a major feature, we see it in its fullest  
possible glory, just as we see our product with every possible  
feature. Just as it would be wrong to build every possible feature, it  
is wrong to start by building each feature out to its fullest  
glory. The desirability of our product depends on a set of  
features that is complete enough—and glorious enough—to  
attract and satisfy our users. What’s left. We can’t build all the foundation first, and we  
can’t build all the features first. Are we doomed. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 55]  
  
Build simple yet functional versions first. It’s safer by far to build a simple yet functional version of  
each feature first. To get the best product in the time available, we need to  
work on all the features that are important to our users, not  
just a few. For each feature, we need to fill in as much capa-  
bility as time permits, using our judgment as to what’s  
needed and how much time we have left. We can do the best possible job of this if we do small versions  
of each necessary feature, with just enough foundation to  
be solid. We build what’s called a “minimum viable product” as  
quickly as we can. Chapter 7. Build Features and Foundation in Parallel • 56  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Build simple yet functional versions first. It’s safer by far to build a simple yet functional version of  
each feature first. To get the best product in the time available, we need to  
work on all the features that are important to our users, not  
just a few.

## Translated Text:

ما نمی‌توانیم ابتدا تمام foundation را بسازیم و نمی‌توانیم ابتدا تمام features را بسازیم. آیا محکوم به شکست هستیم؟   
report erratum • discuss   
فصل ۷. Build Features and Foundation in Parallel • ۵۵   
  
[متن قبلی: ساختن features کامل به صورت یکی یکی هنوز به این معناست که تعداد کمی از features به بازار می‌رسند. اگر features کامل را یکی پس از دیگری بسازیم، احتمالاً قابلیت‌های کلیدی زمانی که وقت تمام شود، از دست خواهند رفت. وقتی یک feature بزرگ را تصور می‌کنیم، آن را در کامل‌ترین شکل ممکن می‌بینیم، همان‌طور که محصول خود را با هر feature ممکن می‌بینیم. همان‌طور که ساختن هر feature ممکن اشتباه است، شروع کردن با ساختن هر feature به کامل‌ترین شکل آن نیز اشتباه است. جذابیت محصول ما به مجموعه‌ای از features بستگی دارد که به اندازه کافی کامل و به اندازه کافی جذاب باشند تا کاربران ما را جذب و راضی کنند. چه چیزی باقی می‌ماند؟ ما نمی‌توانیم ابتدا تمام foundation را بسازیم و نمی‌توانیم ابتدا تمام features را بسازیم. آیا محکوم به شکست هستیم؟   
report erratum • discuss   
فصل ۷. Build Features and Foundation in Parallel • ۵۵]   
  
نسخه‌های ساده اما کاربردی را ابتدا بسازید. ساختن یک نسخه ساده اما کاربردی از هر feature بسیار ایمن‌تر است. برای دستیابی به بهترین محصول در زمان موجود، باید روی تمام features که برای کاربران ما مهم هستند کار کنیم، نه فقط چند مورد.   
  
برای هر feature، باید تا جایی که زمان اجازه می‌دهد، قابلیت‌ها را پر کنیم، با استفاده از قضاوت خود در مورد آنچه لازم است و زمانی که باقی مانده است. ما می‌توانیم بهترین کار ممکن را انجام دهیم اگر نسخه‌های کوچک از هر feature ضروری بسازیم، با foundation کافی برای اینکه محکم باشد.   
  
ما چیزی را می‌سازیم که به آن “minimum viable product” گفته می‌شود، به سریع‌ترین شکل ممکن.   
فصل ۷. Build Features and Foundation in Parallel • ۵۶   
report erratum • discuss   
  
[متن قبلی: نسخه‌های ساده اما کاربردی را ابتدا بسازید. ساختن یک نسخه ساده اما کاربردی از هر feature بسیار ایمن‌تر است. برای دستیابی به بهترین محصول در زمان موجود، باید روی تمام features که برای کاربران ما مهم هستند کار کنیم، نه فقط چند مورد.]

================================================================================

# Section 62

## Original Text:

For each feature, we need to fill in as much capa-  
bility as time permits, using our judgment as to what’s  
needed and how much time we have left. We can do the best possible job of this if we do small versions  
of each necessary feature, with just enough foundation to  
be solid. We build what’s called a “minimum viable product” as  
quickly as we can. Chapter 7. Build Features and Foundation in Parallel • 56  
report erratum • discuss]  
  
Refine each feature in multiple iterations. We fill in the features smoothly, always with our best judg-  
ment of what’s most important, building just enough  
infrastructure as we go. Rather than take chances with how much infrastructure to  
build, or how many features we will be able to get done by  
our finish date, we work in small versions. Each feature  
version makes the product a bit better so that we always  
have the best possible product at every moment. We can repeat this in every iteration, making priority deci-  
sions as we go, until time and money tell us that it’s time to  
stop or switch attention to a new product, as discussed in  
Chapter 2, Value Is What We Want , on page 6. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 57  
  
[CONTEXT FROM PREVIOUS PAGE: Refine each feature in multiple iterations. We fill in the features smoothly, always with our best judg-  
ment of what’s most important, building just enough  
infrastructure as we go. Rather than take chances with how much infrastructure to  
build, or how many features we will be able to get done by  
our finish date, we work in small versions. Each feature  
version makes the product a bit better so that we always  
have the best possible product at every moment. We can repeat this in every iteration, making priority deci-  
sions as we go, until time and money tell us that it’s time to  
stop or switch attention to a new product, as discussed in  
Chapter 2, Value Is What We Want , on page 6. report erratum • discussChapter 7.

## Translated Text:

برای هر Feature، باید تا جایی که زمان اجازه می‌دهد، قابلیت‌ها را تکمیل کنیم و از قضاوت خود برای تعیین نیازها و میزان زمانی که باقی مانده استفاده کنیم. ما می‌توانیم بهترین عملکرد ممکن را داشته باشیم اگر نسخه‌های کوچک‌تری از هر Feature ضروری ایجاد کنیم، با پایه‌ای که به اندازه کافی محکم باشد. ما چیزی که به آن “minimum viable product” گفته می‌شود را در سریع‌ترین زمان ممکن Build می‌کنیم.   
Chapter 7. Build Features and Foundation in Parallel • 56   
report erratum • discuss   
  
هر Feature را در چندین Iteration بهبود دهید. ما Features را به‌صورت روان تکمیل می‌کنیم، همیشه با بهترین قضاوت در مورد اینکه چه چیزی مهم‌تر است، و به اندازه کافی Infrastructure در حین پیش‌روی ایجاد می‌کنیم. به جای ریسک کردن در مورد اینکه چه میزان Infrastructure باید Build شود یا چه تعداد Feature می‌توانیم تا تاریخ پایان خود تکمیل کنیم، ما در نسخه‌های کوچک کار می‌کنیم. هر نسخه از Feature محصول را کمی بهتر می‌کند تا همیشه در هر لحظه بهترین محصول ممکن را داشته باشیم. ما می‌توانیم این کار را در هر Iteration تکرار کنیم، با تصمیم‌گیری‌های اولویت‌بندی در حین پیش‌روی، تا زمانی که زمان و منابع مالی به ما بگویند که وقت آن است که متوقف شویم یا توجه خود را به محصول جدیدی معطوف کنیم، همان‌طور که در Chapter 2، Value Is What We Want، در صفحه 6 بحث شد.   
report erratum • discuss   
Chapter 7. Build Features and Foundation in Parallel • 57   
  
[CONTEXT FROM PREVIOUS PAGE: هر Feature را در چندین Iteration بهبود دهید. ما Features را به‌صورت روان تکمیل می‌کنیم، همیشه با بهترین قضاوت در مورد اینکه چه چیزی مهم‌تر است، و به اندازه کافی Infrastructure در حین پیش‌روی ایجاد می‌کنیم. به جای ریسک کردن در مورد اینکه چه میزان Infrastructure باید Build شود یا چه تعداد Feature می‌توانیم تا تاریخ پایان خود تکمیل کنیم، ما در نسخه‌های کوچک کار می‌کنیم. هر نسخه از Feature محصول را کمی بهتر می‌کند تا همیشه در هر لحظه بهترین محصول ممکن را داشته باشیم. ما می‌توانیم این کار را در هر Iteration تکرار کنیم، با تصمیم‌گیری‌های اولویت‌بندی در حین پیش‌روی، تا زمانی که زمان و منابع مالی به ما بگویند که وقت آن است که متوقف شویم یا توجه خود را به محصول جدیدی معطوف کنیم، همان‌طور که در Chapter 2، Value Is What We Want، در صفحه 6 بحث شد.   
report erratum • discuss   
Chapter 7.]

================================================================================

# Section 63

## Original Text:

Build Features and Foundation in Parallel • 57]  
  
Steer to the best possible result by the desired shipping date. Because we have the best possible product at every moment,  
we have the best possible product at the moment we decide  
to ship it. In fact, because we’re always ready, we can ship  
early if there is any reason to do so—and there often is. Chapter 7. Build Features and Foundation in Parallel • 58  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Steer to the best possible result by the desired shipping date. Because we have the best possible product at every moment,  
we have the best possible product at the moment we decide  
to ship it. In fact, because we’re always ready, we can ship  
early if there is any reason to do so—and there often is. Chapter 7. Build Features and Foundation in Parallel • 58  
report erratum • discuss]  
  
This does take skill. Choosing the best mix of features requires great skill on the  
part of our Product Champion. This skill can be developed,  
however, and the repeated pqractice of choosing the features  
for the next iteration is the best way to develop that skill. An engaged Product Champion sees her vision for the  
product coming into being and learns to make better choices  
by practicing choosing often. The developers need skill as well. Developers are often  
trained to try to design a system up front. As we see here,  
this is never ideal, because we never know up front what  
the system will be. This is especially true when our Product  
Champion is learning what she wants, what she can have,  
and how to grow the best possible system. What are the skills that our whole team needs in order to  
work this way. What guidance must the business-side peo-  
ple provide. How can we help our technical people do their  
part. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 59  
  
[CONTEXT FROM PREVIOUS PAGE: This does take skill.

## Translated Text:

ساخت قابلیت‌ها و زیرساخت به‌صورت موازی • 57  
  
بهترین نتیجه ممکن را تا تاریخ ارسال مورد نظر هدایت کنید. چون ما در هر لحظه بهترین محصول ممکن را داریم، در لحظه‌ای که تصمیم به ارسال آن می‌گیریم، بهترین محصول ممکن را خواهیم داشت. در واقع، چون همیشه آماده هستیم، می‌توانیم زودتر ارسال کنیم اگر دلیلی برای این کار وجود داشته باشد—و اغلب وجود دارد. فصل 7. ساخت قابلیت‌ها و زیرساخت به‌صورت موازی • 58   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: بهترین نتیجه ممکن را تا تاریخ ارسال مورد نظر هدایت کنید. چون ما در هر لحظه بهترین محصول ممکن را داریم، در لحظه‌ای که تصمیم به ارسال آن می‌گیریم، بهترین محصول ممکن را خواهیم داشت. در واقع، چون همیشه آماده هستیم، می‌توانیم زودتر ارسال کنیم اگر دلیلی برای این کار وجود داشته باشد—و اغلب وجود دارد. فصل 7. ساخت قابلیت‌ها و زیرساخت به‌صورت موازی • 58   
گزارش خطا • بحث کنید]   
  
این کار مهارت می‌طلبد. انتخاب بهترین ترکیب از قابلیت‌ها نیازمند مهارت بالایی از سوی Product Champion ما است. با این حال، این مهارت قابل توسعه است و تمرین مکرر انتخاب قابلیت‌ها برای Iteration بعدی بهترین راه برای توسعه این مهارت است. یک Product Champion متعهد، چشم‌انداز خود برای محصول را در حال تحقق می‌بیند و با تمرین انتخاب مکرر، یاد می‌گیرد که تصمیمات بهتری بگیرد.   
  
توسعه‌دهندگان نیز به مهارت نیاز دارند. توسعه‌دهندگان اغلب آموزش دیده‌اند که سعی کنند یک سیستم را از ابتدا طراحی کنند. همان‌طور که در اینجا می‌بینیم، این هرگز ایده‌آل نیست، زیرا ما هرگز از ابتدا نمی‌دانیم سیستم چه خواهد بود. این موضوع به‌ویژه زمانی صادق است که Product Champion ما در حال یادگیری این است که چه می‌خواهد، چه چیزی می‌تواند داشته باشد و چگونه بهترین سیستم ممکن را رشد دهد.   
  
چه مهارت‌هایی برای کل تیم ما لازم است تا بتوانیم به این روش کار کنیم؟ چه راهنمایی‌هایی باید از سوی افراد بخش کسب‌وکار ارائه شود؟ چگونه می‌توانیم به افراد فنی خود کمک کنیم تا نقش خود را ایفا کنند؟   
گزارش خطا • بحث کنید   
فصل 7. ساخت قابلیت‌ها و زیرساخت به‌صورت موازی • 59   
  
[متن از صفحه قبلی: این کار مهارت می‌طلبد.]

================================================================================

# Section 64

## Original Text:

Choosing the best mix of features requires great skill on the  
part of our Product Champion. This skill can be developed,  
however, and the repeated pqractice of choosing the features  
for the next iteration is the best way to develop that skill. An engaged Product Champion sees her vision for the  
product coming into being and learns to make better choices  
by practicing choosing often. The developers need skill as well. Developers are often  
trained to try to design a system up front. As we see here,  
this is never ideal, because we never know up front what  
the system will be. This is especially true when our Product  
Champion is learning what she wants, what she can have,  
and how to grow the best possible system. What are the skills that our whole team needs in order to  
work this way. What guidance must the business-side peo-  
ple provide. How can we help our technical people do their  
part. report erratum • discussChapter 7. Build Features and Foundation in Parallel • 59]  
  
Build walls against bugs using good processes. CHAPTER 8  
Bug-Free and Well Designed  
Good technical practices are required to ensure that our product  
is bug-free and well designed at all times. What are they. Let’s look in a bit more detail at what developers must do  
in order to work feature by feature. Don’t worry—we’re not  
going to drop into programming—but we do want all our  
readers to be aware of what the business needs to expect of  
developers, and what it needs to support. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Build walls against bugs using good processes. CHAPTER 8  
Bug-Free and Well Designed  
Good technical practices are required to ensure that our product  
is bug-free and well designed at all times. What are they. Let’s look in a bit more detail at what developers must do  
in order to work feature by feature.

## Translated Text:

انتخاب بهترین ترکیب از ویژگی‌ها نیازمند مهارت بالایی از سوی Product Champion ما است. با این حال، این مهارت قابل توسعه است و تمرین مکرر در انتخاب ویژگی‌ها برای iteration بعدی بهترین راه برای توسعه این مهارت است. یک Product Champion متعهد، چشم‌انداز خود را برای محصول در حال شکل‌گیری می‌بیند و با تمرین مکرر انتخاب، یاد می‌گیرد تصمیمات بهتری بگیرد.   
  
توسعه‌دهندگان نیز به مهارت نیاز دارند. توسعه‌دهندگان اغلب آموزش دیده‌اند که سعی کنند یک سیستم را از ابتدا طراحی کنند. همانطور که اینجا می‌بینیم، این هرگز ایده‌آل نیست، زیرا ما هرگز نمی‌دانیم که سیستم در ابتدا چه خواهد بود. این موضوع به ویژه زمانی صادق است که Product Champion ما در حال یادگیری این است که چه می‌خواهد، چه چیزی می‌تواند داشته باشد و چگونه بهترین سیستم ممکن را رشد دهد.   
  
تیم ما برای کار به این روش به چه مهارت‌هایی نیاز دارد؟ چه راهنمایی‌هایی باید از سوی افراد بخش کسب‌وکار ارائه شود؟ چگونه می‌توانیم به افراد فنی خود کمک کنیم تا نقش خود را ایفا کنند؟   
report erratum • discuss   
Chapter 7. Build Features and Foundation in Parallel • 59   
  
دیوارهایی در برابر bugs با استفاده از فرآیندهای خوب بسازید.   
CHAPTER 8   
Bug-Free and Well Designed   
  
برای اطمینان از اینکه محصول ما همیشه bug-free و خوب طراحی شده باشد، به practices فنی خوب نیاز است. این practices چه هستند؟ بیایید کمی دقیق‌تر بررسی کنیم که توسعه‌دهندگان برای کار کردن به صورت feature by feature چه کاری باید انجام دهند. نگران نباشید—ما قصد نداریم وارد برنامه‌نویسی شویم—اما می‌خواهیم همه خوانندگان ما آگاه باشند که کسب‌وکار باید چه انتظاراتی از توسعه‌دهندگان داشته باشد و چه چیزی را باید پشتیبانی کند.   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: دیوارهایی در برابر bugs با استفاده از فرآیندهای خوب بسازید.   
CHAPTER 8   
Bug-Free and Well Designed   
  
برای اطمینان از اینکه محصول ما همیشه bug-free و خوب طراحی شده باشد، به practices فنی خوب نیاز است. این practices چه هستند؟ بیایید کمی دقیق‌تر بررسی کنیم که توسعه‌دهندگان برای کار کردن به صورت feature by feature چه کاری باید انجام دهند.]

================================================================================

# Section 65

## Original Text:

Don’t worry—we’re not  
going to drop into programming—but we do want all our  
readers to be aware of what the business needs to expect of  
developers, and what it needs to support. report erratum • discuss]  
  
Our product is made up of a growing set of correctly working  
features, built on a growing, evolving foundation of design. We’re asking our team to build feature by feature, keeping  
everything checked and working, in good order, improving  
the design as they go. This is our best chance to steer the  
project to success. But there are issues. report erratum • discussChapter 8. Bug-Free and Well Designed • 61  
  
[CONTEXT FROM PREVIOUS PAGE: Our product is made up of a growing set of correctly working  
features, built on a growing, evolving foundation of design. We’re asking our team to build feature by feature, keeping  
everything checked and working, in good order, improving  
the design as they go. This is our best chance to steer the  
project to success. But there are issues. report erratum • discussChapter 8. Bug-Free and Well Designed • 61]  
  
Defects amount to negative features. Progress is uncertain. Elim-  
inate defects to provide clarity on what’s done. We’re trying to plan by features, grow by features, and  
manage by features. This means that when we’re told a fea-  
ture is done, it really needs to work. Any defects in our fea-  
tures amount to negative features. They reduce our line of  
apparent progress. If the defects aren’t discovered yet, it’s  
even worse: we don’t even know how bad things are. We cannot work effectively in a world of defects. Chapter 8. Bug-Free and Well Designed • 62  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Defects amount to negative features. Progress is uncertain. Elim-  
inate defects to provide clarity on what’s done. We’re trying to plan by features, grow by features, and  
manage by features. This means that when we’re told a fea-  
ture is done, it really needs to work.

## Translated Text:

نگران نباشید—ما قصد نداریم وارد برنامه‌نویسی شویم—اما می‌خواهیم همه خوانندگان ما آگاه باشند که کسب‌وکار باید چه انتظاراتی از توسعه‌دهندگان داشته باشد و چه چیزی را باید پشتیبانی کند.   
report erratum • discuss   
  
محصول ما از مجموعه‌ای در حال رشد از قابلیت‌هایی که به درستی کار می‌کنند تشکیل شده است، که بر پایه‌ای در حال رشد و تکامل از طراحی ساخته شده‌اند. ما از تیم خود می‌خواهیم که قابلیت‌ها را یکی‌یکی بسازند، همه چیز را بررسی و درست نگه دارند، و طراحی را در حین پیشرفت بهبود دهند. این بهترین شانس ما برای هدایت پروژه به سمت موفقیت است. اما مشکلاتی وجود دارد.   
report erratum • discuss   
Chapter 8. Bug-Free and Well Designed • 61   
  
[CONTEXT FROM PREVIOUS PAGE: محصول ما از مجموعه‌ای در حال رشد از قابلیت‌هایی که به درستی کار می‌کنند تشکیل شده است، که بر پایه‌ای در حال رشد و تکامل از طراحی ساخته شده‌اند. ما از تیم خود می‌خواهیم که قابلیت‌ها را یکی‌یکی بسازند، همه چیز را بررسی و درست نگه دارند، و طراحی را در حین پیشرفت بهبود دهند. این بهترین شانس ما برای هدایت پروژه به سمت موفقیت است. اما مشکلاتی وجود دارد.   
report erratum • discuss   
Chapter 8. Bug-Free and Well Designed • 61]   
  
نقص‌ها به معنای قابلیت‌های منفی هستند. پیشرفت نامشخص است. نقص‌ها را حذف کنید تا شفافیت در مورد آنچه انجام شده است، ایجاد شود. ما سعی داریم بر اساس قابلیت‌ها برنامه‌ریزی کنیم، بر اساس قابلیت‌ها رشد کنیم، و بر اساس قابلیت‌ها مدیریت کنیم. این به این معناست که وقتی به ما گفته می‌شود یک قابلیت تکمیل شده است، واقعاً باید کار کند. هر نقصی در قابلیت‌های ما به معنای قابلیت‌های منفی است. آن‌ها خط پیشرفت ظاهری ما را کاهش می‌دهند. اگر نقص‌ها هنوز کشف نشده باشند، حتی بدتر است: ما حتی نمی‌دانیم اوضاع چقدر بد است. ما نمی‌توانیم در دنیایی پر از نقص‌ها به طور مؤثر کار کنیم.   
Chapter 8. Bug-Free and Well Designed • 62   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: نقص‌ها به معنای قابلیت‌های منفی هستند. پیشرفت نامشخص است. نقص‌ها را حذف کنید تا شفافیت در مورد آنچه انجام شده است، ایجاد شود. ما سعی داریم بر اساس قابلیت‌ها برنامه‌ریزی کنیم، بر اساس قابلیت‌ها رشد کنیم، و بر اساس قابلیت‌ها مدیریت کنیم. این به این معناست که وقتی به ما گفته می‌شود یک قابلیت تکمیل شده است، واقعاً باید کار کند.]

================================================================================

# Section 66

## Original Text:

Any defects in our fea-  
tures amount to negative features. They reduce our line of  
apparent progress. If the defects aren’t discovered yet, it’s  
even worse: we don’t even know how bad things are. We cannot work effectively in a world of defects. Chapter 8. Bug-Free and Well Designed • 62  
report erratum • discuss]  
  
Defect repair adds unknown delay. Repair as you go to provide  
clarity on what’s done. As consultant Tom DeMarco tells us, no one sees a roach on  
the floor of a restaurant and says “There goes the roach. ”  
It’s the same with bugs—that is, with defects. Where there’s  
one, there are likely many. Finding defects takes time. Fixing them takes time. If we  
leave this until the end of the project, our plans for new  
features will be disrupted, and we will still have to decide  
which defects to leave in the system. report erratum • discussChapter 8. Bug-Free and Well Designed • 63  
  
[CONTEXT FROM PREVIOUS PAGE: Defect repair adds unknown delay. Repair as you go to provide  
clarity on what’s done. As consultant Tom DeMarco tells us, no one sees a roach on  
the floor of a restaurant and says “There goes the roach. ”  
It’s the same with bugs—that is, with defects. Where there’s  
one, there are likely many. Finding defects takes time. Fixing them takes time. If we  
leave this until the end of the project, our plans for new  
features will be disrupted, and we will still have to decide  
which defects to leave in the system. report erratum • discussChapter 8. Bug-Free and Well Designed • 63]  
  
If we don’t know in time what’s done, and how well it’s done, we’ll  
have no choice but to ship late with visible defects. That’s bad  
business. “Bug triage. ” No, thanks. We need to prevent as many  
defects as we can, because they add unknown delay right  
at the end of our project. We look like fools for shipping late,  
and we look like fools for shipping buggy software. Let’s  
not go there. Chapter 8.

## Translated Text:

هرگونه نقص در ویژگی‌های ما به معنای ویژگی‌های منفی است. این نقص‌ها خط پیشرفت ظاهری ما را کاهش می‌دهند. اگر این نقص‌ها هنوز کشف نشده باشند، حتی بدتر است: ما حتی نمی‌دانیم اوضاع چقدر بد است. ما نمی‌توانیم در دنیایی پر از نقص‌ها به‌طور مؤثر کار کنیم. فصل ۸. بدون Bug و طراحی‌شده به‌خوبی • ۶۲   
گزارش خطا • بحث کنید]   
  
رفع نقص باعث تأخیر ناشناخته می‌شود. نقص‌ها را در حین کار رفع کنید تا شفافیت بیشتری در مورد آنچه انجام شده است به دست آورید. همان‌طور که مشاور Tom DeMarco به ما می‌گوید، هیچ‌کس یک سوسک روی کف رستوران را نمی‌بیند و نمی‌گوید «آن سوسک است که می‌رود.»   
این موضوع در مورد Bugها—یعنی نقص‌ها—نیز صدق می‌کند. جایی که یک نقص وجود دارد، احتمالاً نقص‌های بیشتری هم وجود دارند. پیدا کردن نقص‌ها زمان می‌برد. رفع آن‌ها زمان می‌برد. اگر این کار را به انتهای پروژه موکول کنیم، برنامه‌های ما برای ویژگی‌های جدید مختل خواهد شد و همچنان باید تصمیم بگیریم که کدام نقص‌ها را در سیستم باقی بگذاریم. گزارش خطا • بحث کنید فصل ۸. بدون Bug و طراحی‌شده به‌خوبی • ۶۳   
  
[متن از صفحه قبلی: رفع نقص باعث تأخیر ناشناخته می‌شود. نقص‌ها را در حین کار رفع کنید تا شفافیت بیشتری در مورد آنچه انجام شده است به دست آورید. همان‌طور که مشاور Tom DeMarco به ما می‌گوید، هیچ‌کس یک سوسک روی کف رستوران را نمی‌بیند و نمی‌گوید «آن سوسک است که می‌رود.»   
این موضوع در مورد Bugها—یعنی نقص‌ها—نیز صدق می‌کند. جایی که یک نقص وجود دارد، احتمالاً نقص‌های بیشتری هم وجود دارند. پیدا کردن نقص‌ها زمان می‌برد. رفع آن‌ها زمان می‌برد. اگر این کار را به انتهای پروژه موکول کنیم، برنامه‌های ما برای ویژگی‌های جدید مختل خواهد شد و همچنان باید تصمیم بگیریم که کدام نقص‌ها را در سیستم باقی بگذاریم. گزارش خطا • بحث کنید فصل ۸. بدون Bug و طراحی‌شده به‌خوبی • ۶۳]   
  
اگر به‌موقع ندانیم چه چیزی انجام شده و چقدر خوب انجام شده است، چاره‌ای نخواهیم داشت جز اینکه با تأخیر و همراه با نقص‌های قابل مشاهده محصول را ارائه دهیم. این برای کسب‌وکار بد است. «Bug triage.» نه، ممنون. ما باید تا حد امکان از بروز نقص‌ها جلوگیری کنیم، زیرا آن‌ها باعث تأخیر ناشناخته درست در انتهای پروژه می‌شوند. ما با ارائه دیرهنگام احمق به نظر می‌رسیم، و با ارائه نرم‌افزار پر از Bug نیز احمق به نظر می‌رسیم. بیایید به آنجا نرویم. فصل ۸.

================================================================================

# Section 67

## Original Text:

Bug-Free and Well Designed • 64  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: If we don’t know in time what’s done, and how well it’s done, we’ll  
have no choice but to ship late with visible defects. That’s bad  
business. “Bug triage. ” No, thanks. We need to prevent as many  
defects as we can, because they add unknown delay right  
at the end of our project. We look like fools for shipping late,  
and we look like fools for shipping buggy software. Let’s  
not go there. Chapter 8. Bug-Free and Well Designed • 64  
report erratum • discuss]  
  
Because features are being added and enhanced, and because the  
design is evolving, we will make mistakes. We need continuous  
comprehensive testing. There’s really no way out. At the end of every iteration, we  
need to have the software as close to defect-free as possible. The only way to get there is to test it. Because we’re growing the system all the time, we need to  
test it more and more, better and better. We need to test the  
new features, but we need to check the old ones as well, to  
make sure they haven’t been broken. We test at two levels, with “Business” tests and “Program-  
mer” tests. report erratum • discussChapter 8. Bug-Free and Well Designed • 65  
  
[CONTEXT FROM PREVIOUS PAGE: Because features are being added and enhanced, and because the  
design is evolving, we will make mistakes. We need continuous  
comprehensive testing. There’s really no way out. At the end of every iteration, we  
need to have the software as close to defect-free as possible. The only way to get there is to test it. Because we’re growing the system all the time, we need to  
test it more and more, better and better. We need to test the  
new features, but we need to check the old ones as well, to  
make sure they haven’t been broken. We test at two levels, with “Business” tests and “Program-  
mer” tests. report erratum • discussChapter 8.

## Translated Text:

بدون باگ و خوب طراحی‌شده • ۶۴   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: اگر به‌موقع ندانیم چه چیزی انجام شده و چقدر خوب انجام شده، چاره‌ای جز ارسال دیرهنگام با نقص‌های آشکار نداریم. این برای کسب‌وکار بد است. «اولویت‌بندی باگ‌ها.» نه، ممنون. ما باید تا حد ممکن از بروز نقص‌ها جلوگیری کنیم، زیرا آن‌ها در انتهای پروژه ما تأخیرهای ناشناخته‌ای ایجاد می‌کنند. ما با ارسال دیرهنگام احمق به نظر می‌رسیم و با ارسال نرم‌افزار پر از باگ هم احمق به نظر می‌رسیم. بیایید به آنجا نرویم. فصل ۸. بدون باگ و خوب طراحی‌شده • ۶۴   
گزارش خطا • بحث کنید]   
  
از آنجا که قابلیت‌ها در حال اضافه شدن و بهبود هستند و طراحی در حال تکامل است، ما اشتباه خواهیم کرد. ما به تست جامع و مداوم نیاز داریم. واقعاً هیچ راه دیگری وجود ندارد. در پایان هر iteration، باید نرم‌افزار را تا حد ممکن بدون نقص کنیم. تنها راه رسیدن به این هدف، Test کردن آن است. از آنجا که ما دائماً در حال گسترش سیستم هستیم، باید بیشتر و بهتر Test کنیم. ما باید قابلیت‌های جدید را Test کنیم، اما همچنین باید قابلیت‌های قدیمی را نیز بررسی کنیم تا مطمئن شویم که خراب نشده‌اند. ما در دو سطح Test می‌کنیم، با تست‌های «Business» و تست‌های «Programmer».   
گزارش خطا • بحث کنید   
فصل ۸. بدون باگ و خوب طراحی‌شده • ۶۵   
  
[متن از صفحه قبلی: از آنجا که قابلیت‌ها در حال اضافه شدن و بهبود هستند و طراحی در حال تکامل است، ما اشتباه خواهیم کرد. ما به تست جامع و مداوم نیاز داریم. واقعاً هیچ راه دیگری وجود ندارد. در پایان هر iteration، باید نرم‌افزار را تا حد ممکن بدون نقص کنیم. تنها راه رسیدن به این هدف، Test کردن آن است. از آنجا که ما دائماً در حال گسترش سیستم هستیم، باید بیشتر و بهتر Test کنیم. ما باید قابلیت‌های جدید را Test کنیم، اما همچنین باید قابلیت‌های قدیمی را نیز بررسی کنیم تا مطمئن شویم که خراب نشده‌اند. ما در دو سطح Test می‌کنیم، با تست‌های «Business» و تست‌های «Programmer».   
گزارش خطا • بحث کنید   
فصل ۸.]

================================================================================

# Section 68

## Original Text:

Bug-Free and Well Designed • 65]  
  
At the end of every iteration, we need business-level tests to verify  
that we have received what we asked for. We’re getting new features every two weeks. We need to  
know whether the new ones work, and we need to be sure  
that old ones haven’t broken. To accomplish this, we must  
build business-level tests and checks for every aspect of  
every feature that we possibly can. If we don’t check something, we don’t know whether it  
works. We have to check everything. The testing burden  
will grow with each new feature. We have to keep up with  
this burden. The best known way to keep up is to express our features  
in terms of the tests they must pass, and to automate the  
tests to give us assurance that the feature works now, and  
from now on. This is often called acceptance test-driven devel-  
opment. Chapter 8. Bug-Free and Well Designed • 66  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: At the end of every iteration, we need business-level tests to verify  
that we have received what we asked for. We’re getting new features every two weeks. We need to  
know whether the new ones work, and we need to be sure  
that old ones haven’t broken. To accomplish this, we must  
build business-level tests and checks for every aspect of  
every feature that we possibly can. If we don’t check something, we don’t know whether it  
works. We have to check everything. The testing burden  
will grow with each new feature. We have to keep up with  
this burden. The best known way to keep up is to express our features  
in terms of the tests they must pass, and to automate the  
tests to give us assurance that the feature works now, and  
from now on. This is often called acceptance test-driven devel-  
opment. Chapter 8. Bug-Free and Well Designed • 66  
report erratum • discuss]  
  
Developers make changes every day. To ensure they don’t waste  
time, developer tests need to be checked much more frequently.

## Translated Text:

بدون باگ و طراحی‌شده به‌خوبی • 65]  
  
در پایان هر iteration، ما به تست‌های سطح کسب‌وکار نیاز داریم تا تأیید کنیم که آنچه درخواست کرده‌ایم را دریافت کرده‌ایم. هر دو هفته یک‌بار ویژگی‌های جدیدی دریافت می‌کنیم. ما باید بدانیم که آیا ویژگی‌های جدید کار می‌کنند و مطمئن شویم که ویژگی‌های قدیمی خراب نشده‌اند. برای انجام این کار، باید تست‌ها و بررسی‌های سطح کسب‌وکار را برای هر جنبه از هر ویژگی که ممکن است، ایجاد کنیم. اگر چیزی را بررسی نکنیم، نمی‌دانیم که آیا کار می‌کند یا نه. ما باید همه چیز را بررسی کنیم. بار تست با هر ویژگی جدید افزایش خواهد یافت. ما باید با این بار همراه شویم. بهترین راه شناخته‌شده برای همراه شدن این است که ویژگی‌های خود را به‌صورت تست‌هایی که باید پاس کنند، بیان کنیم و تست‌ها را اتوماتیک کنیم تا به ما اطمینان دهد که ویژگی اکنون کار می‌کند و از این به بعد نیز کار خواهد کرد. این اغلب به‌عنوان acceptance test-driven development شناخته می‌شود. فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 66   
report erratum • discuss   
  
[متن از صفحه قبلی: در پایان هر iteration، ما به تست‌های سطح کسب‌وکار نیاز داریم تا تأیید کنیم که آنچه درخواست کرده‌ایم را دریافت کرده‌ایم. هر دو هفته یک‌بار ویژگی‌های جدیدی دریافت می‌کنیم. ما باید بدانیم که آیا ویژگی‌های جدید کار می‌کنند و مطمئن شویم که ویژگی‌های قدیمی خراب نشده‌اند. برای انجام این کار، باید تست‌ها و بررسی‌های سطح کسب‌وکار را برای هر جنبه از هر ویژگی که ممکن است، ایجاد کنیم. اگر چیزی را بررسی نکنیم، نمی‌دانیم که آیا کار می‌کند یا نه. ما باید همه چیز را بررسی کنیم. بار تست با هر ویژگی جدید افزایش خواهد یافت. ما باید با این بار همراه شویم. بهترین راه شناخته‌شده برای همراه شدن این است که ویژگی‌های خود را به‌صورت تست‌هایی که باید پاس کنند، بیان کنیم و تست‌ها را اتوماتیک کنیم تا به ما اطمینان دهد که ویژگی اکنون کار می‌کند و از این به بعد نیز کار خواهد کرد. این اغلب به‌عنوان acceptance test-driven development شناخته می‌شود. فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 66   
report erratum • discuss]   
  
توسعه‌دهندگان هر روز تغییراتی ایجاد می‌کنند. برای اطمینان از اینکه وقتشان تلف نمی‌شود، تست‌های توسعه‌دهنده باید بسیار بیشتر و مکرر بررسی شوند.

================================================================================

# Section 69

## Original Text:

The creation of a single feature requires hundreds, even  
thousands of lines of code. Any one of those lines could be  
wrong, and if it is, something, somewhere, will break. We could rely on our business-level tests to catch these  
problems, but they usually take a long time to run, and they  
don’t point to the specific programming error when they do  
fail. Developers need to build a comprehensive network of  
automated tests as well, to be sure that problems are found  
sooner and fixed more readily. The best way we know to do this is to write the developer  
tests first, then make them run. This is called test-driven  
development , orTDD. TDD’s greatest strength is in supporting  
design improvement, which we’ll discuss a bit later. report erratum • discussChapter 8. Bug-Free and Well Designed • 67  
  
[CONTEXT FROM PREVIOUS PAGE: Developers make changes every day. To ensure they don’t waste  
time, developer tests need to be checked much more frequently. The creation of a single feature requires hundreds, even  
thousands of lines of code. Any one of those lines could be  
wrong, and if it is, something, somewhere, will break. We could rely on our business-level tests to catch these  
problems, but they usually take a long time to run, and they  
don’t point to the specific programming error when they do  
fail. Developers need to build a comprehensive network of  
automated tests as well, to be sure that problems are found  
sooner and fixed more readily. The best way we know to do this is to write the developer  
tests first, then make them run. This is called test-driven  
development , orTDD. TDD’s greatest strength is in supporting  
design improvement, which we’ll discuss a bit later. report erratum • discussChapter 8. Bug-Free and Well Designed • 67]  
  
Almost paradoxically, all this testing makes our team go faster. The reason: We make fewer mistakes, and they’re found more  
quickly. It takes longer to find a problem, and to fix it, than it does  
to prevent the problem in the first place.

## Translated Text:

ایجاد یک Feature نیازمند صدها، حتی هزاران خط کد است. هر کدام از این خطوط ممکن است اشتباه باشد و اگر چنین باشد، چیزی در جایی دچار مشکل خواهد شد. ما می‌توانیم به Business-Level Tests خود برای شناسایی این مشکلات تکیه کنیم، اما این تست‌ها معمولاً زمان زیادی برای اجرا نیاز دارند و در صورت شکست، به خطای برنامه‌نویسی خاصی اشاره نمی‌کنند. توسعه‌دهندگان نیاز دارند تا یک شبکه جامع از Automated Tests نیز بسازند تا مطمئن شوند مشکلات زودتر شناسایی و راحت‌تر برطرف می‌شوند. بهترین راهی که ما می‌شناسیم این است که ابتدا Developer Tests را بنویسیم و سپس آن‌ها را Run کنیم. به این روش Test-Driven Development یا TDD گفته می‌شود. بزرگ‌ترین نقطه قوت TDD در پشتیبانی از بهبود Design است که کمی بعد در مورد آن صحبت خواهیم کرد.   
report erratum • discuss   
Chapter 8. Bug-Free and Well Designed • 67   
  
[متن از صفحه قبلی: توسعه‌دهندگان هر روز تغییراتی ایجاد می‌کنند. برای اطمینان از این که زمان تلف نمی‌شود، Developer Tests باید بسیار بیشتر و مکرر بررسی شوند. ایجاد یک Feature نیازمند صدها، حتی هزاران خط کد است. هر کدام از این خطوط ممکن است اشتباه باشد و اگر چنین باشد، چیزی در جایی دچار مشکل خواهد شد. ما می‌توانیم به Business-Level Tests خود برای شناسایی این مشکلات تکیه کنیم، اما این تست‌ها معمولاً زمان زیادی برای اجرا نیاز دارند و در صورت شکست، به خطای برنامه‌نویسی خاصی اشاره نمی‌کنند. توسعه‌دهندگان نیاز دارند تا یک شبکه جامع از Automated Tests نیز بسازند تا مطمئن شوند مشکلات زودتر شناسایی و راحت‌تر برطرف می‌شوند. بهترین راهی که ما می‌شناسیم این است که ابتدا Developer Tests را بنویسیم و سپس آن‌ها را Run کنیم. به این روش Test-Driven Development یا TDD گفته می‌شود. بزرگ‌ترین نقطه قوت TDD در پشتیبانی از بهبود Design است که کمی بعد در مورد آن صحبت خواهیم کرد.   
report erratum • discuss   
Chapter 8. Bug-Free and Well Designed • 67]   
  
تقریباً به شکلی متناقض، تمام این تست‌ها باعث می‌شوند تیم ما سریع‌تر پیش برود. دلیلش این است که ما اشتباهات کمتری مرتکب می‌شویم و آن‌ها سریع‌تر شناسایی می‌شوند. پیدا کردن یک مشکل و رفع آن زمان بیشتری می‌برد تا جلوگیری از وقوع آن مشکل در وهله اول.

================================================================================

# Section 70

## Original Text:

When we work  
with business-level tests, provided with the feature requests,  
we don’t ship the feature until the test runs. When we work  
with developer-level tests, written before or in conjunction  
with coding, we don’t ship any part of the software until all  
the tests run. Testing as part of our ongoing work prevents defects from  
getting into the program. This lets us go much faster than  
we would if we coded and only tested when we thought the  
code was done. Chapter 8. Bug-Free and Well Designed • 68  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Almost paradoxically, all this testing makes our team go faster. The reason: We make fewer mistakes, and they’re found more  
quickly. It takes longer to find a problem, and to fix it, than it does  
to prevent the problem in the first place. When we work  
with business-level tests, provided with the feature requests,  
we don’t ship the feature until the test runs. When we work  
with developer-level tests, written before or in conjunction  
with coding, we don’t ship any part of the software until all  
the tests run. Testing as part of our ongoing work prevents defects from  
getting into the program. This lets us go much faster than  
we would if we coded and only tested when we thought the  
code was done. Chapter 8. Bug-Free and Well Designed • 68  
report erratum • discuss]  
  
At the beginning, there are only a few small features. The design  
can be simple. Remember, we’re working incrementally. We’re shipping  
real working features every two weeks. We need a good design  
early on, but we only need a small good design. report erratum • discussChapter 8. Bug-Free and Well Designed • 69  
  
[CONTEXT FROM PREVIOUS PAGE: At the beginning, there are only a few small features. The design  
can be simple. Remember, we’re working incrementally. We’re shipping  
real working features every two weeks. We need a good design  
early on, but we only need a small good design. report erratum • discussChapter 8.

## Translated Text:

وقتی با تست‌های سطح کسب‌وکار که همراه با درخواست‌های Feature ارائه می‌شوند کار می‌کنیم، Feature را تا زمانی که Test اجرا نشود، Ship نمی‌کنیم. وقتی با تست‌های سطح توسعه‌دهنده که قبل یا همزمان با کدنویسی نوشته می‌شوند کار می‌کنیم، هیچ بخشی از نرم‌افزار را تا زمانی که تمام تست‌ها اجرا نشوند، Ship نمی‌کنیم. تست کردن به عنوان بخشی از کار مداوم ما، از ورود نقص‌ها به برنامه جلوگیری می‌کند. این کار به ما اجازه می‌دهد خیلی سریع‌تر از زمانی که فقط کدنویسی می‌کردیم و تنها زمانی که فکر می‌کردیم کد تمام شده است، Test می‌کردیم، پیش برویم.   
فصل ۸. بدون باگ و طراحی‌شده به‌خوبی • ۶۸   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: تقریباً به‌طور متناقض، تمام این تست‌ها باعث می‌شوند تیم ما سریع‌تر پیش برود. دلیلش این است که ما اشتباهات کمتری می‌کنیم و آن‌ها سریع‌تر پیدا می‌شوند. پیدا کردن یک مشکل و رفع آن بیشتر طول می‌کشد تا جلوگیری از بروز مشکل در وهله اول. وقتی با تست‌های سطح کسب‌وکار که همراه با درخواست‌های Feature ارائه می‌شوند کار می‌کنیم، Feature را تا زمانی که Test اجرا نشود، Ship نمی‌کنیم. وقتی با تست‌های سطح توسعه‌دهنده که قبل یا همزمان با کدنویسی نوشته می‌شوند کار می‌کنیم، هیچ بخشی از نرم‌افزار را تا زمانی که تمام تست‌ها اجرا نشوند، Ship نمی‌کنیم. تست کردن به عنوان بخشی از کار مداوم ما، از ورود نقص‌ها به برنامه جلوگیری می‌کند. این کار به ما اجازه می‌دهد خیلی سریع‌تر از زمانی که فقط کدنویسی می‌کردیم و تنها زمانی که فکر می‌کردیم کد تمام شده است، Test می‌کردیم، پیش برویم.   
فصل ۸. بدون باگ و طراحی‌شده به‌خوبی • ۶۸   
گزارش خطا • بحث کنید]   
  
در ابتدا، فقط چند Feature کوچک وجود دارد. طراحی می‌تواند ساده باشد. به یاد داشته باشید که ما به‌صورت تدریجی کار می‌کنیم. ما هر دو هفته یک بار Featureهای واقعی و کارآمد را Ship می‌کنیم. ما در ابتدا به یک طراحی خوب نیاز داریم، اما فقط به یک طراحی کوچک و خوب نیاز داریم.   
گزارش خطا • بحث کنید   
فصل ۸. بدون باگ و طراحی‌شده به‌خوبی • ۶۹   
  
[متن از صفحه قبلی: در ابتدا، فقط چند Feature کوچک وجود دارد. طراحی می‌تواند ساده باشد. به یاد داشته باشید که ما به‌صورت تدریجی کار می‌کنیم. ما هر دو هفته یک بار Featureهای واقعی و کارآمد را Ship می‌کنیم. ما در ابتدا به یک طراحی خوب نیاز داریم، اما فقط به یک طراحی کوچک و خوب نیاز داریم.   
گزارش خطا • بحث کنید   
فصل ۸.]

================================================================================

# Section 71

## Original Text:

Bug-Free and Well Designed • 69]  
  
The design can easily deteriorate. As we add software, even to a good design, things tend to  
get worse. We hammer this here, bend that there. Good  
designs go bad one decision at a time. We can prevent this,  
but we must work skillfully and fix it as we go. Inferior design will slow us down. Skill and care are required  
to keep our project alive. Chapter 8. Bug-Free and Well Designed • 70  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: The design can easily deteriorate. As we add software, even to a good design, things tend to  
get worse. We hammer this here, bend that there. Good  
designs go bad one decision at a time. We can prevent this,  
but we must work skillfully and fix it as we go. Inferior design will slow us down. Skill and care are required  
to keep our project alive. Chapter 8. Bug-Free and Well Designed • 70  
report erratum • discuss]  
  
As the system grows feature by feature, the design needs to grow. We need to have a good design at all times. A bad design  
slows us down, because it is hard to change. As the project  
grows, as the features grow, we must grow the design to  
support those features. We need a high-quality design at  
every moment. report erratum • discussChapter 8. Bug-Free and Well Designed • 71  
  
[CONTEXT FROM PREVIOUS PAGE: As the system grows feature by feature, the design needs to grow. We need to have a good design at all times. A bad design  
slows us down, because it is hard to change. As the project  
grows, as the features grow, we must grow the design to  
support those features. We need a high-quality design at  
every moment. report erratum • discussChapter 8. Bug-Free and Well Designed • 71]  
  
At each stage of feature building, the team must do enough design  
improvement to keep up. Every change tends to break our current design. To keep the  
design good, we need to improve it as we go. Perhaps we  
make a nice clean space for our new feature.

## Translated Text:

بدون باگ و طراحی خوب • 69]  
  
طراحی به‌راحتی می‌تواند خراب شود. وقتی نرم‌افزار را اضافه می‌کنیم، حتی به یک طراحی خوب، شرایط معمولاً بدتر می‌شود. اینجا را ضربه می‌زنیم، آنجا را خم می‌کنیم. طراحی‌های خوب یک تصمیم در یک زمان خراب می‌شوند. می‌توانیم از این جلوگیری کنیم، اما باید با مهارت کار کنیم و در طول مسیر آن را اصلاح کنیم. طراحی ضعیف ما را کند می‌کند. مهارت و دقت لازم است تا پروژه‌مان زنده بماند. فصل 8. بدون باگ و طراحی خوب • 70   
report erratum • discuss   
  
[متن از صفحه قبلی: طراحی به‌راحتی می‌تواند خراب شود. وقتی نرم‌افزار را اضافه می‌کنیم، حتی به یک طراحی خوب، شرایط معمولاً بدتر می‌شود. اینجا را ضربه می‌زنیم، آنجا را خم می‌کنیم. طراحی‌های خوب یک تصمیم در یک زمان خراب می‌شوند. می‌توانیم از این جلوگیری کنیم، اما باید با مهارت کار کنیم و در طول مسیر آن را اصلاح کنیم. طراحی ضعیف ما را کند می‌کند. مهارت و دقت لازم است تا پروژه‌مان زنده بماند. فصل 8. بدون باگ و طراحی خوب • 70   
report erratum • discuss]   
  
همزمان با رشد سیستم، ویژگی به ویژگی، طراحی نیز باید رشد کند. باید همیشه طراحی خوبی داشته باشیم. یک طراحی ضعیف ما را کند می‌کند، زیرا تغییر آن دشوار است. با رشد پروژه، با رشد ویژگی‌ها، باید طراحی را برای پشتیبانی از آن ویژگی‌ها توسعه دهیم. در هر لحظه به طراحی باکیفیت نیاز داریم. report erratum • discuss   
فصل 8. بدون باگ و طراحی خوب • 71   
  
[متن از صفحه قبلی: همزمان با رشد سیستم، ویژگی به ویژگی، طراحی نیز باید رشد کند. باید همیشه طراحی خوبی داشته باشیم. یک طراحی ضعیف ما را کند می‌کند، زیرا تغییر آن دشوار است. با رشد پروژه، با رشد ویژگی‌ها، باید طراحی را برای پشتیبانی از آن ویژگی‌ها توسعه دهیم. در هر لحظه به طراحی باکیفیت نیاز داریم. report erratum • discuss   
فصل 8. بدون باگ و طراحی خوب • 71]   
  
در هر مرحله از ساخت ویژگی‌ها، تیم باید به اندازه کافی طراحی را بهبود دهد تا هماهنگ بماند. هر تغییر تمایل دارد طراحی فعلی ما را بشکند. برای حفظ طراحی خوب، باید آن را در طول مسیر بهبود دهیم. شاید یک فضای تمیز و مناسب برای ویژگی جدیدمان ایجاد کنیم.

================================================================================

# Section 72

## Original Text:

Perhaps we  
push the feature in and then rearrange things to get a good  
design. Either way works. What doesn’t work is to let the  
design go. Keeping the design good as it changes is called refactoring. Refactoring is a necessary skill for this way of developing  
software. We’ll talk more about refactoring elsewhere. For  
now, let’s focus on what it’s good for. Chapter 8. Bug-Free and Well Designed • 72  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: At each stage of feature building, the team must do enough design  
improvement to keep up. Every change tends to break our current design. To keep the  
design good, we need to improve it as we go. Perhaps we  
make a nice clean space for our new feature. Perhaps we  
push the feature in and then rearrange things to get a good  
design. Either way works. What doesn’t work is to let the  
design go. Keeping the design good as it changes is called refactoring. Refactoring is a necessary skill for this way of developing  
software. We’ll talk more about refactoring elsewhere. For  
now, let’s focus on what it’s good for. Chapter 8. Bug-Free and Well Designed • 72  
report erratum • discuss]  
  
Failure to keep the design good enough will slow progress, or even  
stop it. If our design is allowed to deteriorate, progress will slow. Features will cost more than they should. Unless we keep  
the design fresh, we’ll just get slower and slower. Testing and refactoring work together to make feature-by-feature  
development possible. The nature of the work requires us to  
test and refactor. There is no better way known today. Business needs to know that the program works. Acceptance  
test-driven development is one good way to keep current on  
what’s working. Developers need to know precisely what’s  
broken when something breaks. Test-driven development is a  
key tool in accomplishing that. A strong suite of automated  
tests keeps us certain that things work, and having two layers  
of test, a Business layer and a Developer layer, works best.

## Translated Text:

شاید ما Feature را Push کنیم و سپس چیزها را مرتب کنیم تا یک Design خوب به دست آوریم. هر دو روش کار می‌کنند. چیزی که کار نمی‌کند این است که Design را رها کنیم. حفظ Design خوب در حالی که تغییر می‌کند، Refactoring نامیده می‌شود. Refactoring یک مهارت ضروری برای این روش توسعه نرم‌افزار است. ما در جای دیگری بیشتر درباره Refactoring صحبت خواهیم کرد. فعلاً بیایید روی اینکه چه چیزی مفید است تمرکز کنیم.   
فصل ۸. بدون Bug و خوب طراحی شده • ۷۲   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: در هر مرحله از ساخت Feature، تیم باید به اندازه کافی بهبود Design انجام دهد تا پیشرفت کند. هر تغییر تمایل دارد Design فعلی ما را خراب کند. برای حفظ Design خوب، باید آن را در طول مسیر بهبود دهیم. شاید ما یک فضای تمیز و مناسب برای Feature جدیدمان ایجاد کنیم. شاید ما Feature را Push کنیم و سپس چیزها را مرتب کنیم تا یک Design خوب به دست آوریم. هر دو روش کار می‌کنند. چیزی که کار نمی‌کند این است که Design را رها کنیم. حفظ Design خوب در حالی که تغییر می‌کند، Refactoring نامیده می‌شود. Refactoring یک مهارت ضروری برای این روش توسعه نرم‌افزار است. ما در جای دیگری بیشتر درباره Refactoring صحبت خواهیم کرد. فعلاً بیایید روی اینکه چه چیزی مفید است تمرکز کنیم.   
فصل ۸. بدون Bug و خوب طراحی شده • ۷۲   
گزارش خطا • بحث کنید]   
  
عدم حفظ Design به اندازه کافی خوب باعث کندی پیشرفت یا حتی توقف آن خواهد شد. اگر اجازه دهیم Design خراب شود، پیشرفت کند خواهد شد. Features هزینه بیشتری نسبت به آنچه باید خواهند داشت. مگر اینکه Design را تازه نگه داریم، فقط کندتر و کندتر خواهیم شد. Testing و Refactoring با هم کار می‌کنند تا توسعه Feature به Feature ممکن شود. ماهیت کار ما را ملزم می‌کند که Test و Refactor کنیم. هیچ روش بهتری که امروز شناخته شده باشد وجود ندارد. کسب‌وکار باید بداند که برنامه کار می‌کند. Acceptance Test-Driven Development یک روش خوب برای به‌روز ماندن در مورد آنچه کار می‌کند است. توسعه‌دهندگان باید دقیقاً بدانند چه چیزی خراب شده وقتی چیزی خراب می‌شود. Test-Driven Development یک ابزار کلیدی برای دستیابی به این هدف است. یک مجموعه قوی از Automated Tests به ما اطمینان می‌دهد که چیزها کار می‌کنند، و داشتن دو لایه Test، یک لایه Business و یک لایه Developer، بهترین عملکرد را دارد.

================================================================================

# Section 73

## Original Text:

To progress, the design must improve continually. Improv-  
ing the design by refactoring is the technique for doing this. But even the best refactoring can break something. Our  
business-side and technical-side test suites give us the confi-  
dence to do the design improvement that we need to do. For the best quality, smoothest progress, and greatest pre-  
dictability, this is the best known way to work. Testing and  
refactoring are critical tools in feature-by-feature develop-  
ment. Don’t leave home without them. report erratum • discussChapter 8. Bug-Free and Well Designed • 73  
  
[CONTEXT FROM PREVIOUS PAGE: Failure to keep the design good enough will slow progress, or even  
stop it. If our design is allowed to deteriorate, progress will slow. Features will cost more than they should. Unless we keep  
the design fresh, we’ll just get slower and slower. Testing and refactoring work together to make feature-by-feature  
development possible. The nature of the work requires us to  
test and refactor. There is no better way known today. Business needs to know that the program works. Acceptance  
test-driven development is one good way to keep current on  
what’s working. Developers need to know precisely what’s  
broken when something breaks. Test-driven development is a  
key tool in accomplishing that. A strong suite of automated  
tests keeps us certain that things work, and having two layers  
of test, a Business layer and a Developer layer, works best. To progress, the design must improve continually. Improv-  
ing the design by refactoring is the technique for doing this. But even the best refactoring can break something. Our  
business-side and technical-side test suites give us the confi-  
dence to do the design improvement that we need to do. For the best quality, smoothest progress, and greatest pre-  
dictability, this is the best known way to work. Testing and  
refactoring are critical tools in feature-by-feature develop-  
ment. Don’t leave home without them.

## Translated Text:

برای پیشرفت، طراحی باید به طور مداوم بهبود یابد. بهبود طراحی از طریق refactoring تکنیکی است که برای این کار استفاده می‌شود. اما حتی بهترین refactoring هم ممکن است چیزی را خراب کند. مجموعه تست‌های business-side و technical-side به ما اطمینان می‌دهند که بهبود طراحی مورد نیاز را انجام دهیم. برای بهترین کیفیت، روان‌ترین پیشرفت، و بیشترین قابلیت پیش‌بینی، این بهترین روش شناخته‌شده برای کار است. Testing و refactoring ابزارهای حیاتی در توسعه feature-by-feature هستند. بدون آن‌ها از خانه خارج نشوید.  
  
[متن از صفحه قبلی: عدم حفظ طراحی در سطح کافی خوب، پیشرفت را کند یا حتی متوقف می‌کند. اگر اجازه دهیم طراحی خراب شود، پیشرفت کند خواهد شد. ویژگی‌ها بیشتر از آنچه باید هزینه خواهند داشت. مگر اینکه طراحی را تازه نگه داریم، فقط کندتر و کندتر خواهیم شد. Testing و refactoring با هم کار می‌کنند تا توسعه feature-by-feature ممکن شود. ماهیت این کار ما را ملزم به Testing و refactoring می‌کند. امروزه روش بهتری شناخته نشده است. کسب‌وکار باید بداند که برنامه کار می‌کند. Acceptance test-driven development یکی از راه‌های خوب برای به‌روز ماندن در مورد آنچه کار می‌کند است. توسعه‌دهندگان باید دقیقاً بدانند چه چیزی خراب شده است وقتی چیزی خراب می‌شود. Test-driven development یک ابزار کلیدی برای دستیابی به این هدف است. یک مجموعه قوی از automated tests ما را مطمئن می‌کند که همه چیز کار می‌کند، و داشتن دو لایه تست، یک لایه Business و یک لایه Developer، بهترین کارایی را دارد. برای پیشرفت، طراحی باید به طور مداوم بهبود یابد. بهبود طراحی از طریق refactoring تکنیکی است که برای این کار استفاده می‌شود. اما حتی بهترین refactoring هم ممکن است چیزی را خراب کند. مجموعه تست‌های business-side و technical-side به ما اطمینان می‌دهند که بهبود طراحی مورد نیاز را انجام دهیم. برای بهترین کیفیت، روان‌ترین پیشرفت، و بیشترین قابلیت پیش‌بینی، این بهترین روش شناخته‌شده برای کار است. Testing و refactoring ابزارهای حیاتی در توسعه feature-by-feature هستند. بدون آن‌ها از خانه خارج نشوید.]

================================================================================

# Section 74

## Original Text:

report erratum • discussChapter 8. Bug-Free and Well Designed • 73]  
  
Further reading:  
•Chapter 18, To Speed Up, Build with Skill , on page 116  
•Chapter 19, Refactoring , on page 120  
•Chapter 14, Creating Teams That Thrive , on page 94  
•Chapter 17, Whip the Ponies Harder , on page 112Chapter 8. Bug-Free and Well Designed • 74  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Further reading:  
•Chapter 18, To Speed Up, Build with Skill , on page 116  
•Chapter 19, Refactoring , on page 120  
•Chapter 14, Creating Teams That Thrive , on page 94  
•Chapter 17, Whip the Ponies Harder , on page 112Chapter 8. Bug-Free and Well Designed • 74  
report erratum • discuss]  
  
report erratum • discussChapter 8. Bug-Free and Well Designed • 75  
  
[CONTEXT FROM PREVIOUS PAGE: report erratum • discussChapter 8. Bug-Free and Well Designed • 75]  
  
Let me explain. No, there is too much. Let me sum up. CHAPTER 9  
Full Circle  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Let me explain. No, there is too much. Let me sum up. CHAPTER 9  
Full Circle  
report erratum • discuss]  
  
1. Value is what we want. Features deliver value. Deliver-  
ing features early gives us value sooner. 2. Managing by looking at value works better than manag-  
ing by dates or artifacts that don’t deliver value. 3. Planning features is easy enough to do. Estimate if you  
must. Selecting the work based on Yesterday’s Weather  
works better. 4. Building by features requires us to build a small, com-  
plete product, every couple of weeks. That product must  
always work correctly, and it must always be well  
designed. 5. Development must deliver real working features. The  
product must be well tested. Business-side people and  
developers contribute to testing. The product must be  
well designed. Developers keep the design alive all the  
time. That’s all there is to it. Very simple. A commitment from the  
top of the business, down to the individual managers and  
developers, is all it takes. Let’s get going.

## Translated Text:

گزارش خطا • بحث   
فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 73   
  
مطالعه بیشتر:   
• فصل 18، To Speed Up, Build with Skill، در صفحه 116   
• فصل 19، Refactoring، در صفحه 120   
• فصل 14، Creating Teams That Thrive، در صفحه 94   
• فصل 17، Whip the Ponies Harder، در صفحه 112   
فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 74   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: مطالعه بیشتر:   
• فصل 18، To Speed Up, Build with Skill، در صفحه 116   
• فصل 19، Refactoring، در صفحه 120   
• فصل 14، Creating Teams That Thrive، در صفحه 94   
• فصل 17، Whip the Ponies Harder، در صفحه 112   
فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 74   
گزارش خطا • بحث]   
  
گزارش خطا • بحث   
فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 75   
  
[متن از صفحه قبلی: گزارش خطا • بحث   
فصل 8. بدون باگ و طراحی‌شده به‌خوبی • 75]   
  
اجازه دهید توضیح دهم. نه، خیلی زیاد است. اجازه دهید خلاصه کنم.   
فصل 9   
Full Circle   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: اجازه دهید توضیح دهم. نه، خیلی زیاد است. اجازه دهید خلاصه کنم.   
فصل 9   
Full Circle   
گزارش خطا • بحث]   
  
1. ارزش چیزی است که ما می‌خواهیم. Features ارزش را ارائه می‌دهند. ارائه Features زودتر، ارزش را زودتر به ما می‌دهد.   
2. مدیریت با نگاه به ارزش بهتر از مدیریت با تاریخ‌ها یا مصنوعات است که ارزش ارائه نمی‌دهند.   
3. برنامه‌ریزی Features به اندازه کافی آسان است. اگر لازم است، تخمین بزنید. انتخاب کار بر اساس Yesterday’s Weather بهتر عمل می‌کند.   
4. ساختن با Features نیازمند این است که هر چند هفته یک محصول کوچک و کامل بسازیم. آن محصول باید همیشه به‌درستی کار کند و باید همیشه به‌خوبی طراحی شده باشد.   
5. توسعه باید Features واقعی و کاربردی ارائه دهد. محصول باید به‌خوبی تست شود. افراد سمت کسب‌وکار و توسعه‌دهندگان در تست مشارکت می‌کنند. محصول باید به‌خوبی طراحی شده باشد. توسعه‌دهندگان طراحی را همیشه زنده نگه می‌دارند.   
  
همین است. بسیار ساده. یک تعهد از بالاترین سطح کسب‌وکار تا مدیران و توسعه‌دهندگان فردی کافی است. بیایید شروع کنیم.

================================================================================

# Section 75

## Original Text:

Show me the soft-  
ware. Further reading:  
•Chapter 13, Not That Simple , on page 92  
report erratum • discussChapter 9. Full Circle • 77  
  
[CONTEXT FROM PREVIOUS PAGE: 1. Value is what we want. Features deliver value. Deliver-  
ing features early gives us value sooner. 2. Managing by looking at value works better than manag-  
ing by dates or artifacts that don’t deliver value. 3. Planning features is easy enough to do. Estimate if you  
must. Selecting the work based on Yesterday’s Weather  
works better. 4. Building by features requires us to build a small, com-  
plete product, every couple of weeks. That product must  
always work correctly, and it must always be well  
designed. 5. Development must deliver real working features. The  
product must be well tested. Business-side people and  
developers contribute to testing. The product must be  
well designed. Developers keep the design alive all the  
time. That’s all there is to it. Very simple. A commitment from the  
top of the business, down to the individual managers and  
developers, is all it takes. Let’s get going. Show me the soft-  
ware. Further reading:  
•Chapter 13, Not That Simple , on page 92  
report erratum • discussChapter 9. Full Circle • 77]  
  
Part II  
Notes and Essays  
The essential flow of software development is simple, as we’ve dis-  
cussed in Part I. Good work is simple; it is not easy. The details, of  
course, are endless. In this part we take a closer look at a few of those  
details. Here you’ll find short notes and essays on topics that interest  
me—and, I hope, you. [CONTEXT FROM PREVIOUS PAGE: Part II  
Notes and Essays  
The essential flow of software development is simple, as we’ve dis-  
cussed in Part I. Good work is simple; it is not easy. The details, of  
course, are endless. In this part we take a closer look at a few of those  
details. Here you’ll find short notes and essays on topics that interest  
me—and, I hope, you. ]  
  
CHAPTER 10  
Value—What Is It.

## Translated Text:

نرم‌افزار را به من نشان دهید. مطالعه بیشتر:   
• فصل 13، Not That Simple، در صفحه 92   
report erratum • discuss   
فصل 9. Full Circle • 77   
  
[متن از صفحه قبلی:   
1. ارزش چیزی است که ما می‌خواهیم. Features ارزش را ارائه می‌دهند. ارائه Features زودتر، ارزش را زودتر به ما می‌دهد.   
2. مدیریت با نگاه به ارزش بهتر از مدیریت با تاریخ‌ها یا مواردی است که ارزش ارائه نمی‌دهند.   
3. برنامه‌ریزی Features به اندازه کافی آسان است. اگر لازم است، تخمین بزنید. انتخاب کار بر اساس Yesterday’s Weather بهتر عمل می‌کند.   
4. ساختن بر اساس Features نیازمند ساخت یک محصول کوچک و کامل هر چند هفته یک بار است. آن محصول باید همیشه به درستی کار کند و همیشه خوب طراحی شده باشد.   
5. توسعه باید Features واقعی و کاربردی ارائه دهد. محصول باید به خوبی تست شده باشد. افراد بخش کسب‌وکار و توسعه‌دهندگان در تست کردن مشارکت می‌کنند. محصول باید به خوبی طراحی شده باشد. توسعه‌دهندگان طراحی را همیشه زنده نگه می‌دارند. همین است. بسیار ساده. تعهد از بالاترین سطح کسب‌وکار تا مدیران و توسعه‌دهندگان فردی، تنها چیزی است که لازم است. بیایید شروع کنیم. نرم‌افزار را به من نشان دهید. مطالعه بیشتر:   
• فصل 13، Not That Simple، در صفحه 92   
report erratum • discuss   
فصل 9. Full Circle • 77]   
  
بخش دوم   
یادداشت‌ها و مقالات   
جریان اصلی توسعه نرم‌افزار ساده است، همانطور که در بخش اول بحث کردیم. کار خوب ساده است؛ آسان نیست. جزئیات، البته، بی‌پایان هستند. در این بخش، نگاهی دقیق‌تر به برخی از این جزئیات خواهیم داشت. اینجا یادداشت‌ها و مقالات کوتاهی درباره موضوعاتی که برای من جالب هستند—و امیدوارم برای شما نیز جالب باشند—پیدا خواهید کرد.   
  
[متن از صفحه قبلی:   
بخش دوم   
یادداشت‌ها و مقالات   
جریان اصلی توسعه نرم‌افزار ساده است، همانطور که در بخش اول بحث کردیم. کار خوب ساده است؛ آسان نیست. جزئیات، البته، بی‌پایان هستند. در این بخش، نگاهی دقیق‌تر به برخی از این جزئیات خواهیم داشت. اینجا یادداشت‌ها و مقالات کوتاهی درباره موضوعاتی که برای من جالب هستند—و امیدوارم برای شما نیز جالب باشند—پیدا خواهید کرد.]   
  
فصل 10   
ارزش—چیست؟

================================================================================

# Section 76

## Original Text:

A reader asked: “Ron, in Chapter 2, Value Is What We Want, on  
page 6 , you just say that value is what we want. Isn’t that an  
oversimplification. ”  
I replied: “Well, yes, as is everything in this book. ” Let me say a  
few more words, however, to help out a bit. See also Chapter 11,  
Value—How Can We Measure It. , on page 84. In the book Zen and the Art of Motorcycle Maintenance [Pir00] ,  
Robert Pirsig’s protagonist, Phaedrus, is exploring the idea of  
quality. At one point he reaches the statement: “Quality is what  
you like. ” I’m offering a similar thought here. In Agile software development—as in many other realms—we  
talk about the notion of value. We make decisions about what to  
do, or what not to do, based on value. We do things sooner if they  
are of higher value, and we do things later if their value is lower. What do we mean by value. Value is, simply, “what you want. ”  
You may at first feel that this statement is too Zen, or that it has  
no meaning at all. Let’s explore here what we might mean by value,  
and how we work with it. My aim is to help you see that value is  
in fact, what you want, what you care about. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 10  
Value—What Is It. A reader asked: “Ron, in Chapter 2, Value Is What We Want, on  
page 6 , you just say that value is what we want. Isn’t that an  
oversimplification. ”  
I replied: “Well, yes, as is everything in this book. ” Let me say a  
few more words, however, to help out a bit. See also Chapter 11,  
Value—How Can We Measure It. , on page 84. In the book Zen and the Art of Motorcycle Maintenance [Pir00] ,  
Robert Pirsig’s protagonist, Phaedrus, is exploring the idea of  
quality. At one point he reaches the statement: “Quality is what  
you like. ” I’m offering a similar thought here. In Agile software development—as in many other realms—we  
talk about the notion of value. We make decisions about what to  
do, or what not to do, based on value.

## Translated Text:

یک خواننده پرسید: «رون، در فصل ۲، ارزش همان چیزی است که می‌خواهیم، در صفحه ۶، شما فقط می‌گویید که ارزش همان چیزی است که می‌خواهیم. آیا این یک ساده‌سازی بیش از حد نیست؟»   
من پاسخ دادم: «خب، بله، همان‌طور که همه چیز در این کتاب ساده‌سازی شده است.» با این حال، اجازه دهید چند کلمه بیشتر بگویم تا کمی کمک کنم. همچنین به فصل ۱۱، ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم، در صفحه ۸۴ مراجعه کنید.   
  
در کتاب «ذن و هنر تعمیر و نگهداری موتورسیکلت» [Pir00]، شخصیت اصلی رابرت پیرسیگ، فایدروس، در حال بررسی مفهوم کیفیت است. در یک نقطه، او به این جمله می‌رسد: «کیفیت همان چیزی است که دوست دارید.» من در اینجا یک فکر مشابه ارائه می‌دهم.   
  
در توسعه نرم‌افزار Agile—همانند بسیاری از حوزه‌های دیگر—ما درباره مفهوم ارزش صحبت می‌کنیم. ما تصمیم می‌گیریم که چه کاری انجام دهیم یا چه کاری انجام ندهیم، بر اساس ارزش. ما کارها را زودتر انجام می‌دهیم اگر ارزش بیشتری داشته باشند، و کارها را دیرتر انجام می‌دهیم اگر ارزش کمتری داشته باشند.   
  
منظور ما از ارزش چیست؟ ارزش، به سادگی، «چیزی است که می‌خواهید.»   
ممکن است در ابتدا احساس کنید که این جمله بیش از حد ذن است، یا اینکه اصلاً معنایی ندارد. بیایید در اینجا بررسی کنیم که ممکن است منظور ما از ارزش چیست و چگونه با آن کار می‌کنیم. هدف من این است که به شما کمک کنم ببینید که ارزش در واقع همان چیزی است که می‌خواهید، چیزی که برایتان اهمیت دارد.   
  
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: فصل ۱۰   
ارزش—چیست؟   
یک خواننده پرسید: «رون، در فصل ۲، ارزش همان چیزی است که می‌خواهیم، در صفحه ۶، شما فقط می‌گویید که ارزش همان چیزی است که می‌خواهیم. آیا این یک ساده‌سازی بیش از حد نیست؟»   
من پاسخ دادم: «خب، بله، همان‌طور که همه چیز در این کتاب ساده‌سازی شده است.» با این حال، اجازه دهید چند کلمه بیشتر بگویم تا کمی کمک کنم. همچنین به فصل ۱۱، ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم، در صفحه ۸۴ مراجعه کنید.   
  
در کتاب «ذن و هنر تعمیر و نگهداری موتورسیکلت» [Pir00]، شخصیت اصلی رابرت پیرسیگ، فایدروس، در حال بررسی مفهوم کیفیت است. در یک نقطه، او به این جمله می‌رسد: «کیفیت همان چیزی است که دوست دارید.» من در اینجا یک فکر مشابه ارائه می‌دهم.   
  
در توسعه نرم‌افزار Agile—همانند بسیاری از حوزه‌های دیگر—ما درباره مفهوم ارزش صحبت می‌کنیم. ما تصمیم می‌گیریم که چه کاری انجام دهیم یا چه کاری انجام ندهیم، بر اساس ارزش.]

================================================================================

# Section 77

## Original Text:

We do things sooner if they  
are of higher value, and we do things later if their value is lower. What do we mean by value. Value is, simply, “what you want. ”  
You may at first feel that this statement is too Zen, or that it has  
no meaning at all. Let’s explore here what we might mean by value,  
and how we work with it. My aim is to help you see that value is  
in fact, what you want, what you care about. report erratum • discuss]  
  
Agile methods ask us to choose the order in which we do  
things based on value. Sometimes we might say business  
value orcustomer value , as if these qualifications help. In a  
way, they do help, because they may cause us to think about  
things we value in terms of “good for the business” or “good  
for the customer. ” But these are far from the only kinds of  
value we might consider. Let’s look at just a few more. We might be choosing a strategic direction for our product. We decide that we need information about what users would  
prefer, so we create some prototypes and show them to  
potential users. We value information. Our product might be aimed at saving lives; perhaps it helps  
us ship vaccines around the world quickly. We decide to  
choose our next features based on the number of lives saved  
by those features. We value human life. report erratum • discussChapter 10. Value—What Is It. • 81  
  
[CONTEXT FROM PREVIOUS PAGE: Agile methods ask us to choose the order in which we do  
things based on value. Sometimes we might say business  
value orcustomer value , as if these qualifications help. In a  
way, they do help, because they may cause us to think about  
things we value in terms of “good for the business” or “good  
for the customer. ” But these are far from the only kinds of  
value we might consider. Let’s look at just a few more. We might be choosing a strategic direction for our product. We decide that we need information about what users would  
prefer, so we create some prototypes and show them to  
potential users. We value information.

## Translated Text:

ما کارها را زودتر انجام می‌دهیم اگر ارزش بالاتری داشته باشند، و کارها را دیرتر انجام می‌دهیم اگر ارزش آن‌ها پایین‌تر باشد. منظور ما از ارزش چیست؟ ارزش به‌سادگی همان چیزی است که شما می‌خواهید.   
ممکن است ابتدا احساس کنید که این جمله بیش از حد فلسفی است یا اصلاً هیچ معنایی ندارد. بیایید اینجا بررسی کنیم که منظور ما از ارزش چیست و چگونه با آن کار می‌کنیم. هدف من این است که به شما نشان دهم که ارزش در واقع همان چیزی است که شما می‌خواهید، چیزی که برای شما مهم است.   
گزارش خطا • بحث کنید   
  
روش‌های Agile از ما می‌خواهند که ترتیب انجام کارها را بر اساس ارزش انتخاب کنیم. گاهی ممکن است بگوییم business value یا customer value، انگار این مشخصه‌ها کمک‌کننده هستند. به نوعی، آن‌ها کمک می‌کنند، زیرا ممکن است باعث شوند که درباره چیزهایی که ارزشمند می‌دانیم از نظر "مفید برای کسب‌وکار" یا "مفید برای مشتری" فکر کنیم. اما این‌ها تنها انواع ارزش‌هایی نیستند که ممکن است در نظر بگیریم. بیایید فقط چند مورد دیگر را بررسی کنیم.   
ممکن است در حال انتخاب یک جهت استراتژیک برای محصول خود باشیم. تصمیم می‌گیریم که به اطلاعاتی درباره ترجیحات کاربران نیاز داریم، بنابراین چند prototype ایجاد می‌کنیم و آن‌ها را به کاربران بالقوه نشان می‌دهیم. ما به اطلاعات ارزش می‌دهیم.   
ممکن است محصول ما برای نجات جان‌ها طراحی شده باشد؛ شاید به ما کمک کند واکسن‌ها را به‌سرعت در سراسر جهان ارسال کنیم. تصمیم می‌گیریم که ویژگی‌های بعدی خود را بر اساس تعداد جان‌هایی که توسط آن ویژگی‌ها نجات داده می‌شوند انتخاب کنیم. ما به زندگی انسان‌ها ارزش می‌دهیم.   
گزارش خطا • بحث کنید   
فصل ۱۰. ارزش—چیست؟ • ۸۱   
  
[متن از صفحه قبلی: روش‌های Agile از ما می‌خواهند که ترتیب انجام کارها را بر اساس ارزش انتخاب کنیم. گاهی ممکن است بگوییم business value یا customer value، انگار این مشخصه‌ها کمک‌کننده هستند. به نوعی، آن‌ها کمک می‌کنند، زیرا ممکن است باعث شوند که درباره چیزهایی که ارزشمند می‌دانیم از نظر "مفید برای کسب‌وکار" یا "مفید برای مشتری" فکر کنیم. اما این‌ها تنها انواع ارزش‌هایی نیستند که ممکن است در نظر بگیریم. بیایید فقط چند مورد دیگر را بررسی کنیم.   
ممکن است در حال انتخاب یک جهت استراتژیک برای محصول خود باشیم. تصمیم می‌گیریم که به اطلاعاتی درباره ترجیحات کاربران نیاز داریم، بنابراین چند prototype ایجاد می‌کنیم و آن‌ها را به کاربران بالقوه نشان می‌دهیم. ما به اطلاعات ارزش می‌دهیم.]

================================================================================

# Section 78

## Original Text:

Our product might be aimed at saving lives; perhaps it helps  
us ship vaccines around the world quickly. We decide to  
choose our next features based on the number of lives saved  
by those features. We value human life. report erratum • discussChapter 10. Value—What Is It. • 81]  
  
Our company might be running out of cash. We decide to  
get some venture capital and to produce a sample product  
quickly to show to would-be investors. We value capital. We value company survival. We value the  
ability to help the customers we may someday have if we  
stay in business. Our product might be too slow. Customers are using alter-  
natives because they do not like the speed of our product. We decide to defer features to speed up the software. We value product speed. Our progress might be too slow, taking too long to get things  
done. We decide to defer features to clean up the software  
so that we can develop faster. We value rapid progress. Our product might display funny cat pictures. Our purpose  
might be to make people smile, to provide a bit of happiness  
in their day. We value people’s happiness. We value joy. We value creativity. We value collaboration. We value money. We value revenue. We value the ability  
to keep working. We value the ability to be with people we  
care about, doing things we care about. We value human  
life, or even kitty life. Chapter 10. Value—What Is It. • 82  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Our company might be running out of cash. We decide to  
get some venture capital and to produce a sample product  
quickly to show to would-be investors. We value capital. We value company survival. We value the  
ability to help the customers we may someday have if we  
stay in business. Our product might be too slow. Customers are using alter-  
natives because they do not like the speed of our product. We decide to defer features to speed up the software. We value product speed. Our progress might be too slow, taking too long to get things  
done.

## Translated Text:

محصول ما ممکن است برای نجات جان‌ها طراحی شده باشد؛ شاید به ما کمک کند واکسن‌ها را به سرعت در سراسر جهان ارسال کنیم. ما تصمیم می‌گیریم ویژگی‌های بعدی را بر اساس تعداد جان‌هایی که توسط آن ویژگی‌ها نجات داده می‌شوند، انتخاب کنیم. ما به زندگی انسان‌ها ارزش می‌دهیم.   
report erratum • discuss   
فصل ۱۰. ارزش—ارزش چیست. • ۸۱   
  
شرکت ما ممکن است در حال تمام شدن سرمایه باشد. ما تصمیم می‌گیریم سرمایه‌گذاری خطرپذیر دریافت کنیم و یک محصول نمونه را به سرعت تولید کنیم تا به سرمایه‌گذاران احتمالی نشان دهیم. ما به سرمایه ارزش می‌دهیم. ما به بقای شرکت ارزش می‌دهیم. ما به توانایی کمک به مشتریانی که ممکن است روزی داشته باشیم، اگر در کسب‌وکار بمانیم، ارزش می‌دهیم.   
محصول ما ممکن است خیلی کند باشد. مشتریان از جایگزین‌ها استفاده می‌کنند زیرا از سرعت محصول ما راضی نیستند. ما تصمیم می‌گیریم ویژگی‌ها را به تعویق بیندازیم تا نرم‌افزار را سریع‌تر کنیم. ما به سرعت محصول ارزش می‌دهیم.   
پیشرفت ما ممکن است خیلی کند باشد و زمان زیادی برای انجام کارها صرف شود. ما تصمیم می‌گیریم ویژگی‌ها را به تعویق بیندازیم تا نرم‌افزار را تمیز کنیم و بتوانیم سریع‌تر توسعه دهیم. ما به پیشرفت سریع ارزش می‌دهیم.   
محصول ما ممکن است تصاویر خنده‌دار از گربه‌ها نمایش دهد. هدف ما ممکن است این باشد که مردم را بخندانیم و کمی شادی به روزشان بیاوریم. ما به خوشحالی مردم ارزش می‌دهیم. ما به شادی ارزش می‌دهیم. ما به خلاقیت ارزش می‌دهیم. ما به همکاری ارزش می‌دهیم. ما به پول ارزش می‌دهیم. ما به درآمد ارزش می‌دهیم. ما به توانایی ادامه کار ارزش می‌دهیم. ما به توانایی بودن با افرادی که برایمان مهم هستند و انجام کارهایی که برایمان اهمیت دارند، ارزش می‌دهیم. ما به زندگی انسان‌ها، یا حتی زندگی گربه‌ها، ارزش می‌دهیم.   
فصل ۱۰. ارزش—ارزش چیست. • ۸۲   
report erratum • discuss

================================================================================

# Section 79

## Original Text:

We decide to defer features to clean up the software  
so that we can develop faster. We value rapid progress. Our product might display funny cat pictures. Our purpose  
might be to make people smile, to provide a bit of happiness  
in their day. We value people’s happiness. We value joy. We value creativity. We value collaboration. We value money. We value revenue. We value the ability  
to keep working. We value the ability to be with people we  
care about, doing things we care about. We value human  
life, or even kitty life. Chapter 10. Value—What Is It. • 82  
report erratum • discuss]  
  
These are just a few of the things that make up value. The  
problem of the “Product Owner,” of management, of all the  
people who decide what we should do next, is to look deeply  
at the many things we value, and to choose a sequence of  
development that gives us the best possible result in return  
for our time, money, and effort in building our product. It would be nice if it were easier. It would be nice if we could  
just say “Value is revenue over the next 90 days,” or “Value  
is what the VP of Sales wants. ” It might even work for some  
people, some companies, sometimes. But no such definition  
will work for everyone, and in my opinion, it won’t work  
very well for anyone. If we are to survive as a company or  
as individuals, we need to look deeply at value, and to  
choose the things that matter, among all the things we might  
do. Choosing value is choosing what matters to you. Value…is what you want. report erratum • discussChapter 10. Value—What Is It. • 83  
  
[CONTEXT FROM PREVIOUS PAGE: These are just a few of the things that make up value. The  
problem of the “Product Owner,” of management, of all the  
people who decide what we should do next, is to look deeply  
at the many things we value, and to choose a sequence of  
development that gives us the best possible result in return  
for our time, money, and effort in building our product. It would be nice if it were easier.

## Translated Text:

ما تصمیم می‌گیریم که ویژگی‌ها را به تعویق بیندازیم تا نرم‌افزار را پاکسازی کنیم و بتوانیم سریع‌تر توسعه دهیم. ما به پیشرفت سریع ارزش می‌دهیم. محصول ما ممکن است تصاویر خنده‌دار گربه‌ها را نمایش دهد. هدف ما ممکن است این باشد که مردم را بخندانیم، کمی شادی به روزشان بیاوریم. ما به خوشحالی مردم ارزش می‌دهیم. ما به شادی ارزش می‌دهیم. ما به خلاقیت ارزش می‌دهیم. ما به همکاری ارزش می‌دهیم. ما به پول ارزش می‌دهیم. ما به درآمد ارزش می‌دهیم. ما به توانایی ادامه کار ارزش می‌دهیم. ما به توانایی بودن با افرادی که برایمان مهم هستند و انجام کارهایی که برایمان اهمیت دارند، ارزش می‌دهیم. ما به زندگی انسان‌ها، یا حتی زندگی گربه‌ها ارزش می‌دهیم. فصل ۱۰. ارزش—ارزش چیست؟ • ۸۲   
گزارش خطا • بحث کنید   
  
این‌ها تنها چند مورد از چیزهایی هستند که ارزش را تشکیل می‌دهند. مشکل "Product Owner"، مدیریت، و همه افرادی که تصمیم می‌گیرند چه کاری را بعداً انجام دهیم، این است که عمیقاً به چیزهایی که برایمان ارزشمند است نگاه کنند و یک توالی توسعه را انتخاب کنند که بهترین نتیجه ممکن را در ازای زمان، پول و تلاشی که برای ساخت محصولمان صرف می‌کنیم، به ما بدهد. کاش این کار آسان‌تر بود. کاش می‌توانستیم فقط بگوییم "ارزش، درآمد در ۹۰ روز آینده است"، یا "ارزش همان چیزی است که VP of Sales می‌خواهد." شاید این تعریف برای برخی افراد، برخی شرکت‌ها، گاهی کار کند. اما هیچ تعریف مشابهی برای همه کار نخواهد کرد، و به نظر من، برای هیچ‌کس به خوبی کار نخواهد کرد. اگر بخواهیم به عنوان یک شرکت یا به عنوان افراد زنده بمانیم، باید عمیقاً به ارزش نگاه کنیم و از میان همه کارهایی که ممکن است انجام دهیم، چیزهایی را انتخاب کنیم که اهمیت دارند. انتخاب ارزش یعنی انتخاب آنچه برای شما مهم است. ارزش... همان چیزی است که شما می‌خواهید.   
گزارش خطا • بحث کنید   
فصل ۱۰. ارزش—ارزش چیست؟ • ۸۳   
  
[متن از صفحه قبل: این‌ها تنها چند مورد از چیزهایی هستند که ارزش را تشکیل می‌دهند. مشکل "Product Owner"، مدیریت، و همه افرادی که تصمیم می‌گیرند چه کاری را بعداً انجام دهیم، این است که عمیقاً به چیزهایی که برایمان ارزشمند است نگاه کنند و یک توالی توسعه را انتخاب کنند که بهترین نتیجه ممکن را در ازای زمان، پول و تلاشی که برای ساخت محصولمان صرف می‌کنیم، به ما بدهد. کاش این کار آسان‌تر بود.]

================================================================================

# Section 80

## Original Text:

It would be nice if we could  
just say “Value is revenue over the next 90 days,” or “Value  
is what the VP of Sales wants. ” It might even work for some  
people, some companies, sometimes. But no such definition  
will work for everyone, and in my opinion, it won’t work  
very well for anyone. If we are to survive as a company or  
as individuals, we need to look deeply at value, and to  
choose the things that matter, among all the things we might  
do. Choosing value is choosing what matters to you. Value…is what you want. report erratum • discussChapter 10. Value—What Is It. • 83]  
  
CHAPTER 11  
Value—How Can  
We Measure It. We began by suggesting that “Value is what you want” and  
advising you to produce small bits of real value every couple  
of weeks. We asked you to say “Show us the software” to  
see what’s really going on. The main idea is to concentrate  
on value, not cost, and to see that value in terms of real,  
running software with features we can understand. But Ron, you talk about value as if it’s purely subjective. Shouldn’t  
we be using real, solid numerical information to make our deci-  
sions. What do you have against measurement. Well, it’s a fair cop: I am pushing back against using  
numerical measures, in value and even in cost estimates. There are some reasons for this. We don’t really know the numbers. For almost any product of interest, we don’t know the  
numbers. We don’t know how many users will use our fea-  
ture. We don’t know how many lives will be saved. We don’t  
know if people will rate our latest idea with three stars or  
five. We don’t know if people will buy our product based  
on this idea, or refuse to buy it. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 11  
Value—How Can  
We Measure It. We began by suggesting that “Value is what you want” and  
advising you to produce small bits of real value every couple  
of weeks. We asked you to say “Show us the software” to  
see what’s really going on.

## Translated Text:

خیلی خوب می‌شد اگر می‌توانستیم فقط بگوییم «Value درآمد در ۹۰ روز آینده است» یا «Value چیزی است که VP of Sales می‌خواهد.» شاید این تعریف برای برخی افراد، برخی شرکت‌ها، و در برخی مواقع جواب بدهد. اما هیچ تعریفی وجود ندارد که برای همه کار کند و به نظر من، حتی برای یک نفر هم خیلی خوب کار نخواهد کرد. اگر می‌خواهیم به عنوان یک شرکت یا به عنوان افراد زنده بمانیم، باید به طور عمیق به Value نگاه کنیم و از بین تمام کارهایی که ممکن است انجام دهیم، چیزهایی را انتخاب کنیم که اهمیت دارند. انتخاب Value یعنی انتخاب چیزی که برای شما مهم است. Value... چیزی است که شما می‌خواهید.   
report erratum • discuss   
Chapter 10. Value—What Is It. • 83   
  
CHAPTER 11   
Value—How Can We Measure It   
ما با این پیشنهاد شروع کردیم که «Value چیزی است که شما می‌خواهید» و به شما توصیه کردیم که هر چند هفته یک بار، بخش‌های کوچکی از Value واقعی تولید کنید. از شما خواستیم که بگویید «نرم‌افزار را به ما نشان بدهید» تا ببینید واقعاً چه اتفاقی در حال رخ دادن است. ایده اصلی این است که بر روی Value تمرکز کنید، نه هزینه، و این Value را در قالب نرم‌افزار واقعی و در حال اجرا با قابلیت‌هایی که می‌توانیم درک کنیم، ببینید.   
  
اما رون، شما در مورد Value طوری صحبت می‌کنید که انگار کاملاً ذهنی است. آیا نباید از اطلاعات عددی واقعی و محکم برای تصمیم‌گیری‌هایمان استفاده کنیم؟ شما چه مشکلی با اندازه‌گیری دارید؟   
  
خب، این یک انتقاد منصفانه است: من در حال مقاومت در برابر استفاده از معیارهای عددی هستم، چه در مورد Value و چه حتی در تخمین هزینه‌ها. دلایلی برای این موضوع وجود دارد. ما واقعاً اعداد را نمی‌دانیم. برای تقریباً هر محصولی که مورد توجه است، ما اعداد را نمی‌دانیم. ما نمی‌دانیم چند کاربر از قابلیت ما استفاده خواهند کرد. ما نمی‌دانیم چند زندگی نجات پیدا خواهد کرد. ما نمی‌دانیم آیا مردم ایده جدید ما را با سه ستاره ارزیابی می‌کنند یا پنج ستاره. ما نمی‌دانیم آیا مردم محصول ما را بر اساس این ایده می‌خرند یا از خرید آن امتناع می‌کنند.   
report erratum • discuss

================================================================================

# Section 81

## Original Text:

The main idea is to concentrate  
on value, not cost, and to see that value in terms of real,  
running software with features we can understand. But Ron, you talk about value as if it’s purely subjective. Shouldn’t  
we be using real, solid numerical information to make our deci-  
sions. What do you have against measurement. Well, it’s a fair cop: I am pushing back against using  
numerical measures, in value and even in cost estimates. There are some reasons for this. We don’t really know the numbers. For almost any product of interest, we don’t know the  
numbers. We don’t know how many users will use our fea-  
ture. We don’t know how many lives will be saved. We don’t  
know if people will rate our latest idea with three stars or  
five. We don’t know if people will buy our product based  
on this idea, or refuse to buy it. report erratum • discuss]  
  
Big differences are important; small ones aren’t. When we look at all the feature options before us, some of  
them are incredibly important, and some are really dull and  
boring. That’s the distinction that matters: what’s incredibly  
important versus what’s boring. Different kinds of value aren’t comparable. A Product Champion values many different things. The  
values go up and down as time goes on. Sometimes we need  
customer information: will people find this idea useful. Sometimes we need development information: will this idea  
take a day to do, a week, or ten years. Sometimes we value  
pleasing some customer or prospect: if we can give them  
this feature quickly, they’ll give us their money. Then what  
should we do. report erratum • discussChapter 11. Value—How Can We Measure It. • 85  
  
[CONTEXT FROM PREVIOUS PAGE: Big differences are important; small ones aren’t. When we look at all the feature options before us, some of  
them are incredibly important, and some are really dull and  
boring. That’s the distinction that matters: what’s incredibly  
important versus what’s boring. Different kinds of value aren’t comparable.

## Translated Text:

ایده اصلی این است که بر ارزش تمرکز کنیم، نه هزینه، و این ارزش را به صورت نرم‌افزار واقعی و در حال اجرا با قابلیت‌هایی که می‌توانیم درک کنیم ببینیم. اما رون، تو درباره ارزش صحبت می‌کنی انگار که کاملاً ذهنی است. آیا نباید از اطلاعات عددی واقعی و محکم برای تصمیم‌گیری‌های خود استفاده کنیم؟ چه چیزی علیه اندازه‌گیری داری؟ خب، قبول دارم: من در برابر استفاده از معیارهای عددی، چه در ارزش و چه در تخمین هزینه، مقاومت می‌کنم. دلایلی برای این وجود دارد. ما واقعاً اعداد را نمی‌دانیم. برای تقریباً هر محصولی که مورد علاقه است، ما اعداد را نمی‌دانیم. ما نمی‌دانیم چند کاربر از قابلیت ما استفاده خواهند کرد. ما نمی‌دانیم چند زندگی نجات پیدا خواهند کرد. ما نمی‌دانیم آیا مردم ایده جدید ما را با سه ستاره یا پنج ستاره ارزیابی خواهند کرد. ما نمی‌دانیم آیا مردم محصول ما را بر اساس این ایده خواهند خرید یا از خرید آن امتناع خواهند کرد. گزارش خطا • بحث کنید]  
  
تفاوت‌های بزرگ مهم هستند؛ تفاوت‌های کوچک نیستند. وقتی به تمام گزینه‌های قابلیت‌هایی که پیش روی ما هستند نگاه می‌کنیم، برخی از آن‌ها فوق‌العاده مهم هستند و برخی واقعاً کسل‌کننده و خسته‌کننده. این همان تمایزی است که اهمیت دارد: چه چیزی فوق‌العاده مهم است در مقابل چه چیزی کسل‌کننده است. انواع مختلف ارزش قابل مقایسه نیستند. یک Product Champion چیزهای مختلفی را ارزش‌گذاری می‌کند. این ارزش‌ها با گذشت زمان بالا و پایین می‌روند. گاهی اوقات ما به اطلاعات مشتری نیاز داریم: آیا مردم این ایده را مفید خواهند یافت؟ گاهی اوقات ما به اطلاعات توسعه نیاز داریم: آیا این ایده یک روز طول می‌کشد، یک هفته یا ده سال؟ گاهی اوقات ما ارزش رضایت یک مشتری یا مشتری بالقوه را داریم: اگر بتوانیم این قابلیت را سریع به آن‌ها بدهیم، آن‌ها پولشان را به ما خواهند داد. پس چه کاری باید انجام دهیم؟ گزارش خطا • بحث کنید فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 85  
  
[متن از صفحه قبلی: تفاوت‌های بزرگ مهم هستند؛ تفاوت‌های کوچک نیستند. وقتی به تمام گزینه‌های قابلیت‌هایی که پیش روی ما هستند نگاه می‌کنیم، برخی از آن‌ها فوق‌العاده مهم هستند و برخی واقعاً کسل‌کننده و خسته‌کننده. این همان تمایزی است که اهمیت دارد: چه چیزی فوق‌العاده مهم است در مقابل چه چیزی کسل‌کننده است. انواع مختلف ارزش قابل مقایسه نیستند.]

================================================================================

# Section 82

## Original Text:

A Product Champion values many different things. The  
values go up and down as time goes on. Sometimes we need  
customer information: will people find this idea useful. Sometimes we need development information: will this idea  
take a day to do, a week, or ten years. Sometimes we value  
pleasing some customer or prospect: if we can give them  
this feature quickly, they’ll give us their money. Then what  
should we do. report erratum • discussChapter 11. Value—How Can We Measure It. • 85]  
  
But Ron, then what is “value”. How should we measure it. How  
will we know we have it. Don’t panic. You already have a very good idea what value  
is. Take any two things you might do. Ask yourself which  
one to do next—which one is more valuable right now. It  
turns out you almost always know. If you don’t know, ask  
yourself whether either one of them is worth doing at all. Often, you’ll find they’re not. Then ask yourself why you prefer what you prefer. Write  
down a few notes. Ask again: why. This will give you a sense  
of some of the dimensions of value to you. Ask your stake-  
holders. Whether they agree with you or not, ask why. Build  
up a grasp of what’s important, and why. It’s always tempting to try to express value in numbers, and  
if you have a way to do that, go ahead and use it. Then look  
at two things your numbers suggest and ask yourself  
whether you agree. If you do, go ahead. If not, that’s inter-  
esting. Dig into that until you and your numbers agree. If  
they just won’t, I’d advise throwing out the numbers. Chapter 11. Value—How Can We Measure It. • 86  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: But Ron, then what is “value”. How should we measure it. How  
will we know we have it. Don’t panic. You already have a very good idea what value  
is. Take any two things you might do. Ask yourself which  
one to do next—which one is more valuable right now. It  
turns out you almost always know. If you don’t know, ask  
yourself whether either one of them is worth doing at all.

## Translated Text:

یک Product Champion به چیزهای مختلفی ارزش می‌دهد. این ارزش‌ها با گذشت زمان بالا و پایین می‌روند. گاهی اوقات به اطلاعات مشتری نیاز داریم: آیا مردم این ایده را مفید خواهند یافت؟ گاهی اوقات به اطلاعات توسعه نیاز داریم: آیا این ایده یک روز زمان می‌برد، یک هفته، یا ده سال؟ گاهی اوقات ارزش رضایت یک مشتری یا مشتری بالقوه را داریم: اگر بتوانیم این قابلیت را سریعاً به آنها بدهیم، پولشان را به ما خواهند داد. سپس چه باید کنیم؟   
report erratum • discuss   
فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 85   
  
اما رون، پس «ارزش» چیست؟ چگونه باید آن را اندازه‌گیری کنیم؟ چگونه خواهیم دانست که آن را داریم؟ نگران نباشید. شما همین حالا هم ایده بسیار خوبی درباره اینکه ارزش چیست دارید. دو کاری که ممکن است انجام دهید را در نظر بگیرید. از خودتان بپرسید کدام یک را باید بعدی انجام دهید—کدام یک در حال حاضر ارزشمندتر است. مشخص می‌شود که تقریباً همیشه می‌دانید. اگر نمی‌دانید، از خودتان بپرسید آیا هیچ‌کدام از آنها ارزش انجام دادن را دارند یا نه.   
اغلب، متوجه خواهید شد که اینطور نیستند. سپس از خودتان بپرسید چرا چیزی را که ترجیح می‌دهید، ترجیح می‌دهید. چند یادداشت بنویسید. دوباره بپرسید: چرا؟ این به شما حس برخی از ابعاد ارزش برای خودتان را خواهد داد. از ذی‌نفعان خود بپرسید. چه با شما موافق باشند یا نه، بپرسید چرا. درک کنید چه چیزی مهم است و چرا.   
  
همیشه وسوسه‌انگیز است که سعی کنید ارزش را با اعداد بیان کنید، و اگر راهی برای انجام این کار دارید، ادامه دهید و از آن استفاده کنید. سپس به دو چیزی که اعداد شما پیشنهاد می‌دهند نگاه کنید و از خودتان بپرسید آیا موافق هستید یا نه. اگر موافق هستید، ادامه دهید. اگر نه، این جالب است. به آن موضوع عمیق‌تر بپردازید تا زمانی که شما و اعدادتان موافق شوید. اگر به هیچ وجه موافق نمی‌شوند، توصیه می‌کنم اعداد را کنار بگذارید.   
فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 86   
report erratum • discuss

================================================================================

# Section 83

## Original Text:

Often, you’ll find they’re not. Then ask yourself why you prefer what you prefer. Write  
down a few notes. Ask again: why. This will give you a sense  
of some of the dimensions of value to you. Ask your stake-  
holders. Whether they agree with you or not, ask why. Build  
up a grasp of what’s important, and why. It’s always tempting to try to express value in numbers, and  
if you have a way to do that, go ahead and use it. Then look  
at two things your numbers suggest and ask yourself  
whether you agree. If you do, go ahead. If not, that’s inter-  
esting. Dig into that until you and your numbers agree. If  
they just won’t, I’d advise throwing out the numbers. Chapter 11. Value—How Can We Measure It. • 86  
report erratum • discuss]  
  
Looking for numbers puts us on a bit of a slippery slope. If  
the company’s purpose with the product is to make money,  
then we could measure the product by how much money is  
made. But there’s no useful way to know whether that  
number is a good measure of anything, since it conflates  
sales concerns, product concerns, and of course, customer  
concerns. Even worse, most monetary measures are trailing indicators:  
we don’t get information until it’s too late. We have no real  
way of knowing whether the revenue number is really good  
or really bad. Money is a terrible indicator: too slow, unclear  
when we get it. I wish I had an easy answer, like counting up function points  
or user preference clicks or the like. Use those if you find  
them valuable, but know that the true value of all the mea-  
sures you can think of is to build up an understanding, in  
your Product Champion, your stakeholders, and your team,  
about what’s really valuable. Instead, sit with your developers and stakeholders. Consider  
the things you might do. Select a combination of next things  
to do that the group agrees is most valuable. The true value  
in doing this is the learning that will come from reaching a  
consensus. Then build it, ship it promptly, and listen to your users. Repeat.

## Translated Text:

اغلب، متوجه می‌شوید که اینطور نیست. سپس از خودتان بپرسید چرا چیزی را که ترجیح می‌دهید، ترجیح می‌دهید. چند یادداشت بنویسید. دوباره بپرسید: چرا. این کار به شما درک بهتری از برخی ابعاد ارزش برای خودتان می‌دهد. از ذینفعان خود بپرسید. چه با شما موافق باشند یا نه، دلیلش را بپرسید. درک کنید که چه چیزی مهم است و چرا. همیشه وسوسه‌انگیز است که بخواهید ارزش را با اعداد بیان کنید، و اگر راهی برای این کار دارید، پیش بروید و از آن استفاده کنید. سپس به دو چیزی که اعداد شما پیشنهاد می‌دهند نگاه کنید و از خودتان بپرسید آیا با آن موافق هستید یا نه. اگر موافق هستید، ادامه دهید. اگر نه، این جالب است. به آن عمیق‌تر بپردازید تا زمانی که شما و اعدادتان با هم موافق شوید. اگر به هیچ وجه موافق نشدید، پیشنهاد می‌کنم اعداد را کنار بگذارید.   
فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 86   
گزارش خطا • بحث کنید   
  
جستجو برای اعداد ما را در مسیری لغزنده قرار می‌دهد. اگر هدف شرکت از محصول کسب درآمد باشد، آنگاه می‌توانیم محصول را با میزان درآمدی که ایجاد می‌کند اندازه‌گیری کنیم. اما هیچ راه مفیدی برای دانستن اینکه آیا آن عدد معیار خوبی برای چیزی است یا نه وجود ندارد، زیرا نگرانی‌های فروش، نگرانی‌های محصول، و البته نگرانی‌های مشتری را با هم ترکیب می‌کند. حتی بدتر، بیشتر معیارهای مالی شاخص‌های تأخیری هستند: ما اطلاعات را زمانی دریافت می‌کنیم که خیلی دیر شده است. ما هیچ راه واقعی برای دانستن اینکه آیا عدد درآمد واقعاً خوب است یا بد نداریم. پول یک شاخص وحشتناک است: خیلی کند است و وقتی آن را دریافت می‌کنیم مبهم است.   
ای کاش یک پاسخ آسان داشتم، مثل شمارش Function Points یا کلیک‌های ترجیح کاربران یا چیزهای مشابه. اگر آن‌ها را مفید می‌دانید، از آن‌ها استفاده کنید، اما بدانید که ارزش واقعی همه معیارهایی که می‌توانید به آن‌ها فکر کنید، ایجاد درک در Product Champion، ذینفعان و تیم شما درباره چیزی است که واقعاً ارزشمند است.   
در عوض، با توسعه‌دهندگان و ذینفعان خود بنشینید. به کارهایی که ممکن است انجام دهید فکر کنید. ترکیبی از کارهای بعدی را انتخاب کنید که گروه بر این باور است که بیشترین ارزش را دارد. ارزش واقعی در انجام این کار یادگیری است که از رسیدن به اجماع حاصل می‌شود. سپس آن را Build کنید، سریعاً Ship کنید و به کاربران خود گوش دهید. تکرار کنید.

================================================================================

# Section 84

## Original Text:

report erratum • discussChapter 11. Value—How Can We Measure It. • 87  
  
[CONTEXT FROM PREVIOUS PAGE: Looking for numbers puts us on a bit of a slippery slope. If  
the company’s purpose with the product is to make money,  
then we could measure the product by how much money is  
made. But there’s no useful way to know whether that  
number is a good measure of anything, since it conflates  
sales concerns, product concerns, and of course, customer  
concerns. Even worse, most monetary measures are trailing indicators:  
we don’t get information until it’s too late. We have no real  
way of knowing whether the revenue number is really good  
or really bad. Money is a terrible indicator: too slow, unclear  
when we get it. I wish I had an easy answer, like counting up function points  
or user preference clicks or the like. Use those if you find  
them valuable, but know that the true value of all the mea-  
sures you can think of is to build up an understanding, in  
your Product Champion, your stakeholders, and your team,  
about what’s really valuable. Instead, sit with your developers and stakeholders. Consider  
the things you might do. Select a combination of next things  
to do that the group agrees is most valuable. The true value  
in doing this is the learning that will come from reaching a  
consensus. Then build it, ship it promptly, and listen to your users. Repeat. report erratum • discussChapter 11. Value—How Can We Measure It. • 87]  
  
CHAPTER 12  
Of Course It’s Hard. Early readers of this book have said, “Ron, you make this seem  
clear and simple and compelling. We should focus on value, plan  
by value, manage by value, and build by value. That’s all very well  
and good. But software development is hard. It’s very hard. ”  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 12  
Of Course It’s Hard. Early readers of this book have said, “Ron, you make this seem  
clear and simple and compelling. We should focus on value, plan  
by value, manage by value, and build by value.

## Translated Text:

گزارش خطا • بحث   
فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 87   
  
[متن از صفحه قبلی: جستجوی اعداد ما را وارد یک مسیر لغزنده می‌کند. اگر هدف شرکت از محصول کسب درآمد باشد،   
می‌توانیم محصول را بر اساس میزان پولی که کسب می‌شود اندازه‌گیری کنیم. اما هیچ راه مفیدی برای دانستن اینکه   
آیا آن عدد معیار خوبی برای چیزی است وجود ندارد، زیرا نگرانی‌های فروش، نگرانی‌های محصول، و البته نگرانی‌های   
مشتری را با هم ترکیب می‌کند. حتی بدتر، بیشتر معیارهای پولی شاخص‌های عقب‌مانده هستند:   
ما اطلاعات را زمانی دریافت می‌کنیم که خیلی دیر شده است. هیچ راه واقعی برای دانستن اینکه آیا عدد درآمد واقعاً   
خوب است یا واقعاً بد وجود ندارد. پول یک شاخص وحشتناک است: خیلی کند، نامشخص زمانی که آن را دریافت می‌کنیم.   
ای کاش یک پاسخ آسان داشتم، مانند شمارش Function Points یا کلیک‌های ترجیح کاربر یا موارد مشابه.   
از آن‌ها استفاده کنید اگر آن‌ها را ارزشمند می‌دانید، اما بدانید که ارزش واقعی تمام معیارهایی که می‌توانید به آن‌ها فکر کنید   
این است که درک را در Product Champion، ذینفعان و تیم شما ایجاد کند که واقعاً چه چیزی ارزشمند است.   
در عوض، با توسعه‌دهندگان و ذینفعان خود بنشینید. مواردی را که ممکن است انجام دهید در نظر بگیرید.   
ترکیبی از کارهای بعدی را انتخاب کنید که گروه توافق دارد بیشترین ارزش را دارد. ارزش واقعی در انجام این کار   
یادگیری‌ای است که از رسیدن به اجماع حاصل می‌شود. سپس آن را Build کنید، به سرعت Ship کنید، و به کاربران خود گوش دهید.   
تکرار کنید. گزارش خطا • بحث   
فصل 11. ارزش—چگونه می‌توانیم آن را اندازه‌گیری کنیم. • 87]   
  
فصل 12   
البته سخت است. خوانندگان اولیه این کتاب گفته‌اند: «رون، شما این را واضح، ساده و قانع‌کننده جلوه می‌دهید.   
ما باید بر ارزش تمرکز کنیم، بر اساس ارزش برنامه‌ریزی کنیم، بر اساس ارزش مدیریت کنیم، و بر اساس ارزش Build کنیم.   
این‌ها همه خوب و عالی است. اما توسعه نرم‌افزار سخت است. خیلی سخت است.»   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: فصل 12   
البته سخت است. خوانندگان اولیه این کتاب گفته‌اند: «رون، شما این را واضح، ساده و قانع‌کننده جلوه می‌دهید.   
ما باید بر ارزش تمرکز کنیم، بر اساس ارزش برنامه‌ریزی کنیم، بر اساس ارزش مدیریت کنیم، و بر اساس ارزش Build کنیم.

================================================================================

# Section 85

## Original Text:

That’s all very well  
and good. But software development is hard. It’s very hard. ”  
report erratum • discuss]  
  
Of course it’s hard. It’s the nature of what we’re doing that  
we’re always pushing our own limits. The point of this book,  
however, is that the approach we should take to doing these  
hard things is to insist: “Show us the software. ” That lets us  
accomplish the following:  
• Focus on what we value, so we can get the best results. • Produce real software often so that we can learn what  
we want. • Build what we want in small steps, so we can see how  
we’re doing. • Learn the planning, management, and technical skills  
that we need, so we can build the product rapidly and  
well. Every aspect of this work is complex, but we do not need a  
complex process to do the work. A complex process holds  
us back, imposing rote activity when we need to observe  
what’s going on, adjust it, and improve it. We do need increasing skill, which we can guide best by  
using a simple process aimed at delivering value. You will have to decide specifically “how” to do these things. The point of this book is to help you improve your intuition  
about how. There are no fixed “best” ways, so even if you  
decide to do some specific thing for a while, keep checking  
that it’s really helping. You will need to change things. Take every idea as, maybe, a good way to start doing things  
for a while. Then make the process your own, and build  
your own ideas. But keep it simple. report erratum • discussChapter 12. Of Course It’s Hard. • 89  
  
[CONTEXT FROM PREVIOUS PAGE: Of course it’s hard. It’s the nature of what we’re doing that  
we’re always pushing our own limits. The point of this book,  
however, is that the approach we should take to doing these  
hard things is to insist: “Show us the software. ” That lets us  
accomplish the following:  
• Focus on what we value, so we can get the best results. • Produce real software often so that we can learn what  
we want.

## Translated Text:

این‌ها همه خوب و عالی است. اما توسعه نرم‌افزار سخت است. خیلی سخت است.   
[گزارش خطا • بحث]  
  
البته که سخت است. ماهیت کاری که انجام می‌دهیم این است که همیشه در حال به چالش کشیدن محدودیت‌های خود هستیم. اما نکته این کتاب این است که رویکردی که باید برای انجام این کارهای سخت اتخاذ کنیم این است که اصرار کنیم: «نرم‌افزار را به ما نشان دهید.» این کار به ما اجازه می‌دهد که به موارد زیر دست پیدا کنیم:   
• تمرکز بر روی آنچه ارزشمند است، تا بتوانیم بهترین نتایج را بگیریم.   
• تولید نرم‌افزار واقعی به‌طور مکرر، تا بتوانیم یاد بگیریم که چه می‌خواهیم.   
• ساخت آنچه می‌خواهیم در گام‌های کوچک، تا بتوانیم ببینیم چگونه پیش می‌رویم.   
• یادگیری مهارت‌های برنامه‌ریزی، مدیریت و فنی که نیاز داریم، تا بتوانیم محصول را به‌سرعت و به‌خوبی بسازیم.   
  
هر جنبه‌ای از این کار پیچیده است، اما ما به یک فرآیند پیچیده برای انجام کار نیاز نداریم. یک فرآیند پیچیده ما را عقب نگه می‌دارد و فعالیت‌های تکراری را به ما تحمیل می‌کند، در حالی که ما نیاز داریم مشاهده کنیم چه اتفاقی می‌افتد، آن را تنظیم کنیم و بهبود دهیم.   
  
ما به مهارت‌های رو به افزایش نیاز داریم که می‌توانیم آن‌ها را با استفاده از یک فرآیند ساده که بر ارائه ارزش متمرکز است، به بهترین شکل هدایت کنیم.   
  
شما باید به‌طور خاص تصمیم بگیرید که «چگونه» این کارها را انجام دهید. هدف این کتاب این است که به شما کمک کند شهود خود را درباره چگونگی انجام این کارها بهبود دهید. هیچ روش «بهترین» ثابتی وجود ندارد، بنابراین حتی اگر تصمیم بگیرید برای مدتی یک کار خاص را انجام دهید، همچنان بررسی کنید که آیا واقعاً مفید است یا نه. شما نیاز خواهید داشت که چیزها را تغییر دهید.   
  
هر ایده‌ای را به‌عنوان یک روش خوب احتمالی برای شروع کار در نظر بگیرید. سپس فرآیند را به خود اختصاص دهید و ایده‌های خود را بسازید. اما آن را ساده نگه دارید.   
[گزارش خطا • بحث]   
فصل ۱۲. البته که سخت است. • ۸۹   
  
[متن از صفحه قبلی: البته که سخت است. ماهیت کاری که انجام می‌دهیم این است که همیشه در حال به چالش کشیدن محدودیت‌های خود هستیم. اما نکته این کتاب این است که رویکردی که باید برای انجام این کارهای سخت اتخاذ کنیم این است که اصرار کنیم: «نرم‌افزار را به ما نشان دهید.» این کار به ما اجازه می‌دهد که به موارد زیر دست پیدا کنیم:   
• تمرکز بر روی آنچه ارزشمند است، تا بتوانیم بهترین نتایج را بگیریم.   
• تولید نرم‌افزار واقعی به‌طور مکرر، تا بتوانیم یاد بگیریم که چه می‌خواهیم.]

================================================================================

# Section 86

## Original Text:

• Build what we want in small steps, so we can see how  
we’re doing. • Learn the planning, management, and technical skills  
that we need, so we can build the product rapidly and  
well. Every aspect of this work is complex, but we do not need a  
complex process to do the work. A complex process holds  
us back, imposing rote activity when we need to observe  
what’s going on, adjust it, and improve it. We do need increasing skill, which we can guide best by  
using a simple process aimed at delivering value. You will have to decide specifically “how” to do these things. The point of this book is to help you improve your intuition  
about how. There are no fixed “best” ways, so even if you  
decide to do some specific thing for a while, keep checking  
that it’s really helping. You will need to change things. Take every idea as, maybe, a good way to start doing things  
for a while. Then make the process your own, and build  
your own ideas. But keep it simple. report erratum • discussChapter 12. Of Course It’s Hard. • 89]  
  
It’s still difficult, Ron. Yes, but it gets better. Doing the work in this way results in rapid improvement. As a team we repeatedly decide what to do, then do it, then  
inspect the result. The problems are always hard; building  
good software is always hard. But the team becomes more  
powerful and thus progresses faster. Chapter 12. Of Course It’s Hard. • 90  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: It’s still difficult, Ron. Yes, but it gets better. Doing the work in this way results in rapid improvement. As a team we repeatedly decide what to do, then do it, then  
inspect the result. The problems are always hard; building  
good software is always hard. But the team becomes more  
powerful and thus progresses faster. Chapter 12. Of Course It’s Hard. • 90  
report erratum • discuss]  
  
When I had my knee replaced (long story), I could barely  
walk at first.

## Translated Text:

• چیزی که می‌خواهیم را در گام‌های کوچک Build کنیم تا ببینیم چطور پیش می‌رویم.   
• مهارت‌های برنامه‌ریزی، مدیریت و تکنیکی که نیاز داریم را یاد بگیریم تا بتوانیم محصول را به‌سرعت و به‌خوبی Build کنیم.   
  
هر جنبه‌ای از این کار پیچیده است، اما برای انجام آن به یک فرآیند پیچیده نیاز نداریم. یک فرآیند پیچیده ما را عقب نگه می‌دارد و فعالیت‌های تکراری را تحمیل می‌کند، در حالی که نیاز داریم مشاهده کنیم چه اتفاقی در حال رخ دادن است، آن را تنظیم کنیم و بهبود دهیم.   
  
ما به مهارت‌های بیشتری نیاز داریم که می‌توانیم با استفاده از یک فرآیند ساده که هدف آن ارائه ارزش است، آن را به بهترین شکل هدایت کنیم. شما باید به‌طور خاص تصمیم بگیرید که "چگونه" این کارها را انجام دهید. هدف این کتاب این است که به شما کمک کند شهود خود را در مورد چگونگی بهبود ببخشید.   
  
هیچ راه "بهترین" ثابتی وجود ندارد، بنابراین حتی اگر تصمیم بگیرید برای مدتی یک کار خاص را انجام دهید، مدام بررسی کنید که آیا واقعاً کمک‌کننده است یا نه. شما نیاز خواهید داشت که چیزها را تغییر دهید.   
  
هر ایده‌ای را به‌عنوان شاید یک روش خوب برای شروع کار برای مدتی در نظر بگیرید. سپس فرآیند را به خودتان اختصاص دهید و ایده‌های خودتان را بسازید. اما آن را ساده نگه دارید.   
report erratum • discuss   
Chapter 12. Of Course It’s Hard. • 89   
  
هنوز هم سخت است، Ron.   
بله، اما بهتر می‌شود. انجام کار به این روش منجر به بهبود سریع می‌شود. به‌عنوان یک تیم، ما بارها تصمیم می‌گیریم چه کاری انجام دهیم، سپس آن را انجام می‌دهیم و سپس نتیجه را بررسی می‌کنیم. مشکلات همیشه سخت هستند؛ Build کردن یک نرم‌افزار خوب همیشه سخت است. اما تیم قدرتمندتر می‌شود و بنابراین سریع‌تر پیشرفت می‌کند.   
Chapter 12. Of Course It’s Hard. • 90   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: هنوز هم سخت است، Ron.   
بله، اما بهتر می‌شود. انجام کار به این روش منجر به بهبود سریع می‌شود. به‌عنوان یک تیم، ما بارها تصمیم می‌گیریم چه کاری انجام دهیم، سپس آن را انجام می‌دهیم و سپس نتیجه را بررسی می‌کنیم. مشکلات همیشه سخت هستند؛ Build کردن یک نرم‌افزار خوب همیشه سخت است. اما تیم قدرتمندتر می‌شود و بنابراین سریع‌تر پیشرفت می‌کند.   
Chapter 12. Of Course It’s Hard. • 90   
report erratum • discuss]   
  
وقتی زانویم را تعویض کردم (داستان طولانی)، در ابتدا به‌سختی می‌توانستم راه بروم.

================================================================================

# Section 87

## Original Text:

A few meters caused me pain, and after the  
pain was gone, I was still not strong enough to walk very  
far. I kept trying, kept walking, and it got to be less painful. I was able to go much farther. Today, I can still walk far  
enough to make myself tired, but it’s no longer a matter of  
a few feet. It’s the same with this cyclical development process. At first,  
it hurts and we don’t get much done. After a little while, we  
get a lot more done, and it doesn’t hurt as much. And it’s a  
good kind of hurt, not the kind that tells you you’ve dam-  
aged something. I wish I had better news. The fact is that it’s not easy to be  
excellent. However, you can be as good as you care to be,  
simply by continuing to work and continuing to improve  
how you work. report erratum • discussChapter 12. Of Course It’s Hard. • 91  
  
[CONTEXT FROM PREVIOUS PAGE: When I had my knee replaced (long story), I could barely  
walk at first. A few meters caused me pain, and after the  
pain was gone, I was still not strong enough to walk very  
far. I kept trying, kept walking, and it got to be less painful. I was able to go much farther. Today, I can still walk far  
enough to make myself tired, but it’s no longer a matter of  
a few feet. It’s the same with this cyclical development process. At first,  
it hurts and we don’t get much done. After a little while, we  
get a lot more done, and it doesn’t hurt as much. And it’s a  
good kind of hurt, not the kind that tells you you’ve dam-  
aged something. I wish I had better news. The fact is that it’s not easy to be  
excellent. However, you can be as good as you care to be,  
simply by continuing to work and continuing to improve  
how you work. report erratum • discussChapter 12. Of Course It’s Hard. • 91]  
  
CHAPTER 13  
Not That Simple  
Ron, it’s not that simple. Do you think I’m a fool. —Early  
reviewer, paraphrased. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 13  
Not That Simple  
Ron, it’s not that simple. Do you think I’m a fool.

## Translated Text:

چند متر راه رفتن باعث درد من می‌شد، و بعد از اینکه درد از بین رفت، هنوز به اندازه کافی قوی نبودم که خیلی دور راه بروم. من به تلاش ادامه دادم، به راه رفتن ادامه دادم، و درد کمتر شد. توانستم خیلی بیشتر راه بروم. امروز، هنوز می‌توانم به اندازه‌ای راه بروم که خسته شوم، اما دیگر مسئله چند قدم نیست. این موضوع شبیه به این فرآیند توسعه چرخه‌ای است. در ابتدا، دردناک است و ما کار زیادی انجام نمی‌دهیم. بعد از مدتی، کارهای بیشتری انجام می‌دهیم و دیگر آن‌قدر دردناک نیست. و این یک نوع درد خوب است، نه آن نوعی که به شما می‌گوید چیزی را آسیب زده‌اید.   
  
ای کاش خبرهای بهتری داشتم. واقعیت این است که عالی بودن آسان نیست. با این حال، شما می‌توانید به اندازه‌ای که بخواهید خوب باشید، تنها با ادامه دادن به کار و بهبود نحوه کار خود.   
report erratum • discuss   
فصل ۱۲. البته سخت است. • ۹۱   
  
فصل ۱۳   
آن‌قدر ساده نیست   
رون، آن‌قدر ساده نیست. فکر می‌کنی من احمقم؟   
—بازبین اولیه، نقل به مضمون.   
report erratum • discuss

================================================================================

# Section 88

## Original Text:

—Early  
reviewer, paraphrased. report erratum • discuss]  
  
Of course it’s not that simple. We need to recognize that  
often it can be nearly this simple, and always, we wish it  
could be. As we wander the twisty little passages, all different, let’s  
keep in mind that we’re trying to follow this simple ideal  
path, focusing on value, managing and planning by value,  
building valuable things, and observing them to see what’s  
going on. Things will get hairy while we do that. All the complications  
of real business will show up. All the complications that we  
seem to throw in our own way will show up. And yet, it’s all about deciding what we want, guiding our-  
selves toward what we really want, by building things and  
seeing how they work out. report erratum • discussChapter 13. Not That Simple • 93  
  
[CONTEXT FROM PREVIOUS PAGE: Of course it’s not that simple. We need to recognize that  
often it can be nearly this simple, and always, we wish it  
could be. As we wander the twisty little passages, all different, let’s  
keep in mind that we’re trying to follow this simple ideal  
path, focusing on value, managing and planning by value,  
building valuable things, and observing them to see what’s  
going on. Things will get hairy while we do that. All the complications  
of real business will show up. All the complications that we  
seem to throw in our own way will show up. And yet, it’s all about deciding what we want, guiding our-  
selves toward what we really want, by building things and  
seeing how they work out. report erratum • discussChapter 13. Not That Simple • 93]  
  
Purpose, Autonomy, Mastery  
CHAPTER 14  
Creating Teams That Thrive  
Many of us, when managing something, feel the need to  
provide a lot of direction. But in this book, I ask you to  
ensure that your teams know what to do and to let them  
figure out how. How could this possibly work.

## Translated Text:

—بازبین اولیه، خلاصه‌شده. گزارش خطا • بحث کنید]  
  
البته که این‌قدر ساده نیست. باید بپذیریم که اغلب می‌تواند تقریباً این‌قدر ساده باشد، و همیشه آرزو داریم که کاش این‌قدر ساده بود. همان‌طور که در گذرگاه‌های پیچیده و کوچک، همه متفاوت، سرگردان هستیم، به یاد داشته باشیم که تلاش می‌کنیم این مسیر ساده و ایده‌آل را دنبال کنیم، بر ارزش تمرکز کنیم، با ارزش مدیریت و برنامه‌ریزی کنیم، چیزهای ارزشمند بسازیم و آن‌ها را مشاهده کنیم تا ببینیم چه اتفاقی می‌افتد. در حین انجام این کار، اوضاع پیچیده خواهد شد. تمام پیچیدگی‌های کسب‌وکار واقعی ظاهر خواهند شد. تمام پیچیدگی‌هایی که به نظر می‌رسد خودمان در مسیرمان قرار می‌دهیم، ظاهر خواهند شد. و با این حال، همه چیز درباره تصمیم‌گیری در مورد آنچه می‌خواهیم است، هدایت خودمان به سمت آنچه واقعاً می‌خواهیم، با ساختن چیزها و دیدن اینکه چگونه نتیجه می‌دهند. گزارش خطا • بحث کنید فصل ۱۳. این‌قدر ساده نیست • ۹۳  
  
[متن از صفحه قبلی: البته که این‌قدر ساده نیست. باید بپذیریم که اغلب می‌تواند تقریباً این‌قدر ساده باشد، و همیشه آرزو داریم که کاش این‌قدر ساده بود. همان‌طور که در گذرگاه‌های پیچیده و کوچک، همه متفاوت، سرگردان هستیم، به یاد داشته باشیم که تلاش می‌کنیم این مسیر ساده و ایده‌آل را دنبال کنیم، بر ارزش تمرکز کنیم، با ارزش مدیریت و برنامه‌ریزی کنیم، چیزهای ارزشمند بسازیم و آن‌ها را مشاهده کنیم تا ببینیم چه اتفاقی می‌افتد. در حین انجام این کار، اوضاع پیچیده خواهد شد. تمام پیچیدگی‌های کسب‌وکار واقعی ظاهر خواهند شد. تمام پیچیدگی‌هایی که به نظر می‌رسد خودمان در مسیرمان قرار می‌دهیم، ظاهر خواهند شد. و با این حال، همه چیز درباره تصمیم‌گیری در مورد آنچه می‌خواهیم است، هدایت خودمان به سمت آنچه واقعاً می‌خواهیم، با ساختن چیزها و دیدن اینکه چگونه نتیجه می‌دهند. گزارش خطا • بحث کنید فصل ۱۳. این‌قدر ساده نیست • ۹۳]  
  
هدف، خودمختاری، تسلط   
فصل ۱۴   
ایجاد تیم‌هایی که رشد می‌کنند   
بسیاری از ما هنگام مدیریت چیزی، احساس نیاز می‌کنیم که دستورالعمل‌های زیادی ارائه دهیم. اما در این کتاب، از شما می‌خواهم که مطمئن شوید تیم‌های شما می‌دانند چه کاری باید انجام دهند و به آن‌ها اجازه دهید که خودشان نحوه انجام آن را پیدا کنند. چگونه ممکن است این کار جواب دهد؟

================================================================================

# Section 89

## Original Text:

In his book Drive [Pin09] , Daniel Pink speaks of Purpose,  
Autonomy, and Mastery as drivers of both employee satis-  
faction and improved productivity. These ideas resonate  
with me, and make a lot of sense in the context of the Natural  
Way. Let’s explore. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Purpose, Autonomy, Mastery  
CHAPTER 14  
Creating Teams That Thrive  
Many of us, when managing something, feel the need to  
provide a lot of direction. But in this book, I ask you to  
ensure that your teams know what to do and to let them  
figure out how. How could this possibly work. In his book Drive [Pin09] , Daniel Pink speaks of Purpose,  
Autonomy, and Mastery as drivers of both employee satis-  
faction and improved productivity. These ideas resonate  
with me, and make a lot of sense in the context of the Natural  
Way. Let’s explore. report erratum • discuss]  
  
Purpose comes from the business. A dedicated business-side person is needed to guide your  
development team by identifying what’s to be done and  
what’s to be deferred. This designated person is sometimes  
called the Product Owner, or the Customer. In this book I  
use the term Product Champion, because the best results  
come when the whole team feels ownership in the product,  
with one business-side person providing vision and  
“championing” the product. The Product Champion provides purpose, both broadly and  
in detail, and keeps in daily contact with your team, ensuring  
that they understand why the effort exists, what the most  
important issues are, and how the product can best serve  
those issues. Bring concerns or problems to the team, and  
let the whole team create solutions in concert. In some teams, the Product Champion brings in defined  
solutions rather than concerns or problems. This can work  
but it’s far from ideal. Your team will be slower to gain a  
sense of purpose if they are spoon-fed solutions rather than  
being allowed to creatively solve problems.

## Translated Text:

در کتاب Drive [Pin09]، دنیل پینک درباره‌ی Purpose، Autonomy و Mastery به‌عنوان عوامل مؤثر بر رضایت کارکنان و افزایش بهره‌وری صحبت می‌کند. این ایده‌ها با من هم‌خوانی دارند و در زمینه‌ی Natural Way بسیار منطقی به نظر می‌رسند. بیایید این موضوع را بررسی کنیم.   
report erratum • discuss   
  
[متن از صفحه‌ی قبلی: Purpose، Autonomy، Mastery   
فصل ۱۴   
ایجاد تیم‌هایی که رشد می‌کنند   
بسیاری از ما هنگام مدیریت یک پروژه احساس می‌کنیم که باید دستورالعمل‌های زیادی ارائه دهیم. اما در این کتاب، از شما می‌خواهم که اطمینان حاصل کنید تیم‌های شما می‌دانند چه کاری باید انجام دهند و اجازه دهید خودشان روش انجام آن را پیدا کنند. چگونه این ممکن است؟ در کتاب Drive [Pin09]، دنیل پینک درباره‌ی Purpose، Autonomy و Mastery به‌عنوان عوامل مؤثر بر رضایت کارکنان و افزایش بهره‌وری صحبت می‌کند. این ایده‌ها با من هم‌خوانی دارند و در زمینه‌ی Natural Way بسیار منطقی به نظر می‌رسند. بیایید این موضوع را بررسی کنیم.   
report erratum • discuss]   
  
Purpose از کسب‌وکار نشأت می‌گیرد. یک فرد اختصاصی از سمت کسب‌وکار لازم است تا تیم توسعه‌ی شما را با شناسایی اینکه چه کاری باید انجام شود و چه چیزی باید به تعویق بیفتد، هدایت کند. این فرد تعیین‌شده گاهی اوقات Product Owner یا Customer نامیده می‌شود. در این کتاب، من از اصطلاح Product Champion استفاده می‌کنم، زیرا بهترین نتایج زمانی حاصل می‌شود که کل تیم احساس مالکیت نسبت به محصول داشته باشد و یک فرد از سمت کسب‌وکار، چشم‌انداز و «حمایت» از محصول را فراهم کند.   
  
Product Champion هدف را هم به‌صورت کلی و هم به‌صورت جزئی ارائه می‌دهد و به‌صورت روزانه با تیم شما در ارتباط است تا اطمینان حاصل کند که آن‌ها می‌دانند چرا این تلاش وجود دارد، مهم‌ترین مسائل کدام‌اند و چگونه محصول می‌تواند به بهترین شکل به این مسائل خدمت کند. نگرانی‌ها یا مشکلات را به تیم بیاورید و اجازه دهید کل تیم به‌صورت مشترک راه‌حل‌هایی ایجاد کند.   
  
در برخی تیم‌ها، Product Champion به‌جای نگرانی‌ها یا مشکلات، راه‌حل‌های تعریف‌شده را ارائه می‌دهد. این روش ممکن است کار کند، اما به هیچ وجه ایده‌آل نیست. تیم شما درک هدف را کندتر به دست خواهد آورد اگر به جای اینکه اجازه داده شود به‌صورت خلاقانه مشکلات را حل کنند، راه‌حل‌ها به آن‌ها دیکته شود.

================================================================================

# Section 90

## Original Text:

When the whole team works together on problems, the  
Product Champion learns better what she wants and how  
to express it. The whole team improves, and so does the  
Champion. This is the sweet spot. report erratum • discussChapter 14. Creating Teams That Thrive • 95  
  
[CONTEXT FROM PREVIOUS PAGE: Purpose comes from the business. A dedicated business-side person is needed to guide your  
development team by identifying what’s to be done and  
what’s to be deferred. This designated person is sometimes  
called the Product Owner, or the Customer. In this book I  
use the term Product Champion, because the best results  
come when the whole team feels ownership in the product,  
with one business-side person providing vision and  
“championing” the product. The Product Champion provides purpose, both broadly and  
in detail, and keeps in daily contact with your team, ensuring  
that they understand why the effort exists, what the most  
important issues are, and how the product can best serve  
those issues. Bring concerns or problems to the team, and  
let the whole team create solutions in concert. In some teams, the Product Champion brings in defined  
solutions rather than concerns or problems. This can work  
but it’s far from ideal. Your team will be slower to gain a  
sense of purpose if they are spoon-fed solutions rather than  
being allowed to creatively solve problems. When the whole team works together on problems, the  
Product Champion learns better what she wants and how  
to express it. The whole team improves, and so does the  
Champion. This is the sweet spot. report erratum • discussChapter 14. Creating Teams That Thrive • 95]  
  
Autonomy gives the team responsibility. The Product Champion shows what problem to solve ; the team  
decides how to solve it. With true autonomy, no one double-  
checks the team: they decide, they build, we all see the  
results. Everyone learns. As a team, in each two-week iteration, let’s confer with our  
Product Champion on what needs to be done.

## Translated Text:

وقتی کل تیم با هم روی مشکلات کار می‌کنند، Product Champion بهتر متوجه می‌شود که چه می‌خواهد و چگونه آن را بیان کند. کل تیم پیشرفت می‌کند و Product Champion نیز همین‌طور. این همان نقطه‌ی ایده‌آل است.   
report erratum • discuss   
Chapter 14. Creating Teams That Thrive • 95   
  
[متن از صفحه‌ی قبلی: هدف از کسب‌وکار می‌آید. یک فرد اختصاصی از سمت کسب‌وکار لازم است تا تیم توسعه‌ی شما را با شناسایی آنچه باید انجام شود و آنچه باید به تعویق بیفتد، هدایت کند. این فرد مشخص گاهی Product Owner یا Customer نامیده می‌شود. در این کتاب من از اصطلاح Product Champion استفاده می‌کنم، زیرا بهترین نتایج زمانی حاصل می‌شود که کل تیم احساس مالکیت نسبت به محصول داشته باشد، با یک فرد از سمت کسب‌وکار که چشم‌انداز ارائه می‌دهد و محصول را "champion" می‌کند.   
Product Champion هدف را هم به صورت کلی و هم جزئی فراهم می‌کند و در ارتباط روزانه با تیم شما اطمینان حاصل می‌کند که آن‌ها می‌دانند چرا این تلاش وجود دارد، مهم‌ترین مسائل کدامند و چگونه محصول می‌تواند به بهترین شکل به آن مسائل خدمت کند. نگرانی‌ها یا مشکلات را به تیم بیاورید و اجازه دهید کل تیم به صورت هماهنگ راه‌حل‌هایی ایجاد کنند.   
در برخی تیم‌ها، Product Champion راه‌حل‌های مشخصی به جای نگرانی‌ها یا مشکلات ارائه می‌دهد. این روش می‌تواند کار کند اما به هیچ وجه ایده‌آل نیست. تیم شما کندتر حس هدفمندی پیدا می‌کند اگر به جای اینکه اجازه داده شود خلاقانه مشکلات را حل کنند، راه‌حل‌ها به آن‌ها داده شود.   
وقتی کل تیم با هم روی مشکلات کار می‌کنند، Product Champion بهتر متوجه می‌شود که چه می‌خواهد و چگونه آن را بیان کند. کل تیم پیشرفت می‌کند و Product Champion نیز همین‌طور. این همان نقطه‌ی ایده‌آل است.   
report erratum • discuss   
Chapter 14. Creating Teams That Thrive • 95]   
  
استقلال به تیم مسئولیت می‌دهد. Product Champion نشان می‌دهد که چه مشکلی باید حل شود؛ تیم تصمیم می‌گیرد که چگونه آن را حل کند. با استقلال واقعی، هیچ‌کس تیم را دوباره بررسی نمی‌کند: آن‌ها تصمیم می‌گیرند، آن‌ها می‌سازند، و ما همه نتایج را می‌بینیم. همه یاد می‌گیرند.   
به عنوان یک تیم، در هر iteration دو هفته‌ای، بیایید با Product Champion خود مشورت کنیم که چه کاری باید انجام شود.

================================================================================

# Section 91

## Original Text:

Select a  
quantity of work that we can deliver in good order, figure  
out how to do it, then self-organize to get it done. Then,  
“Show us the software. ”  
As our team becomes self-organizing, we’ll have more  
autonomy, bring more creativity to the problem, and become  
more productive. A self-organizing team with a common understanding of its  
purpose: this is the sweet spot. Chapter 14. Creating Teams That Thrive • 96  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Autonomy gives the team responsibility. The Product Champion shows what problem to solve ; the team  
decides how to solve it. With true autonomy, no one double-  
checks the team: they decide, they build, we all see the  
results. Everyone learns. As a team, in each two-week iteration, let’s confer with our  
Product Champion on what needs to be done. Select a  
quantity of work that we can deliver in good order, figure  
out how to do it, then self-organize to get it done. Then,  
“Show us the software. ”  
As our team becomes self-organizing, we’ll have more  
autonomy, bring more creativity to the problem, and become  
more productive. A self-organizing team with a common understanding of its  
purpose: this is the sweet spot. Chapter 14. Creating Teams That Thrive • 96  
report erratum • discuss]  
  
Mastery comes from the iterative process. In each iteration, our team works to produce a “done”  
increment of software. At first we will have difficulty, but  
in each iteration, we meet and look back on how things went  
and determine how to do better. We move toward mastery. In each iteration, let’s improve our definition of “done,” the  
criteria by which an increment will be considered to be good  
enough. Make it more mature and stringent. Move as a team  
toward mastery. Each iteration constitutes practice in producing “done”  
software. It ends with “Show us the software,” enabling  
everyone to see how good it is, and to contribute to working  
out how to make it better. Again, they move toward mastery.

## Translated Text:

مقداری از کار را انتخاب کنید که بتوانیم به‌صورت منظم تحویل دهیم، روش انجام آن را مشخص کنیم، سپس خودمان را سازماندهی کنیم تا آن را انجام دهیم. سپس، "نرم‌افزار را به ما نشان دهید."  
  
وقتی تیم ما خودسازمان‌ده می‌شود، استقلال بیشتری خواهیم داشت، خلاقیت بیشتری به مسئله خواهیم آورد و بهره‌وری بیشتری خواهیم داشت. تیمی که خودسازمان‌ده است و درک مشترکی از هدف خود دارد: این همان نقطه طلایی است.   
فصل ۱۴. ایجاد تیم‌هایی که رشد می‌کنند • ۹۶   
گزارش خطا • بحث  
  
[متن از صفحه قبلی: استقلال به تیم مسئولیت می‌دهد. Product Champion نشان می‌دهد که چه مشکلی باید حل شود؛ تیم تصمیم می‌گیرد چگونه آن را حل کند. با استقلال واقعی، هیچ‌کس تیم را دوباره بررسی نمی‌کند: آن‌ها تصمیم می‌گیرند، می‌سازند، و همه ما نتایج را می‌بینیم. همه یاد می‌گیرند. به‌عنوان یک تیم، در هر دوره دو هفته‌ای، بیایید با Product Champion خود مشورت کنیم که چه چیزی باید انجام شود. مقداری از کار را انتخاب کنید که بتوانیم به‌صورت منظم تحویل دهیم، روش انجام آن را مشخص کنیم، سپس خودمان را سازماندهی کنیم تا آن را انجام دهیم. سپس، "نرم‌افزار را به ما نشان دهید."  
  
وقتی تیم ما خودسازمان‌ده می‌شود، استقلال بیشتری خواهیم داشت، خلاقیت بیشتری به مسئله خواهیم آورد و بهره‌وری بیشتری خواهیم داشت. تیمی که خودسازمان‌ده است و درک مشترکی از هدف خود دارد: این همان نقطه طلایی است.   
فصل ۱۴. ایجاد تیم‌هایی که رشد می‌کنند • ۹۶   
گزارش خطا • بحث]  
  
تسلط از فرآیند تکراری به دست می‌آید. در هر Iteration، تیم ما تلاش می‌کند یک Increment "انجام‌شده" از نرم‌افزار تولید کند. در ابتدا دشواری خواهیم داشت، اما در هر Iteration، دور هم جمع می‌شویم و به گذشته نگاه می‌کنیم که چگونه پیش رفت و تصمیم می‌گیریم چگونه بهتر عمل کنیم. ما به سمت تسلط حرکت می‌کنیم.   
  
در هر Iteration، تعریف خود از "انجام‌شده" را بهبود دهیم، معیارهایی که بر اساس آن یک Increment به اندازه کافی خوب در نظر گرفته می‌شود. آن را بالغ‌تر و سخت‌گیرانه‌تر کنید. به‌عنوان یک تیم به سمت تسلط حرکت کنید. هر Iteration تمرینی است برای تولید نرم‌افزار "انجام‌شده". این فرآیند با "نرم‌افزار را به ما نشان دهید" به پایان می‌رسد، که به همه اجازه می‌دهد ببینند چقدر خوب است و در یافتن راه‌هایی برای بهتر کردن آن مشارکت کنند. باز هم، آن‌ها به سمت تسلط حرکت می‌کنند.

================================================================================

# Section 92

## Original Text:

The mantra of Scrum, the most popular Agile method, is  
“Inspect and Adapt. ” As a team, we observe what we  
accomplish, note what’s holding us back, and improve the  
situation. As the team improves, we all move toward mas-  
tery. Summing Up  
A masterful self-organizing team with a common understand-  
ing of its purpose. This is the sweet spot. report erratum • discussChapter 14. Creating Teams That Thrive • 97  
  
[CONTEXT FROM PREVIOUS PAGE: Mastery comes from the iterative process. In each iteration, our team works to produce a “done”  
increment of software. At first we will have difficulty, but  
in each iteration, we meet and look back on how things went  
and determine how to do better. We move toward mastery. In each iteration, let’s improve our definition of “done,” the  
criteria by which an increment will be considered to be good  
enough. Make it more mature and stringent. Move as a team  
toward mastery. Each iteration constitutes practice in producing “done”  
software. It ends with “Show us the software,” enabling  
everyone to see how good it is, and to contribute to working  
out how to make it better. Again, they move toward mastery. The mantra of Scrum, the most popular Agile method, is  
“Inspect and Adapt. ” As a team, we observe what we  
accomplish, note what’s holding us back, and improve the  
situation. As the team improves, we all move toward mas-  
tery. Summing Up  
A masterful self-organizing team with a common understand-  
ing of its purpose. This is the sweet spot. report erratum • discussChapter 14. Creating Teams That Thrive • 97]  
  
CHAPTER 15  
The “Five-Card Method”  
for Initial Forecasting  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 15  
The “Five-Card Method”  
for Initial Forecasting  
report erratum • discuss]  
  
If we simply must do initial large-scale forecasting, we can  
use the “Five-Card Method” to get just enough detail but  
not too much.

## Translated Text:

مانترای Scrum، محبوب‌ترین روش Agile، این است:   
«Inspect and Adapt.» به‌عنوان یک تیم، آنچه را که به دست می‌آوریم مشاهده می‌کنیم، مواردی که ما را عقب نگه می‌دارند یادداشت می‌کنیم و وضعیت را بهبود می‌بخشیم. با بهبود تیم، همه ما به سمت تسلط حرکت می‌کنیم.   
  
\*\*جمع‌بندی\*\*   
یک تیم خودسازمان‌دهی‌شده‌ی ماهر با درک مشترک از هدف خود. این نقطه‌ی ایده‌آل است.   
\*report erratum • discuss\*   
\*Chapter 14. Creating Teams That Thrive • 97\*   
  
[متن از صفحه‌ی قبلی: تسلط از فرآیند تکراری به دست می‌آید. در هر iteration، تیم ما تلاش می‌کند یک increment از نرم‌افزار «done» تولید کند. در ابتدا با دشواری مواجه خواهیم شد، اما در هر iteration، ملاقات می‌کنیم و به گذشته نگاه می‌کنیم که چگونه پیش رفت و تصمیم می‌گیریم چگونه بهتر عمل کنیم. ما به سمت تسلط حرکت می‌کنیم. در هر iteration، تعریف خود از «done» را بهبود دهیم، معیارهایی که بر اساس آن یک increment به اندازه کافی خوب در نظر گرفته می‌شود. آن را بالغ‌تر و سخت‌گیرانه‌تر کنید. به‌عنوان یک تیم به سمت تسلط حرکت کنید. هر iteration تمرینی برای تولید نرم‌افزار «done» است. این فرآیند با «نرم‌افزار را به ما نشان دهید» پایان می‌یابد، که به همه اجازه می‌دهد ببینند چقدر خوب است و در یافتن راه‌هایی برای بهتر کردن آن مشارکت کنند. باز هم، آن‌ها به سمت تسلط حرکت می‌کنند. مانترای Scrum، محبوب‌ترین روش Agile، این است:   
«Inspect and Adapt.» به‌عنوان یک تیم، آنچه را که به دست می‌آوریم مشاهده می‌کنیم، مواردی که ما را عقب نگه می‌دارند یادداشت می‌کنیم و وضعیت را بهبود می‌بخشیم. با بهبود تیم، همه ما به سمت تسلط حرکت می‌کنیم.   
  
\*\*جمع‌بندی\*\*   
یک تیم خودسازمان‌دهی‌شده‌ی ماهر با درک مشترک از هدف خود. این نقطه‌ی ایده‌آل است.   
\*report erratum • discuss\*   
\*Chapter 14. Creating Teams That Thrive • 97\*]   
  
\*\*فصل 15\*\*   
\*\*روش «Five-Card» برای پیش‌بینی اولیه\*\*   
\*report erratum • discuss\*   
  
[متن از صفحه‌ی قبلی: \*\*فصل 15\*\*   
\*\*روش «Five-Card» برای پیش‌بینی اولیه\*\*   
\*report erratum • discuss\*]   
  
اگر مجبور باشیم پیش‌بینی اولیه در مقیاس بزرگ انجام دهیم، می‌توانیم از روش «Five-Card» استفاده کنیم تا جزئیات کافی، اما نه بیش از حد، به دست آوریم.

================================================================================

# Section 93

## Original Text:

We take our grand vision and break each  
component down into elements small enough for our team  
members to feel that they know what it means and that they  
can build it in a week:  
• Consider the three to five most important “epics”—the  
big things that describe the product you have in mind. On a card, describe each one in a single sentence. • Break down each of these into three to five smaller cards. Make each one more specific and clear—and, of course,  
smaller. Make sure that each item makes business sense:  
it should be a “feature,” not some technical idea or  
widget. • Repeat the breakdown until each feature seems to be  
about the same size. What’s a good size. Something the  
team members think they can build in about a week. That’s small enough. \* As you break things down, keep  
a close eye out for things that bear high value and things  
that don’t. Set low-value things aside. Remember that  
you’re going to make the date, not by whipping the  
ponies harder (as we’ve discussed in Chapter 17, Whip  
the Ponies Harder , on page 112 ), but by managing carefully  
what we do and what we defer until later. What are the top-level bullet-point features of your product. What are the bullet points for the detailed slide for each of  
them. What more do you need to know to decide to go  
ahead. report erratum • discussChapter 15. The “Five-Card Method”for Initial Forecasting • 99  
  
[CONTEXT FROM PREVIOUS PAGE: If we simply must do initial large-scale forecasting, we can  
use the “Five-Card Method” to get just enough detail but  
not too much. We take our grand vision and break each  
component down into elements small enough for our team  
members to feel that they know what it means and that they  
can build it in a week:  
• Consider the three to five most important “epics”—the  
big things that describe the product you have in mind. On a card, describe each one in a single sentence. • Break down each of these into three to five smaller cards.

## Translated Text:

ما چشم‌انداز بزرگ خود را گرفته و هر مؤلفه را به عناصری کوچک‌تر تقسیم می‌کنیم، به اندازه‌ای که اعضای تیم احساس کنند می‌دانند منظور چیست و می‌توانند آن را در یک هفته Build کنند:   
• سه تا پنج “epic” مهم را در نظر بگیرید—موارد بزرگی که محصول مورد نظر شما را توصیف می‌کنند. هر کدام را روی یک کارت در یک جمله توصیف کنید.   
• هر یک از این موارد را به سه تا پنج کارت کوچک‌تر تقسیم کنید. هر کدام را مشخص‌تر و واضح‌تر کنید—و البته کوچک‌تر. مطمئن شوید که هر آیتم از نظر کسب‌وکار منطقی است: باید یک “feature” باشد، نه یک ایده فنی یا ابزارک.   
• این تقسیم‌بندی را تکرار کنید تا هر feature تقریباً به یک اندازه برسد. اندازه مناسب چیست؟ چیزی که اعضای تیم فکر می‌کنند می‌توانند در حدود یک هفته Build کنند. این اندازه به اندازه کافی کوچک است.   
\* هنگام تقسیم‌بندی، به دقت مراقب مواردی باشید که ارزش بالایی دارند و مواردی که ندارند. موارد کم‌ارزش را کنار بگذارید. به یاد داشته باشید که شما قرار است زمان‌بندی را رعایت کنید، نه با فشار بیشتر (همان‌طور که در فصل 17، Whip the Ponies Harder، در صفحه 112 بحث کردیم)، بلکه با مدیریت دقیق آنچه انجام می‌دهیم و آنچه به بعد موکول می‌کنیم.   
ویژگی‌های اصلی محصول شما چه هستند؟ برای هر یک از آن‌ها، نقاط کلیدی اسلاید جزئیات چیست؟ چه چیز دیگری لازم است بدانید تا تصمیم بگیرید که ادامه دهید؟   
  
report erratum • discuss   
فصل 15. روش “Five-Card” برای پیش‌بینی اولیه • 99

================================================================================

# Section 94

## Original Text:

Make each one more specific and clear—and, of course,  
smaller. Make sure that each item makes business sense:  
it should be a “feature,” not some technical idea or  
widget. • Repeat the breakdown until each feature seems to be  
about the same size. What’s a good size. Something the  
team members think they can build in about a week. That’s small enough. \* As you break things down, keep  
a close eye out for things that bear high value and things  
that don’t. Set low-value things aside. Remember that  
you’re going to make the date, not by whipping the  
ponies harder (as we’ve discussed in Chapter 17, Whip  
the Ponies Harder , on page 112 ), but by managing carefully  
what we do and what we defer until later. What are the top-level bullet-point features of your product. What are the bullet points for the detailed slide for each of  
them. What more do you need to know to decide to go  
ahead. report erratum • discussChapter 15. The “Five-Card Method”for Initial Forecasting • 99]  
  
CHAPTER 16  
Managing Natural  
Software Development  
Someone said to me, “Ron, in the real world there are managers. They’re there for a reason. In your model you don’t say much about  
them. What about managers. ”  
Well, I like to think that we’re in the business of creating the  
“real world,” not putting up with it. Management does need  
to happen. It turns out, it doesn’t need to be “management  
as usual. ” Let’s explore. When we work in the Natural Way, much of the “managing”  
is done inside the team. The Product Champion sets the  
vision and, working with stakeholders and team members,  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 16  
Managing Natural  
Software Development  
Someone said to me, “Ron, in the real world there are managers. They’re there for a reason. In your model you don’t say much about  
them. What about managers. ”  
Well, I like to think that we’re in the business of creating the  
“real world,” not putting up with it. Management does need  
to happen.

## Translated Text:

هر کدام را مشخص‌تر و واضح‌تر کنید—و البته، کوچک‌تر. اطمینان حاصل کنید که هر آیتم از نظر کسب‌وکار منطقی باشد: باید یک "Feature" باشد، نه یک ایده یا ابزار فنی.   
• این تقسیم‌بندی را تکرار کنید تا هر Feature به نظر به یک اندازه برسد. اندازه مناسب چیست؟ چیزی که اعضای تیم فکر می‌کنند می‌توانند در حدود یک هفته Build کنند. این به اندازه کافی کوچک است.   
\* هنگام تقسیم‌بندی، با دقت به مواردی که ارزش بالایی دارند و مواردی که ندارند توجه کنید. موارد با ارزش پایین را کنار بگذارید. به یاد داشته باشید که شما قرار است به تاریخ برسید، نه با فشار آوردن بیشتر (همان‌طور که در فصل 17، \*\*Whip the Ponies Harder\*\*، صفحه 112 بحث کردیم)، بلکه با مدیریت دقیق آنچه انجام می‌دهیم و آنچه به بعد موکول می‌کنیم.   
ویژگی‌های اصلی محصول شما چیست؟ نکات کلیدی برای اسلاید جزئیات هر کدام چیست؟ برای تصمیم‌گیری برای ادامه چه چیز دیگری باید بدانید؟   
  
\*\*report erratum • discuss\*\*   
\*\*Chapter 15. The “Five-Card Method” for Initial Forecasting • 99\*\*   
  
---  
  
\*\*CHAPTER 16   
مدیریت توسعه نرم‌افزار طبیعی\*\*   
  
کسی به من گفت: "ران، در دنیای واقعی مدیرانی وجود دارند. آنها به دلیلی آنجا هستند. در مدل شما زیاد درباره آنها صحبت نمی‌کنید. مدیران چه می‌شوند؟"   
خب، من دوست دارم فکر کنم که ما در کار خلق "دنیای واقعی" هستیم، نه تحمل کردن آن. مدیریت باید انجام شود. اما مشخص می‌شود که نیازی نیست به شکل "مدیریت معمول" باشد. بیایید بررسی کنیم.   
  
وقتی ما به روش طبیعی کار می‌کنیم، بخش زیادی از "مدیریت" در داخل تیم انجام می‌شود. \*\*Product Champion\*\* چشم‌انداز را تعیین می‌کند و با همکاری ذینفعان و اعضای تیم...   
  
\*\*report erratum • discuss\*\*

================================================================================

# Section 95

## Original Text:

It turns out, it doesn’t need to be “management  
as usual. ” Let’s explore. When we work in the Natural Way, much of the “managing”  
is done inside the team. The Product Champion sets the  
vision and, working with stakeholders and team members,  
report erratum • discuss]  
  
sets the priorities. The team “shows us the software” every  
couple of weeks so that everyone can tell how things are  
going. This lets us coordinate with stakeholders outside the  
team and allows us to easily judge whether the team needs  
help of some kind. The team is fully cross-functional and has all the skills and  
capacity needed to deliver each increment of software. Team  
members do their own testing, their own documentation,  
everything. The closer we get to this ideal, the less coordina-  
tion we need. The team is self-organizing as well. Team members decide  
how to parcel out the work, and then ensure that it gets done  
well. When teams are working as described here, they  
require very little ongoing management. Management work still needs to be done, of course. Staffing  
selections should be made by the team to the highest degree  
possible. However, staffing decisions , whether to hire or fire,  
need to be made by management. Budget proposals might  
be made by the team together with the Product Champion. Budget decisions, within and across projects, are a manage-  
ment function. Managers sometimes wonder whether they’ll work them-  
selves out of a job by letting go of so much of their work. But think about it. In management, this is called “delega-  
tion. ” If a manager creates an effective team, building a  
valuable product, visibly and smoothly, that manager is  
going to get more to do, not less. Build another team, and  
another. Let’s consider Peter Drucker’s components of management:  
planning, organizing, staffing, directing, and controlling. We’ll look at each of these, and what these ideas mean for  
managers, in the context of the Natural Way. report erratum • discussChapter 16.

## Translated Text:

مشخص شده است که نیازی نیست که مدیریت به شیوه‌ی معمول انجام شود. بیایید بررسی کنیم. زمانی که ما به روش طبیعی کار می‌کنیم، بخش زیادی از "مدیریت" در داخل تیم انجام می‌شود. Product Champion چشم‌انداز را تعیین می‌کند و با همکاری ذینفعان و اعضای تیم، اولویت‌ها را مشخص می‌کند.  
  
تیم هر چند هفته یک بار "نرم‌افزار را به ما نشان می‌دهد" تا همه بتوانند بفهمند اوضاع چگونه پیش می‌رود. این کار به ما اجازه می‌دهد تا با ذینفعان خارج از تیم هماهنگ شویم و به راحتی قضاوت کنیم که آیا تیم به نوعی کمک نیاز دارد یا خیر. تیم کاملاً چندمهارتی است و تمام مهارت‌ها و ظرفیت‌های لازم برای ارائه هر بخش از نرم‌افزار را داراست. اعضای تیم خودشان تست، مستندسازی و همه چیز را انجام می‌دهند. هرچه بیشتر به این ایده‌آل نزدیک شویم، نیاز به هماهنگی کمتری خواهیم داشت.  
  
تیم همچنین خودسازمان‌ده است. اعضای تیم تصمیم می‌گیرند که چگونه کار را تقسیم کنند و سپس اطمینان حاصل می‌کنند که کار به خوبی انجام شود. وقتی تیم‌ها به این شیوه کار می‌کنند، به مدیریت مداوم بسیار کمی نیاز دارند.  
  
البته کارهای مدیریتی همچنان باید انجام شوند. انتخاب‌های مربوط به نیروی انسانی باید تا حد ممکن توسط تیم انجام شود. با این حال، تصمیمات مربوط به نیروی انسانی، چه برای استخدام و چه برای اخراج، باید توسط مدیریت گرفته شود. پیشنهادات بودجه ممکن است توسط تیم همراه با Product Champion ارائه شود. تصمیمات بودجه، چه درون پروژه‌ها و چه بین پروژه‌ها، یک وظیفه مدیریتی است.  
  
مدیران گاهی تعجب می‌کنند که آیا با واگذاری این حجم از کار، خود را از شغلشان کنار می‌گذارند. اما به این موضوع فکر کنید. در مدیریت، این کار "واگذاری" نامیده می‌شود. اگر یک مدیر تیمی مؤثر ایجاد کند که محصولی ارزشمند را به‌صورت واضح و روان تولید کند، آن مدیر کار بیشتری برای انجام خواهد داشت، نه کمتر. یک تیم دیگر بسازید، و سپس یکی دیگر.  
  
بیایید به اجزای مدیریت از دیدگاه Peter Drucker نگاه کنیم: برنامه‌ریزی، سازمان‌دهی، تأمین نیروی انسانی، هدایت و کنترل. ما هر یک از این موارد و معنای آن‌ها برای مدیران را در زمینه روش طبیعی بررسی خواهیم کرد.

================================================================================

# Section 96

## Original Text:

Managing Natural Software Development • 101  
  
[CONTEXT FROM PREVIOUS PAGE: sets the priorities. The team “shows us the software” every  
couple of weeks so that everyone can tell how things are  
going. This lets us coordinate with stakeholders outside the  
team and allows us to easily judge whether the team needs  
help of some kind. The team is fully cross-functional and has all the skills and  
capacity needed to deliver each increment of software. Team  
members do their own testing, their own documentation,  
everything. The closer we get to this ideal, the less coordina-  
tion we need. The team is self-organizing as well. Team members decide  
how to parcel out the work, and then ensure that it gets done  
well. When teams are working as described here, they  
require very little ongoing management. Management work still needs to be done, of course. Staffing  
selections should be made by the team to the highest degree  
possible. However, staffing decisions , whether to hire or fire,  
need to be made by management. Budget proposals might  
be made by the team together with the Product Champion. Budget decisions, within and across projects, are a manage-  
ment function. Managers sometimes wonder whether they’ll work them-  
selves out of a job by letting go of so much of their work. But think about it. In management, this is called “delega-  
tion. ” If a manager creates an effective team, building a  
valuable product, visibly and smoothly, that manager is  
going to get more to do, not less. Build another team, and  
another. Let’s consider Peter Drucker’s components of management:  
planning, organizing, staffing, directing, and controlling. We’ll look at each of these, and what these ideas mean for  
managers, in the context of the Natural Way. report erratum • discussChapter 16. Managing Natural Software Development • 101]  
  
Ron, getting beyond “management as usual” sounds fine, but I  
work for a company that has long-term goals and I can assure you  
we have a need for longer-term planning.

## Translated Text:

مدیریت توسعه نرم‌افزار طبیعی • ۱۰۱  
  
[متن از صفحه قبلی: اولویت‌ها را تعیین می‌کند. تیم هر چند هفته یک بار "نرم‌افزار را به ما نشان می‌دهد" تا همه بتوانند بفهمند اوضاع چگونه پیش می‌رود. این به ما اجازه می‌دهد با ذینفعان خارج از تیم هماهنگ شویم و به راحتی قضاوت کنیم که آیا تیم به نوعی کمک نیاز دارد یا خیر. تیم کاملاً چندمهارتی است و تمام مهارت‌ها و ظرفیت‌های لازم برای تحویل هر Increment از نرم‌افزار را دارد. اعضای تیم تست‌های خودشان را انجام می‌دهند، مستندات خودشان را می‌نویسند، همه چیز. هرچه بیشتر به این ایده‌آل نزدیک شویم، نیاز به هماهنگی کمتری خواهیم داشت. تیم همچنین خودسازمان‌ده است. اعضای تیم تصمیم می‌گیرند چگونه کارها را تقسیم کنند و سپس اطمینان حاصل می‌کنند که کارها به خوبی انجام می‌شود. وقتی تیم‌ها به این شکل کار می‌کنند، به مدیریت مداوم بسیار کمی نیاز دارند. البته کار مدیریتی همچنان باید انجام شود. انتخاب‌های مربوط به نیروی انسانی باید تا حد ممکن توسط تیم انجام شود. با این حال، تصمیمات مربوط به نیروی انسانی، چه استخدام و چه اخراج، باید توسط مدیریت گرفته شود. پیشنهادات بودجه ممکن است توسط تیم همراه با Product Champion ارائه شود. تصمیمات بودجه، چه درون پروژه‌ها و چه بین پروژه‌ها، یک وظیفه مدیریتی است. مدیران گاهی تعجب می‌کنند که آیا با واگذاری این حجم از کارها، خودشان را از شغلشان کنار می‌گذارند. اما به این فکر کنید. در مدیریت، این "واگذاری" نامیده می‌شود. اگر یک مدیر تیمی مؤثر ایجاد کند که محصولی ارزشمند را به طور واضح و روان بسازد، آن مدیر کار بیشتری برای انجام خواهد داشت، نه کمتر. یک تیم دیگر بسازید، و یکی دیگر. بیایید به اجزای مدیریت از نظر پیتر دراکر نگاه کنیم: برنامه‌ریزی، سازمان‌دهی، نیروی انسانی، هدایت و کنترل. ما به هر یک از این موارد و معنای این ایده‌ها برای مدیران در زمینه روش طبیعی نگاه خواهیم کرد. report erratum • discussChapter 16. Managing Natural Software Development • 101]  
  
رون، فراتر رفتن از "مدیریت به روش معمول" خوب به نظر می‌رسد، اما من برای شرکتی کار می‌کنم که اهداف بلندمدت دارد و می‌توانم به شما اطمینان دهم که ما به برنامه‌ریزی بلندمدت نیاز داریم.

================================================================================

# Section 97

## Original Text:

How do you suggest we  
approach that. Somewhere “upward” in the company, management decides  
what the business of the organization is. They decide in  
general terms what will be done, and who will do it. This  
begins with an overall planning activity, selecting the suite  
of problems to be solved and opportunities to be undertaken,  
and determining the size of each of those efforts in terms of  
time, staffing, and budget. From the viewpoint of the Natural Way, long-term planning  
can be done in nearly any way at all. People from all disci-  
plines should be involved in this activity: management  
people, finance people, product people, technical people. Probably the most important thing to do from the top. Limit the number of products and programs you take on. Get those done, and then add more. Working on lots of  
things in parallel just slows everything down. Chapter 16. Managing Natural Software Development • 102  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Ron, getting beyond “management as usual” sounds fine, but I  
work for a company that has long-term goals and I can assure you  
we have a need for longer-term planning. How do you suggest we  
approach that. Somewhere “upward” in the company, management decides  
what the business of the organization is. They decide in  
general terms what will be done, and who will do it. This  
begins with an overall planning activity, selecting the suite  
of problems to be solved and opportunities to be undertaken,  
and determining the size of each of those efforts in terms of  
time, staffing, and budget. From the viewpoint of the Natural Way, long-term planning  
can be done in nearly any way at all. People from all disci-  
plines should be involved in this activity: management  
people, finance people, product people, technical people. Probably the most important thing to do from the top. Limit the number of products and programs you take on. Get those done, and then add more.

## Translated Text:

چگونه پیشنهاد می‌کنید که به این موضوع نزدیک شویم؟ در جایی "بالاتر" در شرکت، مدیریت تصمیم می‌گیرد که کسب‌وکار سازمان چیست. آنها به طور کلی تصمیم می‌گیرند که چه کاری انجام شود و چه کسی آن را انجام دهد. این فرآیند با یک فعالیت برنامه‌ریزی کلی آغاز می‌شود، انتخاب مجموعه‌ای از مشکلاتی که باید حل شوند و فرصت‌هایی که باید دنبال شوند، و تعیین اندازه هر یک از این تلاش‌ها از نظر زمان، نیروی انسانی و بودجه. از دیدگاه روش طبیعی، برنامه‌ریزی بلندمدت تقریباً به هر شکلی که باشد قابل انجام است. افراد از تمام رشته‌ها باید در این فعالیت مشارکت داشته باشند: افراد مدیریت، افراد مالی، افراد محصول، افراد فنی. احتمالاً مهم‌ترین کاری که باید از بالا انجام شود این است که تعداد محصولات و برنامه‌هایی که می‌پذیرید را محدود کنید. آنها را به اتمام برسانید و سپس موارد بیشتری اضافه کنید. کار کردن روی تعداد زیادی از موارد به صورت موازی فقط همه چیز را کند می‌کند.   
  
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 102   
گزارش خطا • بحث

================================================================================

# Section 98

## Original Text:

Working on lots of  
things in parallel just slows everything down. Chapter 16. Managing Natural Software Development • 102  
report erratum • discuss]  
  
What about planning at the scale of a few months or a year. A medium- or long-term effort starts out with fairly general  
goals. Following the Natural Way, we build high-value fea-  
tures early, and managers can keep an eye on where we are  
without obsessing over details. As high-level planners and  
managers, let’s be clear on the general capabilities needed  
in our large project, and let’s ask our Product Champions  
to “show us the software” that provides those capabilities. Let’s help our Champions cover all the key aspects of the  
product first and then fill in the less important aspects as  
time and money permit. report erratum • discussChapter 16. Managing Natural Software Development • 103  
  
[CONTEXT FROM PREVIOUS PAGE: What about planning at the scale of a few months or a year. A medium- or long-term effort starts out with fairly general  
goals. Following the Natural Way, we build high-value fea-  
tures early, and managers can keep an eye on where we are  
without obsessing over details. As high-level planners and  
managers, let’s be clear on the general capabilities needed  
in our large project, and let’s ask our Product Champions  
to “show us the software” that provides those capabilities. Let’s help our Champions cover all the key aspects of the  
product first and then fill in the less important aspects as  
time and money permit. report erratum • discussChapter 16. Managing Natural Software Development • 103]  
  
What about short-term planning, day to day, week to week. In the Natural Way, short-term planning is continuous at  
the level of development. Valuable features are given prior-  
ities, and they get done in the order of highest value first. Week in and week out, value grows visibly. The plan is  
revised every couple of weeks and we can all see how things  
are going. We do this very simply.

## Translated Text:

انجام دادن کارهای زیاد به صورت همزمان فقط همه چیز را کندتر می‌کند.   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 102   
گزارش خطا • بحث کنید   
  
برنامه‌ریزی در مقیاس چند ماه یا یک سال چطور؟ یک تلاش میان‌مدت یا بلندمدت با اهدافی نسبتاً کلی شروع می‌شود. با پیروی از روش طبیعی، ما ویژگی‌های با ارزش بالا را زودتر Build می‌کنیم و مدیران می‌توانند بدون وسواس روی جزئیات، وضعیت ما را زیر نظر داشته باشند. به عنوان برنامه‌ریزان و مدیران سطح بالا، بیایید در مورد قابلیت‌های کلی موردنیاز در پروژه بزرگ خود شفاف باشیم و از Product Champions خود بخواهیم که "نرم‌افزاری را به ما نشان دهند" که آن قابلیت‌ها را فراهم می‌کند. بیایید به Champions خود کمک کنیم تا ابتدا تمام جنبه‌های کلیدی محصول را پوشش دهند و سپس جنبه‌های کم‌اهمیت‌تر را در صورت اجازه زمان و بودجه تکمیل کنند.   
گزارش خطا • بحث کنید   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 103   
  
[متن از صفحه قبلی: برنامه‌ریزی در مقیاس چند ماه یا یک سال چطور؟ یک تلاش میان‌مدت یا بلندمدت با اهدافی نسبتاً کلی شروع می‌شود. با پیروی از روش طبیعی، ما ویژگی‌های با ارزش بالا را زودتر Build می‌کنیم و مدیران می‌توانند بدون وسواس روی جزئیات، وضعیت ما را زیر نظر داشته باشند. به عنوان برنامه‌ریزان و مدیران سطح بالا، بیایید در مورد قابلیت‌های کلی موردنیاز در پروژه بزرگ خود شفاف باشیم و از Product Champions خود بخواهیم که "نرم‌افزاری را به ما نشان دهند" که آن قابلیت‌ها را فراهم می‌کند. بیایید به Champions خود کمک کنیم تا ابتدا تمام جنبه‌های کلیدی محصول را پوشش دهند و سپس جنبه‌های کم‌اهمیت‌تر را در صورت اجازه زمان و بودجه تکمیل کنند.   
گزارش خطا • بحث کنید   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 103]   
  
برنامه‌ریزی کوتاه‌مدت، روزانه یا هفتگی چطور؟ در روش طبیعی، برنامه‌ریزی کوتاه‌مدت در سطح توسعه به صورت مداوم انجام می‌شود. ویژگی‌های با ارزش اولویت‌بندی می‌شوند و به ترتیب ارزش بالاتر انجام می‌شوند. هفته به هفته، ارزش به صورت قابل مشاهده رشد می‌کند. برنامه هر چند هفته یک بار بازبینی می‌شود و همه ما می‌توانیم ببینیم اوضاع چگونه پیش می‌رود. ما این کار را به سادگی انجام می‌دهیم.

================================================================================

# Section 99

## Original Text:

Every couple of weeks, we observe  
what has been accomplished and we plan the next couple  
of weeks, keeping our eye on the overall vision and goals  
for our effort. Chapter 16. Managing Natural Software Development • 104  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: What about short-term planning, day to day, week to week. In the Natural Way, short-term planning is continuous at  
the level of development. Valuable features are given prior-  
ities, and they get done in the order of highest value first. Week in and week out, value grows visibly. The plan is  
revised every couple of weeks and we can all see how things  
are going. We do this very simply. Every couple of weeks, we observe  
what has been accomplished and we plan the next couple  
of weeks, keeping our eye on the overall vision and goals  
for our effort. Chapter 16. Managing Natural Software Development • 104  
report erratum • discuss]  
  
My experience has been that managing is mostly about trying to  
keep projects on track. How do we ensure that we are keeping to  
the plan. Honestly, our job isn’t to stick to the plan—it’s to steer our  
course for the best result, not some fixed target. When we deploy our products often, delivering real value  
to our users, we often find that we can stop far before the  
time and money run out. Why. Because we’ve already done  
everything customers really need. We often find that, instead  
of whatever we imagined the product should be, new ideas  
have bubbled up. Working in the Natural Way, we steer the  
project—we don’t just plan and then hope. Always keep an eye on the value. At the planning level,  
always ask what the most valuable next things are. To track  
how the plan is going, ask the Product Champion to show  
us the software, and to relate what it does to the value we’re  
asking for. report erratum • discussChapter 16.

## Translated Text:

هر چند هفته یک بار، ما بررسی می‌کنیم که چه کارهایی انجام شده است و برای چند هفته آینده برنامه‌ریزی می‌کنیم، در حالی که چشم‌انداز کلی و اهداف تلاشمان را در نظر داریم.   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 104   
report erratum • discuss   
  
[متن از صفحه قبلی: برنامه‌ریزی کوتاه‌مدت، روزانه، هفتگی چطور؟ در روش طبیعی، برنامه‌ریزی کوتاه‌مدت به صورت مداوم در سطح توسعه انجام می‌شود. قابلیت‌های ارزشمند اولویت‌بندی می‌شوند و به ترتیب بالاترین ارزش انجام می‌شوند. هفته به هفته، ارزش به طور قابل مشاهده‌ای افزایش می‌یابد. برنامه هر چند هفته یک بار بازبینی می‌شود و همه ما می‌توانیم ببینیم که اوضاع چگونه پیش می‌رود. این کار را به سادگی انجام می‌دهیم. هر چند هفته یک بار، ما بررسی می‌کنیم که چه کارهایی انجام شده است و برای چند هفته آینده برنامه‌ریزی می‌کنیم، در حالی که چشم‌انداز کلی و اهداف تلاشمان را در نظر داریم.   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 104   
report erratum • discuss]   
  
تجربه من این بوده است که مدیریت بیشتر درباره تلاش برای نگه داشتن پروژه‌ها در مسیر درست است. چگونه اطمینان حاصل کنیم که به برنامه پایبند هستیم؟ صادقانه بگویم، کار ما پایبندی به برنامه نیست—بلکه هدایت مسیر برای بهترین نتیجه است، نه یک هدف ثابت. وقتی ما محصولات خود را به طور مکرر Deploy می‌کنیم و ارزش واقعی را به کاربرانمان ارائه می‌دهیم، اغلب متوجه می‌شویم که می‌توانیم خیلی زودتر از اتمام زمان و بودجه متوقف شویم. چرا؟ چون ما قبلاً هر چیزی که مشتریان واقعاً نیاز دارند را انجام داده‌ایم. اغلب می‌بینیم که به جای آنچه تصور می‌کردیم محصول باید باشد، ایده‌های جدیدی پدیدار شده‌اند. با کار کردن به روش طبیعی، ما پروژه را هدایت می‌کنیم—فقط برنامه‌ریزی نمی‌کنیم و سپس امیدوار باشیم. همیشه بر ارزش تمرکز کنید. در سطح برنامه‌ریزی، همیشه بپرسید که با ارزش‌ترین کارهای بعدی چیستند. برای پیگیری اینکه برنامه چگونه پیش می‌رود، از Product Champion بخواهید نرم‌افزار را به ما نشان دهد و توضیح دهد که چگونه آنچه انجام می‌دهد به ارزشی که ما درخواست کرده‌ایم مرتبط است.   
report erratum • discuss   
فصل 16.

================================================================================

# Section 100

## Original Text:

Managing Natural Software Development • 105  
  
[CONTEXT FROM PREVIOUS PAGE: My experience has been that managing is mostly about trying to  
keep projects on track. How do we ensure that we are keeping to  
the plan. Honestly, our job isn’t to stick to the plan—it’s to steer our  
course for the best result, not some fixed target. When we deploy our products often, delivering real value  
to our users, we often find that we can stop far before the  
time and money run out. Why. Because we’ve already done  
everything customers really need. We often find that, instead  
of whatever we imagined the product should be, new ideas  
have bubbled up. Working in the Natural Way, we steer the  
project—we don’t just plan and then hope. Always keep an eye on the value. At the planning level,  
always ask what the most valuable next things are. To track  
how the plan is going, ask the Product Champion to show  
us the software, and to relate what it does to the value we’re  
asking for. report erratum • discussChapter 16. Managing Natural Software Development • 105]  
  
Ron, your advice sounds good, but I’ve often found that it comes  
down to having the right people in the right positions. How can  
we best make the necessary organizational decisions. Upper-level management also has the responsibility for  
general organization. They allocate money and people to  
the work. Generally, having determined the budget, man-  
agement selects the Product Champion and (possibly) some  
of the personnel. Often, the Product Champion selects the  
core team members and the team as a whole selects the rest. Within that framework, the Product Champion and team  
self-organize to get the work done. Try to push detailed organizational decisions downward as  
far as possible. Use budgeting to control the size of efforts. Focus on results as much as you can, and allow the people  
closer to the work to make most of the decisions about what  
to do and how to organize to do it. Chapter 16.

## Translated Text:

مدیریت توسعه نرم‌افزار طبیعی • ۱۰۵  
  
[متن از صفحه قبلی: تجربه من این بوده است که مدیریت عمدتاً در مورد تلاش برای نگه داشتن پروژه‌ها در مسیر است. چگونه اطمینان حاصل کنیم که به برنامه پایبند هستیم؟ صادقانه بگویم، کار ما پایبندی به برنامه نیست—بلکه هدایت مسیر برای بهترین نتیجه است، نه یک هدف ثابت. وقتی محصولات خود را به طور مکرر Deploy می‌کنیم و ارزش واقعی را به کاربران خود ارائه می‌دهیم، اغلب متوجه می‌شویم که می‌توانیم خیلی زودتر از اتمام زمان و بودجه متوقف شویم. چرا؟ چون ما قبلاً هر چیزی که مشتریان واقعاً نیاز دارند را انجام داده‌ایم. اغلب متوجه می‌شویم که به جای هر چیزی که تصور می‌کردیم محصول باید باشد، ایده‌های جدیدی ظهور کرده‌اند. با کار کردن به روش طبیعی، پروژه را هدایت می‌کنیم—فقط برنامه‌ریزی نمی‌کنیم و سپس امیدوار باشیم. همیشه به ارزش توجه کنید. در سطح برنامه‌ریزی، همیشه بپرسید که با ارزش‌ترین کارهای بعدی چه هستند. برای پیگیری اینکه برنامه چگونه پیش می‌رود، از Product Champion بخواهید نرم‌افزار را به ما نشان دهد و توضیح دهد که چه کاری انجام می‌دهد و چگونه با ارزشی که درخواست کرده‌ایم مرتبط است. گزارش خطا • بحث فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۰۵]  
  
رون، توصیه‌های شما خوب به نظر می‌رسد، اما من اغلب متوجه شده‌ام که این موضوع به داشتن افراد مناسب در موقعیت‌های مناسب بستگی دارد. چگونه می‌توانیم بهترین تصمیمات سازمانی لازم را بگیریم؟ مدیریت سطح بالا همچنین مسئولیت سازماندهی کلی را بر عهده دارد. آن‌ها پول و افراد را به کارها تخصیص می‌دهند. به طور کلی، پس از تعیین بودجه، مدیریت Product Champion و (احتمالاً) برخی از پرسنل را انتخاب می‌کند. اغلب، Product Champion اعضای اصلی تیم را انتخاب می‌کند و تیم به عنوان یک کل بقیه را انتخاب می‌کند. در چارچوب آن، Product Champion و تیم به صورت خودسازمانده کار را انجام می‌دهند. سعی کنید تصمیمات سازمانی جزئی را تا حد امکان به سطوح پایین‌تر منتقل کنید. از بودجه‌بندی برای کنترل اندازه تلاش‌ها استفاده کنید. تا جایی که می‌توانید بر نتایج تمرکز کنید و به افرادی که به کار نزدیک‌تر هستند اجازه دهید بیشتر تصمیمات را در مورد اینکه چه کاری انجام دهند و چگونه سازماندهی کنند، بگیرند. فصل ۱۶.

================================================================================

# Section 101

## Original Text:

Managing Natural Software Development • 106  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Ron, your advice sounds good, but I’ve often found that it comes  
down to having the right people in the right positions. How can  
we best make the necessary organizational decisions. Upper-level management also has the responsibility for  
general organization. They allocate money and people to  
the work. Generally, having determined the budget, man-  
agement selects the Product Champion and (possibly) some  
of the personnel. Often, the Product Champion selects the  
core team members and the team as a whole selects the rest. Within that framework, the Product Champion and team  
self-organize to get the work done. Try to push detailed organizational decisions downward as  
far as possible. Use budgeting to control the size of efforts. Focus on results as much as you can, and allow the people  
closer to the work to make most of the decisions about what  
to do and how to organize to do it. Chapter 16. Managing Natural Software Development • 106  
report erratum • discuss]  
  
What about staffing decisions. Who decides who should be hired  
or fired. Very likely, personnel actions must be initiated or at least  
approved by management. More and more, however, the  
choices and recommendations are delegated to the team. Let  
your teams determine whether they want to add more peo-  
ple, and let them choose who to add. They understand better  
than you will what they need. Help the team understand policy issues, overall hiring  
guidelines and strategies, and most of all, the value of  
building on the people you already have. The better your  
teams understand the bigger picture, the better things will  
go. report erratum • discussChapter 16. Managing Natural Software Development • 107  
  
[CONTEXT FROM PREVIOUS PAGE: What about staffing decisions. Who decides who should be hired  
or fired. Very likely, personnel actions must be initiated or at least  
approved by management.

## Translated Text:

مدیریت توسعه نرم‌افزار طبیعی • ۱۰۶   
گزارش خطا • بحث کنید   
  
[متن از صفحه قبلی: رون، توصیه‌های شما خوب به نظر می‌رسد، اما من اغلب متوجه شده‌ام که این موضوع به داشتن افراد مناسب در موقعیت‌های مناسب برمی‌گردد. چگونه می‌توانیم بهترین تصمیمات سازمانی لازم را بگیریم؟ مدیریت سطح بالا همچنین مسئولیت سازماندهی کلی را بر عهده دارد. آنها بودجه و نیروی انسانی را به کار اختصاص می‌دهند. به طور کلی، پس از تعیین بودجه، مدیریت Product Champion و (احتمالاً) برخی از اعضای تیم را انتخاب می‌کند. اغلب، Product Champion اعضای اصلی تیم را انتخاب می‌کند و تیم به عنوان یک کل بقیه اعضا را انتخاب می‌کند. در چارچوب این ساختار، Product Champion و تیم به صورت خودسازمانده کار را انجام می‌دهند. سعی کنید تصمیمات سازمانی جزئی را تا حد امکان به سطوح پایین‌تر منتقل کنید. از بودجه‌بندی برای کنترل اندازه تلاش‌ها استفاده کنید. تا جایی که می‌توانید بر نتایج تمرکز کنید و اجازه دهید افرادی که به کار نزدیک‌تر هستند، بیشتر تصمیمات را درباره اینکه چه کاری انجام دهند و چگونه سازماندهی کنند، بگیرند. فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۰۶   
گزارش خطا • بحث کنید]   
  
در مورد تصمیمات مربوط به نیروی انسانی چطور؟ چه کسی تصمیم می‌گیرد که چه کسی باید استخدام یا اخراج شود؟ به احتمال زیاد، اقدامات مربوط به نیروی انسانی باید توسط مدیریت آغاز یا حداقل تأیید شود. با این حال، به طور فزاینده‌ای، انتخاب‌ها و توصیه‌ها به تیم واگذار می‌شود. اجازه دهید تیم‌های شما تصمیم بگیرند که آیا می‌خواهند افراد بیشتری اضافه کنند یا نه، و اجازه دهید آنها انتخاب کنند که چه کسی را اضافه کنند. آنها بهتر از شما می‌دانند که به چه چیزی نیاز دارند. به تیم کمک کنید تا مسائل سیاستی، دستورالعمل‌ها و استراتژی‌های کلی استخدام و از همه مهم‌تر، ارزش استفاده از افرادی که قبلاً دارید را درک کنند. هرچه تیم‌های شما تصویر بزرگ‌تر را بهتر درک کنند، اوضاع بهتر پیش خواهد رفت.   
گزارش خطا • بحث کنید   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۰۷   
  
[متن از صفحه قبلی: در مورد تصمیمات مربوط به نیروی انسانی چطور؟ چه کسی تصمیم می‌گیرد که چه کسی باید استخدام یا اخراج شود؟ به احتمال زیاد، اقدامات مربوط به نیروی انسانی باید توسط مدیریت آغاز یا حداقل تأیید شود.]

================================================================================

# Section 102

## Original Text:

More and more, however, the  
choices and recommendations are delegated to the team. Let  
your teams determine whether they want to add more peo-  
ple, and let them choose who to add. They understand better  
than you will what they need. Help the team understand policy issues, overall hiring  
guidelines and strategies, and most of all, the value of  
building on the people you already have. The better your  
teams understand the bigger picture, the better things will  
go. report erratum • discussChapter 16. Managing Natural Software Development • 107]  
  
Management is supposed to direct what goes on. If all these deci-  
sions are being made at the bottom of the organization, how can  
management provide necessary direction. Based on long-term planning, management has decided  
what products and projects to invest in. They have selected  
a Product Champion who will be accountable for the results  
of each effort. Management views each product as it comes  
into being, and supports and guides the Product Champion,  
making sure they remain aligned with the corporate purpose  
for the product. Direction takes a few primary forms. Sometimes things change in the environment or the compa-  
ny’s priorities. These changes are passed down in terms of  
adjustments to product vision, budget, and the like. Sometimes as managers view the results of the work, they’ll  
discover things that they did not express well or that  
improve their understanding of what’s needed. These, too  
result in adjustments to the team’s vision. Sometimes things may not go as well as they might. Working  
as we recommend in this book, management will see devia-  
tions promptly and can respond by helping the Product  
Champion and team to do better. Big surprises should be  
nearly impossible when you can see the product every couple  
of weeks. Chapter 16. Managing Natural Software Development • 108  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Management is supposed to direct what goes on.

## Translated Text:

با این حال، تصمیم‌گیری‌ها و توصیه‌ها به طور فزاینده‌ای به تیم‌ها واگذار می‌شوند. اجازه دهید تیم‌های شما تصمیم بگیرند که آیا می‌خواهند افراد بیشتری اضافه کنند و اجازه دهید خودشان انتخاب کنند که چه کسی را اضافه کنند. آن‌ها بهتر از شما می‌دانند که به چه چیزی نیاز دارند. به تیم کمک کنید تا مسائل مربوط به سیاست‌ها، دستورالعمل‌ها و استراتژی‌های کلی استخدام و از همه مهم‌تر، ارزش ساختن بر اساس افرادی که از قبل در اختیار دارید را درک کنند. هرچه تیم‌های شما تصویر کلی را بهتر درک کنند، اوضاع بهتر پیش خواهد رفت.   
report erratum • discuss   
Chapter 16. Managing Natural Software Development • 107   
  
مدیریت قرار است آنچه در جریان است را هدایت کند. اگر تمام این تصمیم‌ها در پایین سازمان گرفته می‌شوند، مدیریت چگونه می‌تواند جهت‌گیری لازم را ارائه دهد؟ بر اساس برنامه‌ریزی بلندمدت، مدیریت تصمیم گرفته است که در چه محصولات و پروژه‌هایی سرمایه‌گذاری کند. آن‌ها یک Product Champion انتخاب کرده‌اند که مسئولیت نتایج هر تلاش را بر عهده خواهد داشت. مدیریت هر محصول را در حین ایجاد آن مشاهده می‌کند و از Product Champion حمایت و راهنمایی می‌کند تا اطمینان حاصل کند که با هدف سازمانی محصول همسو باقی می‌ماند.   
  
جهت‌گیری به چند شکل اصلی انجام می‌شود. گاهی اوقات محیط یا اولویت‌های شرکت تغییر می‌کنند. این تغییرات به صورت تنظیماتی در چشم‌انداز محصول، بودجه و موارد مشابه منتقل می‌شوند. گاهی اوقات مدیران هنگام مشاهده نتایج کار، متوجه می‌شوند که چیزی را به خوبی بیان نکرده‌اند یا درک بهتری از نیازها پیدا می‌کنند. این موارد نیز منجر به تنظیماتی در چشم‌انداز تیم می‌شوند.   
  
گاهی اوقات ممکن است اوضاع به خوبی پیش نرود. با کار کردن به روشی که در این کتاب توصیه می‌کنیم، مدیریت انحرافات را به سرعت مشاهده خواهد کرد و می‌تواند با کمک به Product Champion و تیم، عملکرد بهتری ارائه دهد. شگفتی‌های بزرگ تقریباً غیرممکن خواهند بود وقتی که می‌توانید محصول را هر چند هفته یک بار مشاهده کنید.   
Chapter 16. Managing Natural Software Development • 108   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: مدیریت قرار است آنچه در جریان است را هدایت کند.]

================================================================================

# Section 103

## Original Text:

If all these deci-  
sions are being made at the bottom of the organization, how can  
management provide necessary direction. Based on long-term planning, management has decided  
what products and projects to invest in. They have selected  
a Product Champion who will be accountable for the results  
of each effort. Management views each product as it comes  
into being, and supports and guides the Product Champion,  
making sure they remain aligned with the corporate purpose  
for the product. Direction takes a few primary forms. Sometimes things change in the environment or the compa-  
ny’s priorities. These changes are passed down in terms of  
adjustments to product vision, budget, and the like. Sometimes as managers view the results of the work, they’ll  
discover things that they did not express well or that  
improve their understanding of what’s needed. These, too  
result in adjustments to the team’s vision. Sometimes things may not go as well as they might. Working  
as we recommend in this book, management will see devia-  
tions promptly and can respond by helping the Product  
Champion and team to do better. Big surprises should be  
nearly impossible when you can see the product every couple  
of weeks. Chapter 16. Managing Natural Software Development • 108  
report erratum • discuss]  
  
But work has to be controlled. How do we make sure that, day to  
day, things are under control. Day to day, teams control how they do the work. The Prod-  
uct Champion controls, week to week, what they work on. Management observes the results of that work and makes  
sure that progress is commensurate with the time and  
money being spent. If deviations exist, management might  
take action. That action would not be taken by stepping in  
and doing the work, but by providing assistance and train-  
ing, and, if need be, by adjusting budgets, staffing, and  
responsibilities. report erratum • discussChapter 16.

## Translated Text:

اگر تمام این تصمیمات در پایین سازمان گرفته شوند، مدیریت چگونه می‌تواند جهت‌دهی لازم را فراهم کند؟ بر اساس برنامه‌ریزی بلندمدت، مدیریت تصمیم گرفته است که در چه محصولات و پروژه‌هایی سرمایه‌گذاری کند. آن‌ها یک Product Champion انتخاب کرده‌اند که مسئول نتایج هر تلاش خواهد بود. مدیریت هر محصول را همان‌طور که به وجود می‌آید مشاهده می‌کند و از Product Champion حمایت و راهنمایی می‌کند تا اطمینان حاصل کند که آن‌ها با هدف سازمانی محصول هماهنگ باقی می‌مانند. جهت‌دهی به چند شکل اصلی انجام می‌شود. گاهی اوقات تغییراتی در محیط یا اولویت‌های شرکت رخ می‌دهد. این تغییرات به صورت تنظیماتی در چشم‌انداز محصول، بودجه و موارد مشابه منتقل می‌شوند. گاهی اوقات، وقتی مدیران نتایج کار را مشاهده می‌کنند، متوجه چیزهایی می‌شوند که به‌خوبی بیان نکرده‌اند یا درک آن‌ها از نیازها بهبود می‌یابد. این موارد نیز منجر به تنظیماتی در چشم‌انداز تیم می‌شوند. گاهی اوقات ممکن است اوضاع به خوبی پیش نرود. با کار کردن به روشی که در این کتاب توصیه کرده‌ایم، مدیریت انحرافات را به سرعت مشاهده خواهد کرد و می‌تواند با کمک به Product Champion و تیم، پاسخ دهد تا عملکرد بهتری داشته باشند. شگفتی‌های بزرگ باید تقریباً غیرممکن باشند وقتی که می‌توانید محصول را هر چند هفته یک بار مشاهده کنید.   
فصل 16. مدیریت توسعه نرم‌افزار طبیعی • 108   
report erratum • discuss   
  
اما کار باید کنترل شود. چگونه می‌توانیم مطمئن شویم که روزانه همه چیز تحت کنترل است؟ روزانه، تیم‌ها نحوه انجام کار را کنترل می‌کنند. Product Champion به صورت هفتگی کنترل می‌کند که روی چه چیزی کار کنند. مدیریت نتایج آن کار را مشاهده می‌کند و اطمینان حاصل می‌کند که پیشرفت متناسب با زمان و هزینه صرف شده است. اگر انحرافاتی وجود داشته باشد، مدیریت ممکن است اقدام کند. این اقدام با دخالت مستقیم در انجام کار نخواهد بود، بلکه با ارائه کمک و آموزش و در صورت لزوم، با تنظیم بودجه، نیروی انسانی و مسئولیت‌ها انجام می‌شود.   
report erratum • discuss   
فصل 16.

================================================================================

# Section 104

## Original Text:

Managing Natural Software Development • 109  
  
[CONTEXT FROM PREVIOUS PAGE: But work has to be controlled. How do we make sure that, day to  
day, things are under control. Day to day, teams control how they do the work. The Prod-  
uct Champion controls, week to week, what they work on. Management observes the results of that work and makes  
sure that progress is commensurate with the time and  
money being spent. If deviations exist, management might  
take action. That action would not be taken by stepping in  
and doing the work, but by providing assistance and train-  
ing, and, if need be, by adjusting budgets, staffing, and  
responsibilities. report erratum • discussChapter 16. Managing Natural Software Development • 109]  
  
Summing Up  
The Natural Way of software development calls for delegat-  
ing authority to the people who do the work. There’s nothing  
new about this; it’s how management has always been done. True, some managers have been hesitant to fully delegate  
for fear of being unable to help if needed. Fortunately, this  
style of software development provides plenty of visibility  
into what’s going on, which makes delegation quite safe. When our teams “show us the software,” we always know  
where they are, what they’re working on, and how things  
are going. Follow these guidelines thoughtfully and you’ll  
be in good shape. Chapter 16. Managing Natural Software Development • 110  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Summing Up  
The Natural Way of software development calls for delegat-  
ing authority to the people who do the work. There’s nothing  
new about this; it’s how management has always been done. True, some managers have been hesitant to fully delegate  
for fear of being unable to help if needed. Fortunately, this  
style of software development provides plenty of visibility  
into what’s going on, which makes delegation quite safe. When our teams “show us the software,” we always know  
where they are, what they’re working on, and how things  
are going.

## Translated Text:

مدیریت توسعه نرم‌افزار طبیعی • ۱۰۹   
  
[متن از صفحه قبلی: اما کار باید کنترل شود. چگونه مطمئن شویم که روز به روز همه چیز تحت کنترل است؟ روزانه، تیم‌ها کنترل می‌کنند که چگونه کار را انجام دهند. Product Champion به صورت هفتگی کنترل می‌کند که روی چه چیزی کار کنند. مدیریت نتایج آن کار را مشاهده می‌کند و اطمینان حاصل می‌کند که پیشرفت متناسب با زمان و هزینه صرف شده است. اگر انحرافی وجود داشته باشد، مدیریت ممکن است اقدام کند. این اقدام با ورود و انجام کار انجام نمی‌شود، بلکه با ارائه کمک و آموزش، و در صورت لزوم، با تنظیم بودجه‌ها، نیروی انسانی و مسئولیت‌ها انجام می‌شود.   
report erratum • discuss   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۰۹]   
  
جمع‌بندی   
رویکرد طبیعی توسعه نرم‌افزار مستلزم واگذاری اختیار به افرادی است که کار را انجام می‌دهند. این موضوع چیز جدیدی نیست؛ این همان روشی است که مدیریت همیشه انجام شده است. درست است که برخی مدیران ممکن است به دلیل ترس از ناتوانی در کمک در صورت نیاز، از واگذاری کامل اختیار تردید داشته باشند. خوشبختانه، این سبک توسعه نرم‌افزار دید کافی از آنچه در حال وقوع است فراهم می‌کند که واگذاری اختیار را بسیار ایمن می‌سازد. وقتی تیم‌های ما "نرم‌افزار را به ما نشان می‌دهند"، ما همیشه می‌دانیم که کجا هستند، روی چه چیزی کار می‌کنند و وضعیت چگونه است. با دقت این دستورالعمل‌ها را دنبال کنید و در وضعیت خوبی خواهید بود.   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۱۰   
report erratum • discuss

================================================================================

# Section 105

## Original Text:

Follow these guidelines thoughtfully and you’ll  
be in good shape. Chapter 16. Managing Natural Software Development • 110  
report erratum • discuss]  
  
report erratum • discussChapter 16. Managing Natural Software Development • 111  
  
[CONTEXT FROM PREVIOUS PAGE: report erratum • discussChapter 16. Managing Natural Software Development • 111]  
  
CHAPTER 17  
Whip the Ponies Harder  
A high-level manager was told that a project was moving more  
slowly than had been hoped. He retorted, “Well, we’ll just have to  
whip the ponies harder. ”  
I find it difficult to think of a management comment more repug-  
nant than that. In addition, it was sure to backfire. Under pressure, teams give up the wrong things. They don’t  
test enough; they leave the code in poor condition. This  
reduces value, increases the delay to getting the value, and  
reduces the value they can deliver later. Under pressure, teams test less and therefore put more  
defects in. Some of these defects escape the room and affect  
customers. Value is directly reduced. Some defects are discovered before the product ships. This  
usually means that we have a testing phase that comes after  
we think we are “done. ” Already this is delaying delivery  
of value. Worse yet, we have to take time to fix the problems. This  
rework delays and reduces value even more. Finally, when our teams work under pressure, they leave  
the code in bad condition. This makes adding new features  
harder and slows down value delivery even more. How many defects do you have now. Do you really want  
more. How often do developers say that the condition of  
the code slows them down. Did your innocent pressure  
cause those things. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 17  
Whip the Ponies Harder  
A high-level manager was told that a project was moving more  
slowly than had been hoped. He retorted, “Well, we’ll just have to  
whip the ponies harder. ”  
I find it difficult to think of a management comment more repug-  
nant than that.

## Translated Text:

این دستورالعمل‌ها را با دقت دنبال کنید و در وضعیت خوبی قرار خواهید گرفت.   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۱۰   
report erratum • discuss   
  
report erratum • discuss   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۱۱   
  
[متن از صفحه قبلی: report erratum • discuss   
فصل ۱۶. مدیریت توسعه نرم‌افزار طبیعی • ۱۱۱]   
  
فصل ۱۷   
اسب‌ها را سخت‌تر شلاق بزنید   
به یک مدیر ارشد گفته شد که یک پروژه کندتر از حد انتظار پیش می‌رود. او پاسخ داد: «خب، باید اسب‌ها را سخت‌تر شلاق بزنیم.»   
  
فکر کردن به یک اظهار نظر مدیریتی که از این نفرت‌انگیزتر باشد، برایم دشوار است. علاوه بر این، چنین رویکردی قطعاً نتیجه معکوس خواهد داشت. تحت فشار، تیم‌ها چیزهای اشتباهی را کنار می‌گذارند. آنها به اندازه کافی Test نمی‌کنند؛ کد را در وضعیت بدی رها می‌کنند. این امر ارزش را کاهش می‌دهد، تأخیر در دستیابی به ارزش را افزایش می‌دهد و ارزشی که می‌توانند بعداً ارائه دهند را کاهش می‌دهد.   
  
تحت فشار، تیم‌ها کمتر Test می‌کنند و در نتیجه نقص‌های بیشتری ایجاد می‌کنند. برخی از این نقص‌ها از اتاق خارج می‌شوند و بر مشتریان تأثیر می‌گذارند. ارزش به طور مستقیم کاهش می‌یابد. برخی از نقص‌ها قبل از عرضه محصول کشف می‌شوند. این معمولاً به این معنی است که ما یک مرحله Test داریم که پس از این‌که فکر می‌کنیم «تمام شده‌ایم» انجام می‌شود. این موضوع از همین حالا تحویل ارزش را به تأخیر می‌اندازد.   
  
بدتر از آن، ما باید زمانی را برای رفع مشکلات صرف کنیم. این بازکاری، تأخیر بیشتری ایجاد کرده و ارزش را حتی بیشتر کاهش می‌دهد. در نهایت، وقتی تیم‌های ما تحت فشار کار می‌کنند، کد را در وضعیت بدی رها می‌کنند. این امر افزودن قابلیت‌های جدید را سخت‌تر می‌کند و تحویل ارزش را حتی بیشتر کند می‌کند.   
  
در حال حاضر چند نقص دارید؟ آیا واقعاً نقص‌های بیشتری می‌خواهید؟ چند وقت یک‌بار توسعه‌دهندگان می‌گویند که وضعیت کد سرعت آنها را کاهش می‌دهد؟ آیا فشار بی‌گناه شما باعث این مسائل شده است؟   
report erratum • discuss   
  
[متن از صفحه قبلی: فصل ۱۷   
اسب‌ها را سخت‌تر شلاق بزنید   
به یک مدیر ارشد گفته شد که یک پروژه کندتر از حد انتظار پیش می‌رود. او پاسخ داد: «خب، باید اسب‌ها را سخت‌تر شلاق بزنیم.»   
فکر کردن به یک اظهار نظر مدیریتی که از این نفرت‌انگیزتر باشد، برایم دشوار است.]

================================================================================

# Section 106

## Original Text:

In addition, it was sure to backfire. Under pressure, teams give up the wrong things. They don’t  
test enough; they leave the code in poor condition. This  
reduces value, increases the delay to getting the value, and  
reduces the value they can deliver later. Under pressure, teams test less and therefore put more  
defects in. Some of these defects escape the room and affect  
customers. Value is directly reduced. Some defects are discovered before the product ships. This  
usually means that we have a testing phase that comes after  
we think we are “done. ” Already this is delaying delivery  
of value. Worse yet, we have to take time to fix the problems. This  
rework delays and reduces value even more. Finally, when our teams work under pressure, they leave  
the code in bad condition. This makes adding new features  
harder and slows down value delivery even more. How many defects do you have now. Do you really want  
more. How often do developers say that the condition of  
the code slows them down. Did your innocent pressure  
cause those things. report erratum • discuss]  
  
We need more features. Why can’t we just increase velocity. Frankly, when I look into an organization that “needs more  
features,” what I almost invariably find is an organization  
that can’t say no. They have become order takers rather than  
decision makers. They’re working on some things of value,  
perhaps, but much of the work they’re doing doesn’t bring  
much real value to their products—or to their customers. No, really we need more features. The programmers have to go  
faster. Your programmers are going as fast as they reasonably can. There may be ways that the code can be improved over time  
so that they can go faster. There are almost certainly ways  
to reduce defect injection, which saves you all that fixing  
time. Those are improvements that take time to make, and  
they cannot be made by a team under pressure.

## Translated Text:

علاوه بر این، مشخص بود که نتیجه معکوس خواهد داد. تحت فشار، تیم‌ها چیزهای اشتباهی را کنار می‌گذارند. آنها به اندازه کافی Test نمی‌کنند؛ کد را در وضعیت نامناسبی رها می‌کنند. این موضوع ارزش را کاهش می‌دهد، تأخیر در دستیابی به ارزش را افزایش می‌دهد و ارزشی که می‌توانند در آینده ارائه دهند را کاهش می‌دهد. تحت فشار، تیم‌ها کمتر Test می‌کنند و در نتیجه Defect بیشتری ایجاد می‌کنند. برخی از این Defectها از کنترل خارج شده و بر مشتریان تأثیر می‌گذارند. ارزش به‌طور مستقیم کاهش می‌یابد. برخی Defectها قبل از Release محصول کشف می‌شوند. این معمولاً به این معناست که ما یک فاز Testing داریم که بعد از اینکه فکر می‌کنیم "تمام شده‌ایم" انجام می‌شود. همین حالا این موضوع تحویل ارزش را به تأخیر می‌اندازد. بدتر از آن، ما باید زمان صرف کنیم تا مشکلات را برطرف کنیم. این Rework تأخیر بیشتری ایجاد کرده و ارزش را حتی بیشتر کاهش می‌دهد. در نهایت، وقتی تیم‌های ما تحت فشار کار می‌کنند، کد را در وضعیت بدی رها می‌کنند. این موضوع اضافه کردن Featureهای جدید را سخت‌تر کرده و تحویل ارزش را حتی بیشتر کند می‌کند. اکنون چند Defect دارید؟ آیا واقعاً می‌خواهید بیشتر داشته باشید؟ چند وقت یک‌بار توسعه‌دهندگان می‌گویند که وضعیت کد آنها را کند می‌کند؟ آیا فشار بی‌گناه شما باعث این مسائل شده است؟   
[report erratum • discuss]  
  
ما به Featureهای بیشتری نیاز داریم. چرا نمی‌توانیم فقط Velocity را افزایش دهیم؟ صادقانه بگویم، وقتی به یک سازمان که "به Featureهای بیشتری نیاز دارد" نگاه می‌کنم، تقریباً همیشه با سازمانی مواجه می‌شوم که نمی‌تواند "نه" بگوید. آنها به جای تصمیم‌گیرنده، به سفارش‌گیرنده تبدیل شده‌اند. شاید روی برخی چیزهای ارزشمند کار می‌کنند، اما بخش زیادی از کاری که انجام می‌دهند ارزش واقعی زیادی برای محصولاتشان—یا برای مشتریانشان—ایجاد نمی‌کند. نه، واقعاً ما به Featureهای بیشتری نیاز داریم. برنامه‌نویس‌ها باید سریع‌تر کار کنند. برنامه‌نویس‌های شما به سریع‌ترین شکلی که منطقی است کار می‌کنند. ممکن است راه‌هایی وجود داشته باشد که کد با گذشت زمان بهبود یابد تا آنها بتوانند سریع‌تر کار کنند. تقریباً به طور قطع راه‌هایی برای کاهش Defect Injection وجود دارد که زمان صرف‌شده برای Fix کردن را ذخیره می‌کند. این‌ها بهبودهایی هستند که زمان برای انجام آنها لازم است و نمی‌توانند توسط تیمی که تحت فشار است انجام شوند.

================================================================================

# Section 107

## Original Text:

One way that we can subtly cause pressure, leading to the  
problems we just talked about, is to ask the team to “increase  
velocity. ” This means “get more done. ” Teams will try to do  
that. They will do so by turning the invisible dials of quality  
and estimation. If they reduce quality, we’ll get more defects and go slower. We don’t want that. Under pressure, teams will consciously or unconsciously  
begin to be more conservative in what they take on. They  
will begin rating things as a bit larger or a bit harder than  
they used to. This will give the impression that they’re going  
faster, but they are not. What would a team need to do, or need to know, in order  
to produce more work per unit time. Do you believe that  
“work harder” answers that question. What can manage-  
ment do to help a team be truly more productive. Are there any ways to go faster. How can we do it. If you must try to go faster, analyze sources of delay. These  
usually have more impact than individual productivity. Work on team productivity before individual productivity. Ensure that each team has a good mix of skills. Make sure  
report erratum • discussChapter 17. Whip the Ponies Harder • 113  
  
[CONTEXT FROM PREVIOUS PAGE: We need more features. Why can’t we just increase velocity. Frankly, when I look into an organization that “needs more  
features,” what I almost invariably find is an organization  
that can’t say no. They have become order takers rather than  
decision makers. They’re working on some things of value,  
perhaps, but much of the work they’re doing doesn’t bring  
much real value to their products—or to their customers. No, really we need more features. The programmers have to go  
faster. Your programmers are going as fast as they reasonably can. There may be ways that the code can be improved over time  
so that they can go faster. There are almost certainly ways  
to reduce defect injection, which saves you all that fixing  
time.

## Translated Text:

یک راهی که می‌توانیم به‌طور نامحسوس فشار ایجاد کنیم، که منجر به مشکلاتی می‌شود که به‌تازگی درباره آن صحبت کردیم، این است که از تیم بخواهیم "velocity را افزایش دهند." این یعنی "بیشتر کار انجام دهید." تیم‌ها تلاش خواهند کرد این کار را انجام دهند. آنها این کار را با دستکاری نامرئی کیفیت و تخمین انجام می‌دهند. اگر کیفیت را کاهش دهند، ما با نقص‌های بیشتری روبرو خواهیم شد و کندتر پیش خواهیم رفت. ما چنین چیزی نمی‌خواهیم. تحت فشار، تیم‌ها به‌طور آگاهانه یا ناخودآگاه شروع به محافظه‌کارتر شدن در کارهایی که بر عهده می‌گیرند می‌کنند. آنها شروع به ارزیابی کارها به‌عنوان کمی بزرگ‌تر یا کمی سخت‌تر از قبل می‌کنند. این باعث می‌شود که به نظر برسد سریع‌تر پیش می‌روند، اما در واقع این‌طور نیست.  
  
یک تیم برای تولید کار بیشتر در واحد زمان به چه چیزی نیاز دارد یا چه چیزی باید بداند؟ آیا باور دارید که "سخت‌تر کار کردن" پاسخ این سؤال است؟ مدیریت چه کاری می‌تواند انجام دهد تا به تیم کمک کند واقعاً بهره‌وری بیشتری داشته باشد؟ آیا راه‌هایی برای سریع‌تر پیش رفتن وجود دارد؟ چگونه می‌توانیم این کار را انجام دهیم؟ اگر مجبورید تلاش کنید تا سریع‌تر پیش بروید، منابع تأخیر را تحلیل کنید. این منابع معمولاً تأثیر بیشتری نسبت به بهره‌وری فردی دارند. روی بهره‌وری تیمی کار کنید قبل از اینکه به بهره‌وری فردی بپردازید. اطمینان حاصل کنید که هر تیم ترکیب مهارت‌های خوبی دارد.  
  
[متن از صفحه قبلی: ما به ویژگی‌های بیشتری نیاز داریم. چرا نمی‌توانیم فقط velocity را افزایش دهیم؟ صادقانه بگویم، وقتی به سازمانی نگاه می‌کنم که "به ویژگی‌های بیشتری نیاز دارد"، تقریباً همیشه با سازمانی روبرو می‌شوم که نمی‌تواند "نه" بگوید. آنها به جای تصمیم‌گیرنده بودن، به سفارش‌گیرنده تبدیل شده‌اند. آنها شاید روی برخی چیزهای باارزش کار می‌کنند، اما بخش زیادی از کاری که انجام می‌دهند ارزش واقعی زیادی به محصولاتشان – یا به مشتریانشان – نمی‌آورد. نه، واقعاً ما به ویژگی‌های بیشتری نیاز داریم. برنامه‌نویسان باید سریع‌تر کار کنند. برنامه‌نویسان شما تا حد معقولی با بیشترین سرعت ممکن کار می‌کنند. ممکن است راه‌هایی وجود داشته باشد که کد با گذشت زمان بهبود یابد تا آنها بتوانند سریع‌تر کار کنند. تقریباً مطمئناً راه‌هایی برای کاهش تزریق نقص وجود دارد که زمان صرف‌شده برای رفع آنها را ذخیره می‌کند.]

================================================================================

# Section 108

## Original Text:

Those are improvements that take time to make, and  
they cannot be made by a team under pressure. One way that we can subtly cause pressure, leading to the  
problems we just talked about, is to ask the team to “increase  
velocity. ” This means “get more done. ” Teams will try to do  
that. They will do so by turning the invisible dials of quality  
and estimation. If they reduce quality, we’ll get more defects and go slower. We don’t want that. Under pressure, teams will consciously or unconsciously  
begin to be more conservative in what they take on. They  
will begin rating things as a bit larger or a bit harder than  
they used to. This will give the impression that they’re going  
faster, but they are not. What would a team need to do, or need to know, in order  
to produce more work per unit time. Do you believe that  
“work harder” answers that question. What can manage-  
ment do to help a team be truly more productive. Are there any ways to go faster. How can we do it. If you must try to go faster, analyze sources of delay. These  
usually have more impact than individual productivity. Work on team productivity before individual productivity. Ensure that each team has a good mix of skills. Make sure  
report erratum • discussChapter 17. Whip the Ponies Harder • 113]  
  
that each team has all the skills needed to do the work. Team  
members with key skills should be full time, not shared. Use  
specialists to increase other individuals’ abilities. Value  
working together. Increase individual productivity by increasing capability,  
not by urging people to work harder. “Work smarter, not  
harder” means that we have to help them get smarter. What are the real causes of delay on your project. Is decision  
making slowing you down. Defect correction. Handing  
things off between individuals or teams. What’s your delay. Work on that. Can we at least use our velocity to predict when we’ll be done. In a word, no.

## Translated Text:

این بهبودها زمان‌بر هستند و نمی‌توان آن‌ها را تحت فشار انجام داد. یکی از راه‌هایی که می‌توانیم به طور نامحسوس فشار ایجاد کنیم، که منجر به مشکلاتی می‌شود که به آن‌ها اشاره کردیم، این است که از تیم بخواهیم "velocity را افزایش دهند." این به معنای "بیشتر انجام دادن" است. تیم‌ها تلاش می‌کنند این کار را انجام دهند. آن‌ها این کار را با تغییر دادن دکمه‌های نامرئی کیفیت و تخمین انجام می‌دهند. اگر کیفیت را کاهش دهند، با نقص‌های بیشتری مواجه خواهیم شد و کندتر پیش خواهیم رفت. ما چنین چیزی نمی‌خواهیم.   
  
تحت فشار، تیم‌ها به صورت آگاهانه یا ناخودآگاه شروع به محافظه‌کارتر شدن در کارهایی که بر عهده می‌گیرند، می‌کنند. آن‌ها شروع به ارزیابی کارها به عنوان کمی بزرگ‌تر یا سخت‌تر از قبل می‌کنند. این باعث می‌شود که به نظر برسد سریع‌تر پیش می‌روند، اما در واقع این‌گونه نیست.   
  
یک تیم برای تولید کار بیشتر در واحد زمان به چه چیزی نیاز دارد یا باید چه چیزی بداند؟ آیا باور دارید که "سخت‌تر کار کردن" پاسخ این سؤال است؟ مدیریت چه کاری می‌تواند انجام دهد تا به تیم کمک کند واقعاً بهره‌وری بیشتری داشته باشد؟ آیا راه‌هایی برای سریع‌تر شدن وجود دارد؟ چگونه می‌توانیم این کار را انجام دهیم؟ اگر مجبورید سریع‌تر پیش بروید، منابع تأخیر را تحلیل کنید. این منابع معمولاً تأثیر بیشتری نسبت به بهره‌وری فردی دارند. ابتدا روی بهره‌وری تیم کار کنید، سپس بهره‌وری فردی.   
  
اطمینان حاصل کنید که هر تیم ترکیب مهارت‌های خوبی دارد. مطمئن شوید که هر تیم تمام مهارت‌های مورد نیاز برای انجام کار را دارد. اعضای تیم با مهارت‌های کلیدی باید به صورت تمام‌وقت کار کنند، نه به صورت مشترک. از متخصصان استفاده کنید تا توانایی‌های سایر افراد را افزایش دهید. به کار گروهی ارزش دهید.   
  
بهره‌وری فردی را با افزایش قابلیت‌ها افزایش دهید، نه با اصرار بر سخت‌تر کار کردن. "هوشمندانه‌تر کار کردن، نه سخت‌تر" به این معناست که باید به آن‌ها کمک کنیم هوشمندتر شوند. دلایل واقعی تأخیر در پروژه شما چیست؟ آیا تصمیم‌گیری شما را کند می‌کند؟ اصلاح نقص‌ها؟ انتقال کارها بین افراد یا تیم‌ها؟ تأخیر شما چیست؟ روی آن کار کنید.   
  
آیا می‌توانیم حداقل از velocity خود برای پیش‌بینی زمان پایان کار استفاده کنیم؟ در یک کلمه، خیر.

================================================================================

# Section 109

## Original Text:

We do best not when we predict when we’ll  
be done, but when we choose when to be done and then  
ensure that we put the best features in by that date. The  
Natural Way lets you do that by selecting important features  
and building a releasable version every couple of weeks. Whether we use estimated hours, story points, or just count  
stories, using velocity to project when we’ll be “done” is  
nearly always a sign of dysfunction. The premise of this book is that we want value, we want  
value soon, and we want highest value first. The best way  
to do this, as we’ve discussed, is always to be “done. ” If your  
organization has some fixed content in mind, toward which  
your team is working, you are likely “doing it wrong. ” You  
are likely not managing by value but by cost. This is a  
sucker’s game. Provide Real Opportunities to Upgrade Skills  
(See also Chapter 14, Creating Teams That Thrive , on page  
94. )  
What really helps a team be more productive is higher skill. This means that your investment in training and education  
will pay off in productivity. People work pretty hard. Is it sensible to imagine that they  
will then invest in their growth in the evening or on week-Chapter 17. Whip the Ponies Harder • 114  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: that each team has all the skills needed to do the work. Team  
members with key skills should be full time, not shared. Use  
specialists to increase other individuals’ abilities. Value  
working together. Increase individual productivity by increasing capability,  
not by urging people to work harder. “Work smarter, not  
harder” means that we have to help them get smarter. What are the real causes of delay on your project. Is decision  
making slowing you down. Defect correction. Handing  
things off between individuals or teams. What’s your delay. Work on that. Can we at least use our velocity to predict when we’ll be done. In a word, no.

## Translated Text:

ما بهترین عملکرد را زمانی داریم که پیش‌بینی نکنیم چه زمانی کار تمام می‌شود، بلکه زمانی که تصمیم بگیریم چه زمانی کار تمام شود و سپس اطمینان حاصل کنیم که بهترین ویژگی‌ها را تا آن تاریخ اضافه کرده‌ایم. روش طبیعی این امکان را به شما می‌دهد که با انتخاب ویژگی‌های مهم و Build کردن یک نسخه قابل Release هر چند هفته یک بار، این کار را انجام دهید.   
  
چه از estimated hours استفاده کنیم، چه از story points یا فقط شمارش stories، استفاده از velocity برای پیش‌بینی زمان "تمام شدن" تقریباً همیشه نشانه‌ای از ناکارآمدی است. فرض اصلی این کتاب این است که ما به دنبال ارزش هستیم، می‌خواهیم ارزش را زود به دست آوریم و می‌خواهیم بیشترین ارزش را در اولویت قرار دهیم. بهترین راه برای انجام این کار، همان‌طور که بحث کردیم، این است که همیشه "تمام شده" باشیم. اگر سازمان شما محتوای ثابتی را در نظر دارد که تیم شما به سمت آن کار می‌کند، احتمالاً "اشتباه عمل می‌کنید." احتمالاً شما بر اساس ارزش مدیریت نمی‌کنید، بلکه بر اساس هزینه مدیریت می‌کنید. این یک بازی بازنده است.   
  
فرصت‌های واقعی برای ارتقای مهارت‌ها فراهم کنید   
(همچنین به فصل 14، Creating Teams That Thrive، در صفحه 94 مراجعه کنید.)  
  
آنچه واقعاً به یک تیم کمک می‌کند تا بهره‌وری بیشتری داشته باشد، مهارت بالاتر است. این بدان معناست که سرمایه‌گذاری شما در آموزش و تحصیلات در بهره‌وری نتیجه خواهد داد. افراد به‌طور کلی سخت کار می‌کنند. آیا منطقی است که تصور کنیم آن‌ها سپس در عصرها یا آخر هفته‌ها روی رشد خود سرمایه‌گذاری کنند؟   
  
[فصل 17. Whip the Ponies Harder • صفحه 114]   
[گزارش خطا • بحث]  
  
[زمینه از صفحه قبلی: اینکه هر تیم تمام مهارت‌های لازم برای انجام کار را داشته باشد. اعضای تیم با مهارت‌های کلیدی باید تمام‌وقت باشند، نه مشترک. از متخصصان برای افزایش توانایی سایر افراد استفاده کنید. به همکاری ارزش دهید. بهره‌وری فردی را با افزایش قابلیت‌ها افزایش دهید، نه با اصرار به سخت‌تر کار کردن. "هوشمندانه‌تر کار کنید، نه سخت‌تر" به این معناست که باید به آن‌ها کمک کنیم تا هوشمندتر شوند. دلایل واقعی تأخیر در پروژه شما چیست؟ آیا تصمیم‌گیری شما را کند می‌کند؟ اصلاح Defect؟ انتقال کار بین افراد یا تیم‌ها؟ تأخیر شما چیست؟ روی آن کار کنید. آیا می‌توانیم حداقل از velocity خود برای پیش‌بینی زمان تمام شدن استفاده کنیم؟ به یک کلمه، خیر.]

================================================================================

# Section 110

## Original Text:

We do best not when we predict when we’ll  
be done, but when we choose when to be done and then  
ensure that we put the best features in by that date. The  
Natural Way lets you do that by selecting important features  
and building a releasable version every couple of weeks. Whether we use estimated hours, story points, or just count  
stories, using velocity to project when we’ll be “done” is  
nearly always a sign of dysfunction. The premise of this book is that we want value, we want  
value soon, and we want highest value first. The best way  
to do this, as we’ve discussed, is always to be “done. ” If your  
organization has some fixed content in mind, toward which  
your team is working, you are likely “doing it wrong. ” You  
are likely not managing by value but by cost. This is a  
sucker’s game. Provide Real Opportunities to Upgrade Skills  
(See also Chapter 14, Creating Teams That Thrive , on page  
94. )  
What really helps a team be more productive is higher skill. This means that your investment in training and education  
will pay off in productivity. People work pretty hard. Is it sensible to imagine that they  
will then invest in their growth in the evening or on week-Chapter 17. Whip the Ponies Harder • 114  
report erratum • discuss]  
  
ends, out of their limited personal time. What can we do to  
make that more likely. Most employees live on a fairly tight budget. Is it sensible  
to imagine that they’ll use their own vacation time and  
money to take courses or attend valuable conferences. What  
can we do to make sure that time and money for learning  
are available and used. Don’t “whip the ponies. ” Help them improve. report erratum • discussChapter 17. Whip the Ponies Harder • 115  
  
[CONTEXT FROM PREVIOUS PAGE: ends, out of their limited personal time. What can we do to  
make that more likely. Most employees live on a fairly tight budget. Is it sensible  
to imagine that they’ll use their own vacation time and  
money to take courses or attend valuable conferences.

## Translated Text:

ما بهترین عملکرد را زمانی داریم که پیش‌بینی نکنیم چه زمانی کار تمام می‌شود، بلکه زمانی که تصمیم بگیریم چه زمانی کار تمام شود و سپس اطمینان حاصل کنیم که بهترین ویژگی‌ها را تا آن تاریخ اضافه کنیم. روش طبیعی به شما این امکان را می‌دهد که با انتخاب ویژگی‌های مهم و ساخت یک نسخه قابل انتشار هر چند هفته یک بار، این کار را انجام دهید. چه از ساعات تخمینی، Story Points یا فقط شمارش User Storyها استفاده کنیم، استفاده از Velocity برای پیش‌بینی زمان "تمام شدن" تقریباً همیشه نشانه‌ای از ناکارآمدی است.   
  
فرض اصلی این کتاب این است که ما به دنبال ارزش هستیم، ارزش را زود می‌خواهیم و بالاترین ارزش را در اولویت قرار می‌دهیم. بهترین راه برای انجام این کار، همان‌طور که بحث کردیم، این است که همیشه "تمام شده" باشیم. اگر سازمان شما محتوای ثابتی در ذهن دارد که تیم شما به سمت آن کار می‌کند، احتمالاً "اشتباه عمل می‌کنید." احتمالاً شما بر اساس ارزش مدیریت نمی‌کنید، بلکه بر اساس هزینه مدیریت می‌کنید. این یک بازی بازنده است.  
  
### فراهم کردن فرصت‌های واقعی برای ارتقای مهارت‌ها   
(به فصل ۱۴، \*\*Creating Teams That Thrive\*\*، صفحه ۹۴ نیز مراجعه کنید.)  
  
آنچه واقعاً به یک تیم کمک می‌کند تا بهره‌وری بیشتری داشته باشد، مهارت بالاتر است. این بدان معناست که سرمایه‌گذاری شما در آموزش و تحصیلات، در بهره‌وری نتیجه خواهد داد. افراد به‌سختی کار می‌کنند. آیا منطقی است تصور کنیم که آن‌ها در عصرها یا آخر هفته‌ها، از زمان شخصی محدود خود برای رشد خود سرمایه‌گذاری کنند؟   
  
چه کاری می‌توانیم انجام دهیم تا این احتمال را بیشتر کنیم؟ بیشتر کارمندان با بودجه‌ای نسبتاً محدود زندگی می‌کنند. آیا منطقی است تصور کنیم که آن‌ها از زمان تعطیلات و پول خود برای شرکت در دوره‌ها یا کنفرانس‌های ارزشمند استفاده کنند؟ چه کاری می‌توانیم انجام دهیم تا مطمئن شویم که زمان و پول برای یادگیری در دسترس است و استفاده می‌شود؟   
  
"اسب‌ها را شلاق نزنید." به آن‌ها کمک کنید بهتر شوند.   
\*\*report erratum • discuss\*\*   
\*\*Chapter 17. Whip the Ponies Harder • 115\*\*

================================================================================

# Section 111

## Original Text:

What  
can we do to make sure that time and money for learning  
are available and used. Don’t “whip the ponies. ” Help them improve. report erratum • discussChapter 17. Whip the Ponies Harder • 115]  
  
CHAPTER 18  
To Speed Up, Build with Skill  
At first, when we set out to build a complete but tiny product  
in easy, small cycles, things will go badly. That’s normal. Let’s look at a few of the main things that can go wrong and  
what we can do about them. Think in terms of the style of  
these suggestions, not just the specifics. Our team says: We can’t get all these features done in two weeks. Yes, this is a common problem. Often when starting out, a  
team can’t deliver a fully integrated product increment in  
two weeks and will ask for more time. Instead of giving your  
team more time, I suggest giving them less. Ask your team  
to produce a fully integrated product increment in one week  
instead of two. Chances are, they’ll figure out how to do it. They’ll see that  
they need to make just a few actual changes, and then focus  
their attention on getting those changes integrated, tested,  
and running. Once in a while—and I see this more when talking about it  
than when doing it—the team will come to me and say, “We  
can’t get anything done in a week. ” I generally ask them,  
“Well, can you get anything done in a day, then. ” They’ll  
reply that they can’t, and I’ll ask them why they plan to come  
in tomorrow if they aren’t going to do anything. Ha, ha, very  
funny. Except I’m serious. This generally gets the conversation aimed at why it’s diffi-  
cult to produce an increment. And once we’re dealing with  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 18  
To Speed Up, Build with Skill  
At first, when we set out to build a complete but tiny product  
in easy, small cycles, things will go badly. That’s normal. Let’s look at a few of the main things that can go wrong and  
what we can do about them. Think in terms of the style of  
these suggestions, not just the specifics.

## Translated Text:

چه کاری می‌توانیم انجام دهیم تا اطمینان حاصل کنیم که زمان و هزینه برای یادگیری در دسترس بوده و استفاده می‌شود؟ "اسب‌ها را شلاق نزنید." به آنها کمک کنید تا پیشرفت کنند.   
گزارش خطا • بحث   
فصل ۱۷. سخت‌تر شلاق زدن به اسب‌ها • ۱۱۵   
  
### فصل ۱۸   
برای سرعت بخشیدن، با مهارت Build کنید   
  
در ابتدا، وقتی تصمیم می‌گیریم یک محصول کامل اما کوچک را در چرخه‌های آسان و کوچک Build کنیم، اوضاع بد پیش خواهد رفت. این طبیعی است. بیایید به چند مورد از مشکلات اصلی که ممکن است پیش بیاید و کارهایی که می‌توانیم در مورد آنها انجام دهیم نگاه کنیم. به سبک این پیشنهادات فکر کنید، نه فقط به جزئیات خاص آنها.   
  
تیم ما می‌گوید: ما نمی‌توانیم تمام این Features را در دو هفته انجام دهیم.   
بله، این یک مشکل رایج است. اغلب وقتی شروع می‌کنیم، یک تیم نمی‌تواند یک Product Increment کاملاً یکپارچه را در دو هفته تحویل دهد و درخواست زمان بیشتری می‌کند. به جای دادن زمان بیشتر به تیم خود، پیشنهاد می‌کنم زمان کمتری به آنها بدهید. از تیم خود بخواهید که یک Product Increment کاملاً یکپارچه را در یک هفته به جای دو هفته تولید کند.   
  
احتمالاً آنها راهی برای انجام آن پیدا خواهند کرد. آنها خواهند دید که فقط نیاز به ایجاد چند تغییر واقعی دارند و سپس تمرکز خود را بر روی یکپارچه‌سازی، Test کردن و اجرای آن تغییرات قرار خواهند داد.   
  
گاهی اوقات—و من این را بیشتر هنگام صحبت در مورد آن می‌بینم تا هنگام انجام آن—تیم به من می‌گوید: "ما نمی‌توانیم هیچ کاری را در یک هفته انجام دهیم." من معمولاً از آنها می‌پرسم: "خب، آیا می‌توانید کاری را در یک روز انجام دهید؟" آنها پاسخ خواهند داد که نمی‌توانند، و من از آنها می‌پرسم چرا برنامه دارند فردا بیایند اگر قرار نیست کاری انجام دهند. ها، ها، خیلی خنده‌دار. به جز اینکه من جدی هستم.   
  
این معمولاً گفتگو را به سمت اینکه چرا تولید یک Increment دشوار است هدایت می‌کند. و وقتی که ما با این موضوع سروکار داریم...   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: فصل ۱۸   
برای سرعت بخشیدن، با مهارت Build کنید   
در ابتدا، وقتی تصمیم می‌گیریم یک محصول کامل اما کوچک را در چرخه‌های آسان و کوچک Build کنیم، اوضاع بد پیش خواهد رفت. این طبیعی است. بیایید به چند مورد از مشکلات اصلی که ممکن است پیش بیاید و کارهایی که می‌توانیم در مورد آنها انجام دهیم نگاه کنیم. به سبک این پیشنهادات فکر کنید، نه فقط به جزئیات خاص آنها.]

================================================================================

# Section 112

## Original Text:

Our team says: We can’t get all these features done in two weeks. Yes, this is a common problem. Often when starting out, a  
team can’t deliver a fully integrated product increment in  
two weeks and will ask for more time. Instead of giving your  
team more time, I suggest giving them less. Ask your team  
to produce a fully integrated product increment in one week  
instead of two. Chances are, they’ll figure out how to do it. They’ll see that  
they need to make just a few actual changes, and then focus  
their attention on getting those changes integrated, tested,  
and running. Once in a while—and I see this more when talking about it  
than when doing it—the team will come to me and say, “We  
can’t get anything done in a week. ” I generally ask them,  
“Well, can you get anything done in a day, then. ” They’ll  
reply that they can’t, and I’ll ask them why they plan to come  
in tomorrow if they aren’t going to do anything. Ha, ha, very  
funny. Except I’m serious. This generally gets the conversation aimed at why it’s diffi-  
cult to produce an increment. And once we’re dealing with  
report erratum • discuss]  
  
why, it’s easy to figure out things to do. The answers usually  
come down to build times, integration times, and test times. If the problems still aren’t clear, let’s ask the team to rebuild  
the system and get it ready to deploy without making any  
changes at all. If we can’t do that in a week, we’ll sure know  
where the problem is, won’t we. One way or another our software-building process is slow  
and unreliable. Great. We’re computer programmers and  
we can fix that. As Product Champion or manager, you want this fixed more  
than the team does. You need to see a completed product  
increment, and you need to see it on a regular basis. So  
there’s incredibly high value in getting that increment built. That’s why asking for a single feature, or a tiny feature, or  
even no feature at all is a good business decision. You’re not  
asking them to focus on some technical thing: never do that.

## Translated Text:

تیم ما می‌گوید: ما نمی‌توانیم همه این Features را در دو هفته انجام دهیم. بله، این یک مشکل رایج است. اغلب وقتی کار را شروع می‌کنید، یک تیم نمی‌تواند یک Product Increment کاملاً یکپارچه را در دو هفته ارائه دهد و درخواست زمان بیشتری می‌کند. به جای دادن زمان بیشتر به تیم، پیشنهاد می‌کنم زمان کمتری بدهید. از تیم خود بخواهید که یک Product Increment کاملاً یکپارچه را در یک هفته به جای دو هفته تولید کند. احتمالاً آن‌ها راهی برای انجام این کار پیدا خواهند کرد. آن‌ها خواهند دید که فقط نیاز به ایجاد چند تغییر واقعی دارند و سپس تمرکز خود را بر روی یکپارچه‌سازی، Test کردن و Run کردن آن تغییرات قرار خواهند داد.  
  
گاهی اوقات—و من این را بیشتر وقتی در موردش صحبت می‌کنم می‌بینم تا وقتی انجامش می‌دهم—تیم به من می‌گوید، "ما نمی‌توانیم هیچ کاری را در یک هفته انجام دهیم." من معمولاً از آن‌ها می‌پرسم، "خب، آیا می‌توانید کاری را در یک روز انجام دهید؟" آن‌ها پاسخ می‌دهند که نمی‌توانند، و من از آن‌ها می‌پرسم چرا برنامه دارند فردا بیایند اگر قرار نیست کاری انجام دهند. ها ها، خیلی خنده‌دار. اما من جدی هستم. این معمولاً مکالمه را به سمت دلیل دشواری تولید یک Increment هدایت می‌کند. و وقتی داریم با "چرا" سروکار داریم، پیدا کردن کارهایی که باید انجام شود آسان می‌شود. پاسخ‌ها معمولاً به Build Times، Integration Times و Test Times برمی‌گردد.  
  
اگر مشکلات هنوز واضح نیستند، از تیم بخواهیم که سیستم را دوباره Build کند و آن را بدون ایجاد هیچ تغییری برای Deploy آماده کند. اگر نتوانیم این کار را در یک هفته انجام دهیم، مطمئناً می‌دانیم مشکل کجاست، اینطور نیست؟ به هر حال، فرآیند Software-Building ما کند و غیرقابل اعتماد است. عالی. ما برنامه‌نویس کامپیوتر هستیم و می‌توانیم این را درست کنیم. به عنوان Product Champion یا مدیر، شما بیشتر از تیم می‌خواهید که این مشکل حل شود. شما نیاز دارید یک Product Increment کامل شده را ببینید، و باید آن را به طور منظم ببینید. بنابراین ارزش فوق‌العاده‌ای در ساخته شدن آن Increment وجود دارد.  
  
به همین دلیل درخواست یک Feature، یا یک Feature کوچک، یا حتی هیچ Feature اصلاً، یک تصمیم تجاری خوب است. شما از آن‌ها نمی‌خواهید که بر روی یک چیز فنی تمرکز کنند: هرگز این کار را نکنید.

================================================================================

# Section 113

## Original Text:

You’re asking them to show you the software. Our team says: We can’t get anything ready for release until after  
“they” test it. As your team’s Product Champion or manager, you know  
that’s just not acceptable. When defects come back from QA,  
they interfere with our business planning, because we have  
to do things over again and cannot focus our attention on  
our business purpose. So the solution is very simple, and very difficult: When the  
team ships software to QA, we must already be certain that  
it will not be coming back with defects in it. We do that by testing it ourselves. The technical practices of  
acceptance test-driven development (ATDD) and test-driven  
development (TDD) enable us to do this quite nicely. We  
can be increasingly confident that when we pass something  
on, it will work as desired. Our teams must be experts in  
ATDD and TDD. Sometimes the external dependency is not on testing but on  
some other “resource,” perhaps a DBA department or a user  
interface design department. The fix for this is to make your  
team cross-functional. That is, import these skills to your  
report erratum • discussChapter 18. To Speed Up, Build with Skill • 117  
  
[CONTEXT FROM PREVIOUS PAGE: why, it’s easy to figure out things to do. The answers usually  
come down to build times, integration times, and test times. If the problems still aren’t clear, let’s ask the team to rebuild  
the system and get it ready to deploy without making any  
changes at all. If we can’t do that in a week, we’ll sure know  
where the problem is, won’t we. One way or another our software-building process is slow  
and unreliable. Great. We’re computer programmers and  
we can fix that. As Product Champion or manager, you want this fixed more  
than the team does. You need to see a completed product  
increment, and you need to see it on a regular basis. So  
there’s incredibly high value in getting that increment built.

## Translated Text:

شما از آن‌ها می‌خواهید نرم‌افزار را به شما نشان دهند. تیم ما می‌گوید: ما نمی‌توانیم چیزی برای Release آماده کنیم تا زمانی که "آن‌ها" آن را Test کنند. به‌عنوان Product Champion یا مدیر تیم خود، می‌دانید که این اصلاً قابل قبول نیست. وقتی که اشکالات از QA برمی‌گردند، در برنامه‌ریزی کسب‌وکار ما اختلال ایجاد می‌کنند، زیرا مجبوریم دوباره کارها را انجام دهیم و نمی‌توانیم تمرکز خود را روی اهداف کسب‌وکارمان بگذاریم.   
  
بنابراین راه‌حل بسیار ساده و در عین حال بسیار دشوار است: وقتی تیم نرم‌افزار را به QA ارسال می‌کند، باید از قبل مطمئن باشیم که با اشکالات باز نخواهد گشت. ما این کار را با Test کردن توسط خودمان انجام می‌دهیم. روش‌های فنی acceptance test-driven development (ATDD) و test-driven development (TDD) به ما این امکان را می‌دهند که این کار را به‌خوبی انجام دهیم. ما می‌توانیم با اطمینان بیشتری بگوییم که وقتی چیزی را منتقل می‌کنیم، همان‌طور که انتظار می‌رود کار خواهد کرد. تیم‌های ما باید در ATDD و TDD متخصص باشند.   
  
گاهی اوقات وابستگی خارجی به Test نیست، بلکه به یک "منبع" دیگر مانند یک DBA department یا یک user interface design department مربوط می‌شود. راه‌حل این است که تیم خود را cross-functional کنید. به این معنا که این مهارت‌ها را به تیم خود وارد کنید.  
  
[متن از صفحه قبلی: چرا، پیدا کردن کارهایی که باید انجام شود آسان است. پاسخ‌ها معمولاً به زمان‌های Build، زمان‌های Integration و زمان‌های Test برمی‌گردند. اگر مشکلات هنوز مشخص نیستند، از تیم بخواهیم سیستم را دوباره Build کنند و آن را برای Deploy آماده کنند بدون اینکه هیچ تغییری ایجاد کنند. اگر نتوانیم این کار را در یک هفته انجام دهیم، مطمئناً متوجه خواهیم شد که مشکل کجاست، این‌طور نیست؟ به هر طریقی، فرآیند ساخت نرم‌افزار ما کند و غیرقابل اعتماد است. عالی. ما برنامه‌نویس کامپیوتر هستیم و می‌توانیم این مشکل را حل کنیم. به‌عنوان Product Champion یا مدیر، شما بیشتر از تیم می‌خواهید که این مشکل حل شود. شما نیاز دارید یک محصول کامل شده را ببینید و آن را به‌طور منظم مشاهده کنید. بنابراین ارزش بسیار بالایی در ساخت آن increment وجود دارد.]

================================================================================

# Section 114

## Original Text:

That’s why asking for a single feature, or a tiny feature, or  
even no feature at all is a good business decision. You’re not  
asking them to focus on some technical thing: never do that. You’re asking them to show you the software. Our team says: We can’t get anything ready for release until after  
“they” test it. As your team’s Product Champion or manager, you know  
that’s just not acceptable. When defects come back from QA,  
they interfere with our business planning, because we have  
to do things over again and cannot focus our attention on  
our business purpose. So the solution is very simple, and very difficult: When the  
team ships software to QA, we must already be certain that  
it will not be coming back with defects in it. We do that by testing it ourselves. The technical practices of  
acceptance test-driven development (ATDD) and test-driven  
development (TDD) enable us to do this quite nicely. We  
can be increasingly confident that when we pass something  
on, it will work as desired. Our teams must be experts in  
ATDD and TDD. Sometimes the external dependency is not on testing but on  
some other “resource,” perhaps a DBA department or a user  
interface design department. The fix for this is to make your  
team cross-functional. That is, import these skills to your  
report erratum • discussChapter 18. To Speed Up, Build with Skill • 117]  
  
team. And the best way to do that is to assign people from  
those departments as real team members on our important  
projects. But the team says: It still takes time to get really done. That’s OK. We want to take all the time needed to get done  
“the first time. ” Learning to build a solid product increment  
will take time, especially if our team is struggling with a  
load of code written in old-fashioned ways. That’s not  
unsolvable. The Agile method Scrum has a notion that will  
help us here: the evolving “Definition of Done. ”  
We need our team to produce a product increment in every  
“Sprint.

## Translated Text:

به همین دلیل درخواست یک Feature واحد، یا یک Feature کوچک، یا حتی هیچ Feature، یک تصمیم تجاری خوب است. شما از آن‌ها نمی‌خواهید که روی یک موضوع فنی تمرکز کنند: هرگز این کار را نکنید. شما از آن‌ها می‌خواهید که نرم‌افزار را به شما نشان دهند. تیم ما می‌گوید: ما نمی‌توانیم چیزی را برای Release آماده کنیم تا زمانی که "آن‌ها" آن را Test کنند. به‌عنوان Product Champion یا مدیر تیم خود، می‌دانید که این قابل‌قبول نیست. وقتی Defectها از QA برمی‌گردند، در برنامه‌ریزی تجاری ما اختلال ایجاد می‌کنند، زیرا مجبوریم دوباره کارها را انجام دهیم و نمی‌توانیم تمرکز خود را بر هدف تجاری‌مان حفظ کنیم.  
  
بنابراین راه‌حل بسیار ساده و در عین حال بسیار دشوار است: وقتی تیم نرم‌افزار را به QA ارسال می‌کند، باید از قبل مطمئن باشیم که با Defect برنمی‌گردد. این کار را با Test کردن توسط خودمان انجام می‌دهیم. روش‌های فنی مانند Acceptance Test-Driven Development (ATDD) و Test-Driven Development (TDD) به ما این امکان را می‌دهند که این کار را به‌خوبی انجام دهیم. ما می‌توانیم با اطمینان بیشتری بدانیم که وقتی چیزی را انتقال می‌دهیم، همان‌طور که انتظار می‌رود کار خواهد کرد. تیم‌های ما باید در ATDD و TDD متخصص باشند.  
  
گاهی اوقات وابستگی خارجی به Testing نیست، بلکه به یک "منبع" دیگر است، شاید یک DBA department یا یک User Interface Design department. راه‌حل این است که تیم خود را Cross-Functional کنید. یعنی این مهارت‌ها را به تیم خود وارد کنید. و بهترین راه برای انجام این کار این است که افراد آن دپارتمان‌ها را به‌عنوان اعضای واقعی تیم در پروژه‌های مهم خود اختصاص دهید.  
  
اما تیم می‌گوید: هنوز زمان می‌برد تا واقعاً کار تمام شود. مشکلی نیست. ما می‌خواهیم تمام زمانی که لازم است برای انجام کار "در اولین بار" صرف کنیم. یادگیری ساخت یک Product Increment محکم زمان می‌برد، به‌ویژه اگر تیم ما با حجم زیادی از کدی که به روش‌های قدیمی نوشته شده است، دست‌وپنجه نرم می‌کند. این غیرقابل‌حل نیست. روش Agile به نام Scrum یک مفهوم دارد که در اینجا به ما کمک خواهد کرد: "Definition of Done" که در حال تکامل است.  
  
ما به تیم خود نیاز داریم که یک Product Increment را در هر "Sprint" تولید کند.

================================================================================

# Section 115

## Original Text:

” That increment must meet the current Definition  
of Done. And the Definition of Done must become more  
stringent over time. So in early cycles, we might not even require that the incre-  
ment be integrated. We look at each feature one at a time,  
running on some developer’s machine. This should make a  
business-side person Very Nervous: how do we know that  
these things will work together. So we improve our Defini-  
tion of Done to say that all the features must work on a single  
development machine, in a single build. The team will have  
to take on fewer features for a bit while they get this to  
happen, but since we want them to show us the software,  
it’s a good business decision. Next, we may find that it’s integrated, but it doesn’t work  
very well. It needs more testing. So we add some testing  
criteria to our Definition of Done. This will lead us to discover that we need to know at the  
beginning of the cycle how the software will be tested at the  
end. This leads us to get better at getting examples from the  
Product Champion, and to get better at building a common  
understanding of what we need. And so it goes. With a single relentless focus on “show us  
the software,” we improve the team’s process so that when  
they take on some amount of work, they’ll actually get it  
done. Now we all say: It seems so SLOW. Chapter 18. To Speed Up, Build with Skill • 118  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: team. And the best way to do that is to assign people from  
those departments as real team members on our important  
projects. But the team says: It still takes time to get really done. That’s OK. We want to take all the time needed to get done  
“the first time. ” Learning to build a solid product increment  
will take time, especially if our team is struggling with a  
load of code written in old-fashioned ways. That’s not  
unsolvable. The Agile method Scrum has a notion that will  
help us here: the evolving “Definition of Done.

## Translated Text:

آن Increment باید با Definition of Done فعلی مطابقت داشته باشد. و Definition of Done باید به مرور زمان سخت‌گیرانه‌تر شود. بنابراین در چرخه‌های اولیه، ممکن است حتی نیازی نباشد که Increment یکپارچه (Integrated) باشد. ما هر Feature را به صورت جداگانه بررسی می‌کنیم که روی ماشین یکی از توسعه‌دهندگان Run می‌شود. این موضوع باید یک فرد در سمت کسب‌وکار را بسیار نگران کند: چگونه می‌دانیم که این موارد با هم کار خواهند کرد؟ بنابراین Definition of Done خود را بهبود می‌دهیم تا بگوییم که تمام Features باید روی یک ماشین توسعه و در یک Build کار کنند. تیم باید برای مدتی Features کمتری را بر عهده بگیرد تا این اتفاق بیفتد، اما از آنجا که می‌خواهیم نرم‌افزار را به ما نشان دهند، این یک تصمیم تجاری خوب است.  
  
سپس ممکن است متوجه شویم که یکپارچه شده است، اما خیلی خوب کار نمی‌کند. به تست بیشتری نیاز دارد. بنابراین برخی معیارهای تست را به Definition of Done خود اضافه می‌کنیم. این ما را به این کشف می‌رساند که باید در ابتدای چرخه بدانیم که نرم‌افزار در انتها چگونه تست خواهد شد. این ما را به بهبود در گرفتن مثال‌ها از Product Champion و بهبود در ایجاد یک درک مشترک از آنچه نیاز داریم، هدایت می‌کند. و این روند ادامه دارد. با تمرکز بی‌وقفه بر "نرم‌افزار را به ما نشان دهید"، فرآیند تیم را بهبود می‌دهیم تا زمانی که مقداری کار را بر عهده می‌گیرند، واقعاً آن را انجام دهند. حالا همه ما می‌گوییم: به نظر می‌رسد خیلی کند است.  
  
فصل ۱۸. برای سرعت بخشیدن، با مهارت Build کنید • ۱۱۸   
report erratum • discuss   
  
[متن از صفحه قبلی: تیم. و بهترین راه برای انجام این کار این است که افراد آن بخش‌ها را به عنوان اعضای واقعی تیم در پروژه‌های مهم خود اختصاص دهیم. اما تیم می‌گوید: هنوز زمان می‌برد تا واقعاً Done شود. مشکلی نیست. ما می‌خواهیم تمام زمانی که لازم است را صرف کنیم تا "بار اول" Done شود. یادگیری ساخت یک Product Increment محکم زمان می‌برد، به‌ویژه اگر تیم ما با حجم زیادی از کدی که به روش‌های قدیمی نوشته شده است، دست و پنجه نرم می‌کند. این غیرقابل حل نیست. روش Agile به نام Scrum مفهومی دارد که در اینجا به ما کمک می‌کند: Definition of Done که به مرور تکامل می‌یابد.]

================================================================================

# Section 116

## Original Text:

”  
We need our team to produce a product increment in every  
“Sprint. ” That increment must meet the current Definition  
of Done. And the Definition of Done must become more  
stringent over time. So in early cycles, we might not even require that the incre-  
ment be integrated. We look at each feature one at a time,  
running on some developer’s machine. This should make a  
business-side person Very Nervous: how do we know that  
these things will work together. So we improve our Defini-  
tion of Done to say that all the features must work on a single  
development machine, in a single build. The team will have  
to take on fewer features for a bit while they get this to  
happen, but since we want them to show us the software,  
it’s a good business decision. Next, we may find that it’s integrated, but it doesn’t work  
very well. It needs more testing. So we add some testing  
criteria to our Definition of Done. This will lead us to discover that we need to know at the  
beginning of the cycle how the software will be tested at the  
end. This leads us to get better at getting examples from the  
Product Champion, and to get better at building a common  
understanding of what we need. And so it goes. With a single relentless focus on “show us  
the software,” we improve the team’s process so that when  
they take on some amount of work, they’ll actually get it  
done. Now we all say: It seems so SLOW. Chapter 18. To Speed Up, Build with Skill • 118  
report erratum • discuss]  
  
It should feel slow at first. Later on, it will feel smooth and  
efficient, never rushed. We absolutely need a releasable,  
done, product increment every couple of weeks. To accom-  
plish this, our work must be fully designed, fully tested,  
complete in every way. This means the team must take on  
less work in each cycle than we’re used to. It will feel slow. The difference is, when we’re done, we’re really done. Since we’re getting closer to really done, the apparent  
slowness is an illusion.

## Translated Text:

ما به تیم خود نیاز داریم که در هر Sprint یک product increment تولید کند. این increment باید با Definition of Done فعلی مطابقت داشته باشد. و Definition of Done باید به مرور زمان سخت‌گیرانه‌تر شود. بنابراین در چرخه‌های اولیه، ممکن است حتی نیازی نباشد که increment یکپارچه (integrated) باشد. ما هر feature را به صورت جداگانه و روی ماشین یکی از توسعه‌دهندگان بررسی می‌کنیم. این موضوع باید یک فرد از بخش کسب‌وکار را بسیار نگران کند: چگونه می‌دانیم که این موارد با هم کار خواهند کرد؟ بنابراین Definition of Done خود را بهبود می‌دهیم تا بگوییم که همه‌ی features باید روی یک ماشین توسعه و در یک build کار کنند. تیم باید برای مدتی تعداد کمتری feature را انجام دهد تا این اتفاق بیفتد، اما از آنجا که ما می‌خواهیم نرم‌افزار را به ما نشان دهند، این یک تصمیم تجاری خوب است.  
  
سپس ممکن است متوجه شویم که نرم‌افزار یکپارچه شده است، اما خیلی خوب کار نمی‌کند. به تست بیشتری نیاز دارد. بنابراین برخی معیارهای تست را به Definition of Done اضافه می‌کنیم. این ما را به این نتیجه می‌رساند که باید در ابتدای چرخه بدانیم که نرم‌افزار در انتها چگونه تست خواهد شد. این ما را به بهبود در دریافت مثال‌ها از Product Champion و بهبود در ایجاد یک درک مشترک از آنچه نیاز داریم، هدایت می‌کند. و این روند ادامه دارد. با تمرکز بی‌وقفه بر "نرم‌افزار را به ما نشان دهید"، فرآیند تیم را بهبود می‌دهیم تا زمانی که مقداری کار را بر عهده می‌گیرند، واقعاً آن را به پایان برسانند. اکنون همه ما می‌گوییم: به نظر می‌رسد خیلی کند است.  
  
فصل ۱۸. برای سرعت بخشیدن، با مهارت بسازید • ۱۱۸   
[report erratum • discuss]  
  
در ابتدا باید احساس کندی کنید. بعداً این روند روان و کارآمد خواهد شد، بدون هیچ عجله‌ای. ما مطلقاً به یک product increment که قابل انتشار (releasable) و تمام‌شده (done) باشد، هر چند هفته یک‌بار نیاز داریم. برای دستیابی به این هدف، کار ما باید به طور کامل طراحی شده، به طور کامل تست شده و از هر نظر کامل باشد. این بدان معناست که تیم باید در هر چرخه کار کمتری نسبت به آنچه عادت داریم بر عهده بگیرد. این روند کند به نظر خواهد رسید. تفاوت این است که وقتی کار تمام می‌شود، واقعاً تمام شده است. از آنجا که ما به "واقعاً تمام‌شده" نزدیک‌تر می‌شویم، کندی ظاهری یک توهم است.

================================================================================

# Section 117

## Original Text:

We’re be eliminating most of that  
long, painful test-and-fix finish that wears on long after we  
were supposed to be done. Did you forget that little detail,  
all that time you spend after your application was supposed  
to launch, or after it does launch. Second, since we’re working incrementally, high value first,  
there’s every chance to deliver useful value before the  
deadline. But we can only do that if the software actually  
works. In early days, yes, it will feel slow. But even in early days,  
it may not really be slower: defects are being reduced, and  
cleaner code will keep the work flowing. Let it be. As your  
team gets good at really being done, they’ll be able to take  
on more and more features, and things will improve more  
and more. Summary  
The most valuable thing you can do to speed up develop-  
ment is to build skill in the development team. That invest-  
ment will pay off rapidly in less time lost to fixing defects,  
and in smoother development. Don’t confuse thrashing with  
effective performance. The fastest teams move smoothly and  
gracefully. report erratum • discussChapter 18. To Speed Up, Build with Skill • 119  
  
[CONTEXT FROM PREVIOUS PAGE: It should feel slow at first. Later on, it will feel smooth and  
efficient, never rushed. We absolutely need a releasable,  
done, product increment every couple of weeks. To accom-  
plish this, our work must be fully designed, fully tested,  
complete in every way. This means the team must take on  
less work in each cycle than we’re used to. It will feel slow. The difference is, when we’re done, we’re really done. Since we’re getting closer to really done, the apparent  
slowness is an illusion. We’re be eliminating most of that  
long, painful test-and-fix finish that wears on long after we  
were supposed to be done. Did you forget that little detail,  
all that time you spend after your application was supposed  
to launch, or after it does launch.

## Translated Text:

ما بیشتر آن فرآیند طولانی و دردناک تست و رفع اشکال که مدت‌ها پس از زمانی که قرار بود کار تمام شود ادامه پیدا می‌کند را حذف خواهیم کرد. آیا آن جزئیات کوچک را فراموش کرده‌اید؟ تمام آن زمانی که پس از زمانی که قرار بود اپلیکیشن شما Launch شود یا حتی پس از Launch شدن آن صرف می‌کنید؟   
  
دوم، از آنجا که ما به صورت تدریجی و با اولویت‌بندی ارزش‌های بالا کار می‌کنیم، هر فرصتی وجود دارد که ارزش مفیدی را پیش از Deadline ارائه دهیم. اما ما فقط در صورتی می‌توانیم این کار را انجام دهیم که نرم‌افزار واقعاً کار کند. در روزهای ابتدایی، بله، ممکن است کند به نظر برسد. اما حتی در روزهای ابتدایی، ممکن است واقعاً کندتر نباشد: Defectها در حال کاهش هستند و کد تمیزتر باعث می‌شود کار به جریان بیفتد. اجازه دهید همین‌طور باشد. وقتی تیم شما در واقعاً تمام کردن کارها مهارت پیدا کند، قادر خواهند بود Featureهای بیشتری را به عهده بگیرند و اوضاع بیشتر و بیشتر بهبود خواهد یافت.   
  
\*\*خلاصه\*\*   
باارزش‌ترین کاری که می‌توانید برای سرعت بخشیدن به توسعه انجام دهید، ایجاد مهارت در تیم توسعه است. این سرمایه‌گذاری به سرعت با کاهش زمان از دست رفته برای رفع Defectها و توسعه روان‌تر نتیجه خواهد داد. سردرگمی را با عملکرد مؤثر اشتباه نگیرید. سریع‌ترین تیم‌ها به صورت روان و با ظرافت حرکت می‌کنند.   
  
\*\*گزارش خطا • بحث\*\*   
فصل ۱۸. برای سرعت بخشیدن، با مهارت بسازید • ۱۱۹   
  
[متن از صفحه قبلی: در ابتدا ممکن است کند به نظر برسد. بعداً، احساس روان و کارآمدی خواهد داشت، هرگز عجله‌ای در کار نخواهد بود. ما به طور قطع به یک Product Increment قابل Release و تمام‌شده هر چند هفته یک بار نیاز داریم. برای دستیابی به این هدف، کار ما باید به طور کامل طراحی شده، به طور کامل Test شده و از هر نظر کامل باشد. این بدان معناست که تیم باید در هر چرخه کاری کمتری را نسبت به آنچه به آن عادت داریم به عهده بگیرد. این کار کند به نظر خواهد رسید. تفاوت این است که وقتی کار ما تمام می‌شود، واقعاً تمام شده است. از آنجا که ما به واقعاً تمام شدن نزدیک‌تر می‌شویم، کندی ظاهری یک توهم است. ما بیشتر آن فرآیند طولانی و دردناک تست و رفع اشکال که مدت‌ها پس از زمانی که قرار بود کار تمام شود ادامه پیدا می‌کند را حذف خواهیم کرد. آیا آن جزئیات کوچک را فراموش کرده‌اید؟ تمام آن زمانی که پس از زمانی که قرار بود اپلیکیشن شما Launch شود یا حتی پس از Launch شدن آن صرف می‌کنید؟]

================================================================================

# Section 118

## Original Text:

Second, since we’re working incrementally, high value first,  
there’s every chance to deliver useful value before the  
deadline. But we can only do that if the software actually  
works. In early days, yes, it will feel slow. But even in early days,  
it may not really be slower: defects are being reduced, and  
cleaner code will keep the work flowing. Let it be. As your  
team gets good at really being done, they’ll be able to take  
on more and more features, and things will improve more  
and more. Summary  
The most valuable thing you can do to speed up develop-  
ment is to build skill in the development team. That invest-  
ment will pay off rapidly in less time lost to fixing defects,  
and in smoother development. Don’t confuse thrashing with  
effective performance. The fastest teams move smoothly and  
gracefully. report erratum • discussChapter 18. To Speed Up, Build with Skill • 119]  
  
CHAPTER 19  
Refactoring  
Reminder: We need steady progress. To keep progress steady, we  
need a clear, clean design all the time. To accomplish this, we must  
refactor. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 19  
Refactoring  
Reminder: We need steady progress. To keep progress steady, we  
need a clear, clean design all the time. To accomplish this, we must  
refactor. report erratum • discuss]  
  
The Natural Way asks us to show steady progress in terms  
of visible, working software. Seeing real software come into  
being at a steady pace gives us a sense of where we are. This  
helps us decide what to do next and what to defer until later. Often, though, our pace is not steady. Even though our work  
items seem all about equally difficult, our pace may become  
erratic. Shortly after this happens, our pace almost inevitably  
slows down. This is demoralizing. Even worse, it makes planning difficult. Worse still, we’ll usually see an increase in defects as this  
happens. And worst of all, our ability to get the best possible  
product by our deadline is at risk.

## Translated Text:

دوم، از آنجا که ما به صورت تدریجی و با اولویت دادن به ارزش بالا کار می‌کنیم، همیشه این شانس وجود دارد که قبل از موعد مقرر، ارزش مفیدی ارائه دهیم. اما ما فقط زمانی می‌توانیم این کار را انجام دهیم که نرم‌افزار واقعاً کار کند. در روزهای اولیه، بله، ممکن است کند به نظر برسد. اما حتی در روزهای اولیه، ممکن است واقعاً کندتر نباشد: نقص‌ها کاهش می‌یابند و کد تمیزتر جریان کار را حفظ می‌کند. اجازه دهید همین‌طور باشد. وقتی تیم شما در واقعاً به پایان رساندن کارها مهارت پیدا کند، قادر خواهند بود ویژگی‌های بیشتری را بر عهده بگیرند و اوضاع بیشتر و بیشتر بهبود خواهد یافت.   
  
\*\*خلاصه\*\*   
باارزش‌ترین کاری که می‌توانید برای تسریع توسعه انجام دهید، ایجاد مهارت در تیم توسعه است. این سرمایه‌گذاری به سرعت نتیجه می‌دهد، با کاهش زمان از دست رفته برای رفع نقص‌ها و توسعه روان‌تر. سردرگمی را با عملکرد مؤثر اشتباه نگیرید. سریع‌ترین تیم‌ها با روانی و ظرافت حرکت می‌کنند.   
\*report erratum • discuss\*   
\*Chapter 18. To Speed Up, Build with Skill • 119\*  
  
---  
  
\*\*فصل ۱۹\*\*   
\*\*Refactoring\*\*   
یادآوری: ما به پیشرفت مداوم نیاز داریم. برای حفظ پیشرفت مداوم، ما همیشه به یک طراحی واضح و تمیز نیاز داریم. برای دستیابی به این هدف، باید Refactor کنیم.   
\*report erratum • discuss\*  
  
---  
  
\*\*[متن از صفحه قبلی: فصل ۱۹   
Refactoring   
یادآوری: ما به پیشرفت مداوم نیاز داریم. برای حفظ پیشرفت مداوم، ما همیشه به یک طراحی واضح و تمیز نیاز داریم. برای دستیابی به این هدف، باید Refactor کنیم.   
report erratum • discuss]\*  
  
روش طبیعی از ما می‌خواهد که پیشرفت مداوم را از نظر نرم‌افزار قابل مشاهده و کارآمد نشان دهیم. دیدن نرم‌افزار واقعی که با سرعتی ثابت در حال شکل‌گیری است، به ما حس می‌دهد که کجا هستیم. این به ما کمک می‌کند تصمیم بگیریم چه کاری را بعداً انجام دهیم و چه چیزی را به تعویق بیندازیم.   
  
با این حال، اغلب اوقات، سرعت ما ثابت نیست. حتی اگر آیتم‌های کاری ما به نظر به یک اندازه دشوار باشند، ممکن است سرعت ما نامنظم شود. کمی بعد از این اتفاق، سرعت ما تقریباً به طور اجتناب‌ناپذیری کاهش می‌یابد. این موضوع ناامیدکننده است. حتی بدتر، برنامه‌ریزی را دشوار می‌کند. بدتر از آن، معمولاً با افزایش نقص‌ها مواجه خواهیم شد. و بدترین حالت این است که توانایی ما برای دستیابی به بهترین محصول ممکن تا موعد مقرر در خطر خواهد بود.

================================================================================

# Section 119

## Original Text:

report erratum • discussChapter 19. Refactoring • 121  
  
[CONTEXT FROM PREVIOUS PAGE: The Natural Way asks us to show steady progress in terms  
of visible, working software. Seeing real software come into  
being at a steady pace gives us a sense of where we are. This  
helps us decide what to do next and what to defer until later. Often, though, our pace is not steady. Even though our work  
items seem all about equally difficult, our pace may become  
erratic. Shortly after this happens, our pace almost inevitably  
slows down. This is demoralizing. Even worse, it makes planning difficult. Worse still, we’ll usually see an increase in defects as this  
happens. And worst of all, our ability to get the best possible  
product by our deadline is at risk. report erratum • discussChapter 19. Refactoring • 121]  
  
Twisty little passages  
The time needed to build a feature comes from two main  
components: its inherent difficulty, and the accidental diffi-  
culty of putting it into whatever code already exists. Teams  
are good at estimating the inherent difficulty. What makes  
us erratic, what makes us slow down, is the accidental diffi-  
culty. We call this difficulty “bad code. ”  
If we allow code quality to decline, some features go in eas-  
ily, sailing right through. Others that seem similar get  
entangled in twisty little passages of bad code. Similar work  
starts taking radically different amounts of time. To keep progress steady, we have to avoid building twisty  
little passages, and when we do build them, we need to  
straighten them out. Chapter 19. Refactoring • 122  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Twisty little passages  
The time needed to build a feature comes from two main  
components: its inherent difficulty, and the accidental diffi-  
culty of putting it into whatever code already exists. Teams  
are good at estimating the inherent difficulty. What makes  
us erratic, what makes us slow down, is the accidental diffi-  
culty. We call this difficulty “bad code.

## Translated Text:

گزارش خطا • بحث   
فصل 19. Refactoring • 121   
  
[متن از صفحه قبلی: روش طبیعی از ما می‌خواهد که پیشرفت پیوسته‌ای را از نظر نرم‌افزار قابل مشاهده و کارآمد نشان دهیم. دیدن نرم‌افزار واقعی که با سرعتی ثابت ایجاد می‌شود، به ما حس می‌دهد که کجا هستیم. این به ما کمک می‌کند تصمیم بگیریم که چه کاری را بعد انجام دهیم و چه چیزی را به بعد موکول کنیم. با این حال، اغلب اوقات سرعت ما ثابت نیست. حتی اگر آیتم‌های کاری ما به نظر به یک اندازه دشوار باشند، ممکن است سرعت ما نامنظم شود. کمی پس از این اتفاق، سرعت ما تقریباً به‌طور اجتناب‌ناپذیری کاهش می‌یابد. این موضوع دلسردکننده است. حتی بدتر، این برنامه‌ریزی را دشوار می‌کند. بدتر از آن، معمولاً با افزایش نقص‌ها مواجه می‌شویم. و بدترین حالت این است که توانایی ما برای ارائه بهترین محصول ممکن تا مهلت مقرر به خطر می‌افتد.]   
  
مسیرهای پیچ‌درپیچ کوچک   
زمان لازم برای Build کردن یک Feature از دو مؤلفه اصلی تشکیل شده است: دشواری ذاتی آن و دشواری تصادفی قرار دادن آن در کدی که قبلاً وجود دارد. تیم‌ها در تخمین دشواری ذاتی خوب هستند. چیزی که ما را نامنظم می‌کند و باعث کاهش سرعت ما می‌شود، دشواری تصادفی است. ما این دشواری را "bad code" می‌نامیم.   
اگر اجازه دهیم کیفیت Code کاهش یابد، برخی Features به راحتی و بدون مشکل اضافه می‌شوند. برخی دیگر که به نظر مشابه می‌رسند، در مسیرهای پیچ‌درپیچ bad code گرفتار می‌شوند. کار مشابه شروع به گرفتن زمان‌های بسیار متفاوت می‌کند. برای حفظ پیشرفت ثابت، باید از ایجاد مسیرهای پیچ‌درپیچ اجتناب کنیم و وقتی آن‌ها را ایجاد کردیم، باید آن‌ها را صاف کنیم.   
فصل 19. Refactoring • 122   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: مسیرهای پیچ‌درپیچ کوچک   
زمان لازم برای Build کردن یک Feature از دو مؤلفه اصلی تشکیل شده است: دشواری ذاتی آن و دشواری تصادفی قرار دادن آن در کدی که قبلاً وجود دارد. تیم‌ها در تخمین دشواری ذاتی خوب هستند. چیزی که ما را نامنظم می‌کند و باعث کاهش سرعت ما می‌شود، دشواری تصادفی است. ما این دشواری را "bad code" می‌نامیم.]

================================================================================

# Section 120

## Original Text:

”  
If we allow code quality to decline, some features go in eas-  
ily, sailing right through. Others that seem similar get  
entangled in twisty little passages of bad code. Similar work  
starts taking radically different amounts of time. To keep progress steady, we have to avoid building twisty  
little passages, and when we do build them, we need to  
straighten them out. Chapter 19. Refactoring • 122  
report erratum • discuss]  
  
Refactoring straightens the passages. The word refactoring refers to a simple, regular process of  
keeping the code clean. We try not to create the twisty little  
passages that slow us down. When twists do crop up, we  
straighten them out. It’s easy to cause our teams to skimp on refactoring: all we  
have to do is put them under pressure to get a few more  
features done than they realistically can. When we do that,  
they turn secret programmer dials called Not Enough Testing  
and Coding Shortcuts, and the result is that there are more  
defects, erratic progress, and, ultimately, a slowdown. Bad work is hard to see. We can’t see it from the outside: it’s  
under the surface. But as businesspeople we need the work  
to be good. We should expect and demand that the team  
keeps the code clean. And if we notice erratic progress or a  
slowdown, it’s probably time to bear down on cleaning  
things up again. That means it’s time to lighten up on the  
pressure. report erratum • discussChapter 19. Refactoring • 123  
  
[CONTEXT FROM PREVIOUS PAGE: Refactoring straightens the passages. The word refactoring refers to a simple, regular process of  
keeping the code clean. We try not to create the twisty little  
passages that slow us down. When twists do crop up, we  
straighten them out. It’s easy to cause our teams to skimp on refactoring: all we  
have to do is put them under pressure to get a few more  
features done than they realistically can.

## Translated Text:

”  
اگر اجازه دهیم کیفیت کد کاهش یابد، برخی از قابلیت‌ها به راحتی و بدون مشکل اضافه می‌شوند. اما قابلیت‌های دیگری که مشابه به نظر می‌رسند، در پیچ‌وخم‌های کد بد گرفتار می‌شوند. کار مشابه شروع به گرفتن زمان‌های بسیار متفاوتی می‌کند. برای حفظ پیشرفت پایدار، باید از ایجاد پیچ‌وخم‌های کد اجتناب کنیم و اگر آنها را ایجاد کردیم، باید آنها را صاف کنیم.   
فصل ۱۹. Refactoring • ۱۲۲   
گزارش خطا • بحث کنید]   
  
Refactoring پیچ‌وخم‌ها را صاف می‌کند. واژه Refactoring به یک فرآیند ساده و منظم برای تمیز نگه داشتن کد اشاره دارد. ما تلاش می‌کنیم پیچ‌وخم‌هایی که ما را کند می‌کنند ایجاد نکنیم. وقتی پیچ‌ها ظاهر می‌شوند، آنها را صاف می‌کنیم.   
به راحتی می‌توان تیم‌ها را به کم‌کاری در Refactoring وادار کرد: تنها کافی است آنها را تحت فشار قرار دهیم تا چند قابلیت بیشتر از آنچه واقع‌بینانه می‌توانند انجام دهند، به اتمام برسانند. وقتی این کار را می‌کنیم، آنها دکمه‌های مخفی برنامه‌نویسان به نام Not Enough Testing و Coding Shortcuts را می‌چرخانند و نتیجه این می‌شود که نقص‌ها بیشتر می‌شوند، پیشرفت نامنظم می‌شود و در نهایت، روند کار کند می‌شود.   
کار بد به سختی دیده می‌شود. ما نمی‌توانیم آن را از بیرون ببینیم: زیر سطح قرار دارد. اما به عنوان افراد تجاری، ما به کار خوب نیاز داریم. باید انتظار داشته باشیم و از تیم بخواهیم که کد را تمیز نگه دارند. و اگر پیشرفت نامنظم یا کندی مشاهده کردیم، احتمالاً زمان آن رسیده است که دوباره بر تمیز کردن تمرکز کنیم. این به معنای کاهش فشار است.   
گزارش خطا • بحث کنید   
فصل ۱۹. Refactoring • ۱۲۳   
  
[متن از صفحه قبلی: Refactoring پیچ‌وخم‌ها را صاف می‌کند. واژه Refactoring به یک فرآیند ساده و منظم برای تمیز نگه داشتن کد اشاره دارد. ما تلاش می‌کنیم پیچ‌وخم‌هایی که ما را کند می‌کنند ایجاد نکنیم. وقتی پیچ‌ها ظاهر می‌شوند، آنها را صاف می‌کنیم.   
به راحتی می‌توان تیم‌ها را به کم‌کاری در Refactoring وادار کرد: تنها کافی است آنها را تحت فشار قرار دهیم تا چند قابلیت بیشتر از آنچه واقع‌بینانه می‌توانند انجام دهند، به اتمام برسانند.]

================================================================================

# Section 121

## Original Text:

When we do that,  
they turn secret programmer dials called Not Enough Testing  
and Coding Shortcuts, and the result is that there are more  
defects, erratic progress, and, ultimately, a slowdown. Bad work is hard to see. We can’t see it from the outside: it’s  
under the surface. But as businesspeople we need the work  
to be good. We should expect and demand that the team  
keeps the code clean. And if we notice erratic progress or a  
slowdown, it’s probably time to bear down on cleaning  
things up again. That means it’s time to lighten up on the  
pressure. report erratum • discussChapter 19. Refactoring • 123]  
  
What to do when the passages have gotten all twisty  
It’s usually not a good idea to start over on some huge part  
of the product. It might be, but I’d bet it isn’t. It’s usually not a good idea to stop doing features entirely,  
to “clean up” the code. It might be, but I’d bet it isn’t. What seems to work best is to follow the Campground Rule:  
Leave the campground a little better than you found it. Every  
time we do a feature, we start by cleaning up the area where  
we are going to do the work. We don’t have to make it per-  
fect, just sufficiently better to help our feature ease in. And  
once our feature works, we clean up the code as we always  
should—plus a bit more. This process does just what we need: areas where we do  
little work aren’t slowing us down, and they are left mostly  
alone. Areas where we do lots of work get more attention;  
they clean up quickly. It’s always tempting to developers to start over and rewrite. It’s almost never the best plan. Instead, get good at refactor-  
ing, and apply the Campground Rule. Chapter 19. Refactoring • 124  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: What to do when the passages have gotten all twisty  
It’s usually not a good idea to start over on some huge part  
of the product. It might be, but I’d bet it isn’t. It’s usually not a good idea to stop doing features entirely,  
to “clean up” the code.

## Translated Text:

وقتی این کار را انجام می‌دهیم،   
آن‌ها دکمه‌های مخفی برنامه‌نویسان به نام Not Enough Testing و Coding Shortcuts را می‌چرخانند، و نتیجه این است که تعداد بیشتری از defects، پیشرفت نامنظم، و در نهایت، کندی به وجود می‌آید. کار بد دیدنش سخت است. نمی‌توانیم آن را از بیرون ببینیم: زیر سطح است. اما به‌عنوان افراد کسب‌وکار، ما به کار خوب نیاز داریم. باید انتظار داشته باشیم و از تیم بخواهیم که کد را تمیز نگه دارد. و اگر پیشرفت نامنظم یا کندی را مشاهده کردیم، احتمالاً زمان آن رسیده که دوباره بر تمیز کردن تمرکز کنیم. این یعنی زمان آن است که فشار را کم کنیم.   
report erratum • discuss   
Chapter 19. Refactoring • 123   
  
### وقتی بخش‌ها پیچیده شده‌اند چه باید کرد   
معمولاً شروع دوباره روی یک بخش بزرگ از محصول ایده خوبی نیست. ممکن است باشد، اما احتمالاً نیست. معمولاً ایده خوبی نیست که به‌طور کامل انجام دادن features را متوقف کنیم تا کد را “تمیز کنیم”. ممکن است باشد، اما احتمالاً نیست. چیزی که به نظر می‌رسد بهترین کار است، دنبال کردن Campground Rule است: کمپ را کمی بهتر از آنچه پیدا کردید، ترک کنید. هر بار که یک feature انجام می‌دهیم، ابتدا منطقه‌ای را که قرار است در آن کار کنیم، تمیز می‌کنیم. لازم نیست آن را کامل کنیم، فقط به اندازه کافی بهتر کنیم تا feature ما راحت‌تر جا بیفتد. و وقتی feature ما کار می‌کند، کد را همان‌طور که همیشه باید تمیز کنیم—و کمی بیشتر. این فرآیند دقیقاً همان چیزی را که نیاز داریم انجام می‌دهد: مناطقی که در آن‌ها کار کمی انجام می‌دهیم، سرعت ما را کاهش نمی‌دهند و عمدتاً دست‌نخورده باقی می‌مانند. مناطقی که در آن‌ها کار زیادی انجام می‌دهیم، توجه بیشتری دریافت می‌کنند؛ آن‌ها سریع‌تر تمیز می‌شوند. همیشه برای توسعه‌دهندگان وسوسه‌انگیز است که دوباره شروع کنند و بازنویسی کنند. این تقریباً هرگز بهترین برنامه نیست. در عوض، در Refactoring مهارت پیدا کنید و Campground Rule را اعمال کنید.   
Chapter 19. Refactoring • 124   
report erratum • discuss   
  
[CONTEXT FROM PREVIOUS PAGE: وقتی بخش‌ها پیچیده شده‌اند چه باید کرد   
معمولاً شروع دوباره روی یک بخش بزرگ از محصول ایده خوبی نیست. ممکن است باشد، اما احتمالاً نیست. معمولاً ایده خوبی نیست که به‌طور کامل انجام دادن features را متوقف کنیم تا کد را “تمیز کنیم”.]

================================================================================

# Section 122

## Original Text:

It might be, but I’d bet it isn’t. What seems to work best is to follow the Campground Rule:  
Leave the campground a little better than you found it. Every  
time we do a feature, we start by cleaning up the area where  
we are going to do the work. We don’t have to make it per-  
fect, just sufficiently better to help our feature ease in. And  
once our feature works, we clean up the code as we always  
should—plus a bit more. This process does just what we need: areas where we do  
little work aren’t slowing us down, and they are left mostly  
alone. Areas where we do lots of work get more attention;  
they clean up quickly. It’s always tempting to developers to start over and rewrite. It’s almost never the best plan. Instead, get good at refactor-  
ing, and apply the Campground Rule. Chapter 19. Refactoring • 124  
report erratum • discuss]  
  
report erratum • discussChapter 19. Refactoring • 125  
  
[CONTEXT FROM PREVIOUS PAGE: report erratum • discussChapter 19. Refactoring • 125]  
  
CHAPTER 20  
Agile Methods  
I am, by good fortune, one of the authors of the Agile Mani-  
festo. 1The story I tell here is in accord with the Manifesto,  
to the best of my ability. The “Natural Way” is a distillation  
of what I’ve learned in a half-century of software develop-  
ment and almost twenty years of doing “ Agile. ” I am not  
trying to create another Agile method here. In Nature , I’m  
describing how I think software should be built, based on  
everything I’ve learned, before, during, and after “ Agile. ”  
If you want to know more about Agile software develop-  
ment, there are many Agile methods or frameworks out  
there. The most popular is surely Scrum, by Jeff Sutherland  
and Ken Schwaber. Scrum, by design, is not focused solely  
on software. As such, Scrum does not explicitly include  
technical practices such as acceptance test-driven develop-  
ment, test-driven development, refactoring, and so on. I’m  
here to tell you that to prosper, you’ll need to add those  
practices to your Scrum projects.

## Translated Text:

ممکن است اینطور باشد، اما شرط می‌بندم که اینطور نیست. چیزی که به نظر می‌رسد بهترین کارایی را دارد، پیروی از قانون Campground است:   
کمپ را کمی بهتر از حالتی که آن را پیدا کردید، ترک کنید. هر بار که یک Feature انجام می‌دهیم، ابتدا با تمیز کردن ناحیه‌ای که قرار است در آن کار کنیم، شروع می‌کنیم. لازم نیست آن را کامل کنیم، فقط به اندازه‌ای بهتر که Feature ما راحت‌تر در آن جا بیفتد. و وقتی Feature ما کار کرد، کد را همانطور که همیشه باید تمیز کنیم—و کمی بیشتر.   
  
این فرآیند دقیقاً همان چیزی را که نیاز داریم انجام می‌دهد: نواحی‌ای که کار کمی در آن‌ها انجام می‌دهیم، ما را کند نمی‌کنند و عمدتاً به حال خود رها می‌شوند. نواحی‌ای که کار زیادی در آن‌ها انجام می‌دهیم، توجه بیشتری دریافت می‌کنند؛ آن‌ها به سرعت تمیز می‌شوند. برای توسعه‌دهندگان همیشه وسوسه‌انگیز است که از ابتدا شروع کنند و دوباره کدنویسی کنند. این تقریباً هرگز بهترین برنامه نیست. در عوض، در Refactoring مهارت پیدا کنید و قانون Campground را اعمال کنید.   
  
Chapter 19. Refactoring • 124   
report erratum • discuss   
  
report erratum • discuss   
Chapter 19. Refactoring • 125   
  
[متن از صفحه قبلی: report erratum • discuss   
Chapter 19. Refactoring • 125]   
  
CHAPTER 20   
Agile Methods   
من، با خوش‌شانسی، یکی از نویسندگان Agile Manifesto هستم. داستانی که اینجا تعریف می‌کنم، تا جایی که توانایی‌ام اجازه می‌دهد، با Manifesto هماهنگ است. روش "طبیعی" یک خلاصه از آن چیزی است که در نیم‌قرن توسعه نرم‌افزار و تقریباً بیست سال انجام "Agile" یاد گرفته‌ام. من اینجا سعی نمی‌کنم یک روش Agile دیگر ایجاد کنم. در Nature، من توضیح می‌دهم که چگونه فکر می‌کنم نرم‌افزار باید ساخته شود، بر اساس تمام چیزهایی که قبل، حین و بعد از "Agile" یاد گرفته‌ام.   
  
اگر می‌خواهید درباره توسعه نرم‌افزار به روش Agile بیشتر بدانید، روش‌ها یا Frameworkهای Agile زیادی وجود دارند. محبوب‌ترین آن‌ها بدون شک Scrum است، توسط Jeff Sutherland و Ken Schwaber. Scrum، به‌طور طراحی‌شده، صرفاً بر نرم‌افزار تمرکز ندارد. به همین دلیل، Scrum به‌طور صریح شامل تمرین‌های فنی مانند Acceptance Test-Driven Development، Test-Driven Development، Refactoring و غیره نمی‌شود. من اینجا هستم تا به شما بگویم که برای موفقیت، باید این تمرین‌ها را به پروژه‌های Scrum خود اضافه کنید.

================================================================================

# Section 123

## Original Text:

XP , or Extreme Programming, created by Kent Beck, is a  
framework that does explicitly include those technical  
practices. Otherwise, XP is much like Scrum. XP does not  
include the Scrum-specific “ScrumMaster” role, but it often  
recommends a coach, which is a similar notion. To me, Scrum  
plus technical practices amounts to XP. Some people would  
disagree with me on that. Alistair Cockburn’s Crystal Clear is an Agile framework that  
is even simpler than Scrum. There are also complex large-  
1. http://agilemanifesto. org  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 20  
Agile Methods  
I am, by good fortune, one of the authors of the Agile Mani-  
festo. 1The story I tell here is in accord with the Manifesto,  
to the best of my ability. The “Natural Way” is a distillation  
of what I’ve learned in a half-century of software develop-  
ment and almost twenty years of doing “ Agile. ” I am not  
trying to create another Agile method here. In Nature , I’m  
describing how I think software should be built, based on  
everything I’ve learned, before, during, and after “ Agile. ”  
If you want to know more about Agile software develop-  
ment, there are many Agile methods or frameworks out  
there. The most popular is surely Scrum, by Jeff Sutherland  
and Ken Schwaber. Scrum, by design, is not focused solely  
on software. As such, Scrum does not explicitly include  
technical practices such as acceptance test-driven develop-  
ment, test-driven development, refactoring, and so on. I’m  
here to tell you that to prosper, you’ll need to add those  
practices to your Scrum projects. XP , or Extreme Programming, created by Kent Beck, is a  
framework that does explicitly include those technical  
practices. Otherwise, XP is much like Scrum. XP does not  
include the Scrum-specific “ScrumMaster” role, but it often  
recommends a coach, which is a similar notion. To me, Scrum  
plus technical practices amounts to XP. Some people would  
disagree with me on that.

## Translated Text:

XP، یا Extreme Programming، که توسط Kent Beck ایجاد شده است، یک Framework است که به‌طور صریح شامل آن Technical Practices می‌شود. در غیر این صورت، XP بسیار شبیه به Scrum است. XP شامل نقش خاص Scrum به نام "ScrumMaster" نمی‌شود، اما اغلب یک Coach را توصیه می‌کند که مفهومی مشابه است. به نظر من، Scrum به‌علاوه Technical Practices برابر با XP است. برخی افراد ممکن است با این نظر من مخالف باشند.   
  
Crystal Clear از Alistair Cockburn یک Agile Framework است که حتی ساده‌تر از Scrum است. همچنین Frameworkهای پیچیده و بزرگ دیگری نیز وجود دارند.   
1. http://agilemanifesto.org   
report erratum • discuss   
  
[متن از صفحه قبلی:   
فصل ۲۰   
Agile Methods   
من، به لطف خوش‌شانسی، یکی از نویسندگان Agile Manifesto هستم. داستانی که اینجا تعریف می‌کنم، تا جایی که توانایی من اجازه می‌دهد، با Manifesto هماهنگ است. "Natural Way" عصاره‌ای از آن چیزی است که در نیم‌قرن توسعه نرم‌افزار و تقریباً بیست سال انجام "Agile" یاد گرفته‌ام. من اینجا قصد ندارم یک Agile Method جدید ایجاد کنم. در Nature، من توضیح می‌دهم که چگونه فکر می‌کنم نرم‌افزار باید ساخته شود، بر اساس تمام چیزهایی که قبل، حین و بعد از "Agile" یاد گرفته‌ام.   
اگر می‌خواهید درباره Agile Software Development بیشتر بدانید، بسیاری از Agile Methods یا Frameworkها وجود دارند. محبوب‌ترین آن‌ها بدون شک Scrum است که توسط Jeff Sutherland و Ken Schwaber ایجاد شده است. Scrum، به‌طور طراحی‌شده، تنها بر نرم‌افزار تمرکز ندارد. بنابراین، Scrum به‌طور صریح شامل Technical Practices مانند Acceptance Test-Driven Development، Test-Driven Development، Refactoring و غیره نمی‌شود. من اینجا هستم تا به شما بگویم که برای موفقیت، باید این Practices را به پروژه‌های Scrum خود اضافه کنید.   
XP، یا Extreme Programming، که توسط Kent Beck ایجاد شده است، یک Framework است که به‌طور صریح شامل آن Technical Practices می‌شود. در غیر این صورت، XP بسیار شبیه به Scrum است. XP شامل نقش خاص Scrum به نام "ScrumMaster" نمی‌شود، اما اغلب یک Coach را توصیه می‌کند که مفهومی مشابه است. به نظر من، Scrum به‌علاوه Technical Practices برابر با XP است. برخی افراد ممکن است با این نظر من مخالف باشند.]

================================================================================

# Section 124

## Original Text:

Alistair Cockburn’s Crystal Clear is an Agile framework that  
is even simpler than Scrum. There are also complex large-  
1. http://agilemanifesto. org  
report erratum • discuss]  
  
scale Agile-like frameworks, such as Dynamic Systems  
Development Method (DSDM), Larman/Vodde’s Large Scale  
Scrum (LeSS), Scott Ambler’s Disciplined Agile Development  
(DAD), or Dean Leffingwell’s Scaled Agile Framework  
(SAFe). And many more. Read about them if you’re interest-  
ed. Again, The Nature of Software Development does not describe  
Yet Another Agile Framework. Instead, I invite you to think  
about what needs to go on in any software project, especially  
any would-be “ Agile” software project, so that you can be  
successful working in whatever framework you choose. I do have some advice about frameworks, however:  
• Try to have too little framework, not too much. Don’t  
even have “enough. ” As the Agile Manifesto says, we  
prefer “individuals and interactions over processes and  
tools. ” A framework for your project wants to fit loosely,  
more like a sweat suit than like spandex. Everyone on  
the project needs room to “slosh around freely,” to  
interact in ways that are not contemplated by any  
framework and that cannot be dictated by rules. • Keep it light. Naturally, if you have a large project, with  
many teams, you’ll need more process than if you have  
just six people in a team room somewhere. But even so,  
keep it light. Use retrospectives to decide what kinds of  
additions to make. Add process elements as experi-  
ments. Be clear what you expect to get from the change,  
and check to see if you get it. If you don’t, or you get  
some undesirable additional effect, make a different  
change next time. • Control the framework; don’t let the framework control  
you. Modify your framework to make your projects  
more effective. However, don’t change the framework  
just because it asks you to do difficult things. The ideas  
in this book, and the ideas in your framework, are there  
to challenge you to improve.

## Translated Text:

کریستال کلیر (Crystal Clear) از الیستر کاکبرن یک Agile framework است که حتی ساده‌تر از Scrum می‌باشد. همچنین Agile-like frameworks پیچیده و بزرگ‌مقیاسی وجود دارند، مانند Dynamic Systems Development Method (DSDM)، Large Scale Scrum (LeSS) از لارمن/ود، Disciplined Agile Development (DAD) از اسکات امبلر، یا Scaled Agile Framework (SAFe) از دین لفینگول. و بسیاری دیگر. اگر علاقه‌مند هستید، درباره آن‌ها مطالعه کنید.   
  
باز هم، کتاب \*The Nature of Software Development\* به توصیف Yet Another Agile Framework نمی‌پردازد. در عوض، از شما دعوت می‌کنم که به این فکر کنید که در هر پروژه نرم‌افزاری، به‌ویژه هر پروژه نرم‌افزاری که می‌خواهد "Agile" باشد، چه چیزهایی باید اتفاق بیفتد تا بتوانید در هر Framework که انتخاب می‌کنید موفق باشید. با این حال، چند توصیه درباره Frameworkها دارم:  
  
- سعی کنید Framework شما کمتر باشد، نه بیشتر. حتی "به اندازه کافی" هم نداشته باشید. همان‌طور که Agile Manifesto می‌گوید، ما "افراد و تعاملات را به فرآیندها و ابزارها ترجیح می‌دهیم." Framework پروژه شما باید به‌گونه‌ای باشد که به‌طور آزادانه و شل عمل کند، بیشتر شبیه یک لباس ورزشی باشد تا یک لباس چسبان. همه افراد پروژه باید فضایی برای "آزادانه حرکت کردن" داشته باشند، تا بتوانند به روش‌هایی تعامل کنند که در هیچ Framework پیش‌بینی نشده و نمی‌توان آن‌ها را با قوانین دیکته کرد.  
  
- آن را سبک نگه دارید. طبیعتاً، اگر پروژه بزرگی دارید که شامل تیم‌های زیادی است، به فرآیند بیشتری نسبت به زمانی که فقط شش نفر در یک اتاق تیم هستند نیاز خواهید داشت. اما حتی در این صورت، آن را سبک نگه دارید. از Retrospectives استفاده کنید تا تصمیم بگیرید چه نوع اضافاتی را اعمال کنید. عناصر فرآیند را به‌عنوان آزمایش اضافه کنید. مشخص کنید که از این تغییر چه انتظاری دارید و بررسی کنید که آیا به آن دست یافته‌اید یا خیر. اگر نه، یا اگر اثر اضافی نامطلوبی به دست آوردید، دفعه بعد تغییر متفاوتی ایجاد کنید.  
  
- Framework را کنترل کنید؛ اجازه ندهید Framework شما را کنترل کند. Framework خود را تغییر دهید تا پروژه‌های شما مؤثرتر شوند. با این حال، فقط به این دلیل که Framework از شما می‌خواهد کارهای دشواری انجام دهید، آن را تغییر ندهید. ایده‌های این کتاب و ایده‌های موجود در Framework شما برای این است که شما را به بهبود تشویق کنند.

================================================================================

# Section 125

## Original Text:

Adjust things to your  
abilities—but stretch yourself a bit. • Keep process changes close to the team. There’s no  
advantage to imposing practices worldwide. You are  
report erratum • discussChapter 20. Agile Methods • 127  
  
[CONTEXT FROM PREVIOUS PAGE: scale Agile-like frameworks, such as Dynamic Systems  
Development Method (DSDM), Larman/Vodde’s Large Scale  
Scrum (LeSS), Scott Ambler’s Disciplined Agile Development  
(DAD), or Dean Leffingwell’s Scaled Agile Framework  
(SAFe). And many more. Read about them if you’re interest-  
ed. Again, The Nature of Software Development does not describe  
Yet Another Agile Framework. Instead, I invite you to think  
about what needs to go on in any software project, especially  
any would-be “ Agile” software project, so that you can be  
successful working in whatever framework you choose. I do have some advice about frameworks, however:  
• Try to have too little framework, not too much. Don’t  
even have “enough. ” As the Agile Manifesto says, we  
prefer “individuals and interactions over processes and  
tools. ” A framework for your project wants to fit loosely,  
more like a sweat suit than like spandex. Everyone on  
the project needs room to “slosh around freely,” to  
interact in ways that are not contemplated by any  
framework and that cannot be dictated by rules. • Keep it light. Naturally, if you have a large project, with  
many teams, you’ll need more process than if you have  
just six people in a team room somewhere. But even so,  
keep it light. Use retrospectives to decide what kinds of  
additions to make. Add process elements as experi-  
ments. Be clear what you expect to get from the change,  
and check to see if you get it. If you don’t, or you get  
some undesirable additional effect, make a different  
change next time. • Control the framework; don’t let the framework control  
you. Modify your framework to make your projects  
more effective. However, don’t change the framework  
just because it asks you to do difficult things.

## Translated Text:

تنظیمات را بر اساس توانایی‌های خود انجام دهید—اما کمی خود را به چالش بکشید.   
• تغییرات فرآیند را نزدیک به تیم نگه دارید. هیچ مزیتی در تحمیل کردن روش‌ها به صورت جهانی وجود ندارد.   
  
گزارش خطا • بحث   
فصل ۲۰. روش‌های Agile • ۱۲۷   
  
[متن از صفحه قبلی: مقیاس فریمورک‌های مشابه Agile، مانند Dynamic Systems Development Method (DSDM)، Large Scale Scrum (LeSS) از Larman/Vodde، Disciplined Agile Development (DAD) از Scott Ambler، یا Scaled Agile Framework (SAFe) از Dean Leffingwell. و بسیاری دیگر. اگر علاقه‌مند هستید، درباره آن‌ها بخوانید. دوباره، کتاب The Nature of Software Development به توصیف Yet Another Agile Framework نمی‌پردازد. در عوض، از شما دعوت می‌کنم به این فکر کنید که در هر پروژه نرم‌افزاری، به‌ویژه هر پروژه نرم‌افزاری که می‌خواهد "Agile" باشد، چه چیزهایی باید اتفاق بیفتد تا بتوانید در هر فریمورکی که انتخاب می‌کنید موفق باشید. با این حال، چند توصیه درباره فریمورک‌ها دارم:   
• سعی کنید فریمورک کمتر داشته باشید، نه بیشتر. حتی "به اندازه کافی" هم نداشته باشید. همان‌طور که Agile Manifesto می‌گوید، ما "افراد و تعاملات را به فرآیندها و ابزارها ترجیح می‌دهیم." یک فریمورک برای پروژه شما باید به‌صورت شُل و آزاد باشد، بیشتر شبیه به یک لباس راحتی تا لباس تنگ. همه اعضای پروژه باید فضایی برای "آزادانه حرکت کردن" داشته باشند، تا به روش‌هایی تعامل کنند که در هیچ فریمورکی پیش‌بینی نشده و نمی‌توان آن‌ها را با قوانین دیکته کرد.   
• آن را سبک نگه دارید. طبیعتاً، اگر پروژه بزرگی دارید که شامل تیم‌های زیادی است، به فرآیند بیشتری نسبت به زمانی که فقط شش نفر در یک اتاق تیم هستند نیاز خواهید داشت. اما حتی در این صورت، آن را سبک نگه دارید. از جلسات بازنگری (Retrospectives) استفاده کنید تا تصمیم بگیرید چه نوع اضافاتی باید انجام شود. عناصر فرآیند را به‌عنوان آزمایش اضافه کنید. مشخص کنید که از تغییر چه انتظاری دارید و بررسی کنید که آیا به آن دست یافته‌اید یا خیر. اگر نه، یا اگر اثر نامطلوبی ایجاد شد، دفعه بعد تغییر دیگری ایجاد کنید.   
• فریمورک را کنترل کنید؛ اجازه ندهید فریمورک شما را کنترل کند. فریمورک خود را برای مؤثرتر کردن پروژه‌هایتان اصلاح کنید. با این حال، فقط به این دلیل که فریمورک از شما می‌خواهد کارهای دشوار انجام دهید، آن را تغییر ندهید.

================================================================================

# Section 126

## Original Text:

The ideas  
in this book, and the ideas in your framework, are there  
to challenge you to improve. Adjust things to your  
abilities—but stretch yourself a bit. • Keep process changes close to the team. There’s no  
advantage to imposing practices worldwide. You are  
report erratum • discussChapter 20. Agile Methods • 127]  
  
managing your project by observing the actual running  
software, coming out feature by feature, correctly exe-  
cuting verification tests that the business defines. You  
are determining what goes on inside your teams by  
observing their results, and the pace of producing  
results. You don’t need to demand specific process ele-  
ments, and doing so will backfire. • Make learning a priority. As we see in this book, skills  
are required at all levels of the project, from the highest  
business elements, through the management, right down  
to the technical people. The technical people, in particu-  
lar, work with the software every day, and they need  
to have specific skills to do it well. Expect to invest in  
training and support for your project. It will pay off in  
faster delivery of better software. • Most of all, think. The work of building a valuable  
product is complex, and it is best accomplished, not by  
trying to anticipate and control everything, but by  
observing what happens and responding to events. It’s  
a bit like a team sport: there may be plans, there may  
even be planned plays. But what happens in the game  
is always different, and success lies in the ability of the  
team members to interact in the moment. Paradoxically,  
our ability to seize the moment comes from the thinking  
we do before the action starts. Think. Chapter 20. Agile Methods • 128  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: managing your project by observing the actual running  
software, coming out feature by feature, correctly exe-  
cuting verification tests that the business defines.

## Translated Text:

ایده‌های این کتاب و ایده‌های موجود در framework شما برای به چالش کشیدن شما جهت بهبود طراحی شده‌اند. چیزها را با توانایی‌های خود تنظیم کنید—اما کمی خود را به جلو برانید.   
• تغییرات فرآیند را نزدیک به تیم نگه دارید. هیچ مزیتی در تحمیل کردن روش‌ها به صورت جهانی وجود ندارد.   
  
[CONTEXT FROM PREVIOUS PAGE: مدیریت پروژه شما با مشاهده نرم‌افزار در حال اجرای واقعی، که feature به feature ارائه می‌شود و به درستی verification test‌هایی که کسب‌وکار تعریف می‌کند را اجرا می‌کند.]  
  
شما با مشاهده نتایج تیم‌ها و سرعت تولید نتایج، تعیین می‌کنید که در داخل تیم‌های شما چه می‌گذرد. نیازی نیست که عناصر فرآیند خاصی را مطالبه کنید، و انجام این کار نتیجه معکوس خواهد داشت.   
• یادگیری را در اولویت قرار دهید. همان‌طور که در این کتاب می‌بینیم، مهارت‌ها در تمام سطوح پروژه مورد نیاز هستند، از بالاترین عناصر کسب‌وکار، از طریق مدیریت، تا افراد فنی. افراد فنی، به‌ویژه، هر روز با نرم‌افزار کار می‌کنند و نیاز دارند که مهارت‌های خاصی برای انجام این کار به خوبی داشته باشند. انتظار داشته باشید که در آموزش و پشتیبانی از پروژه خود سرمایه‌گذاری کنید. این سرمایه‌گذاری با تحویل سریع‌تر نرم‌افزار بهتر نتیجه خواهد داد.   
• مهم‌تر از همه، فکر کنید. کار ساخت یک محصول ارزشمند پیچیده است و بهترین روش برای انجام آن، نه با تلاش برای پیش‌بینی و کنترل همه چیز، بلکه با مشاهده آنچه اتفاق می‌افتد و پاسخ به رویدادها است. این کار شبیه به یک ورزش تیمی است: ممکن است برنامه‌هایی وجود داشته باشد، حتی ممکن است بازی‌های برنامه‌ریزی‌شده‌ای وجود داشته باشد. اما آنچه در بازی اتفاق می‌افتد همیشه متفاوت است و موفقیت در توانایی اعضای تیم برای تعامل در لحظه نهفته است. به‌طور متناقض، توانایی ما برای استفاده از لحظه از تفکری که قبل از شروع عمل انجام می‌دهیم ناشی می‌شود. فکر کنید.   
Chapter 20. Agile Methods • 128   
report erratum • discuss

================================================================================

# Section 127

## Original Text:

You  
are determining what goes on inside your teams by  
observing their results, and the pace of producing  
results. You don’t need to demand specific process ele-  
ments, and doing so will backfire. • Make learning a priority. As we see in this book, skills  
are required at all levels of the project, from the highest  
business elements, through the management, right down  
to the technical people. The technical people, in particu-  
lar, work with the software every day, and they need  
to have specific skills to do it well. Expect to invest in  
training and support for your project. It will pay off in  
faster delivery of better software. • Most of all, think. The work of building a valuable  
product is complex, and it is best accomplished, not by  
trying to anticipate and control everything, but by  
observing what happens and responding to events. It’s  
a bit like a team sport: there may be plans, there may  
even be planned plays. But what happens in the game  
is always different, and success lies in the ability of the  
team members to interact in the moment. Paradoxically,  
our ability to seize the moment comes from the thinking  
we do before the action starts. Think. Chapter 20. Agile Methods • 128  
report erratum • discuss]  
  
report erratum • discussChapter 20. Agile Methods • 129  
  
[CONTEXT FROM PREVIOUS PAGE: report erratum • discussChapter 20. Agile Methods • 129]  
  
You don’t need to “scale” Agile. You just need to do it. CHAPTER 21  
Scaling Agile  
There’s a lot of interest in “scaling” Agile these days, and it  
has become big business. Large companies have heard the  
clarion call of the Agile Buzzword, and just as they did with  
past good ideas like Six Sigma and TQM, now they want to  
go Agile. It has become the thing to do. But they’re large  
companies. So, naturally, they think they need to scale. It turns out that in most cases, they’re wrong. They don’t  
need to scale. They need to do plain old simple Agile soft-  
ware development.

## Translated Text:

شما با مشاهده نتایج و سرعت تولید نتایج تعیین می‌کنید که در تیم‌های شما چه می‌گذرد. نیازی نیست که عناصر خاصی از فرآیند را مطالبه کنید، انجام این کار نتیجه معکوس خواهد داشت.   
• یادگیری را در اولویت قرار دهید. همان‌طور که در این کتاب می‌بینیم، مهارت‌ها در تمام سطوح پروژه مورد نیاز هستند، از بالاترین عناصر کسب‌وکار، از طریق مدیریت، تا افراد فنی. افراد فنی به‌ویژه هر روز با نرم‌افزار کار می‌کنند و برای انجام خوب این کار به مهارت‌های خاصی نیاز دارند. انتظار داشته باشید که برای آموزش و پشتیبانی از پروژه خود سرمایه‌گذاری کنید. این سرمایه‌گذاری منجر به تحویل سریع‌تر نرم‌افزار بهتر خواهد شد.   
• مهم‌تر از همه، فکر کنید. کار ساخت یک محصول ارزشمند پیچیده است و بهتر است نه با تلاش برای پیش‌بینی و کنترل همه چیز، بلکه با مشاهده آنچه اتفاق می‌افتد و پاسخ به رویدادها انجام شود. این کار شبیه یک ورزش تیمی است: ممکن است برنامه‌هایی وجود داشته باشد، حتی ممکن است بازی‌های برنامه‌ریزی‌شده‌ای وجود داشته باشد. اما آنچه در بازی اتفاق می‌افتد همیشه متفاوت است و موفقیت در توانایی اعضای تیم برای تعامل در لحظه نهفته است. به‌طور متناقض، توانایی ما برای استفاده از لحظه از تفکری که قبل از شروع عمل انجام می‌دهیم ناشی می‌شود. فکر کنید.   
  
Chapter 20. Agile Methods • 128   
report erratum • discuss   
  
report erratum • discussChapter 20. Agile Methods • 129   
  
[متن از صفحه قبلی: report erratum • discussChapter 20. Agile Methods • 129]   
  
شما نیازی به "scale" کردن Agile ندارید. فقط باید آن را انجام دهید.   
  
CHAPTER 21   
Scaling Agile   
این روزها علاقه زیادی به "scaling" Agile وجود دارد و این به یک تجارت بزرگ تبدیل شده است. شرکت‌های بزرگ صدای طنین‌انداز Agile Buzzword را شنیده‌اند و همان‌طور که با ایده‌های خوب گذشته مانند Six Sigma و TQM انجام دادند، اکنون می‌خواهند Agile شوند. این به چیزی تبدیل شده که باید انجام شود. اما آن‌ها شرکت‌های بزرگی هستند. بنابراین، به‌طور طبیعی، فکر می‌کنند که نیاز به scale دارند.   
در بیشتر موارد، مشخص می‌شود که آن‌ها اشتباه می‌کنند. آن‌ها نیازی به scale ندارند. آن‌ها نیاز دارند که فقط Agile ساده و قدیمی توسعه نرم‌افزار را انجام دهند.

================================================================================

# Section 128

## Original Text:

report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: You don’t need to “scale” Agile. You just need to do it. CHAPTER 21  
Scaling Agile  
There’s a lot of interest in “scaling” Agile these days, and it  
has become big business. Large companies have heard the  
clarion call of the Agile Buzzword, and just as they did with  
past good ideas like Six Sigma and TQM, now they want to  
go Agile. It has become the thing to do. But they’re large  
companies. So, naturally, they think they need to scale. It turns out that in most cases, they’re wrong. They don’t  
need to scale. They need to do plain old simple Agile soft-  
ware development. report erratum • discuss]  
  
Scaling Agile is good business for scaling vendors. It’s not neces-  
sarily good advice for you. Scaling Agile has become a good business to be in, because  
people think they need it. There has always been a decent  
market for scaling Agile, so there have always been some  
contending approaches to doing it. Now, with the market  
for Big Agile growing, there are even more. I’ll leave it to you to look those up and choose among them  
if you must. What I’d like to do here is to suggest that—with  
one possible exception—these approaches are misguided. That’s not to say that large-scale Agile won’t be successful;  
very likely it will be. It will be successful in the sense that  
large companies will buy scaling products and ideas, and  
consultants and training companies will enrich themselves  
selling what these large companies want. report erratum • discussChapter 21. Scaling Agile • 131  
  
[CONTEXT FROM PREVIOUS PAGE: Scaling Agile is good business for scaling vendors. It’s not neces-  
sarily good advice for you. Scaling Agile has become a good business to be in, because  
people think they need it. There has always been a decent  
market for scaling Agile, so there have always been some  
contending approaches to doing it. Now, with the market  
for Big Agile growing, there are even more.

## Translated Text:

گزارش خطا • بحث کنید  
  
[متن از صفحه قبلی: شما نیازی به "مقیاس‌پذیری" Agile ندارید. فقط باید آن را انجام دهید. فصل ۲۱   
مقیاس‌پذیری Agile   
این روزها علاقه زیادی به "مقیاس‌پذیری" Agile وجود دارد و این موضوع به یک تجارت بزرگ تبدیل شده است. شرکت‌های بزرگ صدای طنین‌انداز Buzzword مربوط به Agile را شنیده‌اند، و همان‌طور که قبلاً با ایده‌های خوب گذشته مانند Six Sigma و TQM عمل کردند، اکنون می‌خواهند Agile شوند. این به کاری تبدیل شده که باید انجام شود. اما آن‌ها شرکت‌های بزرگی هستند. بنابراین، به طور طبیعی فکر می‌کنند که نیاز به مقیاس‌پذیری دارند. در اکثر موارد، مشخص می‌شود که آن‌ها اشتباه می‌کنند. آن‌ها نیازی به مقیاس‌پذیری ندارند. آن‌ها نیاز دارند که توسعه نرم‌افزار ساده و معمولی Agile را انجام دهند. گزارش خطا • بحث کنید]  
  
مقیاس‌پذیری Agile تجارت خوبی برای فروشندگان مقیاس‌پذیری است. لزوماً توصیه خوبی برای شما نیست. مقیاس‌پذیری Agile به یک تجارت خوب تبدیل شده است، زیرا مردم فکر می‌کنند که به آن نیاز دارند. همیشه بازار مناسبی برای مقیاس‌پذیری Agile وجود داشته است، بنابراین همیشه رویکردهای رقابتی برای انجام آن وجود داشته‌اند. اکنون، با رشد بازار Big Agile، تعداد بیشتری از این رویکردها وجود دارد.   
من این را به شما واگذار می‌کنم که آن‌ها را جستجو کنید و اگر مجبورید، یکی را انتخاب کنید. چیزی که می‌خواهم اینجا پیشنهاد کنم این است که—با یک استثنای احتمالی—این رویکردها اشتباه هستند. این به این معنا نیست که Agile در مقیاس بزرگ موفق نخواهد بود؛ احتمالاً موفق خواهد بود. موفق خواهد بود به این معنا که شرکت‌های بزرگ محصولات و ایده‌های مقیاس‌پذیری را خریداری خواهند کرد، و مشاوران و شرکت‌های آموزشی خود را با فروش آنچه این شرکت‌های بزرگ می‌خواهند، غنی خواهند کرد. گزارش خطا • بحث کنید فصل ۲۱. مقیاس‌پذیری Agile • ۱۳۱   
  
[متن از صفحه قبلی: مقیاس‌پذیری Agile تجارت خوبی برای فروشندگان مقیاس‌پذیری است. لزوماً توصیه خوبی برای شما نیست. مقیاس‌پذیری Agile به یک تجارت خوب تبدیل شده است، زیرا مردم فکر می‌کنند که به آن نیاز دارند. همیشه بازار مناسبی برای مقیاس‌پذیری Agile وجود داشته است، بنابراین همیشه رویکردهای رقابتی برای انجام آن وجود داشته‌اند. اکنون، با رشد بازار Big Agile، تعداد بیشتری از این رویکردها وجود دارد.]

================================================================================

# Section 129

## Original Text:

I’ll leave it to you to look those up and choose among them  
if you must. What I’d like to do here is to suggest that—with  
one possible exception—these approaches are misguided. That’s not to say that large-scale Agile won’t be successful;  
very likely it will be. It will be successful in the sense that  
large companies will buy scaling products and ideas, and  
consultants and training companies will enrich themselves  
selling what these large companies want. report erratum • discussChapter 21. Scaling Agile • 131]  
  
As the Rolling Stones remind us, you can’t always get what you  
want. Unfortunately, contrary to the song, a big company can  
always get what it wants—in this case lots of expensive  
training in some heavy approach that touts itself as Scaled  
Agile. And they’ll get some benefit, certainly. Any attention  
to improvement is usually better than no attention at all. And to the extent that these various approaches include  
some real Agile ideas, organizations will get some of that as  
well. I’m here to talk about what the song offers: I’m here to talk  
about what you need. What does a large company really  
need to know about applying Agile ideas throughout. Chapter 21. Scaling Agile • 132  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: As the Rolling Stones remind us, you can’t always get what you  
want. Unfortunately, contrary to the song, a big company can  
always get what it wants—in this case lots of expensive  
training in some heavy approach that touts itself as Scaled  
Agile. And they’ll get some benefit, certainly. Any attention  
to improvement is usually better than no attention at all. And to the extent that these various approaches include  
some real Agile ideas, organizations will get some of that as  
well. I’m here to talk about what the song offers: I’m here to talk  
about what you need. What does a large company really  
need to know about applying Agile ideas throughout. Chapter 21.

## Translated Text:

من این را به شما واگذار می‌کنم که آن‌ها را جستجو کرده و در صورت لزوم از میانشان انتخاب کنید. چیزی که می‌خواهم اینجا پیشنهاد کنم این است که—با یک استثنای احتمالی—این رویکردها اشتباه هستند. این به این معنا نیست که Agile در مقیاس بزرگ موفق نخواهد بود؛ احتمالاً موفق خواهد بود. موفقیت به این معنا که شرکت‌های بزرگ محصولات و ایده‌های مربوط به Scaling را خریداری خواهند کرد و مشاوران و شرکت‌های آموزشی با فروش آنچه این شرکت‌های بزرگ می‌خواهند، خود را غنی خواهند کرد.   
report erratum • discuss   
Chapter 21. Scaling Agile • 131   
  
همانطور که Rolling Stones به ما یادآوری می‌کند، شما همیشه نمی‌توانید آنچه را که می‌خواهید به دست آورید. متأسفانه، برخلاف این آهنگ، یک شرکت بزرگ همیشه می‌تواند آنچه را که می‌خواهد به دست آورد—در این مورد، آموزش‌های گران‌قیمت در یک رویکرد سنگین که خود را به عنوان Scaled Agile معرفی می‌کند. و قطعاً مقداری منفعت خواهند برد. هر توجهی به بهبود معمولاً بهتر از بی‌توجهی کامل است. و تا حدی که این رویکردهای مختلف شامل برخی ایده‌های واقعی Agile باشند، سازمان‌ها مقداری از آن را نیز دریافت خواهند کرد.   
  
من اینجا هستم تا درباره چیزی که آهنگ ارائه می‌دهد صحبت کنم: من اینجا هستم تا درباره چیزی که شما نیاز دارید صحبت کنم. یک شرکت بزرگ واقعاً چه چیزی باید درباره اعمال ایده‌های Agile در سراسر سازمان بداند؟   
Chapter 21. Scaling Agile • 132   
report erratum • discuss

================================================================================

# Section 130

## Original Text:

Scaling Agile • 132  
report erratum • discuss]  
  
Agile issimple—it justisn’t easy. Agile isquite simple. The most popular Agile approach,  
Scrum, hasjustthree roles, ahandful ofactivities, and one  
major artifact: running tested software. That doesn’t mean Agile iseasy. It’sstillhard todecide what  
product would bedesirable, and it’sstillhard towrite soft-  
ware that does what isasked for. Itis,however, quite simple. Simplicity istheessence ofwhat makes upAgility. Soif  
Agile issimple, what about so-called scaled Agile. report erratum • discussChapter 21. Scaling Agile • 133  
  
[CONTEXT FROM PREVIOUS PAGE: Agile issimple—it justisn’t easy. Agile isquite simple. The most popular Agile approach,  
Scrum, hasjustthree roles, ahandful ofactivities, and one  
major artifact: running tested software. That doesn’t mean Agile iseasy. It’sstillhard todecide what  
product would bedesirable, and it’sstillhard towrite soft-  
ware that does what isasked for. Itis,however, quite simple. Simplicity istheessence ofwhat makes upAgility. Soif  
Agile issimple, what about so-called scaled Agile. report erratum • discussChapter 21. Scaling Agile • 133]  
  
Scaled Agile must besimple—or itisn’t Agile. Chet Hendrickson points outthat since Agile issimple, a  
scaled version ofAgile should also bethat simple, oreven  
simpler. Otherwise, itwill nolonger beAgile. W eshould  
look with great suspicion ataso-called “Scaled Agile”  
approach that iscomplex. Along thesame lines, Arlo Belshee suggests that ifallyour  
development teams have become fluent inAgile software  
development, scaling isnotaproblem. Ifallyour teams can  
slice stories small, select anumber that they canaccomplish  
inaSprint (orotherwise within time estimates), and deliver  
integrated software that isfree ofdefects, then “scaling”  
should beeasy. Diana Larsen and JimShore, originators of  
thenotion offluency inthiscontext, make similar points. Let’s explore this. Agile issimple (but noteasy).

## Translated Text:

مقیاس‌بندی Agile • 132   
گزارش خطا • بحث کنید   
  
Agile ساده است—فقط آسان نیست. Agile کاملاً ساده است. محبوب‌ترین روش Agile،   
Scrum، فقط سه نقش، چند فعالیت و یک   
artifact اصلی دارد: اجرای نرم‌افزار تست‌شده. این به این معنا نیست که Agile آسان است. هنوز هم تصمیم‌گیری درباره اینکه چه   
محصولی مطلوب خواهد بود دشوار است، و هنوز هم نوشتن نرم‌افزاری که   
آنچه خواسته شده را انجام دهد سخت است. با این حال، کاملاً ساده است. سادگی جوهره چیزی است که Agile را تشکیل می‌دهد. بنابراین اگر   
Agile ساده است، پس مقیاس‌بندی‌شده Agile چگونه است؟ گزارش خطا • بحث کنید فصل 21. مقیاس‌بندی Agile • 133   
  
[متن از صفحه قبلی: Agile ساده است—فقط آسان نیست. Agile کاملاً ساده است. محبوب‌ترین روش Agile،   
Scrum، فقط سه نقش، چند فعالیت و یک   
artifact اصلی دارد: اجرای نرم‌افزار تست‌شده. این به این معنا نیست که Agile آسان است. هنوز هم تصمیم‌گیری درباره اینکه چه   
محصولی مطلوب خواهد بود دشوار است، و هنوز هم نوشتن نرم‌افزاری که   
آنچه خواسته شده را انجام دهد سخت است. با این حال، کاملاً ساده است. سادگی جوهره چیزی است که Agile را تشکیل می‌دهد. بنابراین اگر   
Agile ساده است، پس مقیاس‌بندی‌شده Agile چگونه است؟ گزارش خطا • بحث کنید فصل 21. مقیاس‌بندی Agile • 133]   
  
مقیاس‌بندی‌شده Agile باید ساده باشد—یا Agile نیست. Chet Hendrickson اشاره می‌کند که از آنجا که Agile ساده است،   
نسخه مقیاس‌بندی‌شده Agile نیز باید به همان اندازه ساده باشد، یا حتی   
ساده‌تر. در غیر این صورت، دیگر Agile نخواهد بود. ما باید   
با شک و تردید زیاد به چیزی که به عنوان "مقیاس‌بندی‌شده Agile"   
پیچیده معرفی می‌شود نگاه کنیم. در همین راستا، Arlo Belshee پیشنهاد می‌کند که اگر همه تیم‌های   
توسعه شما در Agile software development مهارت پیدا کرده باشند، مقیاس‌بندی مشکلی نخواهد بود. اگر همه تیم‌های شما بتوانند   
User Storyها را کوچک کنند، تعدادی را انتخاب کنند که بتوانند در یک Sprint (یا به هر حال در محدوده تخمین‌های زمانی) انجام دهند، و   
نرم‌افزار یکپارچه‌ای که بدون نقص باشد تحویل دهند، پس "مقیاس‌بندی"   
باید آسان باشد. Diana Larsen و Jim Shore، خالقان   
مفهوم fluency در این زمینه، نکات مشابهی را مطرح می‌کنند. بیایید این موضوع را بررسی کنیم. Agile ساده است (اما آسان نیست).

================================================================================

# Section 131

## Original Text:

Ifyour  
individual teams could really execute software development  
intheAgile style, might “Scaling Agile” beeasy. Chapter 21. Scaling Agile • 134  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Scaled Agile must besimple—or itisn’t Agile. Chet Hendrickson points outthat since Agile issimple, a  
scaled version ofAgile should also bethat simple, oreven  
simpler. Otherwise, itwill nolonger beAgile. W eshould  
look with great suspicion ataso-called “Scaled Agile”  
approach that iscomplex. Along thesame lines, Arlo Belshee suggests that ifallyour  
development teams have become fluent inAgile software  
development, scaling isnotaproblem. Ifallyour teams can  
slice stories small, select anumber that they canaccomplish  
inaSprint (orotherwise within time estimates), and deliver  
integrated software that isfree ofdefects, then “scaling”  
should beeasy. Diana Larsen and JimShore, originators of  
thenotion offluency inthiscontext, make similar points. Let’s explore this. Agile issimple (but noteasy). Ifyour  
individual teams could really execute software development  
intheAgile style, might “Scaling Agile” beeasy. Chapter 21. Scaling Agile • 134  
report erratum • discuss]  
  
Ifyour teams aretruly Agile…  
Agile teams work daily with their business-side associates  
(Agile Manifesto Principle 4). They deliver working software  
frequently, every couple ofweeks (Principle 3). They mea-  
sure themselves with working software (Principle 7),work  
inasustainable fashion (Principle 8),and pay constant  
attention totechnical excellence and good design (Principle  
9). And soon. report erratum • discussChapter 21. Scaling Agile • 135  
  
[CONTEXT FROM PREVIOUS PAGE: Ifyour teams aretruly Agile…  
Agile teams work daily with their business-side associates  
(Agile Manifesto Principle 4). They deliver working software  
frequently, every couple ofweeks (Principle 3).

## Translated Text:

اگر تیم‌های شما واقعاً بتوانند توسعه نرم‌افزار را به سبک Agile اجرا کنند، ممکن است "Scaling Agile" آسان باشد.   
فصل ۲۱. Scaling Agile • ۱۳۴   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: Scaled Agile باید ساده باشد—وگرنه Agile نیست. چت هندریکسون اشاره می‌کند که از آنجا که Agile ساده است، نسخه مقیاس‌پذیر Agile نیز باید به همان اندازه ساده یا حتی ساده‌تر باشد. در غیر این صورت، دیگر Agile نخواهد بود. ما باید با شک و تردید زیاد به یک رویکرد به اصطلاح "Scaled Agile" که پیچیده است نگاه کنیم. به همین ترتیب، آرلو بلشی پیشنهاد می‌کند که اگر تمام تیم‌های توسعه شما در توسعه نرم‌افزار Agile مهارت پیدا کرده باشند، مقیاس‌پذیری مشکلی نخواهد بود. اگر تمام تیم‌های شما بتوانند User Storyها را به بخش‌های کوچک تقسیم کنند، تعداد مشخصی را که می‌توانند در یک Sprint (یا در غیر این صورت در محدوده زمانی تخمین زده شده) انجام دهند انتخاب کنند و نرم‌افزار یکپارچه‌ای که بدون نقص است ارائه دهند، در این صورت "مقیاس‌پذیری" باید آسان باشد. دیانا لارسن و جیم شور، که مفهوم مهارت در این زمینه را مطرح کرده‌اند، نکات مشابهی را بیان می‌کنند. بیایید این موضوع را بررسی کنیم. Agile ساده است (اما آسان نیست). اگر تیم‌های شما واقعاً بتوانند توسعه نرم‌افزار را به سبک Agile اجرا کنند، ممکن است "Scaling Agile" آسان باشد.   
فصل ۲۱. Scaling Agile • ۱۳۴   
گزارش خطا • بحث]   
  
اگر تیم‌های شما واقعاً Agile باشند...   
تیم‌های Agile روزانه با همکاران خود در بخش کسب‌وکار کار می‌کنند (اصل چهارم Agile Manifesto). آن‌ها نرم‌افزار کاربردی را به‌طور مکرر، هر چند هفته یک‌بار ارائه می‌دهند (اصل سوم). آن‌ها خود را با نرم‌افزار کاربردی ارزیابی می‌کنند (اصل هفتم)، به صورت پایدار کار می‌کنند (اصل هشتم)، و توجه مداوم به برتری فنی و طراحی خوب دارند (اصل نهم). و به همین ترتیب.   
گزارش خطا • بحث   
فصل ۲۱. Scaling Agile • ۱۳۵   
  
[متن از صفحه قبلی: اگر تیم‌های شما واقعاً Agile باشند...   
تیم‌های Agile روزانه با همکاران خود در بخش کسب‌وکار کار می‌کنند (اصل چهارم Agile Manifesto). آن‌ها نرم‌افزار کاربردی را به‌طور مکرر، هر چند هفته یک‌بار ارائه می‌دهند (اصل سوم).

================================================================================

# Section 132

## Original Text:

They mea-  
sure themselves with working software (Principle 7),work  
inasustainable fashion (Principle 8),and pay constant  
attention totechnical excellence and good design (Principle  
9). And soon. report erratum • discussChapter 21. Scaling Agile • 135]  
  
…Imean truly Agile…  
Fluent Agile teams, after justalittle jostling when they start  
up,produce aconsistent flow offeatures, and they drive  
defects down tolevels farbelow what the same team  
accomplished before they became fluent inAgile. Fluent Agile teams arevisibly Agile, visibly fluent. They get  
things truly done, ataconsistent and predictable pace. If  
your teams areuptothat…  
…you might already bedone. Chapter 21. Scaling Agile • 136  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: …Imean truly Agile…  
Fluent Agile teams, after justalittle jostling when they start  
up,produce aconsistent flow offeatures, and they drive  
defects down tolevels farbelow what the same team  
accomplished before they became fluent inAgile. Fluent Agile teams arevisibly Agile, visibly fluent. They get  
things truly done, ataconsistent and predictable pace. If  
your teams areuptothat…  
…you might already bedone. Chapter 21. Scaling Agile • 136  
report erratum • discuss]  
  
Sothere y o u are. Allyour teams arecapable ofproducing  
working software, every two weeks, working daily with the  
business-side people who describe what theteam needs to  
build. Y o u might bedone scaling Agile, ifeverything your organi -  
zation builds canbebuilt byasingle Agile team. Really. Think about thisforamoment. Ifeverything y o u do  
could bebuilt byasingle small team, scaling Agile comes  
down tohaving each team learn tobuild intheAgile fashion,  
then hooking them upwith abusiness-side person toguide  
what they build. Done. Agile Scaled. Noextra work beyond thebasics. The  
basics arehard enough ofcourse. We’ve explored that else-  
where inthebook. Butthere’s nobigcorporate rollout/tran -  
sition/Enterprise Agile that y o u need todo.

## Translated Text:

آن‌ها خود را با نرم‌افزارهای کارآمد (Principle 7) می‌سنجند، به شیوه‌ای پایدار کار می‌کنند (Principle 8)، و توجه مداومی به برتری فنی و طراحی خوب دارند (Principle 9). و به همین ترتیب.   
گزارش خطا • بحث   
فصل 21. Scaling Agile • 135   
  
...منظورم واقعاً Agile است...   
تیم‌های Fluent Agile، پس از کمی تکان‌های اولیه هنگام شروع، جریان پایداری از Features تولید می‌کنند و Defects را به سطحی بسیار پایین‌تر از آنچه همان تیم قبل از Fluent شدن در Agile به آن دست یافته بود، کاهش می‌دهند. تیم‌های Fluent Agile به وضوح Agile هستند، به وضوح Fluent. آن‌ها کارها را واقعاً انجام می‌دهند، با سرعتی پایدار و قابل پیش‌بینی. اگر تیم‌های شما به این سطح رسیده‌اند...   
...شاید شما کارتان تمام شده باشد.   
فصل 21. Scaling Agile • 136   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: ...منظورم واقعاً Agile است...   
تیم‌های Fluent Agile، پس از کمی تکان‌های اولیه هنگام شروع، جریان پایداری از Features تولید می‌کنند و Defects را به سطحی بسیار پایین‌تر از آنچه همان تیم قبل از Fluent شدن در Agile به آن دست یافته بود، کاهش می‌دهند. تیم‌های Fluent Agile به وضوح Agile هستند، به وضوح Fluent. آن‌ها کارها را واقعاً انجام می‌دهند، با سرعتی پایدار و قابل پیش‌بینی. اگر تیم‌های شما به این سطح رسیده‌اند...   
...شاید شما کارتان تمام شده باشد.   
فصل 21. Scaling Agile • 136   
گزارش خطا • بحث]   
  
پس اینجا هستید. همه تیم‌های شما قادر به تولید نرم‌افزارهای کارآمد هستند، هر دو هفته یک بار، و روزانه با افراد سمت کسب‌وکار که توصیف می‌کنند تیم چه چیزی باید بسازد، کار می‌کنند. شاید شما کارتان با Scaling Agile تمام شده باشد، اگر هر چیزی که سازمان شما می‌سازد بتواند توسط یک تیم Agile ساخته شود. واقعاً. لحظه‌ای به این فکر کنید. اگر هر چیزی که انجام می‌دهید بتواند توسط یک تیم کوچک ساخته شود، Scaling Agile به این خلاصه می‌شود که هر تیم یاد بگیرد به شیوه Agile بسازد، سپس آن‌ها را به یک فرد سمت کسب‌وکار متصل کنید تا آنچه می‌سازند را راهنمایی کند. تمام. Agile Scaled. هیچ کار اضافی فراتر از اصول اولیه. البته اصول اولیه به اندازه کافی سخت هستند. ما این را در جای دیگری از کتاب بررسی کرده‌ایم. اما هیچ تغییر بزرگ سازمانی/انتقال/Enterprise Agile که نیاز به انجام آن داشته باشید وجود ندارد.

================================================================================

# Section 133

## Original Text:

report erratum • discussChapter 21. Scaling Agile • 137  
  
[CONTEXT FROM PREVIOUS PAGE: Sothere y o u are. Allyour teams arecapable ofproducing  
working software, every two weeks, working daily with the  
business-side people who describe what theteam needs to  
build. Y o u might bedone scaling Agile, ifeverything your organi -  
zation builds canbebuilt byasingle Agile team. Really. Think about thisforamoment. Ifeverything y o u do  
could bebuilt byasingle small team, scaling Agile comes  
down tohaving each team learn tobuild intheAgile fashion,  
then hooking them upwith abusiness-side person toguide  
what they build. Done. Agile Scaled. Noextra work beyond thebasics. The  
basics arehard enough ofcourse. We’ve explored that else-  
where inthebook. Butthere’s nobigcorporate rollout/tran -  
sition/Enterprise Agile that y o u need todo. report erratum • discussChapter 21. Scaling Agile • 137]  
  
What ifyouwant more than oneteam cando. Asingle Agile team that canreally dothis stuff produces  
multiple features every couple ofweeks. It’snoteasy tokeep  
even one team working atcapacity: y o u have tohave alot  
ofproduct ideas todoit. Butmaybe y o u have ahuge prod -  
uct, like aword processor orsome graphics program for  
editing photographs. Y o u feel there isenough work there  
tokeep multiple teams busy. Well, first ofall,prove it. Getasingle team working onyour  
product uptoAgile quality. Then look attherate atwhich  
they deliver features. Seeify o u really need more features  
than that. Odds are, y o u won’t: your customers probably  
can’t absorb new capability faster than asingle team can  
deliver. Butmaybe there isenough work tokeep more than  
oneteam busy. Aha. Now we’ll have toscale Agile…won’t w e. Chapter 21. Scaling Agile • 138  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: What ifyouwant more than oneteam cando. Asingle Agile team that canreally dothis stuff produces  
multiple features every couple ofweeks.

## Translated Text:

گزارش خطا • بحث   
فصل ۲۱. مقیاس‌بندی Agile • ۱۳۷   
  
[متن قبلی: پس اینجا هستید. تمام تیم‌های شما قادر به تولید نرم‌افزار کاربردی هر دو هفته یک بار هستند، و روزانه با افراد سمت کسب‌وکار که توضیح می‌دهند تیم باید چه چیزی Build کند، کار می‌کنند. ممکن است مقیاس‌بندی Agile را به پایان رسانده باشید، اگر هر چیزی که سازمان شما می‌سازد بتواند توسط یک تیم Agile ساخته شود. واقعاً. برای یک لحظه به این فکر کنید. اگر هر چیزی که انجام می‌دهید بتواند توسط یک تیم کوچک ساخته شود، مقیاس‌بندی Agile به این خلاصه می‌شود که هر تیم یاد بگیرد چگونه به روش Agile Build کند، سپس آن‌ها را به یک فرد سمت کسب‌وکار متصل کنید تا راهنمایی کند چه چیزی Build کنند. تمام شد. Agile مقیاس‌بندی شد. هیچ کار اضافی فراتر از اصول پایه وجود ندارد. البته اصول پایه به اندازه کافی سخت هستند. ما این را در جای دیگری از کتاب بررسی کرده‌ایم. اما هیچ راه‌اندازی/انتقال/Agile سازمانی بزرگی که نیاز باشد انجام دهید وجود ندارد. گزارش خطا • بحث   
فصل ۲۱. مقیاس‌بندی Agile • ۱۳۷]   
  
اگر بخواهید بیشتر از یک تیم بتواند کاری انجام دهد چه؟ یک تیم Agile که واقعاً بتواند این کار را انجام دهد، هر چند هفته یک بار چندین Feature تولید می‌کند. نگه داشتن حتی یک تیم در ظرفیت کامل آسان نیست: شما باید ایده‌های زیادی برای Product داشته باشید تا این کار را انجام دهید. اما شاید شما یک Product بزرگ داشته باشید، مانند یک Word Processor یا یک برنامه گرافیکی برای ویرایش عکس‌ها. شما احساس می‌کنید که کار کافی وجود دارد تا چندین تیم مشغول باشند. خب، اول از همه، این را ثابت کنید. یک تیم را روی Product خود به سطح کیفیت Agile برسانید. سپس نرخ تحویل Featureها توسط آن‌ها را بررسی کنید. ببینید آیا واقعاً به Featureهای بیشتری از آن نیاز دارید یا نه. احتمالاً نیاز ندارید: مشتریان شما احتمالاً نمی‌توانند قابلیت‌های جدید را سریع‌تر از آنچه یک تیم می‌تواند Deliver کند، جذب کنند. اما شاید کار کافی وجود داشته باشد تا بیش از یک تیم مشغول باشد. آها. حالا باید Agile را مقیاس‌بندی کنیم... مگر نه؟   
فصل ۲۱. مقیاس‌بندی Agile • ۱۳۸   
گزارش خطا • بحث   
  
[متن قبلی: اگر بخواهید بیشتر از یک تیم بتواند کاری انجام دهد چه؟ یک تیم Agile که واقعاً بتواند این کار را انجام دهد، هر چند هفته یک بار چندین Feature تولید می‌کند.]

================================================================================

# Section 134

## Original Text:

It’snoteasy tokeep  
even one team working atcapacity: y o u have tohave alot  
ofproduct ideas todoit. Butmaybe y o u have ahuge prod -  
uct, like aword processor orsome graphics program for  
editing photographs. Y o u feel there isenough work there  
tokeep multiple teams busy. Well, first ofall,prove it. Getasingle team working onyour  
product uptoAgile quality. Then look attherate atwhich  
they deliver features. Seeify o u really need more features  
than that. Odds are, y o u won’t: your customers probably  
can’t absorb new capability faster than asingle team can  
deliver. Butmaybe there isenough work tokeep more than  
oneteam busy. Aha. Now we’ll have toscale Agile…won’t w e. Chapter 21. Scaling Agile • 138  
report erratum • discuss]  
  
Feature teams  
Well, maybe not. W a y back inthelastcentury, theidea of  
thefeature team w a s devised. Afeature team isasmall team  
whose jobitistodeliver features into aproduct. T ogetmore  
features, y o uadd more feature teams, alldelivering software  
into asingle product. Want more features perunit time. Add another feature team, getmore features. There’s notmuch involved inscaling this w a y , isthere. If  
every team knows how todowhat areal Agile team knows  
how todo,y o u can add feature teams, and any product  
made offeatures cangoasfastasy o u want. Aren’t w eskipping something. How dothose teams coordi -  
nate. Now we’ve gotmultiple teams doing features. How  
canthey avoid stepping oneach other’s toes. report erratum • discussChapter 21. Scaling Agile • 139  
  
[CONTEXT FROM PREVIOUS PAGE: Feature teams  
Well, maybe not. W a y back inthelastcentury, theidea of  
thefeature team w a s devised. Afeature team isasmall team  
whose jobitistodeliver features into aproduct. T ogetmore  
features, y o uadd more feature teams, alldelivering software  
into asingle product. Want more features perunit time. Add another feature team, getmore features. There’s notmuch involved inscaling this w a y , isthere.

## Translated Text:

ترجمه:  
  
نگه داشتن حتی یک تیم در ظرفیت کامل آسان نیست: شما باید ایده‌های زیادی برای محصول داشته باشید تا این کار را انجام دهید. اما شاید شما یک محصول بزرگ داشته باشید، مانند یک واژه‌پرداز یا یک برنامه گرافیکی برای ویرایش عکس‌ها. شما احساس می‌کنید که کار کافی وجود دارد تا چندین تیم مشغول شوند. خب، اول از همه، ثابت کنید. یک تیم را روی محصول خود تا کیفیت Agile کار کنید. سپس به نرخ تحویل ویژگی‌ها توسط آن‌ها نگاه کنید. ببینید آیا واقعاً به ویژگی‌های بیشتری از آن نیاز دارید یا نه. احتمالاً نیاز ندارید: مشتریان شما احتمالاً نمی‌توانند قابلیت‌های جدید را سریع‌تر از آنچه یک تیم بتواند تحویل دهد جذب کنند. اما شاید کار کافی وجود داشته باشد تا بیش از یک تیم مشغول شود. آها. حالا باید Agile را Scale کنیم... مگر نه؟  
  
فصل ۲۱. Scaling Agile • ۱۳۸   
گزارش خطا • بحث کنید  
  
تیم‌های ویژگی   
خب، شاید نه. خیلی دور در قرن گذشته، ایده تیم ویژگی مطرح شد. یک تیم ویژگی یک تیم کوچک است که وظیفه‌اش تحویل ویژگی‌ها به یک محصول است. برای دریافت ویژگی‌های بیشتر، شما تیم‌های ویژگی بیشتری اضافه می‌کنید، همه در حال تحویل نرم‌افزار به یک محصول واحد. می‌خواهید ویژگی‌های بیشتری در واحد زمان داشته باشید؟ یک تیم ویژگی دیگر اضافه کنید، ویژگی‌های بیشتری دریافت کنید. در این روش برای Scaling چیز زیادی درگیر نیست، مگر نه؟ اگر هر تیم بداند چگونه کاری که یک تیم واقعی Agile می‌داند انجام دهد، شما می‌توانید تیم‌های ویژگی اضافه کنید و هر محصولی که از ویژگی‌ها ساخته شده باشد می‌تواند به همان سرعتی که شما می‌خواهید پیش برود. آیا چیزی را جا نمی‌اندازیم؟ این تیم‌ها چگونه هماهنگ می‌شوند؟ حالا ما چندین تیم داریم که ویژگی‌ها را انجام می‌دهند. آن‌ها چگونه می‌توانند از زیر پا گذاشتن یکدیگر جلوگیری کنند؟  
  
گزارش خطا • بحث کنید   
فصل ۲۱. Scaling Agile • ۱۳۹

================================================================================

# Section 135

## Original Text:

If  
every team knows how todowhat areal Agile team knows  
how todo,y o u can add feature teams, and any product  
made offeatures cangoasfastasy o u want. Aren’t w eskipping something. How dothose teams coordi -  
nate. Now we’ve gotmultiple teams doing features. How  
canthey avoid stepping oneach other’s toes. report erratum • discussChapter 21. Scaling Agile • 139]  
  
Agile teams coordinate using tests. Remember that Agile teams doalarge number ofsmall  
features every twoweeks. Asingle team caneasily dofifteen  
ortwenty such features inatwo-week iteration. How do  
they manage nottogetineach other’s way. Itturns outtobesimple. Fluent Agile teams build agrowing  
container ofautomated checks, using acceptance test-driven  
development and test-driven development. These checks  
help teams know when they have completed afeature. However, they also serve asagrowing collection ofregres -  
sion checks that ensure allthefeatures built keep onwork -  
ing. Ifwe’re using multiple feature teams, itworks thesame w a y. Each team, every time itbuilds anew small feature, adds  
that feature, with itsautomated checks, into thecommon  
codebase. Alltheteams dothis daily, just asasingle team  
would. They keep allthechecks running, allthetime. Iffrom  
time totime ateam tries tocheck something inand tests fail,  
they fixtheproblem before checking insothat thecurrent  
codebase always runs allthechecks. Chapter 21. Scaling Agile • 140  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Agile teams coordinate using tests. Remember that Agile teams doalarge number ofsmall  
features every twoweeks. Asingle team caneasily dofifteen  
ortwenty such features inatwo-week iteration. How do  
they manage nottogetineach other’s way. Itturns outtobesimple. Fluent Agile teams build agrowing  
container ofautomated checks, using acceptance test-driven  
development and test-driven development. These checks  
help teams know when they have completed afeature.

## Translated Text:

اگر   
هر تیم بداند چگونه کاری را انجام دهد که یک تیم واقعی Agile می‌داند چگونه انجام دهد، شما می‌توانید تیم‌های Feature اضافه کنید و هر محصولی که از Featureها ساخته شده باشد می‌تواند به همان سرعتی که می‌خواهید پیش برود. آیا چیزی را جا نینداخته‌ایم؟ این تیم‌ها چگونه هماهنگ می‌شوند؟ حالا ما چندین تیم داریم که روی Featureها کار می‌کنند. چگونه می‌توانند از تداخل با یکدیگر جلوگیری کنند؟   
\*\*Chapter 21. Scaling Agile • 139\*\*   
  
تیم‌های Agile با استفاده از Testها هماهنگ می‌شوند. به یاد داشته باشید که تیم‌های Agile تعداد زیادی Feature کوچک را هر دو هفته انجام می‌دهند. یک تیم به‌راحتی می‌تواند پانزده یا بیست Feature را در یک Iteration دو هفته‌ای انجام دهد. چگونه از تداخل با یکدیگر جلوگیری می‌کنند؟ معلوم می‌شود که این کار ساده است. تیم‌های Agile ماهر یک مجموعه در حال رشد از Automated Checks را می‌سازند، با استفاده از Acceptance Test-Driven Development و Test-Driven Development. این Checkها به تیم‌ها کمک می‌کنند بدانند چه زمانی یک Feature را کامل کرده‌اند.   
  
با این حال، این Checkها همچنین به‌عنوان یک مجموعه در حال رشد از Regression Checkها عمل می‌کنند که اطمینان می‌دهند تمام Featureهای ساخته‌شده همچنان به درستی کار می‌کنند. اگر ما از چندین تیم Feature استفاده کنیم، این کار به همان روش عمل می‌کند. هر تیم، هر بار که یک Feature کوچک جدید می‌سازد، آن Feature را به همراه Automated Checks آن به Common Codebase اضافه می‌کند. همه تیم‌ها این کار را به‌صورت روزانه انجام می‌دهند، دقیقاً مانند یک تیم واحد. آنها همه Checkها را همیشه در حال اجرا نگه می‌دارند. اگر گاهی اوقات یک تیم سعی کند چیزی را Check-in کند و Testها Fail شوند، آنها مشکل را قبل از Check-in حل می‌کنند تا Codebase فعلی همیشه تمام Checkها را به درستی اجرا کند.   
\*\*Chapter 21. Scaling Agile • 140\*\*

================================================================================

# Section 136

## Original Text:

However, they also serve asagrowing collection ofregres -  
sion checks that ensure allthefeatures built keep onwork -  
ing. Ifwe’re using multiple feature teams, itworks thesame w a y. Each team, every time itbuilds anew small feature, adds  
that feature, with itsautomated checks, into thecommon  
codebase. Alltheteams dothis daily, just asasingle team  
would. They keep allthechecks running, allthetime. Iffrom  
time totime ateam tries tocheck something inand tests fail,  
they fixtheproblem before checking insothat thecurrent  
codebase always runs allthechecks. Chapter 21. Scaling Agile • 140  
report erratum • discuss]  
  
Might there beaconflict between things done across teams. Possibly, and ifthat happens, theteams coordinate tosee  
what happened. Butthegeneral practice isquite simple: if  
thechecks were running before y o uputyour change in,and  
they’re not running after y o u put your change in,your  
change broke something. Y o u find that something, and fix  
it,sothat allthechecks run—yours plus allthehistorical  
ones. Agile teams dothis asamatter ofcourse. They learn todo  
smaller and smaller releases. When they usesmall releases,  
thechance that they break something isvery small. When  
(rarely) they dobreak something, it’seasy tofind theissue,  
because only asmall amount ofcode has been added or  
changed. report erratum • discussChapter 21. Scaling Agile • 141  
  
[CONTEXT FROM PREVIOUS PAGE: Might there beaconflict between things done across teams. Possibly, and ifthat happens, theteams coordinate tosee  
what happened. Butthegeneral practice isquite simple: if  
thechecks were running before y o uputyour change in,and  
they’re not running after y o u put your change in,your  
change broke something. Y o u find that something, and fix  
it,sothat allthechecks run—yours plus allthehistorical  
ones. Agile teams dothis asamatter ofcourse. They learn todo  
smaller and smaller releases. When they usesmall releases,  
thechance that they break something isvery small.

## Translated Text:

با این حال، آنها همچنین به عنوان یک مجموعه در حال رشد از regression checks عمل می‌کنند که اطمینان می‌دهند تمام features ساخته‌شده به کار خود ادامه می‌دهند. اگر ما از چندین تیم feature استفاده کنیم، همین روند به همان شکل کار می‌کند. هر تیم، هر بار که یک feature کوچک جدید می‌سازد، آن feature را همراه با automated checks آن به common codebase اضافه می‌کند. تمام تیم‌ها این کار را به صورت روزانه انجام می‌دهند، درست مانند یک تیم واحد. آنها تمام checks را همیشه در حال اجرا نگه می‌دارند. اگر گاهی اوقات یک تیم بخواهد چیزی را check کند و tests شکست بخورند، آنها مشکل را قبل از check-in حل می‌کنند تا اطمینان حاصل شود که current codebase همیشه تمام checks را اجرا می‌کند.   
فصل ۲۱. Scaling Agile • ۱۴۰   
گزارش خطا • بحث کنید   
  
آیا ممکن است بین کارهایی که توسط تیم‌ها انجام می‌شود، تضادی وجود داشته باشد؟ احتمالاً، و اگر این اتفاق بیفتد، تیم‌ها هماهنگ می‌شوند تا ببینند چه اتفاقی افتاده است. اما رویه کلی بسیار ساده است: اگر checks قبل از اینکه شما تغییر خود را اضافه کنید در حال اجرا بودند و بعد از اضافه کردن تغییر شما دیگر اجرا نمی‌شوند، تغییر شما چیزی را خراب کرده است. شما آن مشکل را پیدا می‌کنید و آن را حل می‌کنید تا تمام checks اجرا شوند—هم checks شما و هم تمام historical ones. تیم‌های Agile این کار را به صورت عادی انجام می‌دهند. آنها یاد می‌گیرند که releases کوچک‌تر و کوچک‌تری انجام دهند. وقتی آنها از releases کوچک استفاده می‌کنند، احتمال اینکه چیزی را خراب کنند بسیار کم است. وقتی (به ندرت) چیزی را خراب می‌کنند، پیدا کردن مشکل آسان است، زیرا فقط مقدار کمی از code اضافه یا تغییر داده شده است.   
گزارش خطا • بحث کنید   
فصل ۲۱. Scaling Agile • ۱۴۱

================================================================================

# Section 137

## Original Text:

When  
(rarely) they dobreak something, it’seasy tofind theissue,  
because only asmall amount ofcode has been added or  
changed. report erratum • discussChapter 21. Scaling Agile • 141]  
  
OK, feature teams, butwhat about infrastructure. Ifyour product isreally bigenough tousemultiple feature  
teams, they’ll berelying onsome common infrastructure. What about changes tothat. Same w a y. Agile teams change their infrastructure asneeded. They dosofreely, every couple ofweeks, bysupporting  
their changes with automated checks. Feature teams cando  
thesame thing, with each team making thechanges itneeds,  
adding checks tothepool, and checking incode frequently. Will y o uneed aspecialized infrastructure team. Quite often,  
ifyou’re fluent inAgile, y o u won’t. Specialist teams very  
often dissolve inanAgile situation. Butify o u dochoose to  
have such ateam, and they’re Agile, they can smoothly  
produce infrastructure changes, supported byautomated  
checks, insupport ofmultiple feature teams. Irecommend  
letting your feature teams handle infrastructure changes,  
coordinating among themselves asneeded. But ify o u do  
choose tohave aspecialized team forinfrastructure, despite  
that advice, there’s stillnoneed forspecial scaling. Remember, it’sunlikely that you’ll need feature teams atall  
ifyour individual teams candoAgile. But ify o u do,y o u  
won’t need any special infrastructure tohave feature  
teams—you justneed multiple empowered teams who can,  
and will, coordinate among themselves asneeded. Chapter 21. Scaling Agile • 142  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: OK, feature teams, butwhat about infrastructure. Ifyour product isreally bigenough tousemultiple feature  
teams, they’ll berelying onsome common infrastructure. What about changes tothat. Same w a y. Agile teams change their infrastructure asneeded. They dosofreely, every couple ofweeks, bysupporting  
their changes with automated checks.

## Translated Text:

وقتی (به‌ندرت) چیزی را خراب می‌کنند، پیدا کردن مشکل آسان است، زیرا فقط مقدار کمی از کد اضافه یا تغییر داده شده است.   
report erratum • discuss   
فصل 21. Scaling Agile • 141   
  
خب، تیم‌های feature، اما زیرساخت چه می‌شود؟ اگر محصول شما واقعاً به اندازه‌ای بزرگ باشد که از چندین تیم feature استفاده کند، آن‌ها به یک زیرساخت مشترک متکی خواهند بود. تغییرات در آن چه می‌شود؟ همان‌طور. تیم‌های Agile زیرساخت خود را طبق نیاز تغییر می‌دهند. آن‌ها این کار را آزادانه، هر چند هفته یک‌بار، با پشتیبانی از تغییرات خود توسط automated checks انجام می‌دهند. تیم‌های feature می‌توانند همین کار را انجام دهند، با این تفاوت که هر تیم تغییرات مورد نیاز خود را ایجاد می‌کند، checks را به pool اضافه می‌کند و کد را به‌طور مکرر check-in می‌کند.   
  
آیا به یک تیم زیرساخت تخصصی نیاز خواهید داشت؟ اغلب، اگر در Agile مهارت داشته باشید، نیازی نخواهید داشت. تیم‌های تخصصی اغلب در شرایط Agile منحل می‌شوند. اما اگر تصمیم بگیرید که چنین تیمی داشته باشید و آن‌ها Agile باشند، می‌توانند تغییرات زیرساخت را به‌صورت روان، با پشتیبانی از automated checks، برای حمایت از چندین تیم feature تولید کنند. من توصیه می‌کنم اجازه دهید تیم‌های feature شما تغییرات زیرساخت را مدیریت کنند و در صورت نیاز بین خود هماهنگی کنند. اما اگر تصمیم بگیرید که علی‌رغم این توصیه یک تیم تخصصی برای زیرساخت داشته باشید، همچنان نیازی به scaling خاص نخواهید داشت.   
  
به یاد داشته باشید، بعید است که اصلاً به تیم‌های feature نیاز داشته باشید اگر تیم‌های فردی شما بتوانند Agile باشند. اما اگر نیاز داشته باشید، به هیچ زیرساخت خاصی برای داشتن تیم‌های feature نیاز نخواهید داشت—فقط به چندین تیم توانمند نیاز دارید که بتوانند و بخواهند در صورت نیاز بین خود هماهنگی کنند.   
فصل 21. Scaling Agile • 142   
report erratum • discuss

================================================================================

# Section 138

## Original Text:

Feature teams cando  
thesame thing, with each team making thechanges itneeds,  
adding checks tothepool, and checking incode frequently. Will y o uneed aspecialized infrastructure team. Quite often,  
ifyou’re fluent inAgile, y o u won’t. Specialist teams very  
often dissolve inanAgile situation. Butify o u dochoose to  
have such ateam, and they’re Agile, they can smoothly  
produce infrastructure changes, supported byautomated  
checks, insupport ofmultiple feature teams. Irecommend  
letting your feature teams handle infrastructure changes,  
coordinating among themselves asneeded. But ify o u do  
choose tohave aspecialized team forinfrastructure, despite  
that advice, there’s stillnoneed forspecial scaling. Remember, it’sunlikely that you’ll need feature teams atall  
ifyour individual teams candoAgile. But ify o u do,y o u  
won’t need any special infrastructure tohave feature  
teams—you justneed multiple empowered teams who can,  
and will, coordinate among themselves asneeded. Chapter 21. Scaling Agile • 142  
report erratum • discuss]  
  
Sofar,sogood  
Acompany whose work canbedone byasingle team does  
notneed anything special toscale Agile. Acompany with a  
need formore features than asingle team canhandle can  
build feature teams, and they won’t need anything else to  
scale their Agile process. Inmost organizations I’ve seen, themajority ofthework is  
done bysingle teams already. Inafew, Ihave seen aproduct  
that isintegrated enough, and large enough, where feature  
teams might beneeded. What else isthere. report erratum • discussChapter 21. Scaling Agile • 143  
  
[CONTEXT FROM PREVIOUS PAGE: Sofar,sogood  
Acompany whose work canbedone byasingle team does  
notneed anything special toscale Agile. Acompany with a  
need formore features than asingle team canhandle can  
build feature teams, and they won’t need anything else to  
scale their Agile process. Inmost organizations I’ve seen, themajority ofthework is  
done bysingle teams already.

## Translated Text:

تیم‌های Feature می‌توانند همین کار را انجام دهند، با هر تیم که تغییرات مورد نیاز خود را ایجاد می‌کند، چک‌ها را به pool اضافه می‌کند و کد را به صورت مکرر check-in می‌کند. آیا به یک تیم زیرساخت تخصصی نیاز دارید؟ اغلب، اگر در Agile مهارت دارید، نیازی نخواهید داشت. تیم‌های تخصصی اغلب در شرایط Agile منحل می‌شوند. اما اگر تصمیم بگیرید چنین تیمی داشته باشید و آنها Agile باشند، می‌توانند تغییرات زیرساختی را به‌طور روان تولید کنند، که توسط چک‌های خودکار پشتیبانی می‌شود، و از چندین تیم Feature حمایت کنند. من توصیه می‌کنم اجازه دهید تیم‌های Feature شما تغییرات زیرساختی را مدیریت کنند و در صورت نیاز بین خود هماهنگ شوند. اما اگر تصمیم بگیرید یک تیم تخصصی برای زیرساخت داشته باشید، علی‌رغم این توصیه، همچنان نیازی به مقیاس‌بندی خاص نیست. به یاد داشته باشید، احتمالاً نیازی به تیم‌های Feature نخواهید داشت اگر تیم‌های فردی شما بتوانند Agile را انجام دهند. اما اگر نیاز داشته باشید، نیازی به زیرساخت خاصی برای داشتن تیم‌های Feature نخواهید داشت—فقط به چندین تیم توانمند نیاز دارید که بتوانند و خواهند توانست در صورت نیاز بین خود هماهنگ شوند. فصل 21. Scaling Agile • 142   
گزارش خطا • بحث کنید   
  
تا اینجا، خوب پیش رفته‌ایم   
شرکتی که کارش می‌تواند توسط یک تیم انجام شود، نیازی به هیچ چیز خاصی برای مقیاس‌بندی Agile ندارد. شرکتی که به ویژگی‌های بیشتری از آنچه یک تیم می‌تواند مدیریت کند نیاز دارد، می‌تواند تیم‌های Feature بسازد و نیازی به هیچ چیز دیگری برای مقیاس‌بندی فرآیند Agile خود نخواهد داشت. در بیشتر سازمان‌هایی که دیده‌ام، بیشتر کارها از قبل توسط تیم‌های منفرد انجام می‌شود. در چند مورد، محصولی را دیده‌ام که به اندازه کافی یکپارچه و بزرگ است که ممکن است به تیم‌های Feature نیاز داشته باشد. چه چیز دیگری وجود دارد؟ گزارش خطا • بحث کنید فصل 21. Scaling Agile • 143   
  
[متن از صفحه قبلی: تا اینجا، خوب پیش رفته‌ایم   
شرکتی که کارش می‌تواند توسط یک تیم انجام شود، نیازی به هیچ چیز خاصی برای مقیاس‌بندی Agile ندارد. شرکتی که به ویژگی‌های بیشتری از آنچه یک تیم می‌تواند مدیریت کند نیاز دارد، می‌تواند تیم‌های Feature بسازد و نیازی به هیچ چیز دیگری برای مقیاس‌بندی فرآیند Agile خود نخواهد داشت. در بیشتر سازمان‌هایی که دیده‌ام، بیشتر کارها از قبل توسط تیم‌های منفرد انجام می‌شود.]

================================================================================

# Section 139

## Original Text:

Inafew, Ihave seen aproduct  
that isintegrated enough, and large enough, where feature  
teams might beneeded. What else isthere. report erratum • discussChapter 21. Scaling Agile • 143]  
  
Giant efforts  
Some companies undertake truly large efforts, with hundreds  
ofdevelopers, perhaps even thousands, allworking onone  
thing. Ifyou’re notinacompany like that, maybe ally o u  
need istogetyour individual teams able tooperate inan  
Agile fashion. Y o u could stop reading now, orjump tothe  
conclusion. Butyou’re probably wondering what todoabout  
giant efforts. Chapter 21. Scaling Agile • 144  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Giant efforts  
Some companies undertake truly large efforts, with hundreds  
ofdevelopers, perhaps even thousands, allworking onone  
thing. Ifyou’re notinacompany like that, maybe ally o u  
need istogetyour individual teams able tooperate inan  
Agile fashion. Y o u could stop reading now, orjump tothe  
conclusion. Butyou’re probably wondering what todoabout  
giant efforts. Chapter 21. Scaling Agile • 144  
report erratum • discuss]  
  
First, grow thegiant incrementally. Ifyou’re starting agiant effort, even one built onexisting  
architecture, thestandard Agile approach works. Start with  
asingle team. Build itlarger and larger. Build and extend  
infrastructure asy o u go. Add feature teams asy o u need  
them. report erratum • discussChapter 21. Scaling Agile • 145  
  
[CONTEXT FROM PREVIOUS PAGE: First, grow thegiant incrementally. Ifyou’re starting agiant effort, even one built onexisting  
architecture, thestandard Agile approach works. Start with  
asingle team. Build itlarger and larger. Build and extend  
infrastructure asy o u go. Add feature teams asy o u need  
them. report erratum • discussChapter 21. Scaling Agile • 145]  
  
Finally, divide thegiant, mostly along feature teams. Even ingiant efforts, itturns out that almost everything  
being done isbeing done bysingle teams. W ealready know  
how todothose: standard Agile. Just dothat.

## Translated Text:

در چند مورد، محصولی را دیده‌ام که به اندازه کافی یکپارچه و بزرگ بوده است که تیم‌های Feature ممکن است مورد نیاز باشند. چه چیز دیگری وجود دارد؟   
گزارش خطا • بحث   
فصل ۲۱. Scaling Agile • ۱۴۳   
  
تلاش‌های عظیم   
برخی شرکت‌ها تلاش‌های واقعاً بزرگی را انجام می‌دهند، با صدها توسعه‌دهنده، شاید حتی هزاران نفر، که همگی روی یک چیز کار می‌کنند. اگر شما در شرکتی از این نوع نیستید، شاید تنها چیزی که نیاز دارید این است که تیم‌های فردی خود را قادر سازید تا به صورت Agile عمل کنند.   
می‌توانید همین حالا خواندن را متوقف کنید، یا به نتیجه‌گیری بپرید. اما احتمالاً دارید فکر می‌کنید که با تلاش‌های عظیم چه باید کرد.   
فصل ۲۱. Scaling Agile • ۱۴۴   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: تلاش‌های عظیم   
برخی شرکت‌ها تلاش‌های واقعاً بزرگی را انجام می‌دهند، با صدها توسعه‌دهنده، شاید حتی هزاران نفر، که همگی روی یک چیز کار می‌کنند. اگر شما در شرکتی از این نوع نیستید، شاید تنها چیزی که نیاز دارید این است که تیم‌های فردی خود را قادر سازید تا به صورت Agile عمل کنند.   
می‌توانید همین حالا خواندن را متوقف کنید، یا به نتیجه‌گیری بپرید. اما احتمالاً دارید فکر می‌کنید که با تلاش‌های عظیم چه باید کرد.   
فصل ۲۱. Scaling Agile • ۱۴۴   
گزارش خطا • بحث]   
  
ابتدا، تلاش عظیم را به صورت تدریجی رشد دهید. اگر در حال شروع یک تلاش عظیم هستید، حتی اگر بر پایه معماری موجود ساخته شده باشد، رویکرد استاندارد Agile کار می‌کند. با یک تیم شروع کنید. آن را بزرگ‌تر و بزرگ‌تر کنید. زیرساخت را در حین پیشروی بسازید و گسترش دهید. تیم‌های Feature را هر زمان که نیاز داشتید اضافه کنید.   
گزارش خطا • بحث   
فصل ۲۱. Scaling Agile • ۱۴۵   
  
[متن از صفحه قبلی: ابتدا، تلاش عظیم را به صورت تدریجی رشد دهید. اگر در حال شروع یک تلاش عظیم هستید، حتی اگر بر پایه معماری موجود ساخته شده باشد، رویکرد استاندارد Agile کار می‌کند. با یک تیم شروع کنید. آن را بزرگ‌تر و بزرگ‌تر کنید. زیرساخت را در حین پیشروی بسازید و گسترش دهید. تیم‌های Feature را هر زمان که نیاز داشتید اضافه کنید.   
گزارش خطا • بحث   
فصل ۲۱. Scaling Agile • ۱۴۵]   
  
در نهایت، تلاش عظیم را عمدتاً بر اساس تیم‌های Feature تقسیم کنید. حتی در تلاش‌های عظیم، مشخص می‌شود که تقریباً همه کارها توسط تیم‌های منفرد انجام می‌شود. ما قبلاً می‌دانیم که چگونه این کارها را انجام دهیم: Agile استاندارد. فقط همان را انجام دهید.

================================================================================

# Section 140

## Original Text:

Even ingiant efforts, afew efforts can beimproved by  
adding more working teams. Dothat, and operate them like  
feature teams. Standard Agile. Just dothat. What’s left. Isthere really something, somewhere, that needs  
more than one team, and that can’t bedivided upinto  
smaller efforts that canbedone inAgile fashion. Inmost cases, Idoubt it. Idon’t think there aregiant efforts  
that aretruly irreducible. Ifthere are,nooneknows how to  
dothem, Agile ornot. The very essence ofputting lots of  
people onaneffort istodivide upthework. Ifw edonot  
know how todivide upthework, adding people will not  
help. Ifw edoknow how todivide upthework, then, almost  
always, thebulk ofthework can bedone using standard  
Agile. Isthere enough lefttorequire acomplex approach to  
scaling. Perhaps. Wait and see; that’s myadvice. Chapter 21. Scaling Agile • 146  
report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Finally, divide thegiant, mostly along feature teams. Even ingiant efforts, itturns out that almost everything  
being done isbeing done bysingle teams. W ealready know  
how todothose: standard Agile. Just dothat. Even ingiant efforts, afew efforts can beimproved by  
adding more working teams. Dothat, and operate them like  
feature teams. Standard Agile. Just dothat. What’s left. Isthere really something, somewhere, that needs  
more than one team, and that can’t bedivided upinto  
smaller efforts that canbedone inAgile fashion. Inmost cases, Idoubt it. Idon’t think there aregiant efforts  
that aretruly irreducible. Ifthere are,nooneknows how to  
dothem, Agile ornot. The very essence ofputting lots of  
people onaneffort istodivide upthework. Ifw edonot  
know how todivide upthework, adding people will not  
help. Ifw edoknow how todivide upthework, then, almost  
always, thebulk ofthework can bedone using standard  
Agile. Isthere enough lefttorequire acomplex approach to  
scaling. Perhaps. Wait and see; that’s myadvice. Chapter 21.

## Translated Text:

حتی در تلاش‌های بزرگ، چند تلاش را می‌توان با افزودن تیم‌های کاری بیشتر بهبود بخشید. این کار را انجام دهید و آن‌ها را مانند feature teams مدیریت کنید. Standard Agile. فقط همین کار را انجام دهید. چه چیزی باقی می‌ماند؟ آیا واقعاً چیزی وجود دارد که نیاز به بیش از یک تیم داشته باشد و نتوان آن را به تلاش‌های کوچکتری تقسیم کرد که بتوان آن‌ها را به روش Agile انجام داد؟ در بیشتر موارد، من شک دارم. فکر نمی‌کنم تلاش‌های بزرگی وجود داشته باشد که واقعاً غیرقابل تقسیم باشند. اگر وجود داشته باشند، هیچ‌کس نمی‌داند چگونه آن‌ها را انجام دهد، چه Agile باشد چه نه. ذات اصلی تخصیص افراد زیاد به یک تلاش، تقسیم کار است. اگر ندانیم چگونه کار را تقسیم کنیم، افزودن افراد کمکی نخواهد کرد. اگر بدانیم چگونه کار را تقسیم کنیم، تقریباً همیشه بخش عمده کار را می‌توان با استفاده از Standard Agile انجام داد. آیا چیزی باقی می‌ماند که نیاز به یک رویکرد پیچیده برای scaling داشته باشد؟ شاید. صبر کنید و ببینید؛ این توصیه من است. فصل ۲۱. Scaling Agile • 146   
گزارش خطا • بحث

================================================================================

# Section 141

## Original Text:

Scaling Agile • 146  
report erratum • discuss]  
  
Bottom line  
Ifyour individual teams cannot work inanAgile fashion,  
then clearly you’re notready to“transition” your company  
orto“scale” Agile. Y o u don’t want totransition tosomething  
y o u can’t do,and y o u don’t want toscale something that  
doesn’t work. First, start creating teams that arevery capable ofdoing  
Agile. Then, give them themost important, most valuable work to  
dothat your organization can come upwith. And stand  
back. Keep creating Agile teams, organized byfeatures where  
possible. Y o u may find that y o u have little need toscale  
Agile. More likely, y o u justhave todoit. report erratum • discussChapter 21. Scaling Agile • 147  
  
[CONTEXT FROM PREVIOUS PAGE: Bottom line  
Ifyour individual teams cannot work inanAgile fashion,  
then clearly you’re notready to“transition” your company  
orto“scale” Agile. Y o u don’t want totransition tosomething  
y o u can’t do,and y o u don’t want toscale something that  
doesn’t work. First, start creating teams that arevery capable ofdoing  
Agile. Then, give them themost important, most valuable work to  
dothat your organization can come upwith. And stand  
back. Keep creating Agile teams, organized byfeatures where  
possible. Y o u may find that y o u have little need toscale  
Agile. More likely, y o u justhave todoit. report erratum • discussChapter 21. Scaling Agile • 147]  
  
CHAPTER 22  
Conclusion  
Ifyou’ve made itallthew a y here, congratulations. Letme  
sum upwhat Ibelieve we’ve justbeen through, tohelp y o u  
settle inyour mind justwhat hashappened here. Imagine that you’re climbing amountain called Software  
Development. Maybe you’re abeginner, down near the  
bottom, walking upsteep trails, once inawhile clambering  
over rocks. Maybe you’re pretty advanced, with awhole kit  
ofclimbing equipment and knowledge. Maybe y o u even  
help others with their own climbing journey.

## Translated Text:

مقیاس‌بندی Agile • 146   
گزارش خطا • بحث کنید   
  
نتیجه‌گیری   
اگر تیم‌های فردی شما نمی‌توانند به شیوه‌ای Agile کار کنند،   
پس به وضوح آماده نیستید که شرکت خود را "انتقال" دهید   
یا Agile را "مقیاس‌بندی" کنید. شما نمی‌خواهید به چیزی انتقال پیدا کنید   
که نمی‌توانید انجام دهید، و نمی‌خواهید چیزی را مقیاس‌بندی کنید   
که کار نمی‌کند. ابتدا، شروع به ایجاد تیم‌هایی کنید که بسیار توانمند در انجام   
Agile باشند. سپس، مهم‌ترین و باارزش‌ترین کارهایی را که   
سازمان شما می‌تواند ارائه دهد به آن‌ها بدهید. و کنار بایستید.   
به ایجاد تیم‌های Agile ادامه دهید، که در صورت امکان بر اساس ویژگی‌ها سازمان‌دهی شده‌اند.   
ممکن است متوجه شوید که نیاز کمی به مقیاس‌بندی Agile دارید.   
احتمالاً بیشتر، فقط باید این کار را انجام دهید. گزارش خطا • بحث کنید   
فصل 21. مقیاس‌بندی Agile • 147   
  
[زمینه از صفحه قبلی: نتیجه‌گیری   
اگر تیم‌های فردی شما نمی‌توانند به شیوه‌ای Agile کار کنند،   
پس به وضوح آماده نیستید که شرکت خود را "انتقال" دهید   
یا Agile را "مقیاس‌بندی" کنید. شما نمی‌خواهید به چیزی انتقال پیدا کنید   
که نمی‌توانید انجام دهید، و نمی‌خواهید چیزی را مقیاس‌بندی کنید   
که کار نمی‌کند. ابتدا، شروع به ایجاد تیم‌هایی کنید که بسیار توانمند در انجام   
Agile باشند. سپس، مهم‌ترین و باارزش‌ترین کارهایی را که   
سازمان شما می‌تواند ارائه دهد به آن‌ها بدهید. و کنار بایستید.   
به ایجاد تیم‌های Agile ادامه دهید، که در صورت امکان بر اساس ویژگی‌ها سازمان‌دهی شده‌اند.   
ممکن است متوجه شوید که نیاز کمی به مقیاس‌بندی Agile دارید.   
احتمالاً بیشتر، فقط باید این کار را انجام دهید. گزارش خطا • بحث کنید   
فصل 21. مقیاس‌بندی Agile • 147]   
  
فصل 22   
نتیجه‌گیری   
اگر تا اینجا پیش آمده‌اید، تبریک می‌گویم. اجازه دهید   
خلاصه کنم که به نظر من چه چیزی را پشت سر گذاشته‌ایم، تا به شما کمک کنم   
در ذهن خود تثبیت کنید که اینجا چه اتفاقی افتاده است. تصور کنید که در حال صعود به کوهی به نام   
توسعه نرم‌افزار هستید. شاید شما یک مبتدی هستید، پایین نزدیک   
دامنه، در حال قدم زدن در مسیرهای شیب‌دار، و گاهی اوقات از روی صخره‌ها بالا می‌روید.   
شاید شما کاملاً پیشرفته هستید، با یک مجموعه کامل از تجهیزات   
و دانش صعود. شاید حتی به دیگران در سفر صعود خودشان کمک می‌کنید.

================================================================================

# Section 142

## Original Text:

Maybe you’re  
one ofthose amazing free climbers who can liftyourself  
from asingle hand jam and kick your foot over your head  
tofind thenext tiny crevice inwhat looks likeawall ofglass. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: CHAPTER 22  
Conclusion  
Ifyou’ve made itallthew a y here, congratulations. Letme  
sum upwhat Ibelieve we’ve justbeen through, tohelp y o u  
settle inyour mind justwhat hashappened here. Imagine that you’re climbing amountain called Software  
Development. Maybe you’re abeginner, down near the  
bottom, walking upsteep trails, once inawhile clambering  
over rocks. Maybe you’re pretty advanced, with awhole kit  
ofclimbing equipment and knowledge. Maybe y o u even  
help others with their own climbing journey. Maybe you’re  
one ofthose amazing free climbers who can liftyourself  
from asingle hand jam and kick your foot over your head  
tofind thenext tiny crevice inwhat looks likeawall ofglass. report erratum • discuss]  
  
Whatever your level, ifyou’re like me, y o u spend alotof  
time looking atthemountain face right infront ofyour nose. Y o u spend alotoftime using your mind tofigure outyour  
next move upward and which muscles you’ll usetomake  
themove. This book issome words and pictures from someone onthe  
same mountain, who has found areasonably flat and  
pleasant place tositand look around forawhile. Helooks  
outward and sees theamazing vista that’s almost always  
outofsight behind ourback asw eclimb. Helooks down  
themountain and sees itsshape and sees theclimbers mov -  
ingupward, some notaswell ashedid, some much better. Hesees thepaths onthemountain and cantellwhich ones  
areeasy, which ones arehard, which ones aresafe, and  
which ones aredangerous. Then helooks up,and through theclouds and haze, he  
realizes that there’s more climbing todo,lots more, asw e  
work ourw a y upward. Hesees what look like likely paths  
and other interesting spots tostop and look around.

## Translated Text:

شاید شما یکی از آن صخره‌نوردان شگفت‌انگیز باشید که می‌توانید خودتان را با یک دست در شکاف گیر کرده بالا بکشید و پای خود را از روی سرتان عبور دهید تا شکاف کوچک بعدی را پیدا کنید، در چیزی که شبیه به یک دیوار شیشه‌ای به نظر می‌رسد. گزارش خطا • بحث کنید  
  
[متن از صفحه قبلی: فصل ۲۲   
نتیجه‌گیری   
اگر تا اینجا پیش آمده‌اید، تبریک می‌گویم. اجازه دهید خلاصه کنم که فکر می‌کنم ما از چه چیزی عبور کرده‌ایم، تا به شما کمک کنم در ذهن خودتان آنچه اتفاق افتاده را تثبیت کنید. تصور کنید که در حال صعود از کوهی به نام توسعه نرم‌افزار هستید. شاید شما یک مبتدی باشید، پایین نزدیک به دامنه، در حال قدم زدن در مسیرهای شیب‌دار، و گاهی اوقات از روی صخره‌ها بالا می‌روید. شاید شما نسبتاً پیشرفته باشید، با یک مجموعه کامل از تجهیزات و دانش صخره‌نوردی. شاید حتی به دیگران در مسیر صعودشان کمک کنید. شاید شما یکی از آن صخره‌نوردان شگفت‌انگیز باشید که می‌توانید خودتان را با یک دست در شکاف گیر کرده بالا بکشید و پای خود را از روی سرتان عبور دهید تا شکاف کوچک بعدی را پیدا کنید، در چیزی که شبیه به یک دیوار شیشه‌ای به نظر می‌رسد. گزارش خطا • بحث کنید]  
  
هر سطحی که دارید، اگر مثل من باشید، زمان زیادی را صرف نگاه کردن به سطح کوه درست جلوی بینی‌تان می‌کنید. زمان زیادی را صرف استفاده از ذهن خود برای پیدا کردن حرکت بعدی به سمت بالا و اینکه کدام عضلات را برای انجام آن حرکت استفاده کنید، می‌کنید. این کتاب شامل کلماتی و تصاویری از کسی است که روی همان کوه بوده، کسی که یک جای نسبتاً صاف و دلپذیر پیدا کرده تا مدتی بنشیند و اطراف را نگاه کند. او به بیرون نگاه می‌کند و منظره شگفت‌انگیزی را می‌بیند که تقریباً همیشه پشت سر ما هنگام صعود پنهان است. او به پایین کوه نگاه می‌کند و شکل آن و صعودکنندگان را می‌بیند که بالا می‌روند، برخی نه به خوبی او، برخی بسیار بهتر. او مسیرهای روی کوه را می‌بیند و می‌تواند بگوید کدام‌ها آسان هستند، کدام‌ها سخت، کدام‌ها امن و کدام‌ها خطرناک. سپس او به بالا نگاه می‌کند و از میان ابرها و مه، متوجه می‌شود که هنوز صعود بیشتری باقی مانده است، خیلی بیشتر، همان‌طور که ما به سمت بالا حرکت می‌کنیم. او مسیرهایی که به نظر می‌رسند مناسب هستند و نقاط جالب دیگری برای توقف و نگاه کردن را می‌بیند.

================================================================================

# Section 143

## Original Text:

Hetakes some pictures ofwhat hesees and writes afew  
words about histhoughts and discoveries. Hesketches a  
fewpaths andtells y o uabitabout how herecognizes agood  
path, and what hedoes when apath turns outnotsogood. Heoffers allthistoyou, toenrich your own journey upthe  
mountain, toremind y o u ofthemany beautiful views that  
make thehard climb worthwhile. Hereminds y o u that  
sometimes thereward isjustintheclimbing itself, doing it  
more and more well. Hesays, “Here’s what italllooks like tome. What does it  
look liketoyou. ”  
Thanks forreading. report erratum • discussChapter 22. Conclusion • 149  
  
[CONTEXT FROM PREVIOUS PAGE: Whatever your level, ifyou’re like me, y o u spend alotof  
time looking atthemountain face right infront ofyour nose. Y o u spend alotoftime using your mind tofigure outyour  
next move upward and which muscles you’ll usetomake  
themove. This book issome words and pictures from someone onthe  
same mountain, who has found areasonably flat and  
pleasant place tositand look around forawhile. Helooks  
outward and sees theamazing vista that’s almost always  
outofsight behind ourback asw eclimb. Helooks down  
themountain and sees itsshape and sees theclimbers mov -  
ingupward, some notaswell ashedid, some much better. Hesees thepaths onthemountain and cantellwhich ones  
areeasy, which ones arehard, which ones aresafe, and  
which ones aredangerous. Then helooks up,and through theclouds and haze, he  
realizes that there’s more climbing todo,lots more, asw e  
work ourw a y upward. Hesees what look like likely paths  
and other interesting spots tostop and look around. Hetakes some pictures ofwhat hesees and writes afew  
words about histhoughts and discoveries. Hesketches a  
fewpaths andtells y o uabitabout how herecognizes agood  
path, and what hedoes when apath turns outnotsogood. Heoffers allthistoyou, toenrich your own journey upthe  
mountain, toremind y o u ofthemany beautiful views that  
make thehard climb worthwhile.

## Translated Text:

او از چیزهایی که می‌بیند عکس می‌گیرد و چند کلمه درباره افکار و کشفیاتش می‌نویسد. او چند مسیر را ترسیم می‌کند و کمی درباره اینکه چگونه یک مسیر خوب را تشخیص می‌دهد و چه کاری انجام می‌دهد وقتی یک مسیر چندان خوب از آب درنمی‌آید، به شما می‌گوید. او همه این‌ها را به شما ارائه می‌دهد تا سفر خودتان به قله کوه را غنی‌تر کنید، تا شما را به یاد بسیاری از مناظر زیبایی بیندازد که صعود سخت را ارزشمند می‌کنند. او به شما یادآوری می‌کند که گاهی پاداش فقط در خود صعود است، اینکه آن را بهتر و بهتر انجام دهید. او می‌گوید: «این چیزی است که همه این‌ها برای من به نظر می‌رسد. برای شما چگونه به نظر می‌رسد؟»  
  
متشکرم که خواندید.   
گزارش خطا • بحث   
فصل ۲۲. نتیجه‌گیری • ۱۴۹

================================================================================

# Section 144

## Original Text:

Hereminds y o u that  
sometimes thereward isjustintheclimbing itself, doing it  
more and more well. Hesays, “Here’s what italllooks like tome. What does it  
look liketoyou. ”  
Thanks forreading. report erratum • discussChapter 22. Conclusion • 149]  
  
Bibliography  
[Pin09] Daniel H. Pink. Drive: TheSurprising Truth About What  
Motivates Us. Riverhead Books, New York, NY, USA,  
2009. [Pir00] Robert M. Pirsig. ZenandtheArtofMotorcycle Mainte -  
nance: AnInquiry into Values. Perennial Classics, New  
York, NY, USA, Reprint Edition, 2000. report erratum • discuss  
  
[CONTEXT FROM PREVIOUS PAGE: Bibliography  
[Pin09] Daniel H. Pink. Drive: TheSurprising Truth About What  
Motivates Us. Riverhead Books, New York, NY, USA,  
2009. [Pir00] Robert M. Pirsig. ZenandtheArtofMotorcycle Mainte -  
nance: AnInquiry into Values. Perennial Classics, New  
York, NY, USA, Reprint Edition, 2000.

## Translated Text:

او به شما یادآوری می‌کند که گاهی پاداش فقط در خودِ بالا رفتن است، اینکه آن را بارها و بارها بهتر انجام دهید. او می‌گوید: «این چیزی است که به نظر من این‌طور می‌آید. به نظر شما چگونه است؟»   
ممنون از خواندن. گزارش خطا • بحث   
فصل ۲۲. نتیجه‌گیری • ۱۴۹   
  
کتاب‌شناسی   
[Pin09] دنیل اچ. پینک. Drive: The Surprising Truth About What Motivates Us. انتشارات Riverhead Books، نیویورک، ایالات متحده آمریکا، ۲۰۰۹.   
[Pir00] رابرت ام. پیرسیگ. Zen and the Art of Motorcycle Maintenance: An Inquiry into Values. انتشارات Perennial Classics، نیویورک، ایالات متحده آمریکا، چاپ مجدد، ۲۰۰۰.   
گزارش خطا • بحث   
  
[متن از صفحه قبلی: کتاب‌شناسی   
[Pin09] دنیل اچ. پینک. Drive: The Surprising Truth About What Motivates Us. انتشارات Riverhead Books، نیویورک، ایالات متحده آمریکا، ۲۰۰۹.   
[Pir00] رابرت ام. پیرسیگ. Zen and the Art of Motorcycle Maintenance: An Inquiry into Values. انتشارات Perennial Classics، نیویورک، ایالات متحده آمریکا، چاپ مجدد، ۲۰۰۰.]

================================================================================

# Section 145

## Original Text:

report erratum • discuss]  
  
report erratum • discussBibliography • 151  
  
[CONTEXT FROM PREVIOUS PAGE: report erratum • discussBibliography • 151]  
  
Index  
A  
acceptance test-driven de-  
velopment, 66, 73, 117,   
126, 140  
activity-based planning,  
20–22  
adjusting direction, xv,   
11, 45, 108  
Agile, seealsoscaling;  
Scrum  
Agile Manifesto, 126–  
127  
defining value, 80–81  
methods, viii, 126–128  
resources, 126  
Ambler, Scott, 126  
architecture, seefounda -  
tions; infrastructure  
automated tests, 66–67 ,   
73, 140  
autonomy, 94, 96  
B  
Beck, Kent, 37, 126  
Belshee, Arlo, 134  
blame, 117–118  
budget  
management deci-  
sions, 101, 106  
planning feature by  
feature, 34  
bugs, seedefects  
building  
eliminating test-and -  
fixinterval, 48  
byfeatures, 5, 42–49 ,   
77features and founda -  
tions inparallel,  
51–59  
identifying progress,  
47  
refining design while,  
49  
refining features, 57  
small cycles, 44  
value pyramid, 5, 77  
business-level tests, 65–  
66, 68  
business-side personnel,  
seealsoProduct Champi -  
on  
building features and  
foundations inpar-  
allel, 59  
Natural W a y benefits,  
xv  
planning with stories,  
36  
purpose, 95  
refining product vi-  
sion, 45  
C  
Campground Rule, 124  
coaches, 126  
Cockburn, Alistair, 126  
code, refactoring, 122–124  
Communities ofPractice,  
29–30  
conferences, 115  
consensus, 87  
continuous planning, 36,   
41  
continuous testing, 60–68 ,   
73, 77controlling  
frameworks, 127  
management compo -  
nent, 101, 109  
coordinating, multiple  
teams, 139–147  
costs  
design deterioration,  
73  
monolithic projects,  
22  
planning feature by  
feature, 34  
value byfeature, 13  
Crystal Clear, 126  
cycles  
building insmall, 44  
improvement and  
difficulty, 91  
perception ofslow -  
ness, 116–119  
D  
DAD (Disciplined Agile  
Development), 126  
deadlines  
planning by,19  
stopping before, 34,   
105  
defects  
building byfeatures,  
48  
planning and, 21  
pressure, 39, 112–115 ,   
123  
quality and value  
pyramid, 5, 77  
speed, 39, 63–64 , 68  
stretch goals, 39  
  
[CONTEXT FROM PREVIOUS PAGE: Index  
A  
acceptance test-driven de-  
velopment, 66, 73, 117,   
126, 140  
activity-based planning,  
20–22  
adjusting direction, xv,   
11, 45, 108  
Agile, seealsoscaling;  
Scrum  
Agile Manifesto, 126–  
127  
defining value, 80–81  
methods, viii, 126–128  
resources, 126  
Ambler, Scott, 126  
architecture, seefounda -  
tions; infrastructure  
automated tests, 66–67 ,   
73, 140  
autonomy, 94, 96  
B  
Beck, Kent, 37, 126  
Belshee, Arlo, 134  
blame, 117–118  
budget  
management deci-  
sions, 101, 106  
planning feature by  
feature, 34  
bugs, seedefects  
building  
eliminating test-and -  
fixinterval, 48  
byfeatures, 5, 42–49 ,   
77features and founda -  
tions inparallel,  
51–59  
identifying progress,  
47  
refining design while,  
49  
refining features, 57  
small cycles, 44  
value pyramid, 5, 77  
business-level tests, 65–  
66, 68  
business-side personnel,  
seealsoProduct Champi -  
on  
building features and  
foundations inpar-  
allel, 59  
Natural W a y benefits,  
xv  
planning with stories,  
36  
purpose, 95  
refining product vi-  
sion, 45  
C  
Campground Rule, 124  
coaches, 126  
Cockburn, Alistair, 126  
code, refactoring, 122–124  
Communities ofPractice,  
29–30  
conferences, 115  
consensus, 87  
continuous planning, 36,   
41  
continuous testing, 60–68 ,   
73, 77controlling  
frameworks, 127  
management compo -  
nent, 101, 109  
coordinating, multiple  
teams, 139–147  
costs  
design deterioration,  
73  
monolithic projects,  
22  
planning feature by  
feature, 34  
value byfeature, 13  
Crystal Clear, 126  
cycles  
building insmall, 44  
improvement and  
difficulty, 91  
perception ofslow -  
ness, 116–119  
D  
DAD (Disciplined Agile  
Development), 126  
deadlines  
planning by,19  
stopping before, 34,   
105  
defects  
building byfeatures,  
48  
planning and, 21  
pressure, 39, 112–115 ,   
123  
quality and value  
pyramid, 5, 77  
speed, 39, 63–64 , 68  
stretch goals, 39]  
  
delays  
defects, 63–64 , 112  
evaluating, 113–114  
delegating, 101, 107, 110  
delivery  
building byfeatures,  
5  
building features and  
foundations inpar-  
allel, 58  
value ofearly deliv -  
ery,xiii, 8–11 , 15,   
77  
DeMarco, Tom, 63  
design, seealsofounda -  
tions; infrastructure  
deterioration, 70, 73  
quality, 5, 77  
refactoring, 72–73 ,   
77, 120–124  
refining while build -  
ingbyfeatures, 49  
simplicity, 69  
testing, 60–68 , 73, 77  
deterioration, design, 70,   
73  
developers, seeteams  
directing, management  
component, 101, 108  
direction, changing, xv,   
11, 45, 108  
Disciplined Agile Develop -  
ment (DAD), 126  
done, definition of  
building feature by  
feature, 46–47  
eliminating test-and -  
fixinterval, 48, 119  
identifying progress,  
47  
mastery and, 97  
pressure forspeed,  
114, 118  
Drive ,94  
Drucker, Peter, 101  
Dynamic Systems Develop -  
ment Method (DSDM),  
126  
E  
ease vs.

## Translated Text:

گزارش خطا • بحث کنید  
  
گزارش خطا • بحث کنید  
کتاب‌شناسی • ۱۵۱  
  
[متن از صفحه قبلی: گزارش خطا • بحث کنید  
کتاب‌شناسی • ۱۵۱]  
  
نمایه   
A   
acceptance test-driven development، ۶۶، ۷۳، ۱۱۷، ۱۲۶، ۱۴۰   
activity-based planning، ۲۰–۲۲   
adjusting direction، xv، ۱۱، ۴۵، ۱۰۸   
Agile، همچنین ببینید scaling؛ Scrum   
Agile Manifesto، ۱۲۶–۱۲۷   
تعریف ارزش، ۸۰–۸۱   
روش‌ها، viii، ۱۲۶–۱۲۸   
منابع، ۱۲۶   
Ambler, Scott، ۱۲۶   
architecture، همچنین ببینید foundations؛ infrastructure   
automated tests، ۶۶–۶۷، ۷۳، ۱۴۰   
autonomy، ۹۴، ۹۶   
  
B   
Beck, Kent، ۳۷، ۱۲۶   
Belshee, Arlo، ۱۳۴   
blame، ۱۱۷–۱۱۸   
بودجه   
تصمیمات مدیریتی، ۱۰۱، ۱۰۶   
برنامه‌ریزی feature به feature، ۳۴   
bugs، همچنین ببینید defects   
ساختن   
حذف test-and-fix interval، ۴۸   
by features، ۵، ۴۲–۴۹، ۷۷   
features و foundations به‌صورت موازی، ۵۱–۵۹   
شناسایی پیشرفت، ۴۷   
بهبود طراحی در حین ساخت، ۴۹   
بهبود features، ۵۷   
چرخه‌های کوچک، ۴۴   
هرم ارزش، ۵، ۷۷   
business-level tests، ۶۵–۶۶، ۶۸   
business-side personnel، همچنین ببینید Product Champion   
ساختن features و foundations به‌صورت موازی، ۵۹   
مزایای Natural Way، xv   
برنامه‌ریزی با stories، ۳۶   
هدف، ۹۵   
بهبود vision محصول، ۴۵   
  
C   
Campground Rule، ۱۲۴   
coaches، ۱۲۶   
Cockburn, Alistair، ۱۲۶   
code، refactoring، ۱۲۲–۱۲۴   
Communities of Practice، ۲۹–۳۰   
کنفرانس‌ها، ۱۱۵   
consensus، ۸۷   
continuous planning، ۳۶، ۴۱   
continuous testing، ۶۰–۶۸، ۷۳، ۷۷   
کنترل frameworks، ۱۲۷   
مولفه مدیریتی، ۱۰۱، ۱۰۹   
هماهنگی تیم‌های متعدد، ۱۳۹–۱۴۷   
هزینه‌ها   
تخریب طراحی، ۷۳   
پروژه‌های monolithic، ۲۲   
برنامه‌ریزی feature به feature، ۳۴   
ارزش feature به feature، ۱۳   
Crystal Clear، ۱۲۶   
چرخه‌ها   
ساختن در چرخه‌های کوچک، ۴۴   
بهبود و دشواری، ۹۱   
درک کندی، ۱۱۶–۱۱۹   
  
D   
DAD (Disciplined Agile Development)، ۱۲۶   
deadlines   
برنامه‌ریزی بر اساس، ۱۹   
توقف قبل از، ۳۴، ۱۰۵   
defects   
ساختن by features، ۴۸   
برنامه‌ریزی و، ۲۱   
فشار، ۳۹، ۱۱۲–۱۱۵، ۱۲۳   
کیفیت و هرم ارزش، ۵، ۷۷   
سرعت، ۳۹، ۶۳–۶۴، ۶۸   
stretch goals، ۳۹   
  
تاخیرها   
defects، ۶۳–۶۴، ۱۱۲   
ارزیابی، ۱۱۳–۱۱۴   
واگذاری، ۱۰۱، ۱۰۷، ۱۱۰   
تحویل   
ساختن by features، ۵   
ساختن features و foundations به‌صورت موازی، ۵۸   
ارزش تحویل زودهنگام، xiii، ۸–۱۱، ۱۵، ۷۷   
DeMarco, Tom، ۶۳   
طراحی، همچنین ببینید foundations؛ infrastructure   
تخریب، ۷۰، ۷۳   
کیفیت، ۵، ۷۷   
refactoring، ۷۲–۷۳، ۷۷، ۱۲۰–۱۲۴   
بهبود در حین ساختن by features، ۴۹   
سادگی، ۶۹   
تست، ۶۰–۶۸، ۷۳، ۷۷   
تخریب طراحی، ۷۰، ۷۳   
توسعه‌دهندگان، همچنین ببینید teams   
هدایت، مولفه مدیریتی، ۱۰۱، ۱۰۸   
تغییر جهت، xv، ۱۱، ۴۵، ۱۰۸   
Disciplined Agile Development (DAD)، ۱۲۶   
تعریف done   
ساختن feature به feature، ۴۶–۴۷   
حذف test-and-fix interval، ۴۸، ۱۱۹   
شناسایی پیشرفت، ۴۷   
تسلط و، ۹۷   
فشار برای سرعت، ۱۱۴، ۱۱۸   
Drive، ۹۴   
Drucker, Peter، ۱۰۱   
Dynamic Systems Development Method (DSDM)، ۱۲۶   
  
E   
ease vs.

================================================================================

# Section 146

## Original Text:

simplicity, xvi, 88–  
91, 93, 133–137  
end users, xvestimating, seealsoforecast -  
ing  
disadvantages, 34–  
35, 38  
pressure forspeed,  
113–114  
work pieces, 37  
working without, 40  
experts and specialists  
feature teams, 29–30  
infrastructure teams  
forlarge projects,  
142  
team productivity,  
113  
external dependencies, 117  
Extreme Programming,  
126  
F  
feature teams, defined,  
139,seealsoteams  
features, seealsotesting  
asking for,117  
building by,5, 42–49 ,   
77  
building foundations  
first, 54  
building inparallel  
with foundations,  
51–59  
forecasting with  
“Five-Card  
Method”, 99  
guiding by,19–24  
minimal marketable  
features (MMFs),  
12  
organizing by,5, 26–  
31, 77  
planning by,5, 19–  
24, 32–41 , 77  
prioritizing, 24, 46,   
57, 99, 103–104  
refining multiple iter-  
ations, 57  
separating delivery,  
9–10  
slicing, 5, 40, 99  
small cycles, 44  
splitting, 36  
value and, 7, 12–16 ,   
77  
“Five-Card Method”, 99  
fluency and scaling Agile,  
134–137  
forecasting, 99,seealsoesti-  
matingfoundations, seealsode-  
sign; infrastructure  
building first, 54  
building inparallel  
with features, 51–  
59  
importance of,52  
Fowler, Martin, 37  
frameworks, 126–128  
G  
game, lava, xii–xiv  
giant projects, 144–147  
goals, stretch, 39  
guiding  
feature byfeature,  
19–24  
value pyramid, 5, 77  
H  
Hendrickson, Chet, 134  
hiring policy, 107  
I  
information  
changing direction, 11  
Natural W a y benefits,  
xv  
planning feature by  
feature, 24  
stopping delivery, 11  
infrastructure, seealsode-  
sign; foundations  
building inparallel  
with features, 51–  
59  
multiple teams for  
large projects, 142  
“Inspect and Adapt”  
mantra, 97  
iterations  
building insmall cy-  
cles, 44  
continuous testing,  
66–68  
mastery, 97  
perception ofslow -  
ness, 116–119  
planning workload,  
37  
prioritizing features,  
46  
refining features, 57Index • 154  
  
[CONTEXT FROM PREVIOUS PAGE: delays  
defects, 63–64 , 112  
evaluating, 113–114  
delegating, 101, 107, 110  
delivery  
building byfeatures,  
5  
building features and  
foundations inpar-  
allel, 58  
value ofearly deliv -  
ery,xiii, 8–11 , 15,   
77  
DeMarco, Tom, 63  
design, seealsofounda -  
tions; infrastructure  
deterioration, 70, 73  
quality, 5, 77  
refactoring, 72–73 ,   
77, 120–124  
refining while build -  
ingbyfeatures, 49  
simplicity, 69  
testing, 60–68 , 73, 77  
deterioration, design, 70,   
73  
developers, seeteams  
directing, management  
component, 101, 108  
direction, changing, xv,   
11, 45, 108  
Disciplined Agile Develop -  
ment (DAD), 126  
done, definition of  
building feature by  
feature, 46–47  
eliminating test-and -  
fixinterval, 48, 119  
identifying progress,  
47  
mastery and, 97  
pressure forspeed,  
114, 118  
Drive ,94  
Drucker, Peter, 101  
Dynamic Systems Develop -  
ment Method (DSDM),  
126  
E  
ease vs.

## Translated Text:

سادگی، xvi، 88–91، 93، 133–137   
کاربران نهایی، xv   
برآورد، همچنین ببینید forecasting   
معایب، 34–35، 38   
فشار برای سرعت، 113–114   
قطعات کاری، 37   
کار کردن بدون، 40   
متخصصان و کارشناسان   
feature teams، 29–30   
infrastructure teams برای پروژه‌های بزرگ، 142   
بهره‌وری تیم، 113   
external dependencies، 117   
Extreme Programming، 126   
  
F   
feature teams، تعریف شده، 139، همچنین ببینید teams   
features، همچنین ببینید testing   
درخواست برای، 117   
ساخت توسط، 5، 42–49، 77   
ساخت foundations ابتدا، 54   
ساخت به صورت موازی با foundations، 51–59   
forecasting با “Five-Card Method”، 99   
هدایت توسط، 19–24   
minimal marketable features (MMFs)، 12   
سازماندهی توسط، 5، 26–31، 77   
برنامه‌ریزی توسط، 5، 19–24، 32–41، 77   
اولویت‌بندی، 24، 46، 57، 99، 103–104   
تصفیه چندین iteration، 57   
جدا کردن delivery، 9–10   
slicing، 5، 40، 99   
small cycles، 44   
splitting، 36   
ارزش و، 7، 12–16، 77   
“Five-Card Method”، 99   
fluency و scaling Agile، 134–137   
forecasting، 99، همچنین ببینید estimating   
foundations، همچنین ببینید design؛ infrastructure   
ساخت ابتدا، 54   
ساخت به صورت موازی با features، 51–59   
اهمیت، 52   
Fowler، Martin، 37   
frameworks، 126–128   
  
G   
بازی، lava، xii–xiv   
پروژه‌های عظیم، 144–147   
اهداف، stretch، 39   
هدایت   
feature به feature، 19–24   
value pyramid، 5، 77   
  
H   
Hendrickson، Chet، 134   
سیاست استخدام، 107   
  
I   
اطلاعات   
تغییر جهت، 11   
مزایای Natural Way، xv   
برنامه‌ریزی feature به feature، 24   
توقف delivery، 11   
infrastructure، همچنین ببینید design؛ foundations   
ساخت به صورت موازی با features، 51–59   
تیم‌های متعدد برای پروژه‌های بزرگ، 142   
“Inspect and Adapt” mantra، 97   
  
iterations   
ساخت در small cycles، 44   
continuous testing، 66–68   
mastery، 97   
درک کندی، 116–119   
برنامه‌ریزی workload، 37   
اولویت‌بندی features، 46   
تصفیه features، 57   
  
Index • 154   
  
[CONTEXT FROM PREVIOUS PAGE:   
تاخیرها   
defects، 63–64، 112   
ارزیابی، 113–114   
تفویض، 101، 107، 110   
delivery   
ساخت توسط features، 5   
ساخت features و foundations به صورت موازی، 58   
ارزش early delivery، xiii، 8–11، 15، 77   
DeMarco، Tom، 63   
design، همچنین ببینید foundations؛ infrastructure   
تخریب، 70، 73   
کیفیت، 5، 77   
refactoring، 72–73، 77، 120–124   
تصفیه هنگام ساخت توسط features، 49   
سادگی، 69   
testing، 60–68، 73، 77   
تخریب، design، 70، 73   
developers، همچنین ببینید teams   
هدایت، مؤلفه مدیریت، 101، 108   
جهت، تغییر، xv، 11، 45، 108   
Disciplined Agile Development (DAD)، 126   
done، تعریف   
ساخت feature به feature، 46–47   
حذف test-and-fix interval، 48، 119   
شناسایی پیشرفت، 47   
mastery و، 97   
فشار برای سرعت، 114، 118   
Drive، 94   
Drucker، Peter، 101   
Dynamic Systems Development Method (DSDM)، 126   
  
E   
ease vs.

================================================================================

# Section 147

## Original Text:

simplicity, xvi, 88–  
91, 93, 133–137  
end users, xvestimating, seealsoforecast -  
ing  
disadvantages, 34–  
35, 38  
pressure forspeed,  
113–114  
work pieces, 37  
working without, 40  
experts and specialists  
feature teams, 29–30  
infrastructure teams  
forlarge projects,  
142  
team productivity,  
113  
external dependencies, 117  
Extreme Programming,  
126  
F  
feature teams, defined,  
139,seealsoteams  
features, seealsotesting  
asking for,117  
building by,5, 42–49 ,   
77  
building foundations  
first, 54  
building inparallel  
with foundations,  
51–59  
forecasting with  
“Five-Card  
Method”, 99  
guiding by,19–24  
minimal marketable  
features (MMFs),  
12  
organizing by,5, 26–  
31, 77  
planning by,5, 19–  
24, 32–41 , 77  
prioritizing, 24, 46,   
57, 99, 103–104  
refining multiple iter-  
ations, 57  
separating delivery,  
9–10  
slicing, 5, 40, 99  
small cycles, 44  
splitting, 36  
value and, 7, 12–16 ,   
77  
“Five-Card Method”, 99  
fluency and scaling Agile,  
134–137  
forecasting, 99,seealsoesti-  
matingfoundations, seealsode-  
sign; infrastructure  
building first, 54  
building inparallel  
with features, 51–  
59  
importance of,52  
Fowler, Martin, 37  
frameworks, 126–128  
G  
game, lava, xii–xiv  
giant projects, 144–147  
goals, stretch, 39  
guiding  
feature byfeature,  
19–24  
value pyramid, 5, 77  
H  
Hendrickson, Chet, 134  
hiring policy, 107  
I  
information  
changing direction, 11  
Natural W a y benefits,  
xv  
planning feature by  
feature, 24  
stopping delivery, 11  
infrastructure, seealsode-  
sign; foundations  
building inparallel  
with features, 51–  
59  
multiple teams for  
large projects, 142  
“Inspect and Adapt”  
mantra, 97  
iterations  
building insmall cy-  
cles, 44  
continuous testing,  
66–68  
mastery, 97  
perception ofslow -  
ness, 116–119  
planning workload,  
37  
prioritizing features,  
46  
refining features, 57Index • 154]  
  
L  
Large Scale Scrum (LeSS),  
126  
Larman, Craig, 126  
Larsen, Diana, 134  
lava game, xii–xiv  
learning, 128, 147,seeal-  
sotraining  
Leffingwell, Dean, 126  
LeSS (Large Scale Scrum),  
126  
limiting products and  
programs, 102  
long-term planning, 102,   
108  
M  
management  
budget decisions,  
101, 106  
control component,  
101, 109  
defining value, 83  
direction component,  
101, 108  
Natural W a y benefits,  
xv  
need for,100  
organizational deci-  
sions, 101, 106  
planning component,  
101–105  
pressure and defects,  
112–115  
staffing decisions,  
101, 106  
team purpose, auton -  
omy, and mastery,  
94–97  
mastery, 94, 97  
measuring, value, 84–87  
mid-term planning, 103  
minimal marketable fea-  
tures (MMFs), 12  
minimum viable product,  
56  
monolithic projects, 20–22  
multiple teams, disadvan -  
tages, 26, 28  
N  
Natural W a y  
Agile methods, 126summary ofbenefits,  
xv–xvi  
wandering path, xiii,   
93  
O  
observing, teams, 5  
organizing  
byfeatures, 5, 26–31 ,   
77  
management compo -  
nent, 101, 106  
self-organizing teams,  
96–97 , 101, 106  
value pyramid, 5, 77  
P  
Pink, Daniel, 94  
Pirsig, Robert, 80  
planning  
activity-based, 20–22  
continuous, 36, 41  
byfeatures, 5, 19–24 ,   
32–41 , 77  
“Five-Card Method”,  
99  
level ofdetail, 34  
long-term, 102, 108  
asmanagement com-  
ponent, 101–105  
mid-term, 103  
multiple releases, 22  
need for,33  
risk, 23, 38  
short-term, 104  
starting, 35  
stretch goals, 39  
team workload, 37  
value pyramid, 5, 77  
without estimates, 40  
predictability, planning  
feature byfeature, 23  
pressure  
defects, 39, 112–115 ,   
123  
definition ofdone,  
114, 118  
stretch goals, 39  
prioritizing  
features, 24, 46, 57,   
99, 103–104  
learning, 128  
value, xiv, 86, 88, 93,   
105  
Product Champion  
control component,  
109  
defining purpose, 95defining value, 83, 85  
management support  
of,108  
organizing teams by  
features, 27  
planning feature by  
feature, 37  
product vision, 100  
selecting features, 59  
selecting team mem -  
bers, 106  
productivity  
experts and special -  
ists, 113  
increasing, 113–115  
team purpose, auton -  
omy, and mastery,  
94–97  
professional development,  
seetraining  
programmer tests, 65, 67  
progress, identifying, 47  
purpose, 94–95  
pyramid ofvalue, 5, 77  
Q  
quality  
continuous testing,  
60–68 , 73, 77  
planning and, 21  
pressure forspeed,  
113  
refactoring, 122–124  
value pyramid, 5, 77  
questions  
budget, 34  
building byfeatures,  
43–47 , 49  
building features and  
foundations inpar-  
allel, 52, 59  
delivery, 8–11  
estimating, 35, 38  
features, 12–16 , 99  
identifying progress,  
47  
planning, 19–24 , 33–  
36, 38–39 , 99  
pressure, 39  
product vision, 45  
quality, 21  
refining design, 49  
small cycles, 44  
splitting features, 36  
team skills, 59  
value, 7Index • 155  
  
[CONTEXT FROM PREVIOUS PAGE: L  
Large Scale Scrum (LeSS),  
126  
Larman, Craig, 126  
Larsen, Diana, 134  
lava game, xii–xiv  
learning, 128, 147,seeal-  
sotraining  
Leffingwell, Dean, 126  
LeSS (Large Scale Scrum),  
126  
limiting products and  
programs, 102  
long-term planning, 102,   
108  
M  
management  
budget decisions,  
101, 106  
control component,  
101, 109  
defining value, 83  
direction component,  
101, 108  
Natural W a y benefits,  
xv  
need for,100  
organizational deci-  
sions, 101, 106  
planning component,  
101–105  
pressure and defects,  
112–115  
staffing decisions,  
101, 106  
team purpose, auton -  
omy, and mastery,  
94–97  
mastery, 94, 97  
measuring, value, 84–87  
mid-term planning, 103  
minimal marketable fea-  
tures (MMFs), 12  
minimum viable product,  
56  
monolithic projects, 20–22  
multiple teams, disadvan -  
tages, 26, 28  
N  
Natural W a y  
Agile methods, 126summary ofbenefits,  
xv–xvi  
wandering path, xiii,   
93  
O  
observing, teams, 5  
organizing  
byfeatures, 5, 26–31 ,   
77  
management compo -  
nent, 101, 106  
self-organizing teams,  
96–97 , 101, 106  
value pyramid, 5, 77  
P  
Pink, Daniel, 94  
Pirsig, Robert, 80  
planning  
activity-based, 20–22  
continuous, 36, 41  
byfeatures, 5, 19–24 ,   
32–41 , 77  
“Five-Card Method”,  
99  
level ofdetail, 34  
long-term, 102, 108  
asmanagement com-  
ponent, 101–105  
mid-term, 103  
multiple releases, 22  
need for,33  
risk, 23, 38  
short-term, 104  
starting, 35  
stretch goals, 39  
team workload, 37  
value pyramid, 5, 77  
without estimates, 40  
predictability, planning  
feature byfeature, 23  
pressure  
defects, 39, 112–115 ,   
123  
definition ofdone,  
114, 118  
stretch goals, 39  
prioritizing  
features, 24, 46, 57,   
99, 103–104  
learning, 128  
value, xiv, 86, 88, 93,   
105  
Product Champion  
control component,  
109  
defining purpose, 95defining value, 83, 85  
management support  
of,108  
organizing teams by  
features, 27  
planning feature by  
feature, 37  
product vision, 100  
selecting features, 59  
selecting team mem -  
bers, 106  
productivity  
experts and special -  
ists, 113  
increasing, 113–115  
team purpose, auton -  
omy, and mastery,  
94–97  
professional development,  
seetraining  
programmer tests, 65, 67  
progress, identifying, 47  
purpose, 94–95  
pyramid ofvalue, 5, 77  
Q  
quality  
continuous testing,  
60–68 , 73, 77  
planning and, 21  
pressure forspeed,  
113  
refactoring, 122–124  
value pyramid, 5, 77  
questions  
budget, 34  
building byfeatures,  
43–47 , 49  
building features and  
foundations inpar-  
allel, 52, 59  
delivery, 8–11  
estimating, 35, 38  
features, 12–16 , 99  
identifying progress,  
47  
planning, 19–24 , 33–  
36, 38–39 , 99  
pressure, 39  
product vision, 45  
quality, 21  
refining design, 49  
small cycles, 44  
splitting features, 36  
team skills, 59  
value, 7Index • 155]  
  
R  
refactoring, 72–73 , 77, 120–  
124, 126  
refining  
design while building  
byfeatures, 49  
features inmultiple  
iterations, 57  
product vision, 45  
regression checks, 140  
releases, multiple  
planning for,22  
separating features,  
10  
responsibility  
autonomy and, 96  
delegating manage -  
ment, 110  
return oninvestment, xv  
risk  
estimations, 38  
planning feature by  
feature, 23, 38  
S  
Scaled Agile Framework  
(SAFe), 126  
scaling  
giant projects, 144–  
147  
market for,131  
misunderstanding  
need for,130–132  
multiple teams, 138–  
147  
organizing teams by  
features, 31  
simplicity vs.

## Translated Text:

سادگی، xvi، 88–91، 93، 133–137   
کاربران نهایی، xv   
برآورد، همچنین ببینید forecasting   
معایب، 34–35، 38   
فشار برای سرعت، 113–114   
قطعات کاری، 37   
کار کردن بدون، 40   
  
متخصصان و کارشناسان   
feature teams، 29–30   
infrastructure teams برای پروژه‌های بزرگ، 142   
بهره‌وری تیم، 113   
external dependencies، 117   
Extreme Programming، 126   
  
F   
feature teams، تعریف‌شده، 139، همچنین ببینید teams   
features، همچنین ببینید testing   
درخواست برای، 117   
ساختن توسط، 5، 42–49، 77   
ساختن foundations ابتدا، 54   
ساختن به‌صورت موازی با foundations، 51–59   
forecasting با “Five-Card Method”، 99   
هدایت توسط، 19–24   
minimal marketable features (MMFs)، 12   
سازمان‌دهی توسط، 5، 26–31، 77   
برنامه‌ریزی توسط، 5، 19–24، 32–41، 77   
اولویت‌بندی، 24، 46، 57، 99، 103–104   
بهبود در چندین iteration، 57   
جدا کردن delivery، 9–10   
slicing، 5، 40، 99   
small cycles، 44   
splitting، 36   
ارزش و، 7، 12–16، 77   
  
“Five-Card Method”، 99   
fluency و scaling Agile، 134–137   
forecasting، 99، همچنین ببینید estimating   
foundations، همچنین ببینید design؛ infrastructure   
ساختن ابتدا، 54   
ساختن به‌صورت موازی با features، 51–59   
اهمیت، 52   
Fowler، Martin، 37   
frameworks، 126–128   
  
G   
بازی، lava، xii–xiv   
پروژه‌های عظیم، 144–147   
اهداف، stretch، 39   
  
هدایت   
feature به feature، 19–24   
هرم ارزش، 5، 77   
  
H   
Hendrickson، Chet، 134   
سیاست استخدام، 107   
  
I   
اطلاعات   
تغییر جهت، 11   
مزایای Natural Way، xv   
برنامه‌ریزی feature به feature، 24   
توقف delivery، 11   
  
infrastructure، همچنین ببینید design؛ foundations   
ساختن به‌صورت موازی با features، 51–59   
چندین تیم برای پروژه‌های بزرگ، 142   
مانترای “Inspect and Adapt”، 97   
  
iterations   
ساختن در small cycles، 44   
continuous testing، 66–68   
تسلط، 97   
درک کندی، 116–119   
برنامه‌ریزی بار کاری، 37   
اولویت‌بندی features، 46   
بهبود features، 57   
  
L   
Large Scale Scrum (LeSS)، 126   
Larman، Craig، 126   
Larsen، Diana، 134   
بازی lava، xii–xiv   
یادگیری، 128، 147، همچنین ببینید training   
Leffingwell، Dean، 126   
LeSS (Large Scale Scrum)، 126   
محدود کردن محصولات و برنامه‌ها، 102   
برنامه‌ریزی بلندمدت، 102، 108   
  
M   
مدیریت   
تصمیمات بودجه، 101، 106   
کنترل component، 101، 109   
تعریف ارزش، 83   
جهت‌دهی component، 101، 108   
مزایای Natural Way، xv   
نیاز به، 100   
تصمیمات سازمانی، 101، 106   
برنامه‌ریزی component، 101–105   
فشار و نقص‌ها، 112–115   
تصمیمات استخدام، 101، 106   
هدف تیم، خودمختاری و تسلط، 94–97   
تسلط، 94، 97   
اندازه‌گیری ارزش، 84–87   
برنامه‌ریزی میان‌مدت، 103   
minimal marketable features (MMFs)، 12   
minimum viable product، 56   
پروژه‌های monolithic، 20–22   
چندین تیم، معایب، 26، 28   
  
N   
Natural Way   
روش‌های Agile، 126   
خلاصه مزایا، xv–xvi   
مسیر سرگردان، xiii، 93   
  
O   
مشاهده، تیم‌ها، 5   
سازمان‌دهی   
توسط features، 5، 26–31، 77   
component مدیریت، 101، 106   
تیم‌های خودسازمان‌ده، 96–97، 101، 106   
هرم ارزش، 5، 77   
  
P   
Pink، Daniel، 94   
Pirsig، Robert، 80   
برنامه‌ریزی   
بر اساس فعالیت، 20–22   
continuous، 36، 41   
توسط features، 5، 19–24، 32–41، 77   
“Five-Card Method”، 99   
سطح جزئیات، 34   
بلندمدت، 102، 108   
به‌عنوان component مدیریت، 101–105   
میان‌مدت، 103   
چندین release، 22   
نیاز به، 33   
ریسک، 23، 38   
کوتاه‌مدت، 104   
شروع، 35   
اهداف stretch، 39   
بار کاری تیم، 37   
هرم ارزش، 5، 77   
بدون برآورد، 40   
  
پیش‌بینی‌پذیری، برنامه‌ریزی feature به feature، 23   
فشار   
نقص‌ها، 39، 112–115، 123   
تعریف done، 114، 118   
اهداف stretch، 39   
  
اولویت‌بندی   
features، 24، 46، 57، 99، 103–104   
یادگیری، 128   
ارزش، xiv، 86، 88، 93، 105   
  
Product Champion   
کنترل component، 109   
تعریف هدف، 95   
تعریف ارزش، 83، 85   
حمایت مدیریت از، 108   
سازمان‌دهی تیم‌ها توسط features، 27   
برنامه‌ریزی feature به feature، 37   
چشم‌انداز محصول، 100   
انتخاب features، 59   
انتخاب اعضای تیم، 106   
  
بهره‌وری   
متخصصان و کارشناسان، 113   
افزایش، 113–115   
هدف تیم، خودمختاری و تسلط، 94–97   
  
توسعه حرفه‌ای، همچنین ببینید training   
programmer tests، 65، 67   
پیشرفت، شناسایی، 47   
هدف، 94–95   
هرم ارزش، 5، 77   
  
Q   
کیفیت   
continuous testing، 60–68، 73، 77   
برنامه‌ریزی و، 21   
فشار برای سرعت، 113   
refactoring، 122–124   
هرم ارزش، 5، 77   
  
سؤالات   
بودجه، 34   
ساختن توسط features، 43–47، 49   
ساختن features و foundations به‌صورت موازی، 52، 59   
delivery، 8–11   
برآورد، 35، 38   
features، 12–16، 99   
شناسایی پیشرفت، 47   
برنامه‌ریزی، 19–24، 33–36، 38–39، 99   
فشار، 39   
چشم‌انداز محصول، 45   
کیفیت، 21   
بهبود طراحی، 49   
small cycles، 44   
splitting features، 36   
مهارت‌های تیم، 59   
ارزش، 7   
  
Index • 155

================================================================================

# Section 148

## Original Text:

ease,  
133–137  
Schwaber, Ken, 126  
scope, planning and esti-  
mating, 38  
Scrum  
Definition ofDone,  
118  
“Inspect and Adapt”  
mantra, 97  
practices, 126  
simplicity, 133  
ScrumMaster, 126  
shepherding, 30  
Shore, Jim, 134  
short-term planning, 104simplicity  
building features and  
foundations inpar-  
allel, 56  
design, 69  
vs.

## Translated Text:

آسانی،   
133–137   
Schwaber، Ken، 126   
scope، برنامه‌ریزی و تخمین، 38   
Scrum   
Definition of Done،   
118   
شعار “Inspect and Adapt”، 97   
practices، 126   
simplicity، 133   
ScrumMaster، 126   
shepherding، 30   
Shore، Jim، 134   
برنامه‌ریزی کوتاه‌مدت، 104   
simplicity   
ساختن features و foundations به‌صورت موازی، 56   
design، 69   
vs.

================================================================================

# Section 149

## Original Text:

ease, xvi, 88–91 ,   
93, 133–137  
scaled Agile, 133–137  
size  
budget and, 106  
design, 49  
features, 40, 99  
forecasting, 99  
planning, 102–103  
skills  
building features and  
foundations inpar-  
allel, 59  
building speed with,  
116–119  
mastery, 94, 97  
team productivity,  
113–114  
slicing  
features, 40, 99  
Scaling Agile, 134  
value pyramid, 5  
slowness, perception of,  
116–119  
specialists, seeexperts and  
specialists  
speed  
building features and  
foundations inpar-  
allel, 53–58  
building with skills,  
116–119  
continuous testing, 68  
defects, 39, 63–64 , 68,   
112–115  
definition ofdone,  
114, 118  
design deterioration,  
70, 73  
refining design while  
building byfea-  
tures, 49  
splitting features, 36  
sprints  
defined, 36  
definition ofdone,  
118  
planning, 36  
staff, seeteams  
staffing decisions, 101,   
106–107  
steering, 105stopping  
before deadline, 34,   
105  
value of,11, 15  
stories, planning, 36  
stretch goals, 39  
Sutherland, Jeff, 126  
T  
tasks, planning, 36  
TDD (test-driven develop -  
ment), 67, 73, 117, 126,   
140  
teams  
Communities ofPrac -  
tice, 30  
experts and special -  
ists, 29–30 , 113, 142  
fluency and scaling  
Agile, 134–137  
guiding, 5  
increasing productivi -  
ty,113  
multiple teams and  
disadvantages, 26,   
28  
multiple teams and  
scaling Agile, 138–  
147  
Natural W a y benefits,  
xv  
organizing byfea-  
tures, 26–31  
process changes, 128  
purpose, autonomy,  
and mastery, 94–97  
self-organizing teams,  
96–97 , 101, 106  
staffing decisions,  
101, 106–107  
workload planning,  
37  
test-driven development  
(TDD), 67, 73, 117, 126,   
140  
testing  
acceptance test-driv -  
endevelopment,  
66, 73, 117, 126, 140  
automated, 66–67 , 73,   
140  
business-level tests,  
65–66 , 68  
continuous, 60–68 ,   
73, 77  
coordinating multiple  
teams, 140–147Index • 156  
  
[CONTEXT FROM PREVIOUS PAGE: R  
refactoring, 72–73 , 77, 120–  
124, 126  
refining  
design while building  
byfeatures, 49  
features inmultiple  
iterations, 57  
product vision, 45  
regression checks, 140  
releases, multiple  
planning for,22  
separating features,  
10  
responsibility  
autonomy and, 96  
delegating manage -  
ment, 110  
return oninvestment, xv  
risk  
estimations, 38  
planning feature by  
feature, 23, 38  
S  
Scaled Agile Framework  
(SAFe), 126  
scaling  
giant projects, 144–  
147  
market for,131  
misunderstanding  
need for,130–132  
multiple teams, 138–  
147  
organizing teams by  
features, 31  
simplicity vs.

## Translated Text:

آسانی، xvi، 88–91،   
93، 133–137   
Scaled Agile، 133–137   
  
اندازه   
بودجه و، 106   
طراحی، 49   
ویژگی‌ها، 40، 99   
پیش‌بینی، 99   
برنامه‌ریزی، 102–103   
  
مهارت‌ها   
ساخت ویژگی‌ها و   
پایه‌ها به‌صورت   
موازی، 59   
افزایش سرعت با،   
116–119   
تسلط، 94، 97   
بهره‌وری تیم،   
113–114   
  
برش   
ویژگی‌ها، 40، 99   
Scaling Agile، 134   
هرم ارزش، 5   
  
کندی، ادراک از،   
116–119   
  
متخصصان، به بخش   
experts and specialists   
مراجعه کنید   
  
سرعت   
ساخت ویژگی‌ها و   
پایه‌ها به‌صورت   
موازی، 53–58   
ساخت با مهارت‌ها،   
116–119   
continuous testing، 68   
defects، 39، 63–64، 68،   
112–115   
definition of done،   
114، 118   
design deterioration،   
70، 73   
refining design while   
building by features، 49   
splitting features، 36   
  
sprints   
تعریف‌شده، 36   
definition of done،   
118   
برنامه‌ریزی، 36   
  
کارکنان، به بخش teams   
مراجعه کنید   
  
تصمیم‌گیری‌های   
staffing، 101،   
106–107   
هدایت، 105   
توقف قبل از   
موعد مقرر، 34،   
105   
ارزش، 11، 15   
  
stories، برنامه‌ریزی، 36   
اهداف کششی، 39   
Sutherland، Jeff، 126   
  
T   
tasks، برنامه‌ریزی، 36   
TDD (test-driven development)، 67، 73، 117، 126،   
140   
  
تیم‌ها   
Communities of Practice، 30   
experts and specialists، 29–30، 113، 142   
fluency و Scaling Agile، 134–137   
هدایت، 5   
افزایش بهره‌وری، 113   
تیم‌های متعدد و   
معایب، 26، 28   
تیم‌های متعدد و   
Scaling Agile، 138–147   
مزایای Natural Way، xv   
سازمان‌دهی بر اساس   
ویژگی‌ها، 26–31   
تغییرات فرآیند، 128   
هدف، خودمختاری،   
و تسلط، 94–97   
تیم‌های خودسازمان‌ده،   
96–97، 101، 106   
تصمیم‌گیری‌های   
staffing، 101، 106–107   
برنامه‌ریزی بار کاری،   
37   
  
test-driven development   
(TDD)، 67، 73، 117، 126،   
140   
  
تست   
acceptance test-driven development،   
66، 73، 117، 126، 140   
تست خودکار، 66–67، 73،   
140   
تست‌های سطح کسب‌وکار،   
65–66، 68   
continuous، 60–68،   
73، 77   
هماهنگی تیم‌های   
متعدد، 140–147   
  
[CONTEXT FROM PREVIOUS PAGE: R   
refactoring، 72–73، 77، 120–124، 126   
refining   
طراحی هنگام ساخت   
بر اساس ویژگی‌ها، 49   
ویژگی‌ها در چندین   
تکرار، 57   
vision محصول، 45   
regression checks، 140   
releases، چندگانه   
برنامه‌ریزی برای، 22   
جدا کردن ویژگی‌ها،   
10   
  
مسئولیت   
خودمختاری و، 96   
واگذاری مدیریت، 110   
return on investment، xv   
  
ریسک   
تخمین‌ها، 38   
برنامه‌ریزی ویژگی به   
ویژگی، 23، 38   
  
S   
Scaled Agile Framework   
(SAFe)، 126   
  
scaling   
پروژه‌های بزرگ، 144–147   
بازار برای، 131   
سوءتفاهم در   
نیاز به، 130–132   
تیم‌های متعدد، 138–147   
سازمان‌دهی تیم‌ها بر اساس   
ویژگی‌ها، 31   
سادگی در مقابل.

================================================================================

# Section 150

## Original Text:

ease,  
133–137  
Schwaber, Ken, 126  
scope, planning and esti-  
mating, 38  
Scrum  
Definition ofDone,  
118  
“Inspect and Adapt”  
mantra, 97  
practices, 126  
simplicity, 133  
ScrumMaster, 126  
shepherding, 30  
Shore, Jim, 134  
short-term planning, 104simplicity  
building features and  
foundations inpar-  
allel, 56  
design, 69  
vs.

## Translated Text:

آسانی،   
133–137   
Schwaber, Ken، 126   
scope، برنامه‌ریزی و برآورد، 38   
Scrum   
Definition of Done،   
118   
شعار “Inspect and Adapt”، 97   
practices، 126   
simplicity، 133   
ScrumMaster، 126   
هدایت، 30   
Shore, Jim، 134   
برنامه‌ریزی کوتاه‌مدت، 104   
simplicity   
ساختن features و foundations به‌صورت موازی، 56   
design، 69   
در مقابل.

================================================================================

# Section 151

## Original Text:

ease, xvi, 88–91 ,   
93, 133–137  
scaled Agile, 133–137  
size  
budget and, 106  
design, 49  
features, 40, 99  
forecasting, 99  
planning, 102–103  
skills  
building features and  
foundations inpar-  
allel, 59  
building speed with,  
116–119  
mastery, 94, 97  
team productivity,  
113–114  
slicing  
features, 40, 99  
Scaling Agile, 134  
value pyramid, 5  
slowness, perception of,  
116–119  
specialists, seeexperts and  
specialists  
speed  
building features and  
foundations inpar-  
allel, 53–58  
building with skills,  
116–119  
continuous testing, 68  
defects, 39, 63–64 , 68,   
112–115  
definition ofdone,  
114, 118  
design deterioration,  
70, 73  
refining design while  
building byfea-  
tures, 49  
splitting features, 36  
sprints  
defined, 36  
definition ofdone,  
118  
planning, 36  
staff, seeteams  
staffing decisions, 101,   
106–107  
steering, 105stopping  
before deadline, 34,   
105  
value of,11, 15  
stories, planning, 36  
stretch goals, 39  
Sutherland, Jeff, 126  
T  
tasks, planning, 36  
TDD (test-driven develop -  
ment), 67, 73, 117, 126,   
140  
teams  
Communities ofPrac -  
tice, 30  
experts and special -  
ists, 29–30 , 113, 142  
fluency and scaling  
Agile, 134–137  
guiding, 5  
increasing productivi -  
ty,113  
multiple teams and  
disadvantages, 26,   
28  
multiple teams and  
scaling Agile, 138–  
147  
Natural W a y benefits,  
xv  
organizing byfea-  
tures, 26–31  
process changes, 128  
purpose, autonomy,  
and mastery, 94–97  
self-organizing teams,  
96–97 , 101, 106  
staffing decisions,  
101, 106–107  
workload planning,  
37  
test-driven development  
(TDD), 67, 73, 117, 126,   
140  
testing  
acceptance test-driv -  
endevelopment,  
66, 73, 117, 126, 140  
automated, 66–67 , 73,   
140  
business-level tests,  
65–66 , 68  
continuous, 60–68 ,   
73, 77  
coordinating multiple  
teams, 140–147Index • 156]  
  
eliminating test-and -  
fixinterval, 48, 119  
monolithic projects,  
20–21  
programmer tests,  
65, 67  
Scrum, 126  
speed and, 117  
test-driven develop -  
ment (TDD), 67,   
73, 117, 126, 140  
thinking, seealsoques -  
tions; value  
drawing pictures, ix  
asmountain, 148–149  
need for,xvi, 128  
time  
defects, 63, 68  
eliminating test-and -  
fixinterval, 48, 119  
organizing byskill, 28planning feature by  
feature, 34  
value byfeature, 13  
training  
expertise and feature  
teams, 29  
increasing productivi -  
ty,114  
prioritizing, 128  
scaling training pro-  
grams, 132  
V  
vacations, 115  
value  
defining, 7, 80–83 , 85  
delivering early, xiii,   
8–11 , 15, 77  
features, 7, 12–16 , 77  
focus on,xiv, 86, 88,   
93, 105  
measuring, 84–87overview, 5  
prioritizing features,  
46, 99  
pyramid, 5, 77  
ofstopping, 11, 15  
summary, 77  
vision, refining, 45  
Vodde, Bas, 126  
W  
whipping theponies, 112–  
115  
X  
XP ,126  
Y  
“Yesterday’s Weather”, 37  
Z  
ZenandtheArtofMotorcy -  
cleMaintenance ,80Index • 157  
  
[CONTEXT FROM PREVIOUS PAGE: eliminating test-and -  
fixinterval, 48, 119  
monolithic projects,  
20–21  
programmer tests,  
65, 67  
Scrum, 126  
speed and, 117  
test-driven develop -  
ment (TDD), 67,   
73, 117, 126, 140  
thinking, seealsoques -  
tions; value  
drawing pictures, ix  
asmountain, 148–149  
need for,xvi, 128  
time  
defects, 63, 68  
eliminating test-and -  
fixinterval, 48, 119  
organizing byskill, 28planning feature by  
feature, 34  
value byfeature, 13  
training  
expertise and feature  
teams, 29  
increasing productivi -  
ty,114  
prioritizing, 128  
scaling training pro-  
grams, 132  
V  
vacations, 115  
value  
defining, 7, 80–83 , 85  
delivering early, xiii,   
8–11 , 15, 77  
features, 7, 12–16 , 77  
focus on,xiv, 86, 88,   
93, 105  
measuring, 84–87overview, 5  
prioritizing features,  
46, 99  
pyramid, 5, 77  
ofstopping, 11, 15  
summary, 77  
vision, refining, 45  
Vodde, Bas, 126  
W  
whipping theponies, 112–  
115  
X  
XP ,126  
Y  
“Yesterday’s Weather”, 37  
Z  
ZenandtheArtofMotorcy -  
cleMaintenance ,80Index • 157]  
  
Explore Testing and Cucumber  
Explore theuncharted waters ofexploratory testing and beef upyour automated testing  
with more Cucumber—now forJava, too.

## Translated Text:

آسانی، xvi، 88–91،   
93، 133–137   
Scaled Agile، 133–137   
اندازه   
بودجه و، 106   
طراحی، 49   
ویژگی‌ها، 40، 99   
پیش‌بینی، 99   
برنامه‌ریزی، 102–103   
مهارت‌ها   
ساخت ویژگی‌ها و   
پایه‌ها به‌صورت موازی، 59   
افزایش سرعت با،   
116–119   
تسلط، 94، 97   
بهره‌وری تیم،   
113–114   
بخش‌بندی   
ویژگی‌ها، 40، 99   
Scaling Agile، 134   
هرم ارزش، 5   
کندی، درک از،   
116–119   
متخصصان، به متخصصان و   
کارشناسان مراجعه کنید   
سرعت   
ساخت ویژگی‌ها و   
پایه‌ها به‌صورت موازی، 53–58   
ساخت با مهارت‌ها،   
116–119   
Continuous Testing، 68   
نواقص، 39، 63–64، 68،   
112–115   
تعریف Definition of Done،   
114، 118   
زوال طراحی،   
70، 73   
تکمیل طراحی در حین   
ساخت با ویژگی‌ها، 49   
تقسیم ویژگی‌ها، 36   
Sprints   
تعریف شده، 36   
تعریف Definition of Done،   
118   
برنامه‌ریزی، 36   
کارکنان، به Teams مراجعه کنید   
تصمیمات استخدامی، 101،   
106–107   
هدایت، 105   
توقف قبل از مهلت، 34،   
105   
ارزش، 11، 15   
Stories، برنامه‌ریزی، 36   
اهداف کششی، 39   
Sutherland، Jeff، 126   
T   
Tasks، برنامه‌ریزی، 36   
TDD (Test-Driven Development)، 67، 73، 117، 126،   
140   
Teams   
Communities of Practice، 30   
کارشناسان و متخصصان،   
29–30، 113، 142   
Fluency و Scaling Agile، 134–137   
راهنمایی، 5   
افزایش بهره‌وری،   
113   
چند تیم و   
معایب، 26، 28   
چند تیم و   
Scaling Agile، 138–147   
مزایای Natural Way،   
xv   
سازماندهی بر اساس   
ویژگی‌ها، 26–31   
تغییرات فرآیند، 128   
هدف، استقلال،   
و تسلط، 94–97   
تیم‌های خودسازمان‌ده،   
96–97، 101، 106   
تصمیمات استخدامی،   
101، 106–107   
برنامه‌ریزی بار کاری،   
37   
Test-Driven Development   
(TDD)، 67، 73، 117، 126،   
140   
Testing   
Acceptance Test-Driven Development،   
66، 73، 117، 126، 140   
Automated، 66–67، 73،   
140   
Business-Level Tests،   
65–66، 68   
Continuous، 60–68،   
73، 77   
هماهنگی چند تیم، 140–147   
  
حذف Test-and-Fix Interval، 48، 119   
پروژه‌های Monolithic،   
20–21   
Programmer Tests،   
65، 67   
Scrum، 126   
سرعت و، 117   
Test-Driven Development (TDD)، 67،   
73، 117، 126، 140   
تفکر، همچنین به Questions و   
Value مراجعه کنید   
رسم تصاویر، ix   
به‌عنوان کوه، 148–149   
نیاز به، xvi، 128   
زمان   
نواقص، 63، 68   
حذف Test-and-Fix Interval، 48، 119   
سازماندهی بر اساس مهارت، 28   
برنامه‌ریزی ویژگی به ویژگی، 34   
ارزش بر اساس ویژگی، 13   
Training   
تخصص و تیم‌های ویژگی، 29   
افزایش بهره‌وری،   
114   
اولویت‌بندی، 128   
Scaling Training Programs، 132   
V   
مرخصی‌ها، 115   
ارزش   
تعریف، 7، 80–83، 85   
تحویل زودهنگام، xiii،   
8–11، 15، 77   
ویژگی‌ها، 7، 12–16، 77   
تمرکز بر، xiv، 86، 88،   
93، 105   
اندازه‌گیری، 84–87   
بررسی اجمالی، 5   
اولویت‌بندی ویژگی‌ها،   
46، 99   
هرم، 5، 77   
ارزش توقف، 11، 15   
خلاصه، 77   
تصفیه Vision، 45   
Vodde، Bas، 126   
W   
Whipping the Ponies، 112–115   
X   
XP، 126   
Y   
“Yesterday’s Weather”، 37   
Z   
Zen and the Art of Motorcycle Maintenance، 80   
  
Explore Testing and Cucumber   
آب‌های ناشناخته Exploratory Testing را کشف کنید و Automated Testing خود را با Cucumber بیشتر تقویت کنید—اکنون برای Java نیز.

================================================================================

# Section 152

## Original Text:

Explore It. Uncover surprises, risks, and potentially se-  
rious bugs with exploratory testing. Rather  
than designing alltests inadvance, explorers  
design andexecute small, rapid experiments,  
using what they learned from thelastlittle  
experiment toinform thenext. Learn essen -  
tialskills ofamaster explorer, including how  
toanalyze software todiscover keypoints  
ofvulnerability, how todesign experiments  
onthefly,how tohone your observation  
skills, and how tofocus your efforts. Elisabeth Hendrickson  
(186 pages) ISBN :9781937785024. $29  
https://pragprog. com/book/ehxta  
The Cucumber for Java Book  
Teams working ontheJVM cannow say  
goodbye forever tomisunderstood require -  
ments, tedious manual acceptance tests, and  
out-of-date documentation. Cucumber—the  
popular, open-source tool that helps teams  
communicate more effectively with their  
customers—now hasaJava version, and our  
bestselling Cucumber Book hasbeen updated  
tomatch. TheCucumber forJava Book hasthe  
same great advice about how todeliver rock -  
solid applications collaboratively, butwith  
allcode completely rewritten inJava. New  
chapters cover features unique totheJava  
version ofCucumber, and reflect insights  
from theCucumber team since theoriginal  
book w a s published. SebRose, Matt Wynne &Aslak Hellesoy  
(338 pages) ISBN :9781941222294. $36  
https://pragprog. com/book/srjcuc  
  
[CONTEXT FROM PREVIOUS PAGE: Explore Testing and Cucumber  
Explore theuncharted waters ofexploratory testing and beef upyour automated testing  
with more Cucumber—now forJava, too. Explore It. Uncover surprises, risks, and potentially se-  
rious bugs with exploratory testing. Rather  
than designing alltests inadvance, explorers  
design andexecute small, rapid experiments,  
using what they learned from thelastlittle  
experiment toinform thenext.

## Translated Text:

اکتشاف کنید. با استفاده از exploratory testing، شگفتی‌ها، ریسک‌ها و باگ‌های بالقوه جدی را کشف کنید. به جای طراحی تمام تست‌ها از پیش، کاوشگران آزمایش‌های کوچک و سریع طراحی و Execute می‌کنند و از آنچه در آخرین آزمایش کوچک آموخته‌اند برای اطلاع‌رسانی به آزمایش بعدی استفاده می‌کنند. مهارت‌های ضروری یک کاوشگر حرفه‌ای را بیاموزید، از جمله نحوه تحلیل نرم‌افزار برای کشف نقاط کلیدی آسیب‌پذیری، نحوه طراحی آزمایش‌ها به صورت آنی، نحوه تقویت مهارت‌های مشاهده‌ای خود و نحوه تمرکز تلاش‌های خود.   
الیزابت هندریکسون   
(186 صفحه) ISBN: 9781937785024. $29   
https://pragprog.com/book/ehxta   
  
کتاب The Cucumber for Java   
تیم‌هایی که روی JVM کار می‌کنند اکنون می‌توانند برای همیشه با نیازمندی‌های اشتباه فهمیده شده، تست‌های پذیرش دستی خسته‌کننده و مستندات قدیمی خداحافظی کنند. Cucumber—ابزار محبوب و متن‌باز که به تیم‌ها کمک می‌کند با مشتریان خود موثرتر ارتباط برقرار کنند—اکنون نسخه‌ای برای Java دارد و کتاب پرفروش Cucumber ما به‌روزرسانی شده تا با آن هماهنگ شود. کتاب The Cucumber for Java همان توصیه‌های عالی درباره نحوه ارائه برنامه‌های بی‌نقص به صورت مشارکتی را دارد، اما با تمام کدها که کاملاً در Java بازنویسی شده‌اند. فصل‌های جدید ویژگی‌های منحصر به فرد نسخه Java از Cucumber را پوشش می‌دهند و بینش‌هایی از تیم Cucumber را از زمان انتشار کتاب اصلی منعکس می‌کنند.   
سب رز، مت وین و اسلاک هلسوی   
(338 صفحه) ISBN: 9781941222294. $36   
https://pragprog.com/book/srjcuc

================================================================================

# Section 153

## Original Text:

Learn essen -  
tialskills ofamaster explorer, including how  
toanalyze software todiscover keypoints  
ofvulnerability, how todesign experiments  
onthefly,how tohone your observation  
skills, and how tofocus your efforts. Elisabeth Hendrickson  
(186 pages) ISBN :9781937785024. $29  
https://pragprog. com/book/ehxta  
The Cucumber for Java Book  
Teams working ontheJVM cannow say  
goodbye forever tomisunderstood require -  
ments, tedious manual acceptance tests, and  
out-of-date documentation. Cucumber—the  
popular, open-source tool that helps teams  
communicate more effectively with their  
customers—now hasaJava version, and our  
bestselling Cucumber Book hasbeen updated  
tomatch. TheCucumber forJava Book hasthe  
same great advice about how todeliver rock -  
solid applications collaboratively, butwith  
allcode completely rewritten inJava. New  
chapters cover features unique totheJava  
version ofCucumber, and reflect insights  
from theCucumber team since theoriginal  
book w a s published. SebRose, Matt Wynne &Aslak Hellesoy  
(338 pages) ISBN :9781941222294. $36  
https://pragprog. com/book/srjcuc]  
  
Be Agile  
Don’t just“do” agile; yo u want tobeagile. We’ll show yo u how tomake software better. Your Code As a Crime Scene  
Jack theRipper and legacy codebases have  
more incommon than you’d think. Inspired  
byforensic psychology methods, thisbook  
teaches yo u strategies topredict thefuture  
ofyour codebase, assess refactoring direc -  
tion, and understand how your team influ -  
ences thedesign. With itsunique blend of  
forensic psychology and code analysis, this  
book arms yo u with thestrategies yo u need,  
nomatter what programming language yo u  
use. Adam Tornhill  
(218 pages) ISBN :9781680500387. $36  
https://pragprog. com/book/atcrime  
The Agile Samurai  
Here arethree simple truths about software  
development:  
1. Y o u can’t gather alltherequirements  
upfront. 2. The requirements yo u dogather will  
change. 3. There isalways more todothan time  
and money will allow. Those arethefacts oflife.

## Translated Text:

مهارت‌های ضروری یک کاوشگر حرفه‌ای را بیاموزید، از جمله چگونگی تحلیل نرم‌افزار برای کشف نقاط کلیدی آسیب‌پذیری، طراحی آزمایش‌ها به‌صورت فوری، تقویت مهارت‌های مشاهده و تمرکز بر تلاش‌هایتان.   
الیزابت هندریکسون   
(۱۸۶ صفحه)   
ISBN: 9781937785024   
$۲۹   
[https://pragprog.com/book/ehxta](https://pragprog.com/book/ehxta)   
  
\*\*The Cucumber for Java Book\*\*   
تیم‌هایی که بر روی JVM کار می‌کنند اکنون می‌توانند برای همیشه با نیازمندی‌های اشتباه فهمیده‌شده، تست‌های پذیرش دستی خسته‌کننده، و مستندات قدیمی خداحافظی کنند.   
Cucumber—ابزار محبوب و متن‌باز که به تیم‌ها کمک می‌کند تا با مشتریان خود به‌طور مؤثرتری ارتباط برقرار کنند—اکنون نسخه‌ای برای Java دارد و کتاب پرفروش Cucumber ما به‌روزرسانی شده است تا با آن هماهنگ شود.   
کتاب The Cucumber for Java Book همان توصیه‌های عالی درباره چگونگی ارائه برنامه‌های کاربردی قدرتمند به‌صورت مشارکتی را ارائه می‌دهد، اما با تمام کدهایی که به‌طور کامل در Java بازنویسی شده‌اند.   
فصل‌های جدید ویژگی‌های منحصربه‌فرد نسخه Java از Cucumber را پوشش می‌دهند و بینش‌هایی از تیم Cucumber را از زمان انتشار کتاب اصلی منعکس می‌کنند.   
سب رز، مت وین و اسلاک هل‌سوی   
(۳۳۸ صفحه)   
ISBN: 9781941222294   
$۳۶   
[https://pragprog.com/book/srjcuc](https://pragprog.com/book/srjcuc)   
  
\*\*Be Agile\*\*   
فقط "Agile انجام ندهید؛" شما باید Agile باشید. ما به شما نشان می‌دهیم که چگونه نرم‌افزار بهتری بسازید.   
  
\*\*Your Code As a Crime Scene\*\*   
جک قاتل زنجیره‌ای و legacy codebases بیشتر از آنچه فکر می‌کنید با هم مشترک دارند.   
با الهام از روش‌های روان‌شناسی جنایی، این کتاب به شما استراتژی‌هایی را می‌آموزد تا آینده کدبیس خود را پیش‌بینی کنید، جهت Refactoring را ارزیابی کنید و بفهمید که تیم شما چگونه بر Design تأثیر می‌گذارد.   
با ترکیب منحصربه‌فرد روان‌شناسی جنایی و تحلیل کد، این کتاب شما را با استراتژی‌هایی که نیاز دارید، مجهز می‌کند، مهم نیست از چه زبان برنامه‌نویسی استفاده می‌کنید.   
آدام تورنهیل   
(۲۱۸ صفحه)   
ISBN: 9781680500387   
$۳۶   
[https://pragprog.com/book/atcrime](https://pragprog.com/book/atcrime)   
  
\*\*The Agile Samurai\*\*   
اینجا سه حقیقت ساده درباره توسعه نرم‌افزار وجود دارد:   
۱. شما نمی‌توانید تمام نیازمندی‌ها را از ابتدا جمع‌آوری کنید.   
۲. نیازمندی‌هایی که جمع‌آوری می‌کنید تغییر خواهند کرد.   
۳. همیشه کار بیشتری برای انجام دادن وجود دارد نسبت به زمانی که دارید و بودجه‌ای که در اختیار دارید.   
این‌ها واقعیت‌های زندگی هستند.

================================================================================

# Section 154

## Original Text:

Butyo u candeal  
with those facts (and more) bybecoming a  
fierce software-delivery professional, capable  
ofdispatching themost dire ofsoftware  
projects and thetoughest delivery schedules  
with ease and grace. This title isalso available asanaudio book. Jonathan Rasmusson  
(280 pages) ISBN :9781934356586. $34. 95  
https://pragprog. com/book/jtrap  
  
[CONTEXT FROM PREVIOUS PAGE: Be Agile  
Don’t just“do” agile; yo u want tobeagile. We’ll show yo u how tomake software better. Your Code As a Crime Scene  
Jack theRipper and legacy codebases have  
more incommon than you’d think. Inspired  
byforensic psychology methods, thisbook  
teaches yo u strategies topredict thefuture  
ofyour codebase, assess refactoring direc -  
tion, and understand how your team influ -  
ences thedesign. With itsunique blend of  
forensic psychology and code analysis, this  
book arms yo u with thestrategies yo u need,  
nomatter what programming language yo u  
use. Adam Tornhill  
(218 pages) ISBN :9781680500387. $36  
https://pragprog. com/book/atcrime  
The Agile Samurai  
Here arethree simple truths about software  
development:  
1. Y o u can’t gather alltherequirements  
upfront. 2. The requirements yo u dogather will  
change. 3. There isalways more todothan time  
and money will allow. Those arethefacts oflife. Butyo u candeal  
with those facts (and more) bybecoming a  
fierce software-delivery professional, capable  
ofdispatching themost dire ofsoftware  
projects and thetoughest delivery schedules  
with ease and grace. This title isalso available asanaudio book. Jonathan Rasmusson  
(280 pages) ISBN :9781934356586. $34. 95  
https://pragprog. com/book/jtrap]  
  
Past and Present  
T oseewhere we’re going, remember how wegothere, and learn how totake ahealthier  
approach toprogramming. Fire in the Valley  
Inthe1970s, while their contemporaries were  
protesting thecomputer asatool ofdehu -  
manization and oppression, amotley collec -  
tion ofcollege dropouts, hippies, and elec-  
tronics fanatics were engaged insomething  
much more subversive.

## Translated Text:

اما شما می‌توانید با این حقایق (و بیشتر از آن) کنار بیایید، با تبدیل شدن به یک حرفه‌ای در زمینه تحویل نرم‌افزار، که قادر است سخت‌ترین پروژه‌های نرم‌افزاری و فشرده‌ترین زمان‌بندی‌های تحویل را با سهولت و ظرافت مدیریت کند. این عنوان به صورت کتاب صوتی نیز در دسترس است.   
Jonathan Rasmusson   
(۲۸۰ صفحه)   
ISBN: 9781934356586   
$34.95   
[https://pragprog.com/book/jtrap](https://pragprog.com/book/jtrap)   
  
[متن از صفحه قبلی:   
چابک باشید   
فقط "چابک کار نکنید"؛ شما باید چابک باشید. ما به شما نشان خواهیم داد که چگونه نرم‌افزار بهتری بسازید.   
کد شما به‌عنوان یک صحنه جرم   
Jack the Ripper و کدبیس‌های قدیمی بیشتر از آنچه فکر می‌کنید وجه اشتراک دارند. با الهام از روش‌های روانشناسی جنایی، این کتاب به شما استراتژی‌هایی آموزش می‌دهد تا آینده کدبیس خود را پیش‌بینی کنید، جهت Refactoring را ارزیابی کنید و بفهمید تیم شما چگونه بر طراحی تأثیر می‌گذارد. با ترکیب منحصربه‌فرد روانشناسی جنایی و تحلیل کد، این کتاب شما را به استراتژی‌هایی که نیاز دارید مجهز می‌کند، مهم نیست از چه زبان برنامه‌نویسی استفاده می‌کنید.   
Adam Tornhill   
(۲۱۸ صفحه)   
ISBN: 9781680500387   
$36   
[https://pragprog.com/book/atcrime](https://pragprog.com/book/atcrime)   
  
سامورایی چابک   
اینجا سه حقیقت ساده درباره توسعه نرم‌افزار وجود دارد:   
۱. شما نمی‌توانید تمام نیازمندی‌ها را از ابتدا جمع‌آوری کنید.   
۲. نیازمندی‌هایی که جمع‌آوری می‌کنید تغییر خواهند کرد.   
۳. همیشه کار بیشتری برای انجام دادن نسبت به زمان و پول موجود وجود دارد.   
این‌ها حقایق زندگی هستند. اما شما می‌توانید با این حقایق (و بیشتر از آن) کنار بیایید، با تبدیل شدن به یک حرفه‌ای در زمینه تحویل نرم‌افزار، که قادر است سخت‌ترین پروژه‌های نرم‌افزاری و فشرده‌ترین زمان‌بندی‌های تحویل را با سهولت و ظرافت مدیریت کند. این عنوان به صورت کتاب صوتی نیز در دسترس است.   
Jonathan Rasmusson   
(۲۸۰ صفحه)   
ISBN: 9781934356586   
$34.95   
[https://pragprog.com/book/jtrap](https://pragprog.com/book/jtrap)]   
  
گذشته و حال   
برای دیدن اینکه به کجا می‌رویم، به یاد بیاورید که چگونه به اینجا رسیدیم و یاد بگیرید که چگونه رویکرد سالم‌تری به برنامه‌نویسی داشته باشید.   
آتش در دره   
در دهه ۱۹۷۰، در حالی که هم‌عصرانشان کامپیوتر را به‌عنوان ابزاری برای غیرانسانی‌سازی و سرکوب اعتراض می‌کردند، گروهی متنوع از ترک‌تحصیل‌کردگان دانشگاه، هیپی‌ها و علاقه‌مندان به الکترونیک درگیر چیزی بسیار زیرکانه‌تر بودند.

================================================================================

# Section 155

## Original Text:

Obsessed with the  
idea ofgetting computer power into their  
own hands, they launched from their garages  
ahobbyist movement that grew into anin-  
dustry, and ultimately asocial and techno -  
logical revolution. What they didw a s invent  
thepersonal computer: notjustanew device,  
butawatershed intherelationship between  
man and machine. This istheir story. Michael Swaine and Paul Freiberger  
(424 pages) ISBN :9781937785765. $34  
https://pragprog. com/book/fsfire  
The Healthy Programmer  
T okeep doing what yo u love, yo u need to  
maintain your own systems, notjusttheones  
yo u write code for. Regular exercise and  
proper nutrition help yo u learn, remember,  
concentrate, and becreative—skills critical  
todoing your jobwell. Learn how tochange  
your work habits, master exercises that make  
working atacomputer more comfortable,  
and develop aplan tokeep fit,healthy, and  
sharp foryears tocome. This book isintended only asaninformative  
guide forthose wishing toknow more about  
health issues. Innoway isthisbook intended to  
replace, countermand, orconflict with theadvice  
given toyoubyyour own healthcare provider  
including Physician, Nurse Practitioner, Physi -  
cian Assistant, Registered Dietician, andother  
licensed professionals. JoeKutner  
(254 pages) ISBN :9781937785314. $36  
https://pragprog. com/book/jkthp  
  
[CONTEXT FROM PREVIOUS PAGE: Past and Present  
T oseewhere we’re going, remember how wegothere, and learn how totake ahealthier  
approach toprogramming. Fire in the Valley  
Inthe1970s, while their contemporaries were  
protesting thecomputer asatool ofdehu -  
manization and oppression, amotley collec -  
tion ofcollege dropouts, hippies, and elec-  
tronics fanatics were engaged insomething  
much more subversive. Obsessed with the  
idea ofgetting computer power into their  
own hands, they launched from their garages  
ahobbyist movement that grew into anin-  
dustry, and ultimately asocial and techno -  
logical revolution.

## Translated Text:

شیفته‌ی ایده‌ی به دست آوردن قدرت کامپیوتر در دستان خود، آن‌ها از گاراژهایشان یک حرکت سرگرمی‌محور را آغاز کردند که به یک صنعت تبدیل شد و در نهایت به یک انقلاب اجتماعی و تکنولوژیکی انجامید. کاری که آن‌ها انجام دادند اختراع کامپیوتر شخصی بود: نه فقط یک دستگاه جدید، بلکه نقطه عطفی در رابطه‌ی بین انسان و ماشین. این داستان آن‌هاست.   
مایکل سواین و پاول فریبرگر   
(۴۲۴ صفحه)   
ISBN: 9781937785765   
$34   
https://pragprog.com/book/fsfire   
  
\*\*The Healthy Programmer\*\*   
برای اینکه بتوانید کاری را که دوست دارید ادامه دهید، باید سیستم‌های خودتان را حفظ کنید، نه فقط آن‌هایی که برایشان کدنویسی می‌کنید. ورزش منظم و تغذیه‌ی مناسب به شما کمک می‌کند یاد بگیرید، به خاطر بسپارید، تمرکز کنید و خلاق باشید—مهارت‌هایی که برای انجام خوب کارتان حیاتی هستند. یاد بگیرید چگونه عادات کاری خود را تغییر دهید، تمریناتی را بیاموزید که کار با کامپیوتر را راحت‌تر می‌کنند، و یک برنامه برای حفظ تناسب اندام، سلامتی و ذهنی تیز برای سال‌های آینده توسعه دهید.   
  
این کتاب تنها به عنوان یک راهنمای اطلاعاتی برای کسانی که می‌خواهند درباره مسائل سلامتی بیشتر بدانند، در نظر گرفته شده است. به هیچ وجه این کتاب قصد ندارد جایگزین، نقض یا با توصیه‌های ارائه‌شده توسط ارائه‌دهنده خدمات بهداشتی شما، شامل پزشک، پرستار، دستیار پزشک، متخصص تغذیه ثبت‌شده و سایر متخصصان دارای مجوز، در تضاد باشد.   
جو کاتنر   
(۲۵۴ صفحه)   
ISBN: 9781937785314   
$36   
https://pragprog.com/book/jkthp

================================================================================

# Section 156

## Original Text:

What they didw a s invent  
thepersonal computer: notjustanew device,  
butawatershed intherelationship between  
man and machine. This istheir story. Michael Swaine and Paul Freiberger  
(424 pages) ISBN :9781937785765. $34  
https://pragprog. com/book/fsfire  
The Healthy Programmer  
T okeep doing what yo u love, yo u need to  
maintain your own systems, notjusttheones  
yo u write code for. Regular exercise and  
proper nutrition help yo u learn, remember,  
concentrate, and becreative—skills critical  
todoing your jobwell. Learn how tochange  
your work habits, master exercises that make  
working atacomputer more comfortable,  
and develop aplan tokeep fit,healthy, and  
sharp foryears tocome. This book isintended only asaninformative  
guide forthose wishing toknow more about  
health issues. Innoway isthisbook intended to  
replace, countermand, orconflict with theadvice  
given toyoubyyour own healthcare provider  
including Physician, Nurse Practitioner, Physi -  
cian Assistant, Registered Dietician, andother  
licensed professionals. JoeKutner  
(254 pages) ISBN :9781937785314. $36  
https://pragprog. com/book/jkthp]  
  
Redesign Your Career  
Ready tokick your career uptothenext level. Time torewire your brain and then rein-  
vigorate your jobitself. Pragmatic Thinking and Learning  
Software development happens inyour  
head. Not inaneditor, IDE, ordesign tool. You’re well educated onhow towork with  
software and hardware, butwhat about  
wetware —our own brains. Learning new  
skills and new technology iscritical toyour  
career, and it’sallinyour head. Inthisbook byAndy Hunt, you’ll learn how  
ourbrains arewired, andhow totake advan -  
tage ofyour brain’s architecture. You’ll learn  
new tricks and tipstolearn more, faster, and  
retain more ofwhat yo u learn. Y o u need apragmatic approach tothinking  
and learning. Y o u need toRefactor Your  
Wetware. Printed infullcolor. Andy Hunt  
(252 pages) ISBN :9781934356050. $34. 95  
https://pragprog.

## Translated Text:

آن‌ها چیزی که اختراع کردند کامپیوتر شخصی بود: نه فقط یک دستگاه جدید، بلکه نقطه عطفی در رابطه بین انسان و ماشین. این داستان آن‌هاست.   
مایکل سوین و پاول فریبرگر   
(۴۲۴ صفحه)   
ISBN: 9781937785765   
$34   
[https://pragprog.com/book/fsfire](https://pragprog.com/book/fsfire)   
  
### برنامه‌نویس سالم   
برای اینکه بتوانید کاری که دوست دارید را ادامه دهید، باید سیستم‌های خودتان را حفظ کنید، نه فقط آن‌هایی که برایشان کد می‌نویسید. ورزش منظم و تغذیه مناسب به شما کمک می‌کند یاد بگیرید، به خاطر بسپارید، تمرکز کنید و خلاق باشید—مهارت‌هایی که برای انجام خوب کارتان حیاتی هستند.   
یاد بگیرید که چگونه عادات کاری خود را تغییر دهید، تمریناتی را بیاموزید که کار با کامپیوتر را راحت‌تر می‌کنند، و یک برنامه برای حفظ تناسب اندام، سلامتی و تیزهوشی برای سال‌های آینده توسعه دهید.   
  
این کتاب تنها به عنوان یک راهنمای اطلاعاتی برای کسانی که می‌خواهند درباره مسائل سلامتی بیشتر بدانند، در نظر گرفته شده است. به هیچ وجه این کتاب قصد ندارد جایگزین، نقض یا در تضاد با توصیه‌هایی باشد که از ارائه‌دهنده خدمات سلامت خود، شامل پزشک، پرستار حرفه‌ای، دستیار پزشک، متخصص تغذیه ثبت‌شده و دیگر متخصصان دارای مجوز دریافت می‌کنید.   
جو کاتنر   
(۲۵۴ صفحه)   
ISBN: 9781937785314   
$36   
[https://pragprog.com/book/jkthp](https://pragprog.com/book/jkthp)   
  
### طراحی مجدد شغل شما   
آماده‌اید که شغل خود را به سطح بعدی برسانید؟ وقت آن است که مغز خود را بازسازی کنید و سپس شغل خود را دوباره احیا کنید.   
  
### تفکر و یادگیری عمل‌گرایانه   
توسعه نرم‌افزار در ذهن شما اتفاق می‌افتد. نه در یک ویرایشگر، IDE، یا ابزار طراحی. شما به خوبی آموزش دیده‌اید که چگونه با نرم‌افزار و سخت‌افزار کار کنید، اما درباره "wetware" — مغز خودمان چطور؟ یادگیری مهارت‌های جدید و فناوری‌های جدید برای شغل شما حیاتی است، و همه این‌ها در ذهن شماست.   
  
در این کتاب نوشته اندی هانت، شما یاد می‌گیرید که مغز ما چگونه سیم‌کشی شده است و چگونه از معماری مغز خود بهره‌برداری کنید. شما ترفندها و نکات جدیدی یاد می‌گیرید تا بیشتر، سریع‌تر یاد بگیرید و آنچه را که یاد می‌گیرید بهتر حفظ کنید. شما به یک رویکرد عمل‌گرایانه برای تفکر و یادگیری نیاز دارید. شما باید "Wetware" خود را Refactor کنید.   
چاپ شده به صورت تمام رنگی.   
اندی هانت   
(۲۵۲ صفحه)   
ISBN: 9781934356050   
$34.95   
[https://pragprog.com](https://pragprog.com)

================================================================================

# Section 157

## Original Text:

com/book/ahptl  
The Passionate Programmer (2nd edition)  
This book isabout creating aremarkable ca-  
reer insoftware development. Inmost cases,  
remarkable careers don’t come bychance. They require thought, intention, action, and  
awillingness tochange course when you’ve  
made mistakes. Most ofushave been stum -  
bling around letting ourcareers take us  
where they may. It’stime totake control. This revised andupdated second edition lays  
outastrategy forplanning and creating a  
radically successful lifeinsoftware develop -  
ment. Chad Fowler  
(232 pages) ISBN :9781934356340. $23. 95  
https://pragprog. com/book/cfcar2  
  
[CONTEXT FROM PREVIOUS PAGE: Redesign Your Career  
Ready tokick your career uptothenext level. Time torewire your brain and then rein-  
vigorate your jobitself. Pragmatic Thinking and Learning  
Software development happens inyour  
head. Not inaneditor, IDE, ordesign tool. You’re well educated onhow towork with  
software and hardware, butwhat about  
wetware —our own brains. Learning new  
skills and new technology iscritical toyour  
career, and it’sallinyour head. Inthisbook byAndy Hunt, you’ll learn how  
ourbrains arewired, andhow totake advan -  
tage ofyour brain’s architecture. You’ll learn  
new tricks and tipstolearn more, faster, and  
retain more ofwhat yo u learn. Y o u need apragmatic approach tothinking  
and learning. Y o u need toRefactor Your  
Wetware. Printed infullcolor. Andy Hunt  
(252 pages) ISBN :9781934356050. $34. 95  
https://pragprog. com/book/ahptl  
The Passionate Programmer (2nd edition)  
This book isabout creating aremarkable ca-  
reer insoftware development. Inmost cases,  
remarkable careers don’t come bychance. They require thought, intention, action, and  
awillingness tochange course when you’ve  
made mistakes. Most ofushave been stum -  
bling around letting ourcareers take us  
where they may. It’stime totake control. This revised andupdated second edition lays  
outastrategy forplanning and creating a  
radically successful lifeinsoftware develop -  
ment.

## Translated Text:

com/book/ahptl   
The Passionate Programmer (ویرایش دوم)   
این کتاب درباره‌ی ایجاد یک حرفه‌ی چشمگیر در حوزه‌ی توسعه‌ی نرم‌افزار است. در بیشتر موارد، حرفه‌های چشمگیر به‌طور تصادفی به دست نمی‌آیند. آن‌ها نیازمند تفکر، قصد، عمل و تمایل به تغییر مسیر در مواقعی هستند که اشتباه کرده‌اید. اکثر ما به‌طور سرگردان اجازه داده‌ایم که حرفه‌مان ما را به هر کجا که بخواهد ببرد. وقت آن است که کنترل را به دست بگیریم. این ویرایش بازبینی‌شده و به‌روزرسانی‌شده، استراتژی‌ای را برای برنامه‌ریزی و ایجاد یک زندگی کاملاً موفق در توسعه‌ی نرم‌افزار ارائه می‌دهد.   
چاد فاولر   
(232 صفحه) ISBN: 9781934356340. $23.95   
[https://pragprog.com/book/cfcar2](https://pragprog.com/book/cfcar2)   
  
[متن از صفحه‌ی قبلی: طراحی مجدد حرفه‌ی شما   
آماده‌اید حرفه‌ی خود را به سطح بعدی ارتقا دهید؟ وقت آن است که مغزتان را بازسازی کنید و سپس شغل‌تان را دوباره احیا کنید.   
\*\*Pragmatic Thinking and Learning\*\*   
توسعه‌ی نرم‌افزار در ذهن شما اتفاق می‌افتد، نه در یک ویرایشگر، IDE یا ابزار طراحی. شما به‌خوبی آموزش دیده‌اید که چگونه با نرم‌افزار و سخت‌افزار کار کنید، اما درباره‌ی wetware — مغز خودمان چطور؟ یادگیری مهارت‌های جدید و فناوری‌های جدید برای حرفه‌ی شما حیاتی است و همه‌چیز در ذهن شماست.   
در این کتاب از اندی هانت، یاد خواهید گرفت که مغز ما چگونه طراحی شده است و چگونه از معماری مغزتان بهره‌برداری کنید. شما ترفندها و نکات جدیدی را یاد خواهید گرفت تا بیشتر، سریع‌تر یاد بگیرید و آنچه را که یاد می‌گیرید بهتر حفظ کنید.   
شما به یک رویکرد عمل‌گرایانه برای تفکر و یادگیری نیاز دارید. شما باید Wetware خود را Refactor کنید.   
چاپ شده به‌صورت تمام‌رنگی.   
اندی هانت   
(252 صفحه) ISBN: 9781934356050. $34.95   
[https://pragprog.com/book/ahptl](https://pragprog.com/book/ahptl)   
  
The Passionate Programmer (ویرایش دوم)   
این کتاب درباره‌ی ایجاد یک حرفه‌ی چشمگیر در حوزه‌ی توسعه‌ی نرم‌افزار است. در بیشتر موارد، حرفه‌های چشمگیر به‌طور تصادفی به دست نمی‌آیند. آن‌ها نیازمند تفکر، قصد، عمل و تمایل به تغییر مسیر در مواقعی هستند که اشتباه کرده‌اید. اکثر ما به‌طور سرگردان اجازه داده‌ایم که حرفه‌مان ما را به هر کجا که بخواهد ببرد. وقت آن است که کنترل را به دست بگیریم. این ویرایش بازبینی‌شده و به‌روزرسانی‌شده، استراتژی‌ای را برای برنامه‌ریزی و ایجاد یک زندگی کاملاً موفق در توسعه‌ی نرم‌افزار ارائه می‌دهد.]

================================================================================

# Section 158

## Original Text:

Chad Fowler  
(232 pages) ISBN :9781934356340. $23. 95  
https://pragprog. com/book/cfcar2]  
  
Make it Work  
Doretrospectives theright w a y, and seehow togetnew ideas accepted. Agile Retrospectives  
Seehow tomine theexperience ofyour  
software development team continually  
throughout thelifeoftheproject. The tools  
andrecipes inthisbook willhelp y o uuncov -  
erand solve hidden (and not-so-hidden)  
problems with your technology, your  
methodology, and those difficult “people is-  
sues” onyour team. Esther Derby and Diana Larsen, Foreword  
byKen Schwaber  
(176 pages) ISBN :9780977616640. $29. 95  
https://pragprog. com/book/dlret  
Driving Technical Change  
Ify o uwork with people, y o uneed thisbook. Learn toread co-workers’ andusers’ patterns  
ofresistance and dismantle their objections. With these techniques andstrategies y o ucan  
master theartofevangelizing and help your  
organization adopt your solutions. Terrence Rya n  
(146 pages) ISBN :9781934356609. $32. 95  
https://pragprog. com/book/trevan  
  
[CONTEXT FROM PREVIOUS PAGE: Make it Work  
Doretrospectives theright w a y, and seehow togetnew ideas accepted. Agile Retrospectives  
Seehow tomine theexperience ofyour  
software development team continually  
throughout thelifeoftheproject. The tools  
andrecipes inthisbook willhelp y o uuncov -  
erand solve hidden (and not-so-hidden)  
problems with your technology, your  
methodology, and those difficult “people is-  
sues” onyour team. Esther Derby and Diana Larsen, Foreword  
byKen Schwaber  
(176 pages) ISBN :9780977616640. $29. 95  
https://pragprog. com/book/dlret  
Driving Technical Change  
Ify o uwork with people, y o uneed thisbook. Learn toread co-workers’ andusers’ patterns  
ofresistance and dismantle their objections. With these techniques andstrategies y o ucan  
master theartofevangelizing and help your  
organization adopt your solutions. Terrence Rya n  
(146 pages) ISBN :9781934356609. $32. 95  
https://pragprog.

## Translated Text:

چاد فاولر   
(232 صفحه) ISBN: 9781934356340. $23.95   
[https://pragprog.com/book/cfcar2](https://pragprog.com/book/cfcar2)   
  
\*\*Make it Work\*\*   
بازنگری‌ها را به روش درست انجام دهید و ببینید چگونه ایده‌های جدید پذیرفته می‌شوند.   
  
\*\*Agile Retrospectives\*\*   
ببینید چگونه می‌توانید از تجربیات تیم توسعه نرم‌افزار خود به طور مداوم در طول عمر پروژه بهره‌برداری کنید. ابزارها و راهکارهای موجود در این کتاب به شما کمک می‌کنند مشکلات پنهان (و نه چندان پنهان) در تکنولوژی، متدولوژی و حتی مسائل دشوار مربوط به "افراد" در تیم خود را کشف و حل کنید.   
\*\*استر دربی و دایانا لارسن، مقدمه از کن شوابر\*\*   
(176 صفحه) ISBN: 9780977616640. $29.95   
[https://pragprog.com/book/dlret](https://pragprog.com/book/dlret)   
  
\*\*Driving Technical Change\*\*   
اگر با افراد کار می‌کنید، به این کتاب نیاز دارید. یاد بگیرید الگوهای مقاومت همکاران و کاربران را بخوانید و اعتراضات آن‌ها را برطرف کنید. با استفاده از این تکنیک‌ها و استراتژی‌ها، می‌توانید هنر ترویج ایده‌ها را بیاموزید و به سازمان خود کمک کنید تا راهکارهای شما را بپذیرد.   
\*\*ترنس رایان\*\*   
(146 صفحه) ISBN: 9781934356609. $32.95   
[https://pragprog.com/book/trevan](https://pragprog.com/book/trevan)

================================================================================

# Section 159

## Original Text:

com/book/trevan]  
  
The Pragmatic Bookshelf  
The Pragmatic Bookshelf features books written bydevelopers fordevelopers. The titles  
continue thewell-known Pragmatic Programmer style and continue togarner awards  
andrave reviews. Asdevelopment gets more andmore difficult, thePragmatic Program -  
mers will bethere with more titles and products tohelp yo u stay ontopofyour game. Visit Us Online  
This Book’s Home Page  
https://pragprog. com/book/rjnsd  
Source code from thisbook, errata, and other resources. Come give usfeedback, too. Register forUpdates  
https://pragprog. com/updates  
Benotified when updates and new books become available. JointheCommunity  
https://pragprog. com/community  
Read ourweblogs, join ouronline discussions, participate inourmailing list, interact  
with ourwiki, and benefit from theexperience ofother Pragmatic Programmers. New andNoteworthy  
https://pragprog. com/news  
Check outthelatest pragmatic developments, new titles and other offerings. Buy the Book  
Ifyo u liked thiseBook, perhaps you’d liketohave apaper copy ofthebook. It’savailable  
forpurchase atourstore: https://pragprog. com/book/rjnsd  
Contact Us  
https://pragprog. com/catalog Online Orders:  
support@pragprog. com Customer Service:  
translations@pragprog. com International Rights:  
academic@pragprog. com Academic Use:  
http://write-for-us. pragprog. com Write forUs:  
+1800-699-7764 OrCall:.

## Translated Text:

com/book/trevan]  
  
The Pragmatic Bookshelf   
The Pragmatic Bookshelf شامل کتاب‌هایی است که توسط توسعه‌دهندگان برای توسعه‌دهندگان نوشته شده‌اند. این عناوین به سبک معروف Pragmatic Programmer ادامه می‌دهند و همچنان جوایز و نقدهای مثبت بسیاری دریافت می‌کنند. با سخت‌تر شدن توسعه، Pragmatic Programmers با عناوین و محصولات بیشتری در کنار شما خواهند بود تا به شما کمک کنند در اوج کار خود باقی بمانید.   
  
\*\*Visit Us Online\*\*   
\*\*This Book’s Home Page\*\*   
https://pragprog.com/book/rjnsd   
کد منبع این کتاب، خطاها و منابع دیگر. نظرات خود را نیز با ما به اشتراک بگذارید.   
  
\*\*Register for Updates\*\*   
https://pragprog.com/updates   
از به‌روزرسانی‌ها و کتاب‌های جدید مطلع شوید.   
  
\*\*Join the Community\*\*   
https://pragprog.com/community   
وبلاگ‌های ما را بخوانید، در بحث‌های آنلاین ما شرکت کنید، در لیست ایمیل ما عضو شوید، با ویکی ما تعامل داشته باشید و از تجربیات دیگر Pragmatic Programmers بهره‌مند شوید.   
  
\*\*New and Noteworthy\*\*   
https://pragprog.com/news   
آخرین پیشرفت‌های پراگماتیک، عناوین جدید و دیگر پیشنهادات را بررسی کنید.   
  
\*\*Buy the Book\*\*   
اگر از این eBook خوشتان آمده است، شاید بخواهید یک نسخه کاغذی از کتاب داشته باشید. این کتاب برای خرید در فروشگاه ما موجود است:   
https://pragprog.com/book/rjnsd   
  
\*\*Contact Us\*\*   
https://pragprog.com/catalog   
\*\*Online Orders:\*\*   
support@pragprog.com   
\*\*Customer Service:\*\*   
translations@pragprog.com   
\*\*International Rights:\*\*   
academic@pragprog.com   
\*\*Academic Use:\*\*   
http://write-for-us.pragprog.com   
\*\*Write for Us:\*\*   
+1800-699-7764   
\*\*Or Call:\*\*

================================================================================