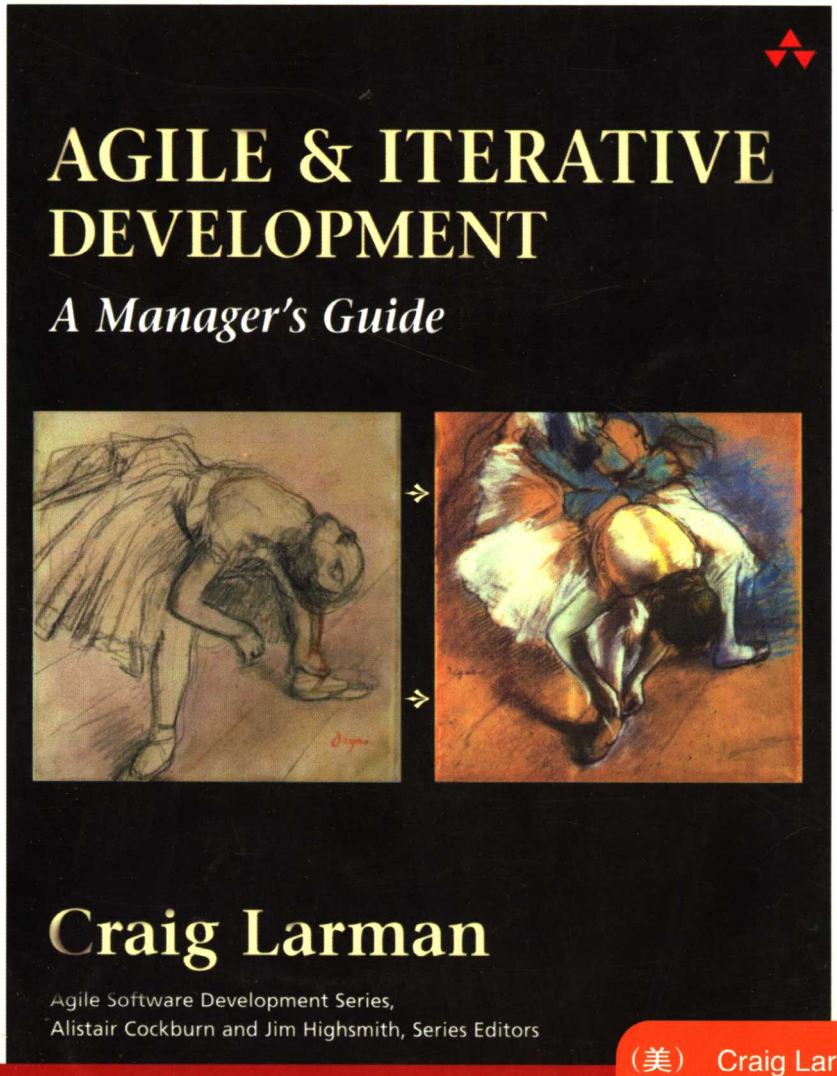
经 典 原 版 书 库

敏捷迭代开发
管理者指南

（英文版）



（美） Craig Larman 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

敏捷迭代开发
管理者指南
（英文版）

Agile & Iterative Development
A Manager's Guide

（美） Craig Larman 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2006-0528

图书在版编目（CIP）数据

敏捷迭代开发：管理者指南（英文版）/（美）拉曼（Larman, C.）著. －北京：机械工业出版社，2006.3

（经典原版书库）

书名原文：Agile & Iterative Development: A Manager's Guide

ISBN 7-111-18483-1

Ⅰ.敏… Ⅱ.拉… Ⅲ.迭代法－应用－软件开发－英文 Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2006）第008912号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京京北制版厂印刷 · 新华书店北京发行所发行

2006年3月第1版第1次印刷

718mm × 1020mm 1/16 · 22.25印张

印数：0 001 - 2 500册

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔

滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下:

电子邮件: hzjsj@hzbook.com
联系电话: (010) 68995264
联系地址: 北京市西城区百万庄南街1号
邮政编码: 100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

For Julie, Haley, and Hannah

with love and thanks

PREFACE

Thank you for reading this book! My sincere aim is that it is useful—quality information, quickly understood.

Some related articles and pointers are at www.craiglarman.com. Please contact me with questions at craig@craiglarman.com.

Typographic Conventions

This is a basic *point of emphasis*. Book titles are also italicized.

This is a ***noticeable point of emphasis*** I wish to make easy for you to see. Usually, so you can skim pages and pick out key ideas.

This is a **new term** in a sentence.

This is a reference [Bob67] in the bibliography.

About the Author

Craig Larman serves as Chief Scientist for Valtech, an international consulting and skills transfer company with divisions in Europe, Asia, and North America. He also works globally as an independent consultant, coach, and speaker.

Craig is the author of *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, the world's best-selling text on OOA/D and iterative development, translated to many languages and used worldwide in industry and colleges.

After a failed career as a wandering street musician, he built systems in APL, PL/I, and 4GLs in the 1970s. Starting in the early 1980s—after a full recovery—he became interested in artificial intelligence (having little of his own) and knowledge representation, and built knowledge systems with Lisp machines, Lisp, Prolog, and Smalltalk. He has played bad lead guitar in his very part-

time band, the *Changing Requirements* (it used to be called the *Requirements*, but some band members changed...).

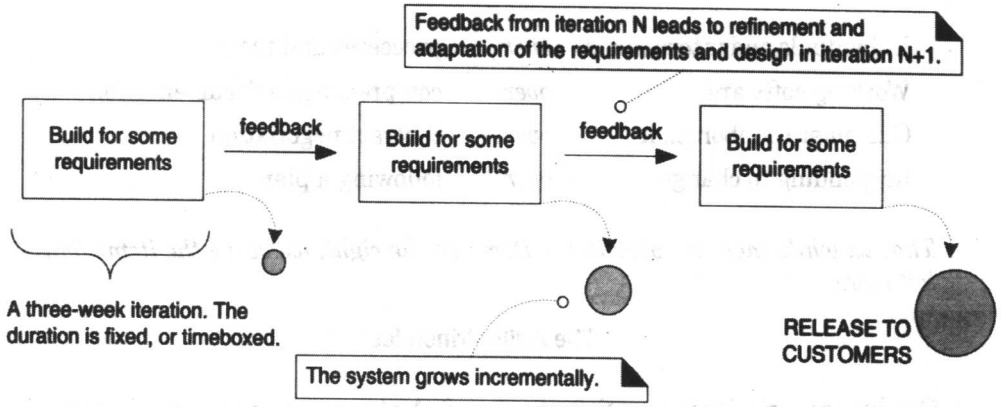
Craig has a B.S. and M.S. in computer science from beautiful Simon Fraser University in Vancouver, Canada.

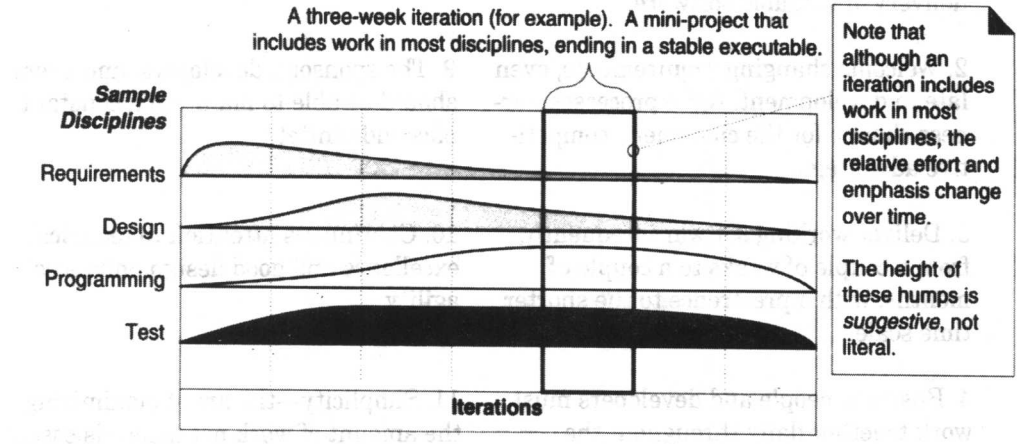
Acknowledgments

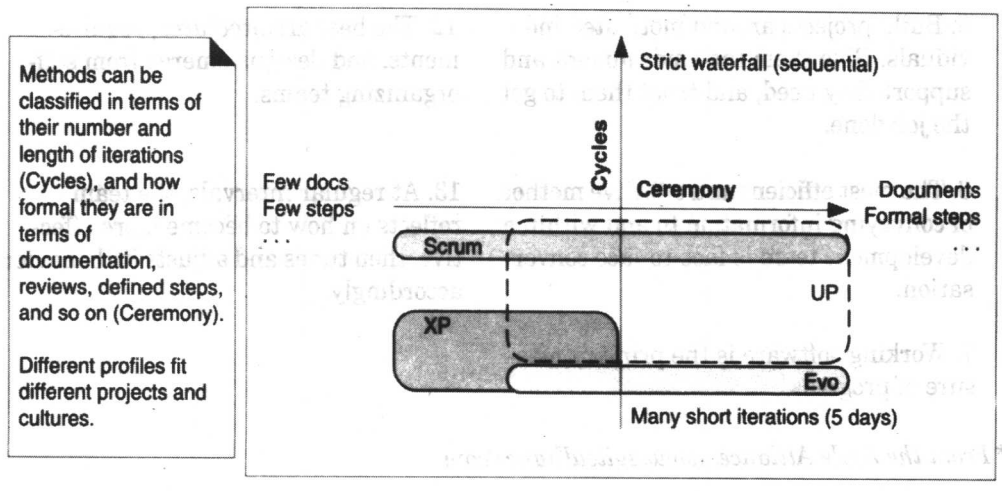
A special thanks to my friends and colleagues at Valtech, world-class iterative developers, especially Tim Snyder.

Many thanks to the reviewers, including Alistair Cockburn, Claudia Frers, Tom Gilb, Jim Highsmith, Ron Jeffries, Philippe Kruchten, Niels Maloteaux, Gary Pollice, Ken Schwaber, and Jeff Sutherland.

Thanks to Paul Petralia and Patti Guerrieri for shepherding.


Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
Build for some requirements
feedback
Build for some requirements
feedback
Build for some requirements
A three-week iteration. The duration is fixed, or timeboxed.
RELEASE TO CUSTOMERS
The system grows incrementally.


A three-week iteration (for example). A mini-project that includes work in most disciplines, ending in a stable executable.
Sample Disciplines
Requirements
Design
Programming
Test
Iterations
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.
The height of these humps is suggestive, not literal.


Methods can be classified in terms of their number and length of iterations (Cycles), and how formal they are in terms of documentation, reviews, defined steps, and so on (Ceremony).
Different profiles fit different projects and cultures.
Strict waterfall (sequential)
Cycles
Few docs
Few steps
...
Ceremony
Documents
Formal steps
...
Scrum
UP
XP
Evo
Many short iterations (5 days)

The Agile Manifesto*

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

That is, while there is value in the items on the right, we value the items on the left more.

The Agile Principles*

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development.

9. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

10. Continuous attention to technical excellence and good design enhances agility.

11. Simplicity—the art of maximizing the amount of work not done—is essential.

12. The best architectures, requirements, and designs emerge from self-organizing teams.

13. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

* *From the Agile Alliance. www.agilealliance.com*

TABLE OF CONTENTS

Chapter

1

INTRODUCTION

Logic is the art of going wrong with confidence.
—Joseph Wood Krutch

OVERVIEW

- What's in this book?
- Predictable versus new product development.

What value will you get from studying this book, an introduction to iterative and agile methods?

First, you will know the key practices of four noteworthy methods, **Scrum**, **Extreme Programming** (**XP**), the **Unified Process** (**UP**), and **Evo** (one of the original iterative methods). This is a "Cliffs Notes" summary, each chapter has something useful to you as a manager, developer, or student of development methods.

Scrum p. 109
XP p. 137
UP p. 173
Evo p. 211

Second, your learning curve will be shortened, as this is a distilled learning aid. The four method chapters have the same structure, to speed comprehension and compare-contrast. There's a FAQ chapter, a "tips" chapter of common practices, and plenty of margin pointers to related pages—paper hyperlinks.

FAQ p. 297
tips p. 247

Third, you will know motivation and evidence. Some organizations accept the value of iterative development, but others are still reluctant. If you need to make a case for an iterative project experiment, you will find in this book the key reasons, research, examples of large projects, standards-body acceptance, a business case, and promotion by well-known thought leaders through the

motivation p. 49
evidence p. 63

decades. The research and history sections are also of value to students of software engineering methods.

> Note that agile methods are a subset of iterative methods; this book covers both types.

The chapters may be read in any order; the big picture is this:

1. Introduction, and predictable vs. inventive development.	5–6. Motivation and evidence chapters for iterative and agile methods; useful for some.
2. Basic iterative and evolutionary method practices.	7–10. Four method summaries on Scrum, XP, UP, and Evo. Note: practices can be mixed.
3. Summary of agile principles and methods.	11. A tips chapter that expands on some of the method practices, plus others.
4. An agile project story to pull some ideas together.	12. A frequently asked questions (FAQ) chapter.

Finally, *people trump process*. Every process book should probably include this standard disclaimer:

> *Process is only a second-order effect.*[1] *The unique people, their feelings, qualities, and communication are more influential.*
>
> *Some problems are just hard, some people are just difficult. These methods are not salvation.*

1. A quote from the agile methodologist Alistair Cockburn.

SOFTWARE IS NEW PRODUCT DEVELOPMENT

Consider building mobile phones on an assembly line: It *is* possible to unambiguously define the specifications and construction steps. After building some phones and measuring things, it is possible to reliably estimate and schedule the building of future phones.

A different problem: Build a custom house. The owner wants to use new environmentally friendly materials and methods, but isn't exactly sure what they want, and is going to change or clarify their decisions as they see the house, costs, and weeks unfold.

At one end of the spectrum, such as manufacturing phones, there are problems with low degrees of novelty or change, and high rates of repeated identical or near-identical creation—*mass manufacturing* or *predictable manufacturing*.

At the other end, there are problems with high degrees of novelty, creativity, and change, and no previous identical cases from which to derive estimates or schedules. This is the realm of *new product development* or *inventive projects*.

The development process, management values, planning and estimation models appropriately associated with these two domains are different (Table 1.1).

Table 1.1 predictable vs. inventive projects

Predictable Manufacturing	New Product Development
It is possible to first complete specifications, and then build.	Rarely possible to create up-front unchanging and detailed specs.

Predictable Manufacturing	New Product Development
Near the start, one can reliably estimate effort and cost.	Near the beginning, it is not possible. As empirical data emerge, it becomes increasingly possible to plan and estimate.
It is possible to identify, define, schedule, and order all the detailed activities.	Near the beginning, it is not possible. Adaptive steps driven by build–feedback cycles are required.
Adaptation to unpredictable change is not the norm, and change-rates are relatively low.	Creative adaptation to unpredictable change is the norm. Change rates are high.

Of course, the point is,

> *Most software is not a predictable or mass manufacturing problem. Software development is new product development.*

Plus, many projects use new and buggy technologies that exacerbate the degree of novelty and unpredictability. Note also it is a new product for the inexperienced even if it has been done before.

Since predictable manufacturing is the wrong paradigm for software, practices and values rooted in it are not helpful.

> This mismatch lies at the heart of many of the challenges associated with traditional approaches to running a software project.
>
> A "waterfall" lifecycle, big up-front specifications, estimates, and speculative plans applicable to predictable manufacturing have been misapplied to software projects, a domain of inventive, high-change, high-novelty work.