经 典 原 版 书 库

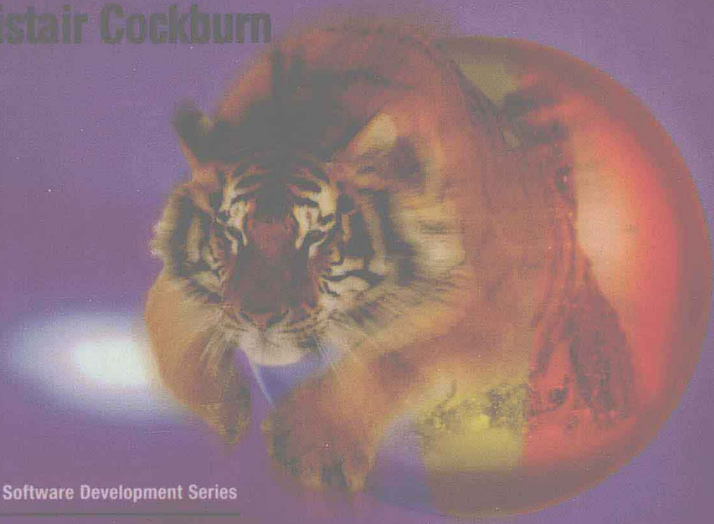
敏捷软件开发

（英文版 · 第2版）

Agile Software Development SECOND EDITION

The Cooperative Game

Alistair Cockburn



Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors

（美）Alistair Cockburn 著

机械工业出版社
China Machine Press

经　典　原　版　书　库

敏捷软件开发

（英文版·第2版）

Agile Software Development
The Cooperative Game

(Second Edition)

（美）Alistair Cockburn 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2007-2431

图书在版编目（CIP）数据

敏捷软件开发（英文版·第2版）/（美）科伯恩（Cockburn，A.）著. －北京：机械工业出版社，2007.6

（经典原版书库）

书名原文：Agile Software Development：The Cooperative Game，Second Edition

ISBN 978-7-111-21457-1

I. 敏… II. 科… III. 软件开发－英文 IV. TP311.52

中国版本图书馆CIP数据核字（2007）第064662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

2007年6月第1版第1次印刷

170mm×242mm · 31.25印张

定价：59.00元

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人一个新的阶段。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			

PREFACE

Is software development an art, a craft, science, engineering, or something else entirely? Does it even matter?

Yes, it does matter, and it matters to you. Your actions and their results will differ depending on which of those is more correct.

The main thing is this: You want your software out soon and defect free, but more than that, you need a way to examine how your team is doing along the way.

PURPOSE

It is time to reexamine the notions underlying software development.

The trouble is that as we look at projects, *what* we notice is constrained by what we know to notice. We learn to distinguish distinct and separable things in the extremely rich stream of experience flowing over us, and we pull those things out of the stream for examination. To the extent that we lack various key distinctions, we overlook things that are right in front of us.

We anchor the distinctions in our memories with words and use those words to reflect on our experiences. To the extent that we lack words to anchor the distinctions, we lack the ability to pull our memories into our conversations and the ability to construct meaningful strategies for dealing with the future.

In other words, to reexamine the notions that underlie software development, we have to reconsider the distinctions that we use to slice up our experience and the words we use to anchor our memories.

This is, of course, a tall order for any book. It means that some of the earlier parts of this book will be rather abstract. I see no way around it, though.

The last time people constructed a vocabulary for software development was in the late 1960s, when they coined the phrase *software engineering*, as both a wish and a direction for the future.

It is significant that at the same time the programming-should-be-engineering pronouncement was made, Gerald Weinberg was writing *The Psychology of Computer Programming*. In that book, software development doesn't look very much like an engineering discipline at all. It appears to be something very human-centric and communication-centric. Of the two, Weinberg's observations match what people have reported in the succeeding 30 years, and *software engineering* remains a wishful term.

In this book, I will

- Build distinctions and vocabulary for talking about software development
- Use that vocabulary to examine and anchor critical aspects of software projects that have been pushed to the sidelines too often
- Work through the ideas and principles of methodologies as "rules of behavior"
- Merge our need for these rules of behavior with the idea that each project is unique, and derive effective and self-evolving rules

I hope that after reading this book, you will be able to use the new vocabulary to look around at your project, notice things you didn't notice before, and express those observations. As you gain facility, you should be able to

- Discuss Extreme Programming, the Capability Maturity Model, the Personal Software Process, or your favorite process
- Determine when each process is more or less applicable
- Understand people who have differing opinions, abilities, and experience

Audience

Each person coming to this book does so with a different experience level, reading style, and role. Here's how you might read the book to use it to your greatest advantage: by experience, by reading style, or by role.

By Experience

This book is written for the more experienced audience. The book does not contain procedures to follow to develop software; in fact, core to the book is the concept that every technique has limitations. Therefore, it is impossible to name one best and correct way to develop software. Ideally, the book helps you reach that understanding and then leads you to constructive ideas about how to deal with this real-world situation.

If you are an intermediate practitioner who has experience with software-development projects, and if you are now looking for the boundaries for the rules you have learned, you will find the following topics most helpful:

- What sorts of methodologies fit what sorts of projects
- Indices for selecting the appropriate methodology category for a project
- The principles behind agile methodologies

Being an intermediate practitioner, you will recognize that you must add your own judgement when applying these ideas.

If you are an advanced practitioner, you already know that all recommendations vary in applicability. You may be looking for words to help you express that. You will find those words where the following topics are presented:

- Managing the incompleteness of communication
- Continuous methodology reinvention
- The manifesto for agile software development

A few topics should be new even to advanced software developers: the vocabulary for describing methodologies and the technique for just-in-time methodology tuning.

By Reading Style

The earlier chapters are more abstract than the later chapters.

If you enjoy abstract material, read the book from beginning to end, watching the play of abstract topics to see the resolution of the impossible questions through the course of the book.

If you want concrete materials in your hands as quickly as possible, you may want to skip over the early chapters on the first read and start with Chapter 4, "Methodologies." Return to the sections about "Cooperative Games" and "Convection Currents of Information" to get the key parts of the new vocabulary. Dip into the introduction and the chapters about individuals and teams to fill in the gaps.

By Role

People who sponsor software development can get from this book an understanding of how various organizational, behavioral, and funding structures affect the rate at which they receive value from their development teams. Project sponsors may pay less attention to the details of methodology construction than people who are directly involved in the projects. They should still understand the consequences of certain sorts of methodology decisions.

Team leads and project managers can see how seating, teaming, and individuality affect their project's outcome. They can also learn what sorts of interventions are more likely to have better or worse consequences. They will need to understand the construction and consequences of their methodology and how to evolve their methodology—making it as light as possible, but still sufficient.

Process and methodology designers can examine and argue with my choice of terms and principles for methodology design. The ensuing discussions should prove useful for the field.

Software developers should come to know this material simply as part of being in the profession. In the normal progression from newcomers to leaders, they will have to notice what works and doesn't work on their projects. They will also have to learn how to adjust their environment to become more effective. "Our methodology" really means "the conventions we follow around here," and so it becomes every professional's responsibility to understand the basics of methodology construction.

Organization of the Book

The book is designed to set up two nearly impossible questions at the beginning and derive answers for those questions by the end of the book:

- If communication is fundamentally impossible, how can people on a project manage to do it?
- If all people and all projects are different, how can we create any rules for productive projects?

To achieve that design, I wrote the book a bit in the "whodunit" style of a mystery. I start with the broadest and most philosophical discussions: "What is communication?" and "What is software development?"

The discussion moves through still fairly abstract topics such as "What are the characteristics of a human?" and "What affects the movement of ideas within a team?"

Eventually, it gets into more concrete territory with "What are the elements and principles of methodologies?" This is a good place for you to start if you are after concrete material early on.

Finally, the discussion gets to the most concrete matter: "What does a light, sufficient, self-evolving methodology look like?" and "How does a group create a custom and agile methodology in time to do the project any good?"

The two appendixes contain supporting material. The first contains the "Agile Software Development Manifesto," signed by 17 very experienced software developers and methodologists.

The second appendix contains extracts from three pieces of writing that are not as widely read as they should be. I include them because they are core to the topics described in the book.

HERITAGE OF THE IDEAS IN THIS BOOK

The ideas in this book are based on 25 years of development experience and 10 years of investigating projects directly.

The IBM Consulting Group asked me to design its first object-oriented methodology in 1991. I looked rather helplessly at the conflicting "methodology" books at the time. My boss, Kathy Ulisse, and I decided that I should debrief project teams to better understand how they really worked. What an eye-opener! The words they used had almost no overlap with the words in the books.

The interviews keep being so valuable that I still visit projects with sufficiently interesting success stories to find out what they encountered, learned, and recommend. The crucial question I ask before the interview is, "And would you like to work the same way again?" When people describe their experiences in words that don't fit my vocabulary, it indicates new areas in which I lack distinctions and words.

The reason for writing this book now is that the words and distinctions finally are correlating with descriptions of project life and project results. They are proving more valuable for diagnosis and intervention than any of the tools that I used previously.

The ideas in this book have been through dozens of development teams, eight methodology designs, and a number of successful projects on which I participated.

AGILITY

I am not the only person who is using these ideas:

- Kent Beck and Ward Cunningham worked through the late 1980s on what became called *Extreme Programming* (XP) in the late 1990s.
- Jim Highsmith studied the language and business use of complex adaptive systems in the mid-1990s and wrote about the application of that language to software development in his *Adaptive Software Development*.
- Ken Schwaber and Jeff Sutherland were constructing the Scrum method of development at about the same time, and many project leaders made similar attempts to describe similar ideas through the same years.

When a group of us met in February 2001 to discuss our differences and similarities, we found we had a surprising number of things in common. We selected the word *agile* to describe our intent and wrote the Agile Software Development Manifesto (Appendix A).

We are still formulating the principles that we share and are finding many other people who could have been at that meeting if they had known about it or if their schedules had permitted their presence.

Core to *agile* software development is the use of light-but-sufficient rules of project behavior and the use of human- and communication-oriented rules.

Agility implies maneuverability, a characteristic that is more important now than ever. Deploying software to the Web has intensified software competition further than before. Staying in business involves not only getting software out and reducing defects but tracking continually moving user and marketplace demands. Winning in business increasingly involves winning at the software-development game. Winning at the game depends on understanding the game being played.

The best description I have found for *agility* in business comes from Goldman (1997):

> "Agility is dynamic, context-specific, aggressively change-embracing, and growth-oriented. It is not about improving efficiency, cutting costs, or battening down the business hatches to ride out fearsome competitive 'storms.' It is about succeeding and about winning: about succeeding in emerging competitive arenas, and about winning profits, market share, and customers in the very center of the competitive storms many companies now fear."

THE AGILE SOFTWARE DEVELOPMENT SERIES

Among the people concerned with *agility* in software development over the last decade, Jim Highsmith and I found so much in common that we joined efforts to bring to press an Agile Software Development Series based around relatively light, effective, human-powered software-development techniques.

We base the series on these two core ideas:

- Different projects need different processes or methodologies.
- Focusing on skills, communication, and community allows the project to be more effective and more agile than focusing on processes.

The series has these three main tracks:

- Techniques to improve the effectiveness of a person who is doing a particular sort of job. This might be a person who is designing a user interface, gathering requirements, planning a project, designing, or testing. Whoever is performing such a job will

want to know how the best people in the world do their jobs. *Writing Effective Use Cases* (Cockburn 2001c) and *GUIs with Glue* (Hohmann, forthcoming) are two individual technique books.
- Techniques to improve the effectiveness of a group of people. These might include techniques for team building, project retrospectives, decision making, and the like. *Improving Software Organizations* (Mathiassen 2002) and *Surviving Object-Oriented Projects* (Cockburn 1998) are two group technique books.
- Examples of particular, successful agile methodologies. Whoever is selecting a base methodology to tailor will want to find one that has already been used successfully in a similar situation. Modifying an existing methodology is easier than creating a new one and is more effective than using one that was designed for a different situation. *Crystal Clear* (Cockburn, forthcoming) is a sample methodology book. We look forward to identifying other examples to publish.

Two books anchor the Agile Software Development Series:

- This one expresses the thoughts about *agile* software development using my favorite vocabulary: that of software development as a cooperative game, methodology as conventions about coordination, and families of methodologies.
- The second book is Highsmith's forthcoming one, *Agile Software Development Ecosystems.* It extends the discussion about problems in software development, common principles in the diverse recommendations of the people who signed the Agile Software Development Manifesto, and common agile practices. Highsmith's previous book, *Adaptive Software Development*, expresses his thoughts about software development using his favorite vocabulary, that of complex adaptive systems.

You can find more about Crystal, Adaptive, and other agile methodologies on the Web. Specific sites and topics are included in the References at the back. A starter set includes these sites:

- www.CrystalMethodologies.org
- www.AdaptiveSD.com
- www.AgileAlliance.org
- My home site, members.aol.com/acockburn

THANKS TO SPECIFIC PEOPLE

Ralph Hodgson has this amazing library of obscure and interesting books. More astounding, though, is how he manages to have in his briefcase just that obscure book I happen to need to read next: Vinoed's *Sketches of Thought* and Wenger and Lave's *Situated Learning*, among others. The interesting and obscure books you find in the References chapter probably came from Ralph's library.

Luke Hohmann tutored me about Karl Weick and Elliot Soloway. Jim Highsmith taught me that "emergent behavior" is a *characteristic* of the rules and not just "lucky." Each spent a disproportionate amount of time influencing the sequencing of topics and accuracy of references, commenting on nearly every page.

Jason Yip beautifully skewered my first attempt to describe information dissemination as gas dispersion. He wrote, "Kim is passing information. Information is green gas. Kim is passing green gas . . ." Yikes! You can guess that those sentences changed!

Bo Leuf came up with the wonderful wordplay of *argh-minutes* (in lieu of *erg-seconds*) as the unit of measure for frustrating communications sessions. He also was kind enough to double-check some of my assertions. For example, he wrote to some Israelis to check my contention that in Israel, "politeness in conversation is considered more of an insult than a compliment." That produced an exciting e-mail exchange, which included (from Israelis): "Definitely wrong on this one, your author. . . . We always say hello and shake hands after not seeing for a few days. . . . I think your author is mistaking a very little tolerance for mistakes at work for a lack of politeness." Another wrote, "Regarding your being flamed. There is no way out of it, no matter what you say. According to me, Israelis would demand of you to have your own opinion and to stand behind it. And of course they have their own (at least one :-)." Benny Sadeh offered the word I finally used, "frankness."

Martin Fowler contributed the handy concept of "visibility" to the methodology discussion, in addition to helping with constructive comments and being very gentle where he thought something was terrible.

Other energetic reviewers I would like to recognize and thank (in first-name alphabetical order) are Alan Harriman, Allen Galleman, Andrea Branca, Andy Sen, Bill Caputo, Charles Herbaut, Charlie Toland, Chris Lopez, Debbie Utley, Glenn Vanderburg, James Hanrahan, Jeff Miller, Jeff Patton, Jesper Kornerup, Jim Sawyer, John Brewer, John Cook, Keith Damon, Laurence Archer, Michael Van Hilst, Nick Fortescue, Patrick Manion, Phil Goodwin, Richard Pfeiffer, Ron Holiday, Scott Jackson, Ted Young, Tom DeMarco, and Tracy Bialik.

The Silicon Valley Patterns Group took the trouble to dissect the draft as a group, for which I doubly thank them.

The Salt Lake production team of Elizabeth Wilcox, Cathy Gilmore, John Roberts, and Malia Howland did a fantastic job of turning the manuscript into a final book in an unreasonably short period of time.

All these people did their best to see that I fixed the weak parts and kept the good parts. If I had another few years to keep reworking the book, I might even have been able to get it to the point that they would have accepted it.

In the absence of those extra years, I thank them for their efforts and apologize for not being able to fix all the awkward spots.

Thank goodness the Beans & Brews coffee shop finally started playing jazz and rock again. I lost several months of writing to heavy metal and country music. Thanks to the Salt Lake Roasting Company for staying open until midnight.

To save us some future embarrassment, my name is pronounced "Cō-burn," with a long *o*.

Additional Copyright Information

PREFACE TO THE SECOND EDITION

The agile model of software development took the world by storm in 2001. Within a year there were books and conferences on it around the world. Within five years, it had influenced everything from project management and how corporate executives write contracts with their clients, to military procurement procedures, and even to college curricula.

It is time to look at the changes and see what we can learn about the agile model, and more generally, the cooperative game.

At the time that the Manifesto for Agile Software Development was written—February 2001—I was already deep into writing about software development as a cooperative game and the tailoring of methodologies to individual projects. The manifesto merely echoed what I and others were already doing.

The agile model took the world by storm. Hundreds of developers signed the online signing board (the original, informal Agile Alliance, not to be confused with the AgileAlliance corporation that was formed later and runs the Agile conferences in the U.S.) to show their alignment with its values and principles.

After the initial questions around "What does this mean?," people asked:

- "Where does it fit in the total set of development situations?"
- "How do we blend these ideas with others?"
- "How do we extend these ideas to other fields?"

These are the questions picked up by the additional text in this second edition.

The *cooperative game* model grew alongside the agile model. Originally constructed to explain software development, it struck a chord with business people, who rightly saw that business is also predominantly a cooperative (and competitive!) game of invention and communication.

You would imagine that during the past five years, something should have come as a surprise to me. I highlight four:

- *Engineering* is also a cooperative game of invention and communication. With a bit of reframing, we can now place software engineering as a proper member of the engineering fields.

I leave intact in the second edition the fairly strong words I wrote originally against the idea that software development should be treated as engineering. I do so because those words still apply to the way in which most people still incorrectly view engineering and draw from it correspondingly incorrect ways of how to manage software development. In this second edition text, I turn the investigation back to the question, "What is engineering?"

After the second world war, discipline envy of applied physics caused wishful thinking in engineering academia, and popular understanding of engineering got off track. Looking afresh at engineering practices, we notice that it is itself a cooperative game of invention and communication. From this, we can create an updated notion of *software engineering* that makes sense both practically and pedagogically.

- The specialty area of *user experience design* has made strong inroads in the last five years. It was ignored by the authors of the manifesto, and deserves serious attention.

All of the early agile methodologies skipped completely over this issue, probably because none of the manifesto authors had strong expertise in that area. That area has been and still is an area of hot debate, with few solid