The Addison Wesley Signature Series

A Mike Cohn Signature Book

Management 3.0

Leading Agile Developers, Developing Agile Leaders

Jurgen Appelo

Forewords by Robert C. Martin and Ed Yourdon

This page intentionally left blank

Library of Congress Cataloging-in-Publication Data

Appelo, Jurgen, 1969-
Management 3.0 : leading Agile developers, developing Agile leaders / Jurgen Appelo. -- 1st ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-71247-9 (pbk. : alk. paper) 1. Management information systems. 2. Agile software development--Management. 3. Leadership. I. Title.
HD30.213.A67 2011
658.4--dc22

2010041778

ISBN-13: 978-0-321-71247-9
ISBN-10: 0-321-71247-1

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing December 2010

Editor-in-Chief
Mark Taub

Executive Editor
Chris Guzikowski

Development Editor
Sheri Cain

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Apostrophe Editing Services

Indexer
Cheryl Lenser

Proofreader
Jennifer Gallant

Publishing Coordinator
Raina Chrobak

Cover Designer
Alan Clements

Compositor
Bumpy Design

This page intentionally left blank

Contents

This page intentionally left blank

Forewords

By Robert C. Martin

I hate management books. I do. People give them to me all the time saying: "You should read this one, it changed my life!" These books are all about 150 pages. They have 14 point type, double-spaced. They have lots of pictures. They have titles like: *Open Locker Management, Management by not Managing*, *First Clean All The Glasses, Now Discover Your Knees*, *The Power of Positive Penalties*, and *Tnemeganam!* They sit on my shelves. I sometimes read them in the John.

They all tell the same story. The author is always some guy who was running a company and failing horribly. When he reaches "bottom" (remember, I read them in the John) he has a critical insight that no human has ever had before. When he describes this idea to others, they think he's crazy; but he tries it anyway, and makes a $1,000,000,000,000 (one trillion dollars—billions are so passé nowadays). And now, out of the goodness of his heart, he wants to share that idea with you (for a small fee) so that you can make your trillion.

These books are usually repetitive, simple-minded, and inane. They are written at a third-grade level for poor saps who think that *one simple insight* is all they need to fix their problems. These unfortunate dweebs hope, against all hope, that if they just read the latest blockbuster: *Blue Pants Management*, and then have everyone in the office wear blue pants on Thursdays, that their management problems will go away.

Like I said, I hate management books. So why am I writing the foreword to a management book? I am writing the foreword to *this* management book because *this* book has the word *Eukaryotic* in it! What does "Eukaryotic" mean? That's not important. The point is that this book has words in it that have more than three syllables! This book talks about the *Red Queen Race* hypothesis. This book has depictions of *tesseracts*. This book talks about *Drunkard's Walks*. In short, this book is *smart*!

Just take a look at the table of contents. You'll see topics like *Complex Systems Theory*, *Game Theory*, *Cybernetics*, *Self-Organization*, and *The Darkness Principle*. You'll see that the author covers issues from team-size and motivation to scaling organizations up vs. scaling them out.

When you read this book you can tell that the author has done his homework. This is not just a simple-minded anecdote about how some old football player turned a department around. Rather, this book is a serious compilation of management ideas, techniques, and disciplines that have been accumulating for over a century. The author has taken these ideas and synthesized them with the *Agile Software Development* movement to form a *memeplex*, an interconnected system of ideas that every student of management will want to absorb. This book is not written for those who want a quick fix. This book is written for *serious* students who have a passion and love for management. This book is written for management craftsmen.

By Ed Yourdon

A long time ago, in a galaxy far far away, my colleagues and I proudly proclaimed that we were the young revolutionaries of the computer field, ushering in a new generation of methods and techniques for software programming, design, and analysis—which seemed to go hand-in-hand with the top-down, command-and-control management approach that prevailed at the time. We weren't clever enough to label our ideas "Software 2.0" in the fashion that subsequent advocates of "Web 2.0" and "Enterprise 2.0" have done … but in any case, Jurgen Appelo's new book, *Management 3.0*, tells me that my generation has been consigned to the dustheap of history.

The issue here, and the subject of Jurgen's book, is not really about software development techniques—though the "agile" development approach that has been growing ever more popular during the past decade does reject the idea that the requirements and architecture for a complex system can be developed in a strictly linear fashion, by following a top-down, hierarchical, deterministic approach. In a complex world where the end-users are not really sure what they want their system to do, and where everything around the users is changing all during the development of that system, we do need an orderly (dare I say "structured"?) approach to develop the boundaries and overall framework of the user's system—but many of the details will remain unknown and unknowable unless an "emergent" approach allows them to be discovered at the right time.

If that is true of the technical job of analyzing, designing, and implementing systems—and I firmly believe it is—then it is also true of the management approach that organizes, motivates, monitors, constrains, and (hopefully) rewards the people who carry out those technical tasks. So the top-down hierarchical style of management that corresponded to our top-down hierarchical "structured" approach to analysis and design in the 1970s is now being referred to as "Management 1.0"; and Jurgen tells us that there was also a phase known as "Management 2.0" that largely consisted of fads (like "Business Process Reengineering" and "Six Sigma") and add-ons to the earlier Management 1.0 approach.

But Management 3.0, which Jurgen's book discusses in detail, is based on complexity theory. It's something that mathematicians and biologists have been studying for the past few decades, and it's now becoming a central part of economics and sociology—and, more generally, management of people and their relationships in an organization. You really need to read Jurgen's summary of this concept—and the related ideas of causality,

determinism, and reductionism – because almost anyone whose education has focused on engineering, mathematics, and/or computer science has been inculcated with these ideas from an early age.

With this grounding, you'll be ready for Jurgen's "model" of modern management, which he portrays as a six-eyed monster named Martie—with a separate "eye" for viewing people, empowerment, alignment, improvement, competence, and structure. You'll need to plow through two more introductory chapters in which Jurgen summarizes agile software development and complex systems theory, but after that he devotes two full chapters to each of these six components of the Management 3.0 approach.

You won't find any of the "traditional" project-management stuff about risk management, estimating, scheduling, and monitoring progress with Microsoft Project; indeed, there is no mention at all of Microsoft Project in this book, and you won't find any references to the standard textbooks on risk management or estimating of schedules and budgets for projects. Those traditional activities still have to be carried out in most cases, and you probably should take a Project Management 101 course to make sure you understand them; but the essence of Jurgen's presentation is that even if you do a perfect job at carrying out the basics of Project Management 101, it's not enough to guarantee success. (Indeed, it may even aggravate the problem of complexity, and help you arrive at a disaster sooner than before!)

You can read the chapters of Jurgen's book somewhat independently, and perhaps even out of sequence—but I recommend that you read them all, and digest them slowly. There is an enormous amount of good advice, practical checklists, and wise counsel (how did someone so young become so wise?) on the nuances of leading, motivating, coaching, and communicating with individual developers, project teams, and the higher-level executives who are often still "stuck" in older ways of managing (e.g., the ones who insist on referring to the employees in their organization as "resources"). You may be tempted to treat some of his advice as glib one-liners (e.g., the advice in Chapter 4 that innovation is a bottom-up phenomenon, and that it cannot be mandated from the top), but if you read the book carefully, you'll see that it's a very sophisticated (and well-researched) discussion of the nuances involved in balancing things like self-organization versus anarchy.

I was amused to see Jurgen's statement, relatively early in his book, that he "wish[ed] a book like this had been available (or known) to me when I created my Internet start-up ten years ago. But then I might have become a millionaire and probably wouldn't have bothered writing this book in the first place." I feel the same way: I wish this book had been available (or known) to me when I first stumbled into the software field some 45 years ago, or at least when someone foolishly promoted me into a project-management position two years later. But then I too might have become a millionaire and probably wouldn't have bothered writing the foreword for this book.

Seriously, the only real problem I foresee with Jurgen's book is that the managers of my generation are still alive, and because the recent financial crisis reduced their 401(k) pension plans to a 201(k) or a 101(k), they're still working—and they're still doing their best to impose a rigid, top-down hierarchical management style on their subordinates. It's also problematic that managers of Jurgen's generation are moving into positions of power—because many of them have been brainwashed into following a top-down hierarchical management approach for such a long time, and they, too, may resist the ideas of Management 3.0.

But if the growing popularity of agile software development techniques is any indication, it's only a matter of time before the equally agile management techniques espoused by Jurgen Appelo in Management 3.0 become equally popular. And if you're determined to become an "agile manager" for dealing successfully with today's ever-more-complex projects, then while Jurgen's book will certainly not be the only book you read, it may well be the first book that you read on the subject.

And more important, it's likely to be the book that you return to, over and over again. I firmly believe that Management 3.0 will become the "Bible" of agile management books in the decade ahead.

This page intentionally left blank

This page intentionally left blank

Preface

This book is about **Agile management**, the managerial counterpart to Agile software development. I believe that Agile management is underrepresented in the Agile world. There are many dozens of books for Agile developers, testers, coaches, and project managers, but next to none exist for Agile managers and team leaders. However, when organizations adopt Agile practices, it is imperative that team leaders and development managers learn a better approach to leading and managing their teams.

Studies indicate that management is the biggest obstacle in transitions to Agile software development [VersionOne 2009]. For software teams, it is hard to be Agile and implement processes such as Scrum, XP, or Kanban when their "leaders" are stuck in old-fashioned management styles. Managers need to understand what their new role is in the 21st century, and how to get the best out of Agile software teams. This book aims at managers who want to become agile, and Agile developers who want to learn about management.

What makes this a unique management book is that it is grounded in science and leans heavily on complex systems theory. Unlike other (general) management books, it will not ask you to open your heart, hold hands, and sing "Kumbaya." Many managers, particularly in technical businesses, are "left-brainers," with a preference for logical, rational, analytical thought. So I wrote a book that appeals, hopefully, to left-brainers. But the right-brainers among you shouldn't fear! The scientific references in this book are explored in a casual manner, with plenty of explanations, metaphors, pictures, and at least two jokes that are actually funny.

One important goal I had for this book was to be *descriptive*, instead of *prescriptive*. Its purpose is to make you understand how organizations and Agile teams work, so you can solve your own problems. The world is too complex to give you merely a list of practices to follow. What managers in the 21st century need most is *insight* so that they can develop their own prescriptions for their own particular needs [Mintzberg 2004:252].

This page intentionally left blank

Chapter 13

How to Grow Structure

In all large corporations, there is a pervasive fear that someone, somewhere is having fun with a computer on company time. Networks help alleviate that fear.

—John C. Dvorak, columnist, broadcaster (1952–)

I love structuring things. You can see it in my file folders, my blog, my financial records, and my paper archives. Everything has a place and a function. I even have a neat white box labeled "Jurgen's junk," to keep things separated from another box labeled "Raoul's junk." It's the same with organizations I work for. I want to know what the structure is and what each part is for. Including the junk.

So that's the purpose of this chapter. It gives you an overview of adaptive principles in organizational design and some ideas on the ways to grow a structure in your own organization. I believe better communication follows from better structure; therefore, this chapter focuses on structure. We see that no single structure is the definitive answer for all organizations and that managers should instead focus on an organizational ability for continuous structural change.

The Management 3.0 model specifically refers to *growing* a structure. In complex systems, structure emerges by itself. However, as a manager, being responsible for the direction the self-organizing system takes, you can recognize that some structures are good and others are bad. The level of steering and intervention needed depends on the maturity and competence of the people in your teams.

About Environment, Products, Size, and People

People often ask me, "How should I structure my business and my teams?" (Well, actually they don't, but I expect they might after reading the previous chapter.) Unfortunately, there's no simple answer to that question. At least not a simple answer that also happens to be right. People might as

well ask, "What is the best form for a species?" The question makes no sense. One cannot claim that a starfish has a better body structure than a spider. Both species exist, and both have found a niche in which to survive. The spider can't survive in the sea. And the starfish won't survive in my cellar. It is the same with organizations. The "best" organizational structure depends on the environment in which the organization needs to survive.

> Thus we see that in today's environment, *no solutions can be independent of either time or context*. This also applies to organizational structures. To the extent that this is true, there is not—and likely may never be—any single form of organizational structure that provides maximum overall effectiveness.[1]

But the structure of an organization not only depends on its environment. The second factor in organizational change is the type of products. **Conway's Law**[2] says:

> Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations.[3]

Conway's interesting observation easily leads to the conclusion that an organization must be adapted to the kinds of products that are being produced [Poppendieck 2009:67]. Therefore, a second driver for organizational design is the set of products developed in the business.

The third relevant factor contributing to organizational structure is the size of the organization. While an organization grows, it regularly needs restructuring to accommodate for its new size, even when environment and product types remain unchanged.

> As a rule, every time a company grows by 50 percent, you should evaluate whether organizational changes are required, and by the time growth reaches 100 percent, you should already have made changes to accommodate that growth.[4]

---

1 This text was published in Organizational Survival in the New World, Alex Bennet and David Bennet, page 9, Copyright Elsevier, 2004. Used with permission. [Bennet 2004:9].

2 http://www.mgt30.com/conway/.

3 Reprinted under the Creative Commons License. Please visit http://creativecommons.org/.

4 © 2009 by Louis Testa and No Starch Press, San Francisco, CA, page 54. Used with permission. [Testa 2009:54].

And finally, the last driver for organizational change is the people. It is no coincidence that new managers and new teams, even when all else remains constant, often result in a restructuring of an organization. Different people need different structures to work with.

Changes in the environment, changes in product types, changes in company size, and changes in people, all lead to (or *should* lead to) changes to the organization's structure. A business that does not change with the times creates its own bubble of reality in which a lot of effort is wasted on stuff that has no value to anyone. A famous example of this phenomenon is **Parkinson's Law**, which says that "work expands so as to fill the time available for its completion." When existing structures in an organization are not abandoned, they will just keep inventing new work simply because they have the capacity available for it.

The people with whom I've worked know that I don't mind regular changes to teams and departments. It's not that things must change for the sake of change. But neither do I think that a structure is better off unchanged for the sake of stability. And when I leave an organization for another job, it doesn't bother me (that much) when my legacy is overhauled again by my successor. Times change with new competitors, new products, new employees, and new managers. I would be worried if a business *stopped* responding to such changes.

I don't believe managers need an overview of best organizational diagrams. What they need is advice on how to achieve adaptability. Species are all different, but they have one thing in common: The principles of adaptability are built into their DNA. That is what we're looking for. We want to know how to have an adaptable business so that it is easier to let an organization morph into different structures depending on context, products, size, and people.

When researching a number of books covering business structures, I noticed that many of them have a description of the "standard" hierarchical functional organization and then go on to describe "alternative" structures that are supposed to be better [Augustine 2005]. Or they describe different organizational archetypes or "forms," where the forms emerge as a result of their environments [Mintzberg 2009:106]. I will attempt a different approach. I will focus on a number of guidelines for adaptable organizations, and you can use these guidelines to grow your own organizational structures.

I believe that, similar to the forms of species, there are a few basic successful patterns with a large number of variations. None of them are

intrinsically "better" than any of the others. The starfish is not better than the spider. Though, I must admit, a poodle is better than a Chihuahua.

Consider Specialization First…

Suppose you are the publisher of a magazine about cooking. It's a glossy magazine with recipes, restaurant reviews, and lots of pictures of expensive cutlery and celebrities tasting trendy oysters. The magazine is released every month, and you have a huge list of recipes and restaurants, and celebrities waiting to make their appearance in one of the upcoming editions. Getting a new edition out the door is always a stressful experience. The celebrities can never commit to any culinary photo shoot. The chefs always complain about the way their dishes are depicted. And some of the recipes are so bad, you wouldn't even want to cook them for your neighbor's dog.

Now the editor walks up to you and tells you he has the solution to all problems. It is called generalization. It's really simple and very effective, he says. The different roles of all people working on the magazine will be turned into one generic role called "team member." There are no real specialists anymore, as everyone on the team is allowed to do any of the jobs needed to get a new edition of the magazine out of the door. The writers are allowed to do the photo shoots, whenever they happen to be in the vicinity of a celebrity. Any chef, with at least one working finger left, is allowed to type restaurant reviews. And if the photographers are finished with their work, they can help out writing and cooking recipes. With such a team of generalists, explains the editor, making a new edition of the magazine will be much less stressful (see Figure 13.1). So…what do you say?

This is what I would say, "Are you completely mad?" If I'm on an operation table having my eyelids corrected, would I want the nurse to take over when the surgeon is having trouble keeping up with his schedule? Would I say, "Yes, thank you nurse, and why don't you remove my tonsils while you're at it?"

I believe generalization is a fine idea. But specialization is your *first* friend. Research has confirmed that teams of specialists are more productive than teams of generalists [Anderson 2004:271]. Building teams of only generalists ignores everything society has learned in the last 235 years, ever since Adam Smith pointed out that specialization leads to higher productivity and prosperity. Specialization is the reason why

software developers do not bake their own bread, fix their own clothes, or grow their own food, a few exceptions notwithstanding. The larger an economy or organization is, the more people will want to (and be able to) specialize in what they are good at. It is a mechanism that has proven to work well, not only for individuals but also for the whole world.

FIGURE 13.1
From specialist to generalist?

…And Generalization Second

On the other hand….

Specialization does have its problems. It can lead to bottlenecks when specialists cannot cope with demand and others cannot take over for them. After all, I once *did* design a corporate web site myself, including interaction design *and* graphics design because our regular designers were unavailable for weeks. And it can lead to stagnation when the specialists are unable (or unwilling) to pick up work that they are unfamiliar with. For example, I once *did* ask a software developer to help me carry out some marketing activities I could not have done on my own. Our marketing efforts would have stalled if he had not willingly co-operated.

I have no use of people telling me they have a "broad range of skills," meaning that they never specialized in any specific area. I clearly prefer specialists over generalists. But I like it even better when the specialists have a few extra areas in which they have built up some knowledge and expertise. Fortunately, I'm not alone in that opinion.

> A generalizing specialist is someone who: 1) Has one or more technical specialties […]. 2) Has at least a general knowledge of software development. 3) Has at least a general knowledge of the business domain in which they work. 4) Actively seeks to gain new skills in both their existing specialties as well as in other areas, including both technical and domain areas.[5]

A **generalizing specialist** does one kind of job very well and some other jobs adequately. With generalizing specialists your teams enjoy the benefits of high productivity, while lowering the risk of bottlenecks and retaining flexibility. Generalizing specialists are sometimes called **T-shaped people**. They have a principal skill that is the vertical leg of the T, but they are also inquisitive and interested in branching out into other skills. Such people are valuable because they can explore insights from multiple perspectives. [Brown 2005]

When hiring people and putting together teams, look for T-shaped people. Always check if they are specialists in at least one useful area, and then verify that they are willing and able to pick up other kinds of work as well. If you're looking for a software developer, make sure it's a good one. But also ask some questions about graphics, design, hardware, and maybe even marketing.

AND SPECIALIZING GENERALISTS? DO THEY EXIST?

They certainly do. They are people who do many jobs reasonably well but have a tendency to do one or two jobs significantly better. They are very much like generalizing specialists but still less of a specialist and more of a generalist. I would consider them almost as valuable as generalizing specialists.

---

5 Ambler, Scott "Generalizing Specialists: Improving Your IT Career Skills" http://www.mgt30.com/specialists/. Agile Modeling. Reprinted by permission of Scott Ambler. [Ambler 2010].

Widen People's Job Titles

In my job as chief information officer, I sometimes clashed with HR people over the chaotic growth of job titles in some parts of the organization. For business units as small as 10 people, I saw never-ending streams of job titles flying by, like Content Developer, Content Manager, Web Editor, Web Designer, Interaction Designer, Front-end Designer, Front-end Developer, Web Manager, and Front-end Manager. I'm sure Interaction Developer had slipped in there somewhere as well. What was the use of all these different titles? I have no idea. And neither did the ones involved. I repeatedly told people that having fewer job titles is better. And all those developers and designers could have been called Esteemed Employee, as far as I'm concerned.

The team I was working on (while I wrote this) had four great people in it. One of them knew all about the API that we were developing. He decided what the interface looked like, how it was deployed, and how it was kept consistent over multiple releases. He was our leader when it came to our programming interfaces. The second person was our youngest team member. But he had proved himself as a promising architect. Our third team member knew all about social media and e-commerce. He was our leader when it came to online marketing and communication strategies. And finally, yours truly played the role of the Product Owner, making decisions about features and priorities, and keeping the others busy so they didn't get bored and started blowing things up.

Each of the members in our team was a leader. We played roles that matched our specialties, but they were not our job titles. We had no titles for Interface Programmers, Software Architects, Marketing Consultants, or Product Owners. In fact, we took over each other's roles whenever the need arose. (And this was a real necessity with me traveling up and down between conferences around the world.)

For improved organizational adaptability, I believe it helps not to lock up responsibilities in job titles. Instead, you need to keep those titles as widely applicable as possible. People's official job titles don't change easily (sometimes only once every few years); therefore, it is wise to decouple job titles from day-to-day responsibilities. For example, the title Software Engineer gives you more freedom in moving responsibilities around than the title Information Analyst. Even when someone asks to be called an Information Analyst, tell her that her contract will say Software Engineer, and that Information Analyst will be her role. For now.

The wide job titles *can* be used as formal boundaries for the informal roles. For example, the job of a Software Engineer can include anything ranging from design, development, and testing, to project management and support [Abran 2004]. Therefore, a Software Engineer in your organization might be allowed to pick up a diverse bunch of roles like Programmer, Tester, Support Engineer, and Business Analyst. But no person with a job title outside the boundary of Software Engineer (like Account Manager or System Administrator) would ever be given such roles.

Flexibility of people is exactly the reason why Scrum calls everyone simply a Team Member. It underlines the requirement that people feel a responsibility to do anything needed to ship their product, no matter their official job titles. Nobody should be able to say, "I won't do that. It's not my job." If releasing a successful product involves cleaning your customer's keyboard, then cleaning keyboards *is* your job. Some organizations even go as far as to have just the title Associate for everyone in the company. It teaches people to be flexible while getting things done.

Note that the idea of widening job titles actively supports the concept of generalizing specialists. People *should* specialize in something, but they must be flexible enough not to claim exclusive job titles in support of their specialization. Such specialist job titles would mean responsibilities get locked into the title and into the person. And that's not what you want in an adaptable organization.

What you want is a small set of job titles and perhaps a few guidelines on which informal roles go with which titles. Any initiatives that tend to increase the number of job titles in the organization, and requests to formalize roles and responsibilities, should be nipped in the bud.

For years, my job title had been CIO, which is a great title because the letters can stand for almost anything. (Depending on the context, the "I" has stood for Information, Ideation, Imagination, Innovation, Inspiration, Insubordination, Interaction, Intimidation, Illustration, and Idolization.) But the things I've specialized in, and the projects I did, often had nothing to do with my title. It was just stuff that had to be done.

Cultivate Informal Leadership

Leaders in a team are sometimes called Leads or Chiefs, like technical leads, project leads, chief programmers, and chief architects. What these people have in common is that they are not the line managers of the others in their teams. Informal leadership is bestowed upon people because of credits earned or commitments made. Or maybe even as a practical joke. It is a responsibility that is completely separate from line management [Testa 2009:53]. When several people take up leadership in different areas, we might call it distributed informal leadership. Informal leadership follows logically from working with generalizing specialists and using wide job titles.

You can actively cultivate informal leadership in your teams by supporting emergent leadership positions, but it is best to refrain from directly assigning such roles yourself. Allow the teams to decide whether they want to appoint Technical Leads, Project Leads, or some other leading role. (Note that many teams tend to flounder when there's no strong leadership inside the team. You may need to push them and help them in solving their own leadership problem.)

None of the roles mentioned would involve a management layer. In fact, that is precisely why informal leadership contributes to the adaptability of an organization. By abstaining from a management layer of Chief Somethings and Lead Whatevers, you make it much easier for the organization to add, move, and delete such responsibilities. Whenever there's a need for a Chief Graphics Designer, she can be appointed on the spot. And when the need fades away, so does the role. Not the person. If the role was a formal job title, the person would have to be kept busy, or she would have been asked to formally change her job, or else she'd have to get fired for lack of work. All these are unpleasant measures that suck productivity out of the organization.

Generalizing specialists, widening job titles, and informal leadership are different but related concepts (see Figure 13.2). Though they tend to reinforce each other, you can introduce one before introducing the others, which might be necessary when gradually changing a bureaucratic organization to a more adaptable one. But please don't ask me what order would be best in such cases. My experience is mainly with organizations in which people were flexible and passionate enough to swallow them all at once.

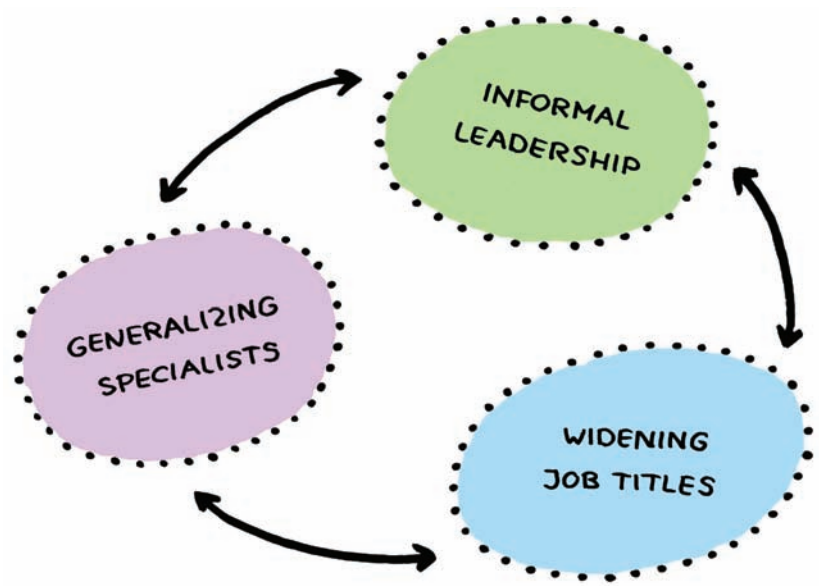


FIGURE 13.2
Different but related concepts.

Watch Team Boundaries

In Chapter 12, "Communication on Structure," we saw that people tend to form groups. And when a group is small enough and has a shared purpose, we may call it a team. The concept of a team is very useful because it is a way of identifying a number of people as one entity. In psychology they call that chunking:

> The idea of "chunking": a group of items is perceived as a single "chunk". The chunk's boundary is a little like a cell membrane or a national border. It establishes a separate identity for the cluster within. According to context, one may wish to ignore the chunk's internal structure or take it into account.[6]

In my last job, with many small projects and dozens of developers and testers in multiple locations, team formation was always a challenge. We changed our team formation approach more often than Madonna changes her image. But management of team boundaries is an important part of a manager's responsibilities, and it's important to try and get things right.

6 Hofstadter, Douglas. Gödel, Escher, Bach. New York: Basic Books, 1979. [Hofstadter 1979:288].

After all, teams don't operate well when people don't know what the teams are and who they can rely on.

There are three aspects to boundary management: the way teams are structured, how individuals relate to teams, and how teams change over time. Self-selection of teams is possible in organizations in which people have a high level of "empowerment maturity" (see Chapter 7, "How to Empower Teams"). In such an organization you create a pool of potential team members, and then you leave team formation to the group. There might be projects that many people want to be on and projects that nobody wants to do. The great thing is that the group has to find its own rules for team selection, and as a manager you can just enjoy the heated discussions from the sideline. Self-selection of teams is something I have rarely seen in real businesses. It *is* worth considering, but you have to be sure that people understand *how* to form teams. One team of 30 developers and one team of 20 testers might not be a good option. Just consider the example of popular boy bands: Though they *can* have 30 members, in which case we tend to call them boy *choirs*, with such a size they rarely have the agility to keep up with trends in entertainment as much as a small team can. So to increase their chance of success, you might want to define and discuss some constraints on team formation first, concerning size, diversity, and other parameters.

How individuals relate to teams is another constraint you should take into account. Is a person allowed to be a member of more than one team? It is common for people not to perform as well as they could when they are asked to spread their loyalty across multiple teams. Mick Jagger never joined the Jackson Five to complement the Rolling Stones, and for good reasons. Such situations lead to task-switching, conflicts of interests, loss of commitment, and loss of motivation. Try to make sure that every person is dedicated to just one team. People cannot act as a team when they do not know what the team is. They may occasionally assist other teams and help out with other people's projects or perform some duets, but each person should have exactly one base team to return to.

Finally, the time span of a team is also an important issue. Research shows that teams perform much better when they are long-lived. Not just in software development [Larman, Vodde 2009:149/153] but also in other businesses, like airlines [Hackman 2002]. It is best for teams to exist for as long as possible because it takes time for communication paths and rules in a team to grow and pay off. It also takes time for them to learn, as a team, which information is important for them and which is not. Just think of this: What is the best pop group ever? And how long did they

stay together? More than a few years? Yes, I thought so. When projects in your organization are by their nature short, try to keep people together in teams with longer life spans, where the same teams work on one project after another.

The Optimal Team Size Is 5 (Maybe)

What is the optimal team size? This is one of the most interesting boundary issues and an important question people have been discussing ever since they teamed up and killed the first mammoth.

I once attended an inspiring conference session hosted by social complexity expert Joseph Pelrine, who told his audience that the sizes 5, 15, and 150 have been mentioned in (or can be derived from) scientific research as being optimal sizes for social groups.

The Agile movement, with Scrum as the leading method at the time of writing of this book, often mentions a preferred team size of "7 plus or minus 2" (which is just a software developer's way of saying "between 5 and 9").

Research into optimal group size for decision making revealed that only numbers below 20 appear to work well [Buchanan 2009:38-39]. Anything from 20 and up can hardly be called a *team*. When the number of people is too large, we should just call it a group. (I'm writing this text secretly while attending sessions at the Scandinavian Developers Conference, which has 600 attendees. That's a group, not a team.)

Buchanan's article makes an exception for team sizes of 8, which do not appear to work very well. That's because eight people frequently find themselves in a deadlock situation over their decisions. It is said that King Charles I, the only British monarch ever to work with a council of eight members, made decisions that were so notoriously bad that he lost his head [Buchanan 2009:39].

Considering these findings, we can easily see that there's only one optimal team size that satisfies all conditions:

Five

Five is one of the three optimal sizes mentioned by Joseph Pelrine. Five also falls within the preferred range of sizes for Scrum teams. Five is less than 20 and yet unequal to 8. Five is also closest to the optimum of 4.6 team members that professor J. Richard Hackman found in his research [Hackman 2002:116-122]. And best of all, 5 happens to be my lucky number. So it must be true.

Five is also my default answer to any question that I cannot answer without more information. You see, I actually cannot tell you what the optimal team size is! Let's revisit Kurt Lewin's equation for a moment (discussed in Chapter 10, "The Craft of Rulemaking"), and you will see why:

$$B = f(P,E)$$

As discussed earlier, this equation means: *a person's behavior is a function of his or her personality and his or her environment*. And because communication is part of a person's behavior, a different version of this equation could look like this:

$$C = f'(P,E)$$

It means *a person's communication is a function of his or her personality and the environment*. And when we're talking about a whole group of people, and realizing that team size is a communication issue, we can rewrite the equation to look like this:

$$S = f''(\{P\},E)$$

This version means *the optimal size of a team is a function of the set of people's personalities and their environment*.

In other words, the value of S can be anything! For the Apollo 11 moon landing, the optimal team size was 3. In rugby, the team size is 15. Apparently, the optimum for team size depends on the project, the people, and their environment. But statistically, across all teams in all businesses, the optimum could be 5, and a few numbers close to 5. And if we want to describe this as a range, we could say "between 3 and 7" (or for software developers, "5 plus or minus 2"), which neatly cuts off the 8 (see Figure 13.3).

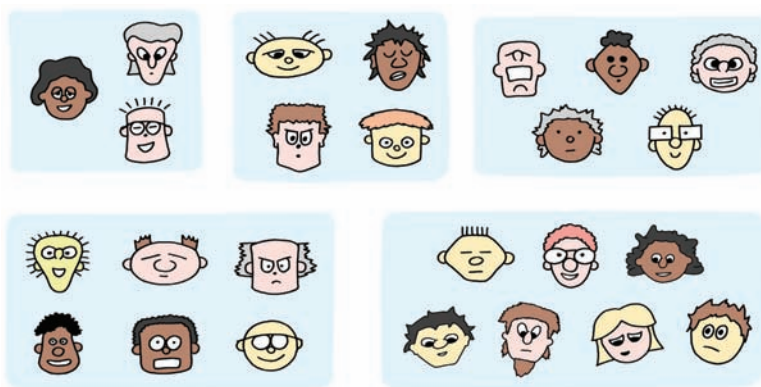


FIGURE 13.3
Optimal team size: 5 plus or minus 2.

So, what can we learn from this?

My suggestion is not to *impose* one "preferred" team size on people; although, you might want to add some constraints to team formation. For example, anything upward of 20 is not allowed, with a *suggestion* to have 5 plus/minus 2 members per team. Then allow self-organization to do its job, and let the people (within their real environment) figure out what their optimum is. Do they want to cut a team of 7 into two teams of 3 and 4? Sure, why not? Are they merging two teams into one big team of 15? Fine, let them see if that works for them. Just make them aware that they might want to reconsider things when the environment *or* the set of personalities in the team has changed. One final world of advice: Keep your axe ready in case they come up with a team size of 8 (plus or minus 0).

Functional Teams versus Cross-Functional Teams

Whether team formation is done by the manager or by the teams, one important question needs to be answered, "*How* should people be grouped together?" Basically there are two main options to choose from: group people by *similar function* or by *similar business*.

Grouping people by similar function means that you put developers with developers, testers with testers, and project managers with project managers. Such groups are called functional units, and the driving motivation behind this kind of structure is efficiency and functional learning [Larman, Vodde 2009:243]. It is easiest for writers of user stories to learn how to be efficient user story writers when they're all put together in one department called User Story Writing.

Grouping people by similar business means that you put everyone together who works on the delivery of the same business value (the same feature, the same product, or the same customer). Such groups are sometimes called cross-functional units because all people involved in the same project(s), from user story writers to binary assembly deployers, end up in the same group.

In Chapter 12, "Communication on Structure," we discussed that good communication is both hard and crucial for any organization. It is therefore imperative that we let communication be one of our guiding principles when choosing between the two variants. Which people need each other most often? The ones with the same job titles? Or the ones working on the same project?

If you were to analyze daily communication between employees, it would quickly become clear that most of that communication is oriented around the business and not around the function. People with different functions but working on the same projects need to communicate more frequently than people with the same functions who work on different projects (see Figure 13.4). We can thus conclude that for *projects* cross-functional teams are a more suitable solution to the grouping problem.

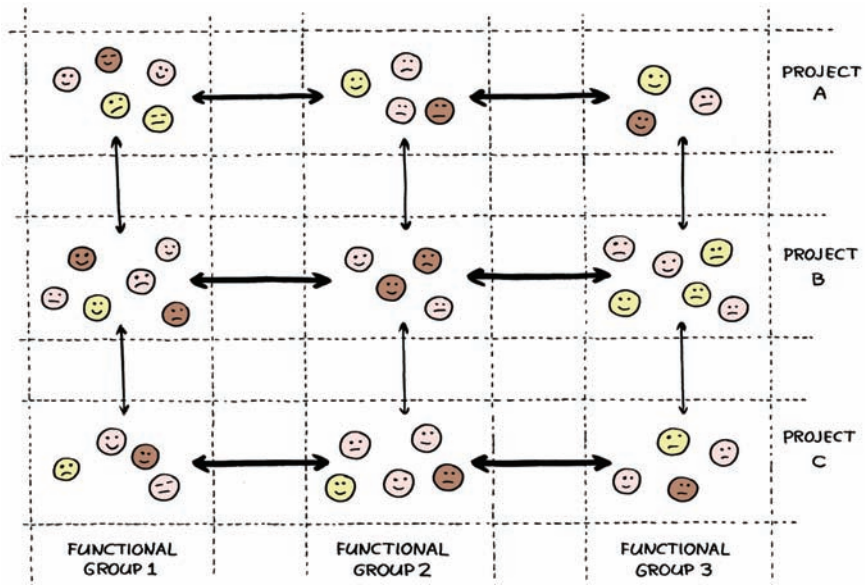


FIGURE 13.4
More communication in projects than within functional groups.

It has been reported that in organizations where people are grouped by function (sometimes referred to as **functional silos**), there are too many dependencies between the functional teams. Delivering even the smallest piece of business value (like one feature of a product) requires communication and coordination across multiple teams [Poppendieck 2009:68]. Functional silos therefore have a high interaction penalty [Augustine 2005:26].

When you build teams *across* the functional silos and not *inside* the silos, the interaction penalty is lower but *not* zero. Donald Reinertsen lists three problems with cross-functional teams: suboptimization at the project level, inefficiencies due to lack of coordination across projects, and reduced expertise because of limited knowledge sharing across specialists [Reinertsen 1997:104]. So it appears that with cross-functional teams the penalty is paid for synchronization of standards, methods, and approaches within one functional discipline across different teams. For example, it

will take a quality assurance manager more effort to co-ordinate best practices in testing, when the testers and QA people are spread over multiple teams. But the price being paid here is generally lower than in the case of functional units.

There are several other advantages to cross-functional teams (varyingly referred to as feature teams, project teams, organic teams, or product teams). Several experts report improved design decisions, reduced waste from hand-offs of intermediate products, improved speed, improved adaptability, simplified planning, and focus on delivering value [Cohn 2009:182–188] [Larman 2009:154].

Two Design Principles

When there is more than one team in your organization, things need to be coordinated. Whether it is the choice of logging framework, the location of the refrigerator, or the availability of the demo room, people need to agree on things that are shared across multiple teams.

Psychologist Fred Emery distinguished two basic patterns for coordination of activities across multiple teams. He named them the *first design principle* and the *second design principle*.

In the first design principle (DP1), the location of the fridge is determined by people who are positioned one level above the teams. They are either the line managers of the teams or else a dedicated Fridge Manager who is appointed by the line managers. Either way, the teams have no say in the location of the fridge. Only the Fridge Manager is authorized to decide (see Figure 13.5).

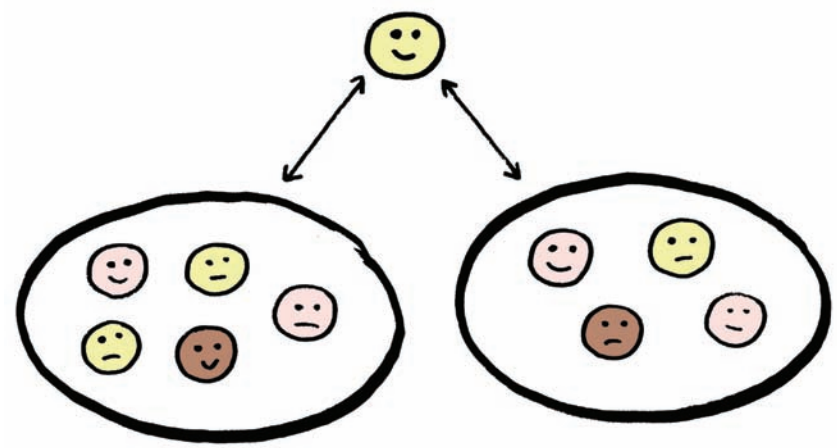


FIGURE 13.5
First design principle: a manager coordinates.

In the second design principle (DP2), regulation of the location of the fridge is built into the teams themselves, meaning that the teams take care of coordination across their boundaries. In practice, this means that teams have to negotiate with each other and agree on some rules, such as voting on the location of the fridge, pricing the availability of the fridge, daily fridge rotation, or fridge roulette. The teams may even agree on their own Fridge Manager and bestow authority on her to make decisions for the teams. With DP2, the authority ultimately lies with the teams, not with the line managers (see Figure 13.6). (And then informal emergent leadership inside the team could become a necessity to prevent a consensus culture with endless discussion.)

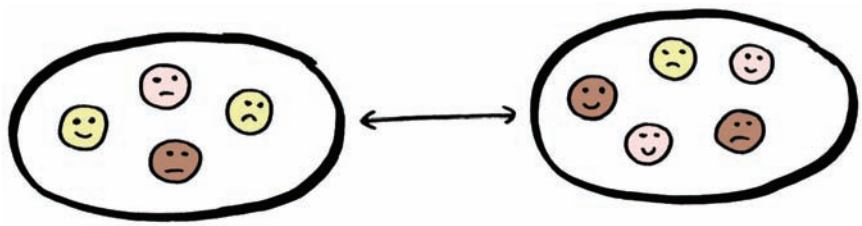


FIGURE 13.6
Second design principle: The teams coordinate.

The second design principle closely resembles the solution that complexity scientist Stuart Kauffman describes as "patches":

> Kauffman says break up the organization into patches, yet emphasizes that these patches must interact. This advice is different from the old management standby of the independent, self-sufficient business unit. It is in the nature and quantity of the interactions that Kauffman finds that the organization as a whole can be moved toward a global optimum, even though each patch is acting selfishly. Interactions require language or some other mechanism of fairly continual communication. He stresses that the patches must be coupled. In management jargon, the pieces must communicate, and not just at quarterly review sessions.[7]

In this analogy, patches are self-organizing teams, not controlled departments. The adaptability of these patches (DP2) compared to hierarchical management (DP1) follows directly from the organic way of problem

7 Lissack, Michael R. "Complexity: the Science, its Vocabulary, and its Relation to Organizations" Emergence. Vol. 1, Issue 1, 1999. Used with permission. [Lissack 1999:114].

solving. Every team tries to solve one part of a bigger problem. But because of the couplings between teams, the solution found in one team will change the problem to be solved in adjacent teams. And the adaptive moves of those teams in turn will alter the problems to be solved by other teams. Ultimately, you end up with an ecosystem of teams, or patches, solving a big problem together. [Kauffman 1995:252]

It is clear that the principle of patches (DP2) is the best option for decisions on the choice of logging framework, the location of the refrigerator, the availability of the demo room, or anything else that needs to be coordinated across teams. When some issue needs to be resolved across multiple teams, tell them to coordinate the solution among themselves. DP1 (that's *you* or some other manager making the decision for them) will only be a viable solution when you realize that DP2 doesn't work well. For example, when competence issues have not been resolved yet.

Choose Your Organizational Style

There is a tremendous amount of praise in literature, and in the blogosphere, for cross-functional teams. It sometimes seems as if it is the best idea since cross-personal interaction. And cross-personal interaction *is* a great idea, until you find out you caught some social disease you would rather have avoided.

I am glad that I have little experience with social diseases, but I do know that at least part of the praise for cross-functional teams is undeserved. There are a number of misconceptions because some authors associate functional teams with hierarchies and cross-functional teams with organic networks. But this is both unrealistic and unfair.

Functional teams require coordination across team boundaries about the projects they are doing, and the business value delivered to customers. On the other hand, cross-functional teams require coordination across team boundaries about practices, standardization, and shared resources, for any similar kind of work that is carried out in different teams. So the question is, "*How* is this coordination across teams taking place?"

In the previous section, we saw that you have two options for coordination: DP1 and DP2. Both can be applied to either functional teams or cross-functional teams. These 2x2 options result in four organizational styles, as shown in Table 13.1 and Figure 13.7.

TABLE 13.1
Four organizational styles

Style	Team Structure	Design Principle	Description
1	Functional	DP1	Coordination between functional teams is performed by managers (typical hierarchical functional silos).
2	Functional	DP2	Coordination between functional teams by the teams themselves (for example, self-organized sysops teams each dedicated to a piece of an infrastructure).
3	Cross-functional	DP1	Coordination between cross-functional teams by a project manager or other authorities above the teams.
4	Cross-functional	DP2	Coordination between cross-functional teams by the teams themselves (for example, a "Scrum of Scrums").

In general, cross-functional teams work better than functional teams, and DP2 works better than DP1, and therefore organizational style 4 is the preferred option for many Agile consultants. But, as always, it depends on the context, and you may want to choose one of the two reasonable alternatives (organizational styles 2 or 3), either because team maturity or prevailing communication paths require it, or to facilitate a gradual organizational transition from style 1 to style 4 (see Figure 13.7).

I have known cross-functional teams that were so young and inexperienced (may I even say irresponsible?) that they could have infected half the company with their problems, if management had let them. Fortunately, organizational style 3 saved the day there. And I have known productive specialist teams responsible for components or assets that were too risky to distribute over multiple teams. (Access to other people's bank accounts is one that comes to my mind.) Yet these small specialist teams were mature enough to organize their own cross-team coordination without a manager.

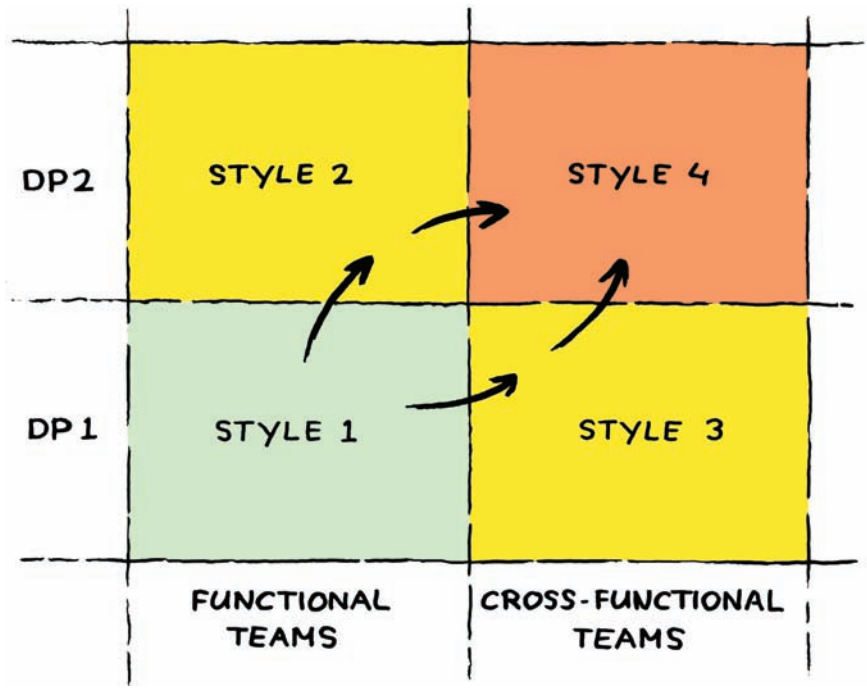


FIGURE 13.7
Quadrant of organizational styles.

Cross-functional teams without management coordination are a great idea. But they can both solve *and* introduce problems. Good managers need to be smart enough to think of their own best approach to an organizational style that is both adaptable *and* safe.

Turn Each Team into a Little Value Unit

The last team of system administrators I worked with was a great team. I really like them, but I think I was their worst customer. It's not that I was behaving badly. (Well, usually I wasn't.) It's just that my aura has an unpredictable effect on electromagnetic fields. People have seen reliable software crash whenever I passed by, and even the sturdiest operating system has an increased tendency to reboot unexpectedly in my presence. And remember those many times you saw a Fail Whale on Twitter? Yes, that was probably me having logged in before you. That's why I liked my system administrators so much. Because no matter how many problems I generated for them, they always treated me as a customer.

It is often claimed that cross-functional teams solve the problem of local optimization, which happens when functional teams optimize their own efficiency. This hurts the overall performance of the business. For

example, a testing team may optimize testing procedures, making sure that all testing for a project is performed in one short period of time. Such an "efficient" practice doesn't take into account the dramatic effect this has on the development and support phases of the projects. But is this really a problem of functional structure? Or is it an example of the testing team not treating the development and support teams as their customers?

The opposite problem is that cross-functional teams tend to optimize for their own projects, which can also hurt the overall performance of the business. For example, there may be problems when different project teams all decide to choose their own architectures and third-party components. This increased variation of technologies makes it difficult for the organization to support all those projects. And I'm sure that when I allowed project teams to purchase their own computers and install their favorite operating systems and development environments, my friendly team of system administrators would have skinned me alive.

But most software developers I have worked with wouldn't dream of inviting system administrators into their cross-functional teams. And that's not because they don't like them. It's because communication within a team of system administrators is usually more intensive than their communication with project teams, even though infrastructure is often an important part of many business solutions. Therefore, it makes more sense to keep these people together in their own functional group, despite the communication penalty paid on any cross-functional communication.

What's important is that every team, both functional *and* cross-functional, should see itself as *delivering value* to a customer, no matter whether that customer is an internal or external one. Our team of system administrators saw itself as a small business unit that tried to serve its customers, by delivering something valuable. And that's why we liked them. They made the other teams feel important, because to them we *were* important, no matter how often I crashed our systems or brought down our servers. Functional teams *and* cross-functional teams should be run as little value units. Then they are truly fractal teams, and there is no limit to the number that can be formed [Leffingwell 2007:96].

Move Stuff out to Separate Teams

The nice thing about not being directly involved with any method, framework, alliance, or consortium, is that I can be a heretic and say anything I want. The worst thing that can happen to me is that I'm being flamed and

grilled when I'm on a conference panel. That is why I have fire-resistant gel in my hair. But I've noticed there's a market for contrary ideas. And as a firm believer in markets, I love exploiting opportunities of dissent whenever I can. Like in this case.

I believe it is *sometimes* better to move specialist work to (functional) specialist teams. This could be necessary for project management, architectural components, user interface design, hardware design, testing, or any other work that deviates significantly from standard activities in a project team. This goes against "accepted" thinking in the Agile community because many strong voices suggest that all work, from story to binary, should better be done by cross-functional project teams, including coordination of efforts across multiple teams. The Scrum of Scrums is a good example. It says that each team sends a person to a daily Scrum of Scrums meeting, and these people then coordinate the work across the teams. Such suggestions have been made for Scrum Masters, technical leads, user interface designers, and lead testers.

But I believe it is simply a matter of balancing communication. If it turns out that user interface designers need each other more often than they need the team members working on delivering business value to customers, then it is right for them to sit together and form their own team. Likewise, project dynamics in a company may be so intense or complex that project leads of different teams require intense collaboration. Then it might be better for them to get together and form their own team. Perhaps even a Project Management Office.

BUT…five things are important here:

- First, when some responsibility, like project management, architecture, or GUI design, is moved outside the project teams, every (cross-functional) project team needs a communication interface to the (functional) team that is formed around the specialist activity [Leffingwell 2007:108]. One can think of regular attendance of the specialists in the project teams' stand-up meetings and/or some designated representative from the project teams in the specialist team. Plenty of options are available and should be applied to address the issue of the bandwidth of communication between the project teams and the specialist team.
- Second, the people who are moved into a specialist team must see themselves as value units, just like system administrators are *servicing* project teams, not *controlling* them. Specialist teams should consider project teams to be their "customers," not their

subordinates, and organize their processes accordingly. They sell their services to their colleagues in the other teams, just like I'm trying to sell my dissenting views to you. (I'm glad you invested in this book *before* you got this far.)

- Third, the project teams should decide whether the specialist team is actually delivering any value. Such a market approach would counterbalance the tendency for support units to suboptimize at their own level. For example, in my last position I could choose to go to our unit of expert interaction designers, or I could choose to do interaction design myself. It all depended on how well (and how soon) our interaction design unit was able to service me and my project. (And note: I have developed some skills in dissent *and* design.)
- Fourth, we know that the total amount of communication in a complex system remains (more or less) the same, no matter how the system reorganizes itself. Therefore the teams and their managers will figure out how many points of contact with other teams they can handle. Both too little and too much is bad for the adaptability of the organization.
- Fifth, a team of specialists can be virtual instead of physical. It can be just a matter of getting all user interaction designers together once in a while, and allowing them to agree on common standards and approaches across the cross-functional teams where they actually do their work. Such virtual teams are called **communities of practice**, and they are a good compromise, bridging the need for cross-functional teams *and* the need for coordination among specialists [Augustine 2005:71–73] [Larman, Vodde 2009:252/253]. (Note: Some organizations have **centers of excellence** with a similar purpose; although these COE tend to be a bit more formal in nature.)

It is possible, and perhaps even preferred, that the formation of specialist teams is a result of self-organization. Specialist teams form themselves organically in an attempt to solve a problem that is shared across multiple teams. For example, a continuous integration (CI) team forms itself as a spin-off in an attempt to provide a more professional CI service to the other teams. Team members from the various project teams then have a choice of full-time, part-time, and/or rotating membership [Highsmith 2009:272/280]. Another example is that of a component team, which

designs, builds, and delivers an architectural part of a solution to the project teams, whereas the project teams together act as customers to the component team [Cohn 2009:185]. The primary reason for the formation of specialist teams is efficiency and effectiveness (productivity through division of labor).

We can even imagine that these specialist units grow and form their own little hierarchies. They may even have a number of rules that apply to project teams *if* these teams decide to make use of their services. But like in any market environment, the specialist teams (and their rules and hierarchies) can and should be dissolved as soon as the need for them evaporates.

In each of these examples it is clear that the project teams are consuming and the specialist teams are providing (see Figure 13.8). And so it should be the same with a project management office (PMO), *if* it exists. A PMO is in the business of *servicing* project teams in getting projects organized. Project managers, like user interface designers, architects, and system administrators, are *not* line managers. And nobody should ever be expected to "report to" the PMO. Instead, the PMO should respectfully *ask* the teams for information and *deliver* something that the teams and their customers can actually use.

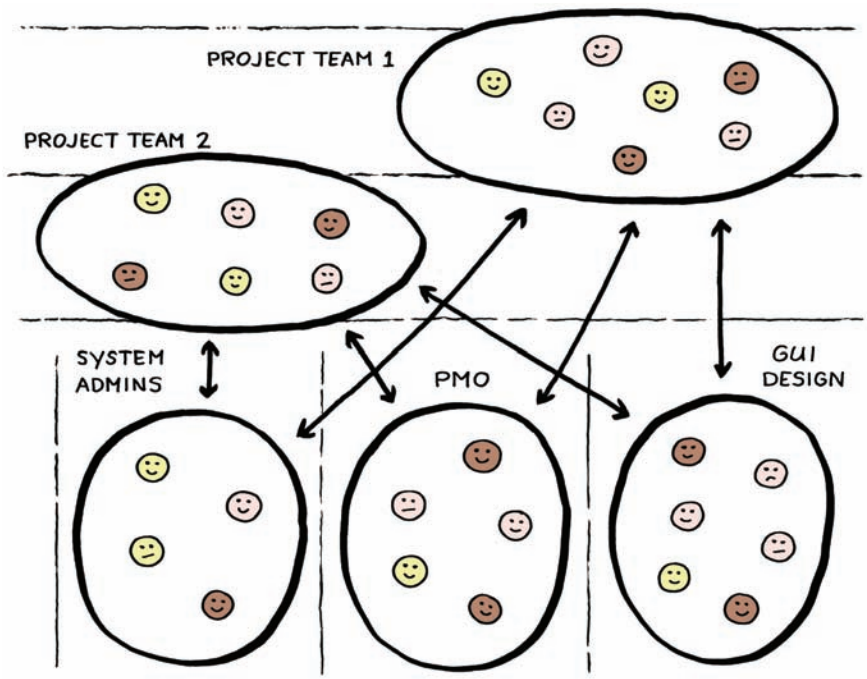


FIGURE 13.8
Project teams serviced by specialist teams.

WHAT IF THE PMO SERVES TOP MANAGEMENT?

That would not be consistent with the picture painted here. The PMO cannot see both the project teams and the management team as their customers. This would lead to a conflict of interest, and usually the project teams get to draw the shortest straw.

I am convinced that project *teams*, not project *managers*, should be held accountable for the results of a project. This requires that top management should work with teams, not with a PMO, either directly or through line management. The PMO, like System Administration and Human Resources, is there to help and coordinate—not to control.

Move Stuff up to Separate Layers

Management hierarchies are like taxi drivers. They are both necessary and evil. Necessary because there needs to be some traceable line of authority between employees and the owners of an organization. And evil because hierarchies are too easily abused, in which case they have terrible effects on information flow. This follows (theoretically) from Emery's first design principle and (practically) from empirical evidence. An example of the latter is found in Malcolm Gladwell's book *Outliers*, in which he described that there is a strong correlation between plane crashes and hierarchical cultures (because of bad communication in cockpits) [Gladwell 2008]. But that doesn't mean that there should be *no* hierarchies. If hierarchies were all bad, we wouldn't find them all around us in nature, as indicated by the **Hierarchy Principle**:

> Complex natural phenomena are organized in hierarchies wherein each level is made up of several integrated systems.[8]

The question is then how to use the benefits of a hierarchy without allowing it to work against us. To me the chain of authority seems to be a valid reason for the existence of a management hierarchy. The owners of

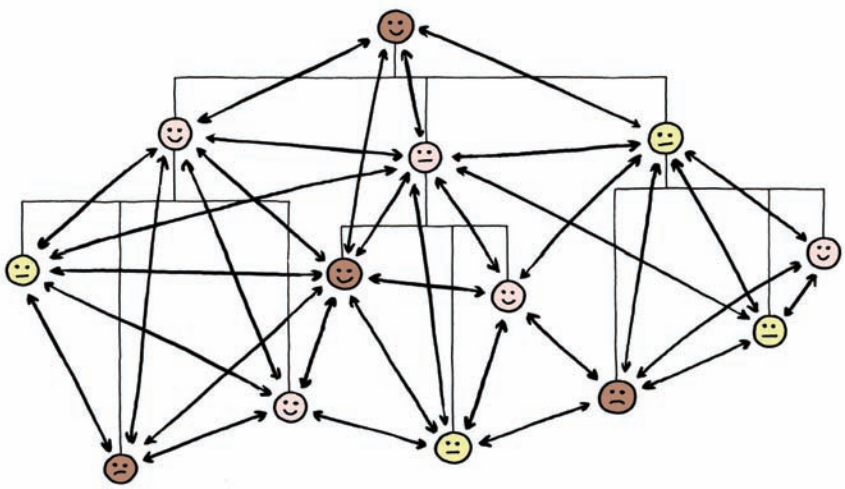
8 Skyttner, L. General systems theory: Ideas and applications, River Edge, NJ: World Scientific. 2001. Used with permission. [Skyttner 2001:93].

an organization hire someone to run their business, and this person hires some other people to delegate part of that work to, and so on. This is a hierarchy. There's no denying it. It is a tree-like structure to facilitate the flow and division of authority.

> The purpose of organization is to reduce the amount of communication and coordination necessary; hence organization is a radical attack on the communication problems…. A tree organization really arises as a structure of authority and responsibility. The principle that no man can serve two masters dictates that the authority structure be tree-like. But the communication structure is not so restricted, and the tree is a barely possible approximation to the communication structure, which is a network.[9]

What we need is a happy marriage of the formal hierarchical structure with the informal network structure [Augustine 2005:48]. Management must acknowledge that information flows through the network and not through the hierarchy. This is not something to be blocked or controlled. Instead it must be nurtured. The hierarchy is needed for authorization; the network is needed for communication (see Figure 13.9).

FIGURE 13.9
Both network (for communication) and hierarchy (for authorization).



9 Brooks, Frederick. The Mythical Man-Month. Reading: Addison-Wesley Pub. Co, 1975/1995. Used with permission. [Brooks 1995:78–79].

Organizational psychologist Elliott Jaques, creator of Requisite Organization Theory, discusses in his works that hierarchies do have a function; although, they are usually badly designed [Jaques 1998]. One important requirement for each management layer is that it must add value to the organizational structure. Just like natural hierarchical layers have new emergent properties at each higher level that did not exist at the lower layers, so must each managerial layer in an organization take care of stuff that the lower levels don't normally concern themselves with.

For example, Jaques describes that each higher level could deal with a different organizational time span [Jaques 1990]. The lowest level deals with all issues that take between 1 day and 3 months to solve; the second level has a time horizon of 3 to 12 months; the third level has work spanning 1 to 3 years, and so on. A project team (usually) has no time to wonder what needs to be done for a business to be successful in 5 years' time. And there are other examples, too, such as hiring people, forging strategic alliances, and balancing budgets, all of which are things that project teams are unlikely to address by themselves. However, it must be noted that management experts don't agree on this matter. Some have noted that even CEOs tend to busy themselves with day-to-day concerns [Mintzberg 2005:110].

I think the real lesson here is that there needs to be *some* separation of concerns between management layers, regardless of whether this separation is by nature temporal, spatial, or anything else. Jaques has shown that organizational problems are often the result of different management layers not clearly adding value. The requirement of adding value is a great starting point when making decisions on management layers. Whenever someone suggests adding a new management layer, ask yourself the question, "What is this layer going to solve that the lower or higher layers cannot do themselves?" If you cannot clearly answer this question, then *don't* add the managers!

How Many Managers Does It Take to Change an Organization?

A trendy thing to say is that having fewer managers is "better" and organizations should be "as flat as possible." True. We all know that. We read it all the time. But the first question people then come up with is, "How many managers should there be?" And the documented answers I could

find range from one for every team [Testa 2009:52] to one for every 100 employees [Larman 2009:241].

But I think the question is a wrong one. The ratio of managers to subordinates in an organization is not some constant you can define. Instead, this ratio is the *outcome* of the measures that managers take when growing the structure of their organization. How many teams are cross-functional and how many are functional? Where is the first design principle applied and in which cases the second? And how free are employees in choosing the teams they want to work for and work with? It is managers who make these decisions. And it is managers who bear the consequences.

> It is a fantasy—a tempting and pervasive one, but a fantasy nonetheless—that it is possible to have great teams without the bother of creating enabling team structures. We hope that markets will make hierarchies unnecessary. That we can have networks rather than organizations. That boundaryless social systems can accomplish work efficiently and effectively. And, when some kind of structure actually is needed, that self-organizing processes of the kind celebrated by complexity theory will create them automatically.[10]

The first concern for managers is growing the best team structures. It makes no sense to discuss the best ratio of managers to subordinates in an organization. But it does make sense to discuss the best rationale for organizational design. The ratio will simply follow the rationale.

Create a Hybrid Organization

The mixing of project teams with specialist teams, and hierarchies with networks, can be called a **hybrid organization**. It is said that hybrid organizations avoid the disadvantages of both functional teams in a purely hierarchical environment and autonomous project teams in a purely networked environment. Companies with less rigid cultures, many projects, and the need for speed, typically arrive at hybrid solutions [Testa 2009:370] [Reinertsen 1997:106].

Some forms of hybrid organizations are called **matrix organizations**. But although I've used that name in the past, I prefer not to use it

---

10 Hackman, J. Leading Teams. Boston: Harvard Business School Press, 2002. Used with permission. [Hackman 2002:130].

anymore. In the available literature on this topic, the term matrix organization for many people seems to imply two organizational "dimensions": line management and project management. Some authors describe the "problems" of matrix organizations, which are conflicts of authority between line managers and project managers, the question of who is the real boss, nasty political situations, and a perceived overhead in the number of managers. [Jones 2001]

Some authors report problems with morale in matrix organizations. If the project manager is in control, the line manager feels demoralized for having responsibility but no control. And it is the same the other way around, with "strong" line managers and "weak" project managers. But I believe all that is just a big misunderstanding. One shouldn't blame the chainsaw for holding it at the wrong end.

The reported problems with matrix organizations are a result of incorrectly implementing hybrid organizations. In a proper implementation, there is *one and only one* line of authority, and it flows through the hierarchy of line managers. Project managers are there to *serve* the teams, not to control them. Project managers are there to manage *projects*, not *people*. I am convinced that the position of project managers should be no different than that of software architects and QA managers, who all have their own responsibilities. By the way, this also makes it clear that there are usually more than two "dimensions" in a hybrid organization. Only one line goes up (through line management), but many lines go sideways.

The Anarchy Is Dead, Long Live the Panarchy

Big projects have a higher chance of failure than small projects, primarily for sociological and communicative reasons [DeMarco, Lister 1999:4]. Some sources even claim that the odds of successful completion of a project disappear almost completely with large-scale projects [Yourdon 2004:4].

But I'm an anarchist and an optimist. I believe we *can* solve these problems by breaking things down and then blowing them up—figuratively speaking, of course.

Agilists and anarchists break up big projects into small projects, and they break up large organizations into small organizations. Then they blow things up by scaling the small working parts to similar-looking big working parts [Highsmith 2009:272]. An Agile organization is the inverse of bureaucracy through top-down planning. It is adaptability through bottom-up growth.

With the rise of global markets, the Internet, social networks, and other network-like developments, there is a global trend that looks similar to the emergence of Agile organizations. On a transnational scale, such a network is called a **panarchy.**[11]. I love the word because it is just one letter removed from my natural state of mind.

> The emerging **complexity** of our social and political structures, composed of many interacting agents, combined with the increasing importance of **network forms of organization**, enabled by technologies that increase **connectivity**, propels the world system towards a transformation that culminates in a global political environment that is made up of a diversity of spheres of governance, the whole of which is called **panarchy**. To clarify, global linkages between individuals and groups create transnational networks consisting of shared norms and goals. […] Panarchy is governance as a complex adaptive system of anarchical networks that relies on diversity and resists hierarchy in order to function and adapt.[12]

A panarchy is a system of overlapping networks of collaboration and authority. As an individual, I subject myself not only (unwillingly) to the authority of my government, but also (willingly) to that of my bank, my Internet and energy providers, Twitter, Facebook, and LinkedIn, sports and game clubs, nonprofit and charity organizations, and foreign governments when I'm traveling abroad. (And other people can add religious organizations to that list.)

There are many sources of authority in the world, and as an individual I choose to subject myself to the rules and norms of any group or organization that I want to participate in. The only one I cannot choose directly is my government. (Unless I pick up my stuff and move somewhere else.)

These days being an anarchist is not what it used to be. I now call myself a panarchist. A panarchist is an anarchist who is acting peacefully. Brian Marick, one of the original signatories of the Agile Manifesto, has similar ideas and calls it *Artisanal Retro-Futurism crossed with Team-Scale Anarcho-Syndicalism*.[13] But I think the word panarchy is easier. And I hope the stickers are cheaper.

---

11 http://www.mgt30.com/panarchy/.

12 Hartzog, Paul B. "Panarchy: Governance in the Network Age" http://www.mgt30.com/network-age/, 2009. Reprinted by permission of Paul B. Hartzog. [Hartzog 2009].

13 http://www.mgt30.com/arxta/.

> The rise of global network governance is a process that is to some extent shaped by states, but it is not controlled by them, and it is also shaped by corporations, individuals, non-governmental organizations, and other groups. It is as yet unclear if any one of those entities trumps the others, although realists would claim the state holds the trump card, and Marxists would claim that it is capital that is in the driver's seat. History has shown that ultimately it is the people who are in charge, and the new connective technologies have only increased their power and ability to organize collective action.[14]

We can now understand why true Agile organizations are panarchies. And because they are networks of value units we may also call them "value networks." They have multiple sources of authority within the Agile organization, including those dealing with architecture, GUI design, project management, and infrastructure. Each value unit can subject itself, *willingly*, to the rules and norms of some specialist groups. But they can also form such functional teams themselves or simply decide to do everything inside their own team. There is plenty of freedom to be anarcho-syndicalist or peacefully anarchist. The only choice people usually cannot make themselves is that of line management. Unless they move to another organization.

A value network is an organic approach to organizational design, resulting in a fractal-like structure of small hierarchies that are all superimposed on one another in one big network. And because it favors scaling out over scaling up, there is no end to the growth of a panarchy.

Have No Secrets

Now that you know what your choices are in designing your organization it is time to spend the last few pages of this chapter on the communication flowing through the structure you created.

As I wrote earlier, most problems in software projects are the result of bad communication. For proper communication people need good information, good relationships, and good feedback.

---

14 Hartzog, Paul B. "Panarchy: Governance in the Network Age" http://www.mgt30.com/network-age/, 2009. Reprinted by permission of Paul B. Hartzog. [Hartzog 2009].

In many organizations, people lack good information, which usually results in people *inventing* it themselves. When they don't know how well their project is doing, they will try to guess. When they don't know how other teams are performing, they will make assumptions. When they don't understand what their colleagues contribute to the organization, they will invent their own reasons. And when they don't know anything about their manager's personal life, they will gossip about it.

To prevent such problems, you should make information available and accessible. And in general, *more is better.* Give everyone access to the Internet, all network folders, project information systems, and source control systems. Make books and magazines available, promote your company's intranet, and publish time registration reports, project burn charts, profit and loss figures, and other kinds of corporate information. Withholding information is (in general) a bad thing. Don't just assume that nobody will be interested in something. You may be right, but keeping information to yourself is not a good thing, because people *will* communicate *something*, and it can only mean that *other* (mis)information gets passed around. And opening up not only applies to your information systems. You have to be honest yourself as well because talented people want to hear the truth about themselves and about the organization. [Kaye, Jordan-Evans 2008:204]

I have often tried to make sure that plenty of information is available for everyone. I want people to see who is working on which projects and which features, bugs, and issues are handled by whom, and what the team members' evaluations are of those projects.

In tough economic times, it is particularly important to make everyone understand what the organization's financial performance is. As Jack Stack wrote in *The Great Game of Business*, only when employees care about financial figures, they will think of ways to improve them [Stack 1994].

Some great managers argue that, ultimately, even people's salaries should be made public, including the salary of the manager. After all, if you cannot explain a person's salary to everyone else in the organization, how can you expect people to trust you as a manager?

I think I can agree with that. But I also understand that you cannot change an organization's culture overnight. It would be unwise to start communicating people's secrets when there's no culture of doing so. But you have to start *somewhere*. Jack Stack lists ten "Higher Laws of Business," of which the last is called "Shit rolls downhill." It means that changing an organization begins with changing management.

Well, *someday* I hope to be a great manager. So I have made sure that my personal "secrets" are published throughout this book. Have you spotted them?

Make Everything Visible

I once started following Ashton Kutcher on Twitter. I didn't really think about the decision for long. It was just that Ashton was the first person in the world to have 1,000,000 followers on Twitter. So, except for the looks, there had to be *something* interesting about this guy, right?

Ashton Kutcher was visible. Stories about his race with CNN to be the first with a million followers could be found all over the Internet. For someone like me, reading many social networking blogs, it was very hard *not* to see this. *That's* why I followed Ashton Kutcher.

So, how do you make people follow practices? Easy. Make them *visible!*

Last year, some managers and I introduced "big visible charts" in the form of task boards for every development team. Anybody walking around the office could easily see them. So, when other (nonsoftware development) teams noticed these task boards, they wanted them as well! *They saw and they followed.* And this principle doesn't just work for task boards. *Any* visible process is an **information radiator**.

My last team did its stand-up meetings in our open office space as well. We first considered doing stand-ups in a more secluded area so as not to disturb our colleagues while discussing our project for 15 minutes. But we decided against that. Then it soon turned out that, again, other teams (including nonsoftware development teams) started following the same practice. They saw our teams doing stand-ups every morning, and they decided to try this interesting practice, too.

To see is to follow….

People copy each other's behaviors, sometimes for no other reason than just seeing them. It's a human thing. It's why I started following Ashton Kutcher. And it's why teenagers start smoking. Scientists say humans often mimic each other unintentionally. But this fact can be used intentionally, too. Mimicry has a great potential to be used for influencing interpersonal persuasion and communication. You can use mimicry to your advantage by making sure that good behavior is visible. If you want people to write better code, plaster the best code you have all over your coffee machine. If you want other people to follow Scrum practices, post

times and locations for sprint planning and review meetings on your company's public calendars. If you want people to use proper source control and branching techniques, draw the source control tree and its branches on your office walls.

People follow what they see, and you must show that which is good.

And perhaps you should refrain from showing examples of bad behavior in your office. People might (unintentionally) follow them.

Connect People

In his book *Fired Up or Burned Out,* Michael L. Stallard shows us that one of the best ways to achieve organizational excellence is to "connect with people." And in their book *Love 'Em or Lose 'Em*, Beverly Kaye and Sharon Jordan-Evans describe the concept of "creating connections," which they call one of the 26 engagement strategies [Kaye, Jordan-Evans 2008:113-122].

Creating and maintaining meaningful connections with employees (and *between* employees) is not just some fancy way of making managers seem more human. As we saw in Chapter 12, the need for connections is rooted in complexity theory.

Resilience and innovation in an organization are the result of people having good relationships with each other so that information flows freely and undistorted. You have to make sure that people enjoy working together. Remove cubicle walls, have informal meetings, facilitate coffee and smoke breaks, and stimulate that people enjoy each other's company at lunch or dinner.

And try and engage in more meaningful relationships with your employees. It doesn't mean you have to be close friends with everyone. That's not even possible. But simply knowing a little more about their life, their families, their home, and their hobbies (and them knowing some more about yours) would be a great start.

Aim for Adaptability

At the beginning of this chapter, I noted that no single structure is the definitive answer for all organizations. Not cross-functional teams, not matrix organizations, nor whatever. The most important thing to take away is that you need to work on the organizational ability to change. It should be OK for functional teams to morph into cross-functional ones

and back. It should be OK for teams to spin off specialist teams, and then break them up again later when they have no need of them anymore. It should be OK for management to try the second design principle in some part of the organization, and then replace it again with DP1 if that didn't work out well. It is only natural that complex adaptive systems constantly revise and rearrange their building blocks as they gain experience. In organizations it is no different [Waldrop 1992:146].

Organizational adaptability calls for a minimum specification of organization. The less that is defined and frozen into formal charts, contracts, and procedures the better.

> Applying a "barely sufficient" principle to your team's organizational design will afford it the flexibility and freedom to self-organize. At times, some managers have tended to go overboard in attempts to comprehensively define organizational elements such as roles, responsibilities, policies, and procedures. Instead, a holographic structure limits design to just the critical minimum specifications.[15]

You know you have achieved organizational adaptability when employees stop complaining about reorganizations and start suggesting new structural changes. Then you can simply enjoy watching the organization grow, and you will have achieved the purpose of the fifth view of Management 3.0.

Summary

Because of changes in the environment, organizational size, products, and people, it is important to change organizational structure regularly. Implementing the concepts of generalizing specialists, wide job titles, and informal leadership greatly improves organizational adaptability.

Team boundaries need to be watched carefully because people cannot identify with a team if team membership is unclear or unstable. Various research studies seem to indicate that between three to seven people is a good team size.

Teams can be organized as either functional or cross-functional units, with the latter being the most obvious choice for optimal communication,

---

15 Augustine, Sanjiv. Managing Agile Projects. Upper Saddle River: Prentice Hall Professional Technical Reference, 2005. Used with permission. [Augustine 2005:58].

though exceptions may exist. Communication between teams happens either via managers or primarily via the teams themselves. Again, the latter is usually preferred.

Organizational structure is most adaptable when teams work as value units, considering other teams as their customers to whom they must deliver value. New teams can be constructed when there is demand, but they must be dissolved when demand among other teams evaporates. Management layers can be beneficial to an organization provided that they too truly add value.

With authority flowing through teams from different directions, we have what is called a hybrid organization. We may also call this a panarchy or value network, when the organization primarily works as a network, with (optionally) multiple overlapping hierarchies.

Last but not least, for optimal communication it is important that managers have as few secrets as possible, make all information they have visible, and make an honest attempt at connecting with their people.

Reflection and Action

Let's see if you can apply some ideas from this chapter to your organization:

- Consider the people in your team. Are they generalizing specialists (or specializing generalists)? If not, what will you do about that?
- Review the official job titles in your organization. Are they wide enough to cover different roles? If not, come up with a plan to change them and make them wider.
- Consider leadership in your team. Are there informal leaders among the team members? Are these leadership roles dynamic enough so that they can change easily when needed?
- Review how teams are constructed in your organization. Are the teams small enough so that people can feel they are really part of a team? Does team membership last long enough for rules and leadership to emerge? Are the teams cross-functional?
- Review the quadrant of organizational styles. Which style are you using now in your organization? If it's not the fourth style, do you have a plan for getting there?

- Discuss value with your team. Does the team see itself as a value-delivering unit? Do they feel that other teams also consider themselves as value units? If not, can you do something about that?
- Review the management positions in your organization. Are all of them adding real value? If not, can you address or influence this issue?
- Draw the organizational structure of your business. Does it look like a hierarchy or like a value network?
- Check your own social skills. Are you connecting with people regularly? If not, how will you change that?

This page intentionally left blank

Index

H

I

N

O

This page intentionally left blank