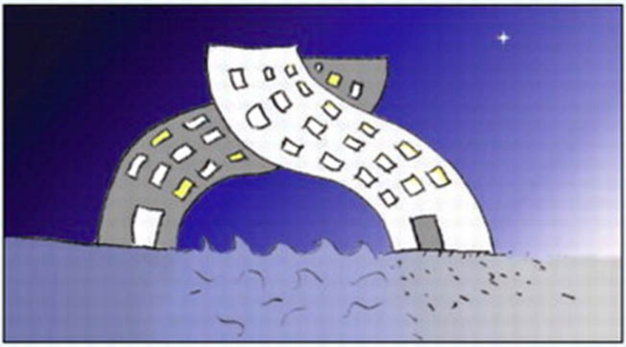
Agile Software Development *with* Distributed Teams



Staying Agile in a Global World

by Jutta Eckstein

DH

Agile Software Development *with* Distributed Teams

Agile Software Development *with* Distributed Teams

Staying Agile in a Global World

Jutta Eckstein

DORSET HOUSE PUBLISHING
3143 BROADWAY, SUITE 2B
NEW YORK, NEW YORK 10027

Library of Congress Cataloging-in-Publication Data

Eckstein, Jutta.
Agile software development with distributed teams : staying agile in a global world / Jutta Eckstein.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-932633-71-2 (alk. paper)
1. Agile software development. I. Title.
QA76.76.D47E2823 2010
005.1--dc22
2010009762

Cover design: Claire Veligdan, Dorset House Publishing
Front cover and chapter illustrations: Katja Gloggengiesser, www.grellgelb.de
Chapter-opening quotations translated from German by Jutta Eckstein.

Distributed in the UK and EEC by Computer Bookshops Ltd., based in Birmingham, England (computerbookshops.com); in the English language in Singapore, the Philippines, and Southeast Asia by Alkem Company (S) Pte. Ltd., Singapore; and in the English language in India, Bangladesh, Sri Lanka, Nepal, and Mauritius by Prism Books Pvt., Ltd., Bangalore, India.

Printed in the United States of America

Library of Congress Catalog Number: 2010009762

ISBN: 978-0-932633-71-2

12 11 10 9 8 7 6 5 4 3 2 1

Digital release by Pearson Education, Inc., June, 2013

Acknowledgments

Books are in the air.
The author provides only a bridge between material and transcript.
—Marguerite Duras

One significant problem with conducting such a project as writing a book is remembering all the people who have supported me. This is, in my opinion, a problem that is unsolvable because almost everyone with whom I communicated during—and even before starting—this project has contributed by sharing experiences or providing feedback.

Unfortunately, it is impossible for me to remember everyone I've met within the past ten years. Therefore, I begin my acknowledgments by apologizing in advance to those I fail to mention explicitly. They are the people on teams all over the world with whom I have had the opportunity to work while learning not only about agile software development, but also about similarities and differences between cultures, interactions, and, not least of all, food. These contributors are as well the countless people who have helped me reflect on my experiences, enabling me to transform and codify what I learned into something tangible and explicit. I am grateful for time spent with these reflection partners, whom I've met at workshops, talks, tutorials, and panels at many different conferences—ACCU in the United Kingdom, Agile in North America, JAOO in Denmark, Retrospective Gathering in North America and Europe, and XP in Europe, to name just a few.

To those whose names gratitude has made indelible in my memory, I also give thanks: David Hussman, Naresh Jain, Nicolai

Josuttis, Daniel Karlström, Michael Kircher, Debra Lavell, Ainsley Nies, Joseph Pelrine, and Linda Rising. I am ever indebted to each of you for writing an expert box for this book, thus sharing your invaluable experience and providing an additional perspective on the subject at hand.

Thanks as well to my reviewers for generously sharing their considerable knowledge and for guiding my attempts to get it right in a book: Jamie Allsop, Joseph Bergin, Magnus Christerson, Lise Hvatum, Carsten Jakobsen, Michael Kircher, Yi Lv, Ken Pugh, Bas Vodde—you are the best.

To Wendy Eakin, Claire Veligdan, and all the folks at Dorset House Publishing, thank you for again proving that the editorial standard still flies high. For professionalism in turning my work into a readable book, *Danke schön.*

Last—but definitely not least—I give thanks and more to my family, who never seem to grow bored listening to my stories of travel and work, forever helping me see differences and similarities among people around the world. Foremost, I give thanks to my partner, Nicolai Josuttis, who not only provides great support for my professional life but, even more importantly, enriches my personal life in most wonderful ways; to my cousin Katja Gloggengiesser, whose delightful illustrations make this book more vivid; and finally, to my sister, Eva Eckstein, whose brilliant recommendation again encouraged me to write about global projects while hidden away on Hiddensee, the same little island in the Baltic Sea where I wrote *Agile Software Development in the Large.* I thank you, all!

Contents

Preface

We are all called to be pontifeces—bridge builders.
Various rivers already have a crossing,
At many others we stand at diverse waters
And look for pontoons,
For a footbridge, for communication.
No sea separates creation and technology,
But, often, speechlessness.
—August Everding

Several years back when I was preparing the manuscript for *Agile Software Development in the Large,* I encountered only a few people scaling agile methods up to use on large projects, teams, and organizations. Since then, many people have discovered that agility works for projects of all sizes, that agile methods are not only applicable on small teams.

There is, I have learned, a big difference between the mostly large projects I worked with five years ago and the ones I work with now. To my mind, the most significant difference is that almost all large projects today comprise work and teams spread over multiple locations and time zones. These days, even small projects are not necessarily collocated.

For me, the biggest change between then and now is that I have to travel a lot when serving distributed projects as their change agent, project coach, or consultant. I find this exhilarating—and yes, at the same time, exhausting—because travel furnishes me with many amazing opportunities to learn about different processes and cultures. One lesson learned is that my experiences in scaling agile processes up can be transferred to distributed projects, because distributed and large projects share some of the same challenges, but there are a lot of other challenges that are more difficult to overcome and that require special attention if you don't want to lose overall agility when spread over the globe.

Consequently, I've made the focal point of this book the many distributed software development projects that successfully follow an agile approach. My objective is to illuminate best practices for applying agility when project members spread over the globe. I hope you will find the material in this book both helpful and enjoyable to read, possibly even taking the book on one of *your* trips to one of *your* distributed teams. Whatever the circumstances, I am curious to learn about your experiences and invite you to visit the book's Website at http://distributed-teams.com. And if you are so moved, please provide comments at feedback@distributed-teams.com.

November 2009 —J.E.
Braunschweig, Germany

Agile Software Development *with* Distributed Teams

This page intentionally left blank

3

Building Teams

It is impossible to create joint plans with people who strive for different goals.
—Confucius

The basic idea behind agile development is to provide to the customer, at any point in time, the highest possible business value in terms of working software. Even collocated teams find it challenging to stay focused on this goal, and it is increasingly difficult the more distributed a project is, especially if the global project is very large, with many people working at different sites.

Building a team with a flexible structure is one key to reaching this goal. However, a large, distributed project structured as one single team is likely to be neither manageable nor flexible. For a manageable and flexible project, divide staff into sub-teams optimally with no more than ten members each.

Typically, distributed projects select a team structure they hope will reduce the channels of communication required between different sites. One approach has been to adopt a traditional linear or waterfall structure, which directs that sub-teams form based on classical project life-cycle phases, activities, or roles such as analysis, design, programming, test, and so on. Following this structured approach, analysts may be located at one site, designers at another, programmers at a third, and so forth.

According to studies reported by Shao and Smith-David,[1] adopting a waterfall approach for distributed software development usually means that each phase is conducted by a specialized

[1]Shao and Smith-David, op. cit., p. 93.

team at one location (phase "headquarters," so to speak), with phase-associated front-end activities (such as preliminary requirements analysis or conceptual architecture design) and phase-related back-end activities (system test, system deployment, or user-training, for example) collocated at the headquarters, whereas mid-life-cycle phase activities (coding, most typically) are transferred offshore to a different location. Forming sub-teams according to activities or phases hinders the collaboration within the whole project. Moreover, distributing the activities across different sites complicates cross-teamwork for the project.

Another approach to dividing distributed projects into sub-teams makes technological know-how its guiding principle. Using architectural layers of the software to define team boundaries, people who concentrate, say, on user interfaces form a sub-team at one site, database specialists work together at another site, and middleware experts function at a third location. Popular on its own, this approach is often combined with the preceding structuring-along-activities approach.

A possible combination of both structuring approaches would be to locate, say, a team of analysts in England, user-interface specialists in India, and acceptance-test developers in Germany. Given such a structure, it is not surprising how often component interfaces, for example, suffer from compatibility issues, or functionality is developed that does not reflect the customer's requirements. Moreover, finger-pointing between sites becomes commonplace because it's completely unclear whose responsibility it is to develop specific functionality. Communication and technological problems such as these can be the result of how teams and sub-teams are structured, making it difficult for anyone to deliver business value during a project's lifetime, and nigh impossible at the end of the project. Without the valuable customer feedback that delivery of business value triggers, project management and staff cannot learn from the customer in order to successfully adapt the system to the customer's needs.

Feature Teams

One proven way to maximize the likelihood of delivering business value to the customer is to organize teams according to customerrequested business functionality. As declared in the Agile Manifesto, "The best architectures, requirements, and designs emerge from self-organizing teams."[2] That is, instead of building teams according to know-how or system life-cycle phases and activities, organize teams according to their experience in delivering the features the customer wants. With expertise in the required domain and delivery of business value in mind, such teams can be chartered to organize both themselves and their work.

Called feature teams, or domain teams, such teams comprise people who possess or are in the process of acquiring all knowledge and skills necessary to deliver a complete feature that provides business value to the customer. Consequently, a feature team may be staffed by analysts, testers, user interface specialists, database experts, and so on, who work together to deliver the required functionality.

Although chosen for their expertise, feature-team members do not work solely in their field, but are encouraged to take instruction from feature-team colleagues to learn different roles. Consequently, each feature team "is a generalist in its domain. . . . But although the team consists of these different experts, those experts will not work solely in their field of specialty. Instead, the team members must take different roles."[3] This way, specialized knowledge can be spread to all members of the team, thereby reducing the risk that the project's success will depend on continued involvement of specific experts. Knowledge-sharing enables feature-team members to contribute to each other's work and fosters a shared vision, allowing them to work toward a common objective.

Single- and Multi-Site Teams

Organizing a feature team across different sites may seem daunting but can be accomplished in the following, relatively straightforward, ways:

- Assemble a *collocated feature team* at one site. This strategy simplifies communication within the team and facilitates synchronization of the shared vision that is

[2]See the Agile Manifesto online: http://agilemanifesto.org.

[3]For more on feature-team roles, see Eckstein, op. cit., pp. 57-58.

Feature team . . .

necessary in order to achieve the goal of delivering the feature at the end of an iteration. A primary requirement for a collocated feature team is that all required knowledge (or the skills for acquiring this knowledge) is available at the one site. If it is not, either ask people who possess whatever knowledge is missing to impart that knowledge to the collocated feature-team members, or move those people temporarily to the site in order to keep all feature-team members physically together. Note, however, that even with collocated teams, every effort must be made to ensure conceptual integrity (that is, to establish the same look and feel throughout the whole application), among other things.

- A *dispersed feature team* is one that is established across different sites, and is optimal when roles that are needed for the team are not available at a single site and educating people so that they may adopt the roles is not feasible. One drawback can be that internal team synchronization may require a high degree of organizational effort. That notwithstanding, Jamie Allsop

> reports from his globally dispersed team that "some of our greatest benefits from using an agile methodology were centered around the better utilization of our communication bandwidth. Included in this would also be the rhythmic synchronization at the daily, iteration, and release time points."[4] By working on the same set of features while at the same time focusing on delivering business functionality at the end of an iteration, members of a dispersed feature team share and are motivated by a common goal. Individuals grow to work together as a team. For this to work well, individuals on dispersed feature teams need to share common ideals (a topic we'll look more closely at in the next section).

Both types of feature-team structures possess inherent advantages and disadvantages. An advantage of dispersed feature teams is that communication across teams is made easier by the physical proximity of members collocated at the same site, which helps ensure conceptual integrity among myriad feature teams. In all honesty, I was originally surprised to learn that dispersed teams are not always at a disadvantage. For example, setting up a dispersed feature team is advantageous if you are working in a large, distributed setting with many teams. Then, dispersed team members can maintain communication with other collocated project members, even though they belong to other (dispersed or not) feature teams. I recently heard about a project distributed over two locations (The Netherlands and India) that intentionally creates only dispersed teams. The project's positive experience is based on the reality that project culture unifies and permeates all teams equally. An additional advantage of dispersed feature teams is that the structure "reduces the us-them thinking between the different development sites."[5]

Carefully balance the advantages and disadvantages of collocated and dispersed teams and use a structure that best fits your setting.

---

[4]Thanks to Jamie Allsop for sharing this experience.

[5]My thanks to Bas Vodde for this insight (personal communication).

Dispersed Teams

Teams accomplish together what individuals cannot accomplish on their own. In order for a feature team to accomplish a task, team members first need to be aware of the fact that they *are* a team, and not several individuals working in common. Together, team members need to do the following:

- *Share a team identity:*[6] Every team member should identify with and feel part of the team.
- *Share a common vision:* Team members need to work toward the same goal.
- *Acknowledge joint responsibility:* Every team member has to commit to sharing responsibility for the work the team promises to deliver.
- *Adhere to collaborative rules and guidelines:* Team members need to have and understand a common protocol that determines how they will work together.
- *Appreciate a joint set of values:* Team members need to agree to what they as a team value (and what they don't value) and use these values to guide their work.

The catch is that no one can instill these characteristics in team members by mere command. Rather, they have to be created, established, and experienced by the team. To achieve this in a timely manner, team members should work together, collocated, for a relatively brief period of time. Collocation provides an artificial environment for members of a dispersed team, yet it still facilitates the process of evolving common rules, guidelines, and an agreed-upon set of values.

Unfortunately, starting with a dispersed team in its *natural* (dispersed) environment will prolong the time it takes team members to gain mutual respect and trust, but the process will focus individuals' attention on the most important challenges they face in their natural environment, thereby creating proximity despite many forms of distance.

[6]For insight into *group* identity, see M.L. Manns and L. Rising, *Fearless Change: Patterns for Introducing New Ideas* (Reading, Mass.: Addison-Wesley, 2004).

Forging a Team

If you are working with a dispersed team, you will have a better chance of establishing a good working relationship among the team members if you enable them to work together, preferably at the beginning of the project. Ambler suggests that in order to help a team jell, members should work together for at least one month; two months would be better.[7] Depending on a team's dispersion, adopt one of the following strategies:

- If the distance between the sites isn't too great (for example, 500 miles or less), team members can travel to one of the feature team's sites weekly, possibly spending several days each week at that site. On one of my recent projects, guest workers stayed at a feature team's site three days a week, a schedule that was feasible because this dispersed sub-team was spread between Austria, the Czech Republic, and Hungary, countries that are only a few hours' train ride or brief flight apart.
- If the distance between sites is great, implement a strategy whereby team members work at a host location less frequently but for longer periods of time. This is also known as the concept of *expatriates.* Imagine a feature team composed of five Chinese and two American developers. A way to help the team jell would be to have the two Americans work for, say, two months in China at the start of the project.

The important thing to remember is that face-to-face meetings are the most effective way to create solidarity and intimacy among team members. McKinney and Whiteside, who conducted a survey of more than two-hundred individuals working in virtual teams, quote one manager regarding dispersed team relationships:[8]

[7]S. Ambler, "Bridging the Distance," Dr. Dobb's Portal, http://www.ddj.com/dept/architect/184414899, August, 2002.

[8]V.R. McKinney and M.M. Whiteside, "Maintaining Distributed Relationships," *Communications of the ACM,* Vol. 49, No. 3 (March 2006), pp. 82-86.

> ". . . electronic partnerships work extremely well as long as there has been a relationship built in advance."

Building a traditional working relationship between team members before initiating dispersed work increases peer awareness, which in turn motivates individual team members to collaborate actively.[9] Magnus Christerson emphasizes the point, noting, "what and how [teams] work is more important than how long they work. Time is not the critical factor—the critical factor is trust and how long it takes to build. Trust depends not only on people, but also on cultures and values."[10]

Teams Happen
***by* David Hussman**[11]

Several years ago, I coached a team "across the pond." As a coach who works in many domains and industries, in many companies and countries, I find it can be hard to know what, or who, will be at the next gig. So when asked to coach a "team," I never know what to expect.

As is the case for many of my gigs, this team was an offshore group, in Ukraine, providing software development services to a U.S.-based management group. The projects were bid on and organized in the U.S., and the requirements were passed on to the developers in Crimea.

When I arrived at the airport, no one was there to meet me. Without knowing much of the language, I was a bit concerned. Then, out of nowhere, a group of friendly faces rolled out of a packed vehicle. This was clearly the team. I immediately felt a team vibe.

Once I was settled in my hotel, we headed to their work space for our first day together. Upon entering the door to their space, I could again sense the team spirit. It was not that the space was flowing with *feng shue* or

[9]For information about team-member collaboration, see B.J. Koh, et al., "Encouraging Participation in Virtual Communities," *Communications of the ACM,* Vol. 50, No. 2 (February 2007), pp. 69-73.

[10]M. Christerson, personal communication.

[11]D. Hussman (USA), software anthropologist and agile coach, www.devjam.com.

beautiful wall art. Instead, there was a palatable presence of work happening and people working together.

They showed me around and told me about the space. It turned out that more than one of the team members, and a family helper or two, had built the room and hung the wallboard. They had even built a long U-shaped table where most of the developers sat together each day.

As we worked together, it was clear they were truly a team. They were struggling with problems common for offshore teams. One of many problems was the lack of face-to-face connections with the customers, the kind of connections that help produce better software products in less time.

As an offshore team and part of a newly formed company, the Ukrainians knew they needed to produce. They had been working hard to find various ways to make better connections with the customers and other users, sometimes with success, but, many times, their efforts returned little.

Around about lunchtime, we all headed to the kitchen. Someone's mother had brought us food and we all sat down to eat a wonderful lunch; the same meal was shared by all. The conversation ranged from coding to life and family. When we finished eating, each developer washed his plate and utensils and put them away. I thought to myself, *This might work well as a team-building exercise for the many large corporate groups who lack any real sense of team, but that is another story.*

As the gig proceeded, I was continually impressed by the way they worked together. Many of the agile values and practices were easy to introduce because of the existing team vibe. The lunchtime experience was only one example of how they kept their team whole. Humor was another, and the need to produce was another.

I did not bring anything that made them a team. They were a team long before my plane landed. What I offered was a set of practices that helped them make better connections within the team and across the pond. Some of the practices we used to help make customer connects were short development cycles, automated acceptance tests, and user stories with personas (short descriptions of people who might use the product and what they value in the product). I am not sure which of the practices were most helpful, but I think the personas truly helped the developers start seeing across the ocean and into the lives and values of their intended audience.

Once we had the personas, we created a collection of user stories with acceptance tests. The tests were another simple and strong way to connect remote developers to product value and improve the iterative output. Instead of hoping that they were producing the right thing, they now had a tool for communicating what would be deemed "done" in a concrete and automated way.

I am not saying that a few personas and automated tests do span the entire Atlantic Ocean, but they do help improve the bandwidth of communication. In this case, they provided a tactic to make connections where there once were none.

I would like to think I did a good job coaching this team. (I was told I did.) As I flew home, I made some notes about the gig. I was surprised how often I typed the word "team." As my sports metaphors are slim, "team" is not a word I often use. I tend to lean more toward "community," as I think it better captures the essence that exists when people bond around creating great products. But the word choice does not matter. This Ukrainian community developed that special combination of people, trust, respect, and skills that helped them work with and for each other at the same time. Agile practices did not make them a team, but such practices did foster experimentation and learning to find ways to improve.

Remote teams should assume that, to succeed, they need to embrace iterative development and agility as simple tools that foster collaboration and community. Instead of saying, "How can we practice agile methods," a better question is, "How can agile methods help us succeed?" Agile methods are tools that remote teams or distributed communities use to make the connections needed to deliver the right product to their customers.

Roles

To be fully functional throughout a project's life cycle, a feature team typically requires that a variety of roles be performed. For each role described below, the person or people responsible for performing it will be most effective if they are able to travel to the different teams, as needed. Direct, face-to-face communication allows role-players to gain a comprehensive understanding of project progress and ensures that key information is not only communicated well, but is also understood by all other members of the team.

There are several basic premises that pertain to feature-team roles, as follows: Most roles need not be fulfilled by one person for the duration of the project; most role-players can assume several roles as time and rationale permit; one role rarely equates exactly to one person's responsibility. Determining and mapping roles and people's responsibilities depends on such factors as the qualified people at hand, their knowledge of the risk implied by the system being built (for instance, their familiarity with specific technologies and the business domain), and the size and distribution of a team.

Feature-Team Constellation

The baseline of operational feature teams is that they either already comprehend or are capable of acquiring required knowledge to complete a unit of business functionality. In addition to each individual's domain and technical know-how, a team must be able to *deliver* functionality. Thus, feature teams typically need to include all or most of the following roles:

- *Architect:* ensures the conceptual integrity of the system (a topic we'll look more closely at in the next section)
- *Database administrator:* creates and/or maintains databases
- *Designer:* conceives and directs a coherent design of required features
- *Documenter:* provides necessary documentation for developed features
- *Domain expert:* helps teammates to understand the domain
- *Infrastructure specialist:* ensures that the development environment is working and supportive
- *Integration expert:* possesses the know-how to integrate, build, deliver, and deploy a feature within the whole

system, and ensures a working configuration management

- *Programmer:* codes the required functionality as well as accompanying unit tests
- *Tester:* works closely with the domain expert to define acceptance criteria for business functionality and tests for acceptance
- *User interface designer:* knows how the user interface should appear and feel, and then creates it accordingly

In many of my projects, the whole feature team shares the responsibility for each role. This means that any feature-team member will take on any role when needed. We seldom have people dedicated to fulfilling only one of these roles. To refer to all of these required roles, we typically use the general term "developer." The number of people on a feature team varies, but I recommend that there be seldom fewer than three members and never more than ten. A good rule to follow for team size is the Miller rule of seven, plus or minus two. Sangwan, et al., further recommend that "no team should be larger than ten staff members, and no single development site should have more than 100 engineers (or ten teams of 10)."[12]

Sometimes, if a feature requires a special technology or connectivity, a project will require temporary assistance from specialists gathered from outside of the feature team. When this happens, I recommend that the experts become members of the feature team within a given time frame (for example, one iteration). This may require that the experts travel to the site where the feature resides. Such experts don't stay with a specific feature team for the whole project but instead work as needed to support a specific feature team.

Establish a similar relationship between a mentor/trainer and a feature team requiring support or specific knowledge: Mentors work with a team for as long as necessary to transfer the knowledge. This may mean a mentor travels to a remote feature team or to several sites to train members of a dispersed team. Efficiency and effectiveness depend upon working together in-person with mentors. Avoid the monopolization of knowledge—that is, a situation in which a few elites are regarded as irreplaceable experts.

[12]R. Sangwan, et al., *Global Software Development Handbook* (New York: Auerbach, 2007), p. 97.

Monopolization presents a big risk in that, just like everyone else, experts take vacation, become sick, change jobs, and so on, leaving behind a project in limbo.

If possible, I recommend that team members stay together within a feature team over the whole lifetime of the project. The biggest benefit is that a team can build its identity and, once it establishes communication paths, knowledge and conversation typically flow more easily during the rest of the collaboration. Dispersed as well as collocated feature teams can reap these benefits.

Architect and Chief Architect

The architect's main task is to ensure conceptual integrity, independent of the number of feature teams or sites involved. Brooks defines conceptual integrity as follows:[13]

> "Every part must reflect the same philosophies and the same balancing of desiderata. Every part must even use the same techniques in syntax and analogous notions in semantics. Ease of use, then, dictates unity of design, conceptual integrity."

Only conceptual integrity enables simplicity—and simplicity in turn enables maintainability. In my experience, "on a typical agile project with a small team, you will often end up without an architectural lead because the whole team is of equal value and takes the same responsibility for the whole project. Although this could or should be a goal for a large team, too, it would never work, because development would diverge, uncoordinated."[14]

This is one reason why the architect's role is so important. In a project with only one collocated team, the whole team can take responsibility for conceptual integrity. In some projects (still not too big), it might be enough that an experienced developer additionally takes the role of the architect. Depending on the technology used, the content of the project, and project size, the demand regarding the architect's role varies:

[13]F.P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering,* 20th anniv. ed. (Reading, Mass.: Addison-Wesley, 1995), p. 44.

[14]Eckstein, op. cit., p. 127.

- *One project architect:* The one architect oversees/advises all technical decisions and also acts as the major contact for the (lead) product owner in order to learn about technical dependencies between features.
- *One architect per feature team:* If the project is rather complex and unknown, one architect per feature team is needed. However, once the team has gained necessary knowledge and experience, it may only need one or a few architects for the whole project. The position of architect on a feature team is rarely full-time, but rather may be an additional role adopted by a developer.
- *One-to-*n *architects to support all teams:* If you have fewer architects than feature teams, each architect should work with a single feature team for a limited period of time (for example, for one iteration) before moving on to support another feature team.

No matter if you have an architect for every feature team or a group of architects supporting many feature teams, it is important that all architects communicate, and one chief—or lead—architect pulls the strings and ensures they work toward the same project vision. Otherwise, it is very likely that every team or every site, or both, will make its own architectural decisions that will probably differ from one to another. The main responsibility of the chief architect is to ensure that the big picture (from a technical viewpoint) is understood by everybody on the team. The architectural lead should also train the team members by helping them, for example, to see the big picture and to take responsibility. So, the lead architect will not only be the one memorizing the key ideas but, more importantly, the one who spreads these ideas so that more and more people will have the same understanding of the system.[15]

Spreading these ideas and helping the other team members to see the big picture does not refer to concepts only, but rather to the actual system. Thus, the chief architect does not just create, say, documents; he or she may also code. No matter whether a project has a single architect, several architects, or even one architect per feature team, architects must always work with the feature teams and understand that they provide a service for the feature teams.

[15]Ibid., pp. 128-29.

Another important lesson to learn is that the worst architectures are often the result of democratic decisions. I don't mean that the chief architect should dictate architectural decisions; rather, he or she ensures that all opinions are heard and, if required, evaluated before a final decision is made. It is a chief architect's responsibility to ensure that all architects accept and respect that final decision. Final agreement on a decision is key to success. In order to accomplish this, I recommend you follow the process of *nemawashi*, noted in *The Toyota Way:*[16]

> "Make decisions slowly by consensus, thoroughly considering all options; implement rapidly."

The basis for this process is that every party gets a fair hearing and is allowed to provide input, and is therefore involved in the decision-making. Ultimately, this ensures that all participants can agree with the decision and prioritize their own objectives below the project's objectives.

Experiences as a Software Architect on Global Agile Projects
***by* Michael Kircher**[17]

In my roles as architect—in this concrete case, as lead architect of a mid-size development project (about 50 developers)—I have been able to gather experiences regarding the scalability of the role of software architect in an agile project.

The project concerned the development of two subsystems: an embedded device to be integrated in vehicles, and an enterprise-scale information system, laid out as a three-tier system combined with batch processing. In this setting, we had multiple challenges. First, three development locations in Europe were involved. Second,

[16]Liker, op. cit., p. 241.

[17]M. Kircher (Germany), Director, syngo Platform Development, Siemens Healthcare (formerly, Principal Engineer, Siemens Corporate Technology).

the developed software was expected to be reused as a platform for similar solutions, with similar requirements, after the first project was completed. Third, the requirements were not fixed: The customer—represented by a product owner—was elaborating the key use cases as development continued. For me, these challenges meant that I needed a way to constantly evolve the architecture on multiple sites while ensuring minimal complexity in order to secure later reuse. In my experience, this is a very typical problem description for a software architect in a global and agile software development project.

****

A brief note on what software architecture is concerned with: Architecture, in my understanding, concerns everything that is expensive to change afterward, so it concerns not only the global structure of the software but also the employed technologies, decisions for internationalization support, and the like. With this interpretation of architecture, it is easy to see that architecture matters in agile projects. The key difference in an agile project is that the specific architecture evolves during the project instead of remaining as planned in advance through to the project's very end. Maybe a good analogy is to compare between *static and up-front-designed architecture* and *dynamic, constantly maturing architecture*. This is not to say that everything changes in the latter case; far from it. Elementary design decisions, such as layering, partitioning principles, and programming idioms, stay the same, most likely including the initial architectural style, such as Broker, Common Repository, and Pipes and Filters.

****

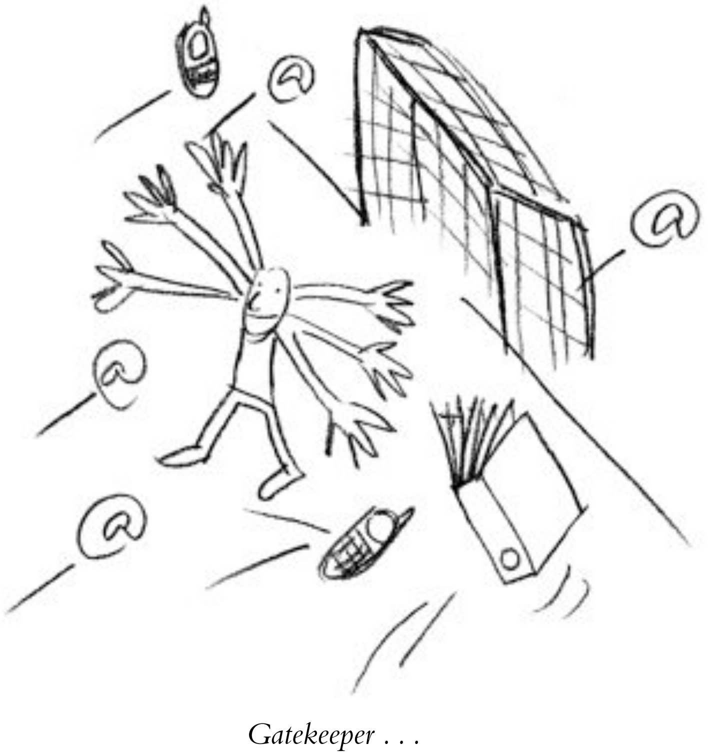
Due to the multiple subsystems and development sites involved on this project, it was obvious that a single architect was not sufficient. We established an architect for every major subsystem and site—which correlated as we tried to avoid splits across sites within a subsystem—coached and guided by me, the lead architect. This orchestration of architect roles allowed the handling of situations that required hard and final decisions. The general strategy is to have one lead architect and subordinate software architects, partitioned according to boundaries of subsystems, problem domains—such as technical architecture (infrastructure) and business (domain) logic—and especially sites.

Concerning the focus and priorities of a software architect, I have a very dedicated opinion: A software architect must ensure, in decreasing order of priority: 1) consistency among design decisions and the resulting architecture, 2) communication among developers and with stakeholders, 3) guidance regarding best-practice in daily design decisions of the team, and, 4) making design decisions. This last point might seem contrary to what many believe, because many expect the architect to make *all* design decisions by herself or himself. In my experience, that is actually the *worst* you could do because of many reasons, the topmost being that developers are not committed to dictated decisions and that the number of required decisions will flood the architect, hence hampering project progress as the architect becomes a bottleneck. The role of the architect can only scale and architects can only maintain control over design decisions if they remove themselves from actual design work and, instead, install themselves as a control, reviewing relevant decisions before implementation. Thus, architects should guide teams to employ wise design practice.

Coming back to agility: The concept of software architects coaching and distributing responsibility aligns quite naturally with the principles of agility, with empowered teams that are aligned to accomplish common goals.

Coach

Every feature team needs a coach, a spokesperson (in Scrum, called the "Scrum Master") to ensure that the team is able to do its work. This doesn't mean that team members cannot take responsibility for resolving matters if there is something in their way (they can), but rather that, in order to move the project forward, there is someone who cares for their issues and who will take problems to the right people. The coach serves also as a firewall or gatekeeper for the team, so that team members won't always be interrupted if someone from outside has a question or request.[18]



Gatekeeper . . .

[18]For more on firewalls and gatekeepers, see J.O. Coplien and N.B. Harrison, *Organizational Patterns of Agile Software Development* (Englewood Cliffs, N.J.: Prentice-Hall, 2004).

Additionally, the coach ensures the location and scheduling of team meetings. This doesn't necessarily mean that the coach personally reserves a conference room—the responsibility should be shared among all team members—but merely makes certain that this type of task is done. A coach also helps team members stay organized and current. For example, he or she may remind team members to verify that a particular agreement is valid, and that if outdated or invalid, it be either eliminated or replaced.

A large distributed project will have several coaches, typically one for each feature team. I find it most effective when a coach is actually a member of the team (avoid, for example, selecting team members' managers) and is collocated with the team. A coach should have an additional role such as developer (the most common case), or tester. However, depending on whether or not team members work smoothly together, the coach may have to defer, say, a development task if too busy fulfilling the job of coach. If the team is dispersed, the coach must maintain adequate contact with *all* team members (by phone and e-mail, and by traveling to their work sites).

Product Owner and Product Manager

Feature-team members need to know which feature needs to be implemented next and whom to ask if they have problems understanding the requirements for the feature. In order to satisfy the customer, the Agile Manifesto stresses, "Business people and developers must work together daily throughout the project."[19]

Not every feature team, nor every customer, can afford to move in together—nor is such a drastic effort necessary. Basically, a feature team needs somebody who represents the customer in a role that often is simply referred to as "the customer" (in XP, called the "on-site-customer"). However, although this role could be filled by an actual customer, it is more often appropriate that the person just represent the customers' perspective (but isn't a customer himself of herself). Having someone represent the customers' perspective is especially helpful if the system will have to serve customers who have differing opinions regarding the future func-

[19]See the Agile Manifesto online: http://agilemanifesto.org/principles.html.

tionality of the system. To differentiate between the real customer and the role-player representing the customer's perspective, some use the label "product manager," or "proxy customer," but I like the Scrum term "product owner" for this role.

The product owner clarifies and assigns priority to the different requirements of all the various customers. In order to do so, the product owner needs thorough knowledge of the customer's business domain, and moreover, an effective communication channel to the different customers. As Magnus Christerson emphasizes: "Product owners/managers need to spend quality time with real customers continuously. I used to have the guideline that 25 percent of the time of a product manager should be spent on direct customer activities."[20]

Product owners on my projects generally come from different areas or departments within the company sponsoring the project. People with a background, say, in marketing, customer support, product management, sales, or business analysis are good candidates. My general rule of thumb is: The person selected must be someone with insight into the customers' domain. Thus, if building a product for developers, a developer is a good candidate to place in the role of the product owner. Who is best qualified depends very much on the system to be built. If several different types of customer need to be served, the product owner's task can be especially demanding. Balancing and assigning priority to the different needs of diverse customers and drawing conclusions from them are major responsibilities of the product owner.

If the system being built is highly complex or the team is not particularly well versed in the customer's business domain, one product owner might only be able to support one feature team. In some of my projects, one product owner was able to support up to three teams; in other projects, we needed one product owner per feature team. Generally, the role of product owner is very demanding! Thus, it is essential to ensure that he or she does not get burned out, caught between the feature team requesting support regarding business knowledge and the real customer whose needs must be met. One way to maintain sanity is to establish a support structure along the following lines:

[20]M. Christerson, personal communication.

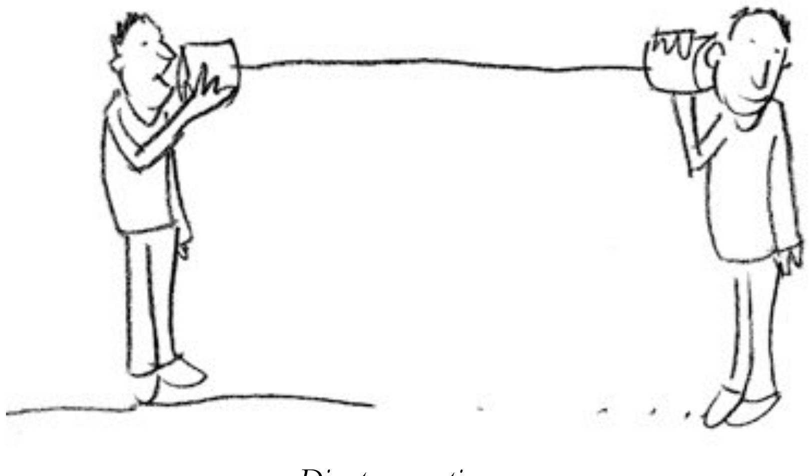
- *Product-owner team:* On a large project, it often is not sufficient to just have *one* product owner because the required tasks are too extensive for one person, but instead to have a team of product owners, each of whom may be able to support one to three feature teams.
- *Lead product owner:* A team of product owners will need at its head a lead product owner to serve as arbiter and final decision-maker in the event of disagreement. The lead product owner's major responsibility is to make priority decisions based on input from customers as well as from the team of product owners. In the same way that the chief architect pulls the strings regarding technical decisions, the lead product owner retains final judgment for business decisions. Depending on the complexity of the system and the project, the lead product owner may have no task other than coordinating the team of product owners and keeping in touch with customers. In most cases, however, a lead product owner also can fulfill the role of a regular product owner, steering one feature team and serving as lead product owner in parallel.

Although responsibility for the motherlode of customer contact belongs to the lead product owner, the other members of the product-owner team need to work with customers in addition to supporting their feature teams. Ideally, product owners are collocated with the feature team or teams they support, and travel to customer sites to get and give feedback and to clarify possible misunderstandings.

The more "business complexity" there is on the project, the closer the product owner should be to the team. Product owners who support one or more dispersed feature teams, of course, will need to travel frequently to the sites involved. At times when the product owner cannot be collocated with his or her feature team, communication can be effectively maintained with off-site contacts and teams by using all kinds of communication media to connect and enrich the feature team's and product owner's conversation. Although not always feasible, product owners should strive to limit the time when they are not collocated with their feature team.

The role of the product owner is the same whether working with an onshore or offshore team. I want to emphasize this point because I every so often hear people state that offshore teams will not have a collocated product owner, gaining access to their product owner only at the project's base location. After all, agility is about focusing on business value and, therefore, it is relatively unimportant where feature teams are located. Every feature team will benefit most from direct support from the business side and thus will always function best if it has the support of a collocated product owner. Matt Simons, project manager for ThoughtWorks, advises: "Assuming that your offshore team is unable to locate a business customer willing to play the role of the onsite customer, you'll have to establish some type of proxy customer."[21]

If there is no direct contact between the product owner and his or her feature team, you need a different product owner—the idea of a product owner is actually to reduce the miscommunication between development and the customer. Otherwise, you're just introducing another middleman between the team and the customer, and nurturing miscommunication all the more.



Direct connection . . .

[21]M. Simons, "Internationally Agile," *Informit.com,* http://www.informit.com/articles/article.asp?p=25929, March 2002.

Project Manager

Several agile methodologies claim that agile projects have evolved and no longer require the role of the project manager. Most traditional agile teams and certainly large and distributed teams, however, stumble without assigning responsibility for project-critical factors such as the following:

- *Politics:* In my experience, the project manager's most important task is to deal with organizational politics. Typically, a project needs support both from inside and outside the company. Establishing and preserving that support is very often a full-time job.
- *Personnel issues:* The project manager should be the knowledgeable and helpful face of human resources for the team, having insight into team pressures and personal problems, giving leeway, say, on personal days and holidays while supporting the team as a whole. Often, the project manager is the one who can solve tough problems that the team coach cannot solve by virtue of being too close to the team or not close enough to company politics.
- *Budgetary control:* The project manager typically is responsible for controlling the budget and providing relevant information to the product owners or the lead product owner. In smaller and less complex projects, this responsibility is and should be in the hands of the (lead) product owner.
- *Hiring:* Project managers usually are also responsible for organizing the search for skilled people based on a project team's input.

The project manager ensures that all project members can do their job so that the project can progress, and, depending on the actual size of the team, several people may be needed to support the project manager fulfilling these tasks.

For a distributed project development team spread across several sites, where the project manager actually is located is comparatively unimportant. He or she will have to travel to all project team locations. However, avoid locating all development in one place and all project management in another. Project management that is not skillfully integrated with the development effort simply amounts to no project management at all: Developers will lack context with which to complete project goals. As Yourdon observes, "the main thing is that the project manager *and* the rest of the team are from the same organization, part of the same culture, and presumably acquainted with one another already."[22]

Collocate Key Roles with Teams

All the responsibilities discussed above—whether formally assigned to an architect, coach, product owner, or project manager—are integral to agile development. I want to reiterate that these key roles function most efficiently and effectively when collocated with their respective team or teams. In fact, each is part of the team.

Every so often, I am surprised by the tendency in some organizations to assign the key roles to project members located at project headquarters. This practice is not really helpful. In order to support their feature teams adequately, the key roles have to be (physically) close to their teams.

Having noted that, there is one exception: When a feature team is dispersed and, consequently, there is no single site where it is located, then there is no specific site where key roles should be located. It then becomes extremely important that people fulfilling these key roles are willing and able to travel (most probably, a lot) to all sites involved, and that they have particularly good communication skills in a dispersed setting.

Ensuring Conceptual Integrity

Correlating the efforts of feature teams will ensure focus on the highest business value. Without an architect, feature teams may grow near-sighted and focus on features rather than on cross-features, like conceptual integrity. Avoid ending up with a system consisting of a hodgepodge of looks and feels, diverse database access, and the like. Conceptual integrity is the basis for a main-

[22]E. Yourdon, *Outsource: Competing in the Global Productivity Race* (Englewood Cliffs, N.J.: Prentice-Hall, 2005), p. 56.

tainable system. Only this allows a simple vision to evolve that facilitates the general understanding of the system as well as future upkeep and changes. Conceptual integrity is also required by the Agile Manifesto declaration of architectural philosophy:

> "Simplicity—the art of maximizing the amount of work not done—is essential."[23]

For every system, and even more so for large systems, simplicity comes from conceptual integrity. Architects are responsible for conceptual integrity. Depending on the size of the project, as well as on the complexity of it, it could be enough to just have one architect or it might be necessary to have a team of architects in place to ensure conceptual integrity.

Starting Team Provides Model

If you have started a project with just one team and this team implements two to three key use cases together with a referential implementation of the architecture, the use cases might already be sufficient to establish conceptual integrity. Referential implementation will act as a role model for all further development, serving as an example on which to expand the system and as a basis for learning more about the business domain and the technology.[24]

Moreover, if you have carefully built this starting team (selecting people from all sites involved), then all sites will learn from it, and knowledge about cross-functional issues will spread across sites.

Technical Service Team

If the complexity and the size of the project make it impossible for the group of architects to maintain conceptual integrity, think about establishing a separate team to provide this support. For example, if the system you are building consists of a complex user interface, you might need to establish a separate team, called a *technical service team,* that is tasked with creating the infrastructure or a

[23]See the Agile Manifesto online: http://agilemanifesto.org/principles.html.

[24]For more on the topic of referential implementation, see Eckstein, op. cit., p. 114.

framework for it. This strategy allows feature teams to more easily build user interfaces. Another reason to form a technical service team is to build different products based on the same architecture. Thus, the technical service team will provide the foundation so that the feature teams can build their features on top of it using the same concepts.

Key to success for a technical service team is that the work it delivers is a real service for the feature teams. This means that the technical service team understands that the feature teams are the "customers" stating the requirements. It is not the technical service team that comes up with ideas independent of their customers' needs (although I have seen the latter far too often). No, it is that technical service teams should always regard themselves as pure service providers for the feature teams.[25]

Feature teams in turn then have to act as customers and must additionally provide a "product owner" for the technical service team. This product owner prioritizes and steers technical-service-team development. The major difference for feature teams is that the customers of a technical service team are also developers, so that the "features" a technical service team provides are technical and not business features. Even the technical features are driven by business features, because the feature teams will still direct their efforts toward developing business functionality.

If the technical service team provides the architecture for different products, the same holds true. That is, the teams that develop these products are the customers for the technical service team, and they state the requirements.

Summary

The most important part of building a team is to ensure that the team is able to deliver a discrete project goal, a whole feature. Charge a whole team responsible for a whole business feature to keep clear where ownership lies if, for instance, toward the end of an iteration, not all tasks are completed. Then, it is the team members' job to work together and finish that feature. Still, in

[25]Ibid., p. 53.

order to deliver a coherent system, conceptual integrity has to be ensured as well. This is the task of the architect(s) and, depending on the project, possibly also of the technical service team.

You might need to balance physical proximity with the need for specific skills and roles within a team by setting up a dispersed feature team. It is very important for dispersed teams to have a common goal: to focus on the features they're responsible for developing, as well allowing time for members to create a team identity so they work effectively together.

Contrary to dispersed feature teams, collocated feature teams need to concentrate more effort on cross-team communication between different sites to focus on the big-picture, common goal of the project. Often, projects end up with some teams collocated and others dispersed. Plan ahead for different configurations.

Three central roles ensure focus on the big picture:

- *(lead) product owner,* who provides the business perspective
- *(chief) architect,* who ensures the technical vision of the product
- *project manager,* who supports the organizational side

In a study of several distributed projects, Biehl summarizes lessons learned from managers who had been responsible for unsuccessful distributed development projects: "they had neglected three critical factors: not using cross-functional teams; not engaging in cross-functional communication; and not bringing end users on board early enough during the project."[26]

The architecture has to be a service for feature teams and therefore take the latter's requirements into account. This is the case regardless of whether the intent is to ensure conceptual integrity with one architect, with a team of architects, or with one or several technical service teams. Every person involved has to thoroughly support this approach.

[26]M. Biehl, "Success Factors for Implementing Global Information Systems," *Communications of the ACM,* Vol. 50, No. 1 (January 2007), p. 57.

Index