# Agile Software Development with Distributed Teams

Staying Agile in a Global World

Jutta Eckstein

# Agile Software Development with Distributed Teams

## Staying Agile in a Global World

Jutta Eckstein

This book is for sale at http://leanpub.com/distributed-teams

This version was published on 2022-03-31

ISBN 978-3-947991-30-3

# Also By Jutta Eckstein

# Contents

# 1. Assessing Agility and Distributed Projects

*All things are connected.*

— Chief Seattle

---

## 1.1 Understanding Distributed Development

My neighborhood grocery store currently displays an advertisement that notes,

> Computer specialists can be found in India; a grocery specialist is just around the corner.

One message to be taken from this ad is that people may need to go far afield to find experts to build or support technology, but they easily can find everything they want in the way of non-technological expertise locally. I don't want to argue for or against the cultural bias of this supposition (there undoubtedly are myriad grocery specialists in India, and I know for certain that there are IT specialists by the thousands in Germany), but I do want to note the implied difference in difficulty between seeking experts locally versus abroad. That's not to imply that the difficulty in looking for talented people in more than one place argues against globalization, merely that global project success involves more than just hiring top performers from around the world. Of course, there are those cynics who say that going global just means that the project will fail cheaper, so if cheap labor is the main goal, then just looking for any kind of cheap help will be enough.

As when shopping for groceries, software customers require at a minimum a local contact from whom to receive the actual product. Globalization does not change this. Although global software development may encompass multiple locations, distributed and dispersed teams, numerous companies, and off-site customers, a "local" coordinator only becomes more important as project scope, distance, and dispersion increase. Simulating proximity is one key to the success of distributed projects, as will be seen in the following sections.

## Working With Several Development Sites

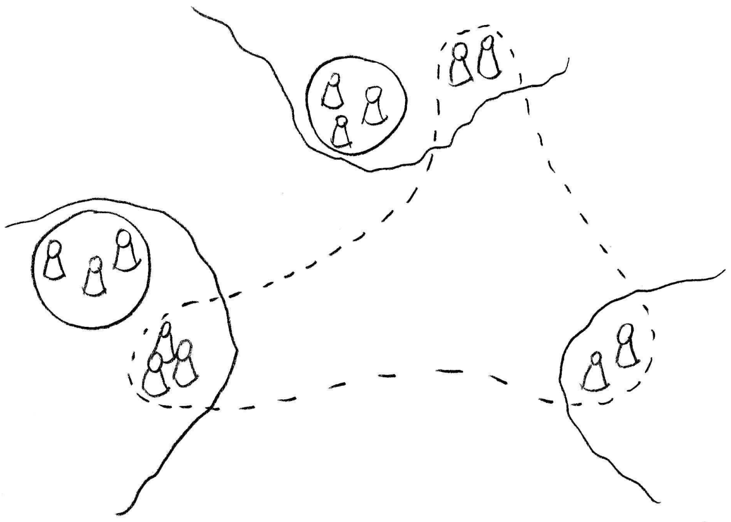
As indicated previously, a typical setting in a distributed software development effort involves multiple development sites. Project experts should not all be physically clustered at one site, but instead can communicate their knowledge virtually, across even several countries. In this way, each expert is the local link to the dispersed project effort.

Obvious difficulties accompany the geographical distribution of project experts. Experts must collaborate despite being located at different sites, but different cultures, time zones, languages, distance, and so on, make collaboration difficult. The goal is for experts to communicate with each other, then translate their specific duties to the local audience.

## Distributed and Dispersed Teams

At the core of distributed development are teams at multiple sites. A distributed project may be staffed by one or both of the following kinds of teams:

- *Distributed teams* might be made up of one group of people located in, say, Bangalore, India, and another group in the United States. This work unit is *distributed* between two sites, and the project is made up of two teams situated at different sites. Staff may work on different aspects of a project, but they form a single work unit (like an offensive and defensive squad on the same sports team).
- A *dispersed team* is one unit that is made up of people working at numerous locations. One team member may be located in India, another one in Northern Ireland, a third in the United States, and a fourth in Russia, with all four working as one development team.


**Dispersed Teams**

Most often, large global software-development projects include a mix of distributed teams as well as dispersed teams. Some groups within such a project may be collocated while others communicate from outposts. Keep in mind that a dispersed team requires a high coordination effort, even if the team is small, in order to ensure effective communication among all team members. Commit to building team identity early on in order to ensure that all members work toward the same goal.

## Large Projects

Developing a project globally usually increases its size. As explored in [Eckstein04], factors that determine a project's size include scope, time, budget, people, and risk. For example, factors other than budget certainly contribute to a project's complexity and stature. The relationship between such factors as people and time also affect the size and nature of projects—few distributed projects are scheduled to last only three months, as it is just not worth the effort to set up a heavily dispersed environment with every team member in a different part of the world. Large projects, whether distributed or not, always garner their own risks. One risk is attendant with the coordination of diverse people and teams. Another is ensuring that a large project remains flexible and efficient, despite the overhead required, for example, to ensure that all project members have the same goals in mind.

## Coordinating Companies

Often, the expertise of more than one company can provide better value for software customers than can be provided by a single entity. Some companies deem appropriate the strategy either of buying a company in order to work with teams at different sites or of founding a subsidiary at a different location, thereby ensuring an enduring cooperation and committing long-term to global development. Other companies regard distributed development as a chance to focus on their own core competency, and they prefer, therefore, to outsource peripheral tasks to other companies. Others use distributed development as an opportunity to better recruit talent.

No matter the group's formal organization, the effort to bring together different companies for a project requires a considerable amount of work to establish cordial, and essentially trusting, relationships between parties. A lack of mutual respect may cause difficulty between different subsidiaries, but it spells disaster when between different companies.

The kind of relationship coordinating companies seek can be defined in a contract, which clarifies in detail what two or more parties expect and how they will resolve conflict. By spelling out penalties, contracts provide a formal means to protect involved parties. Although a contract may serve as a means to establish a successful and trusting relationship, it does not actually *create and preserve* a trustworthy relationship that enables successful cooperation. Establishing a complete and successful relationship requires more than just words in a contract. It requires trust and partnership.

The concept of lean development, which is the root of agile development, favors partnerships.[1] Carsten Ruseng Jakobsen, Project Manager at the Danish firm Systematic Software Engineering, notes the use and role of contracts in building trusting relationships:

"Toyota has proven that treating sub-contractors as partners instead of continuously seeking [whoever offers] the lowest possible price is more profitable in the long term. . . . Even though you have a contract between different parts of the project, you want the complete project to collaborate toward the same vision."[2]

## Different Sites

Distributed projects can encompass any of the following intra- and inter-corporate relationships, each of which engenders unique

[1]For more information on lean development, see J.K. Liker's *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer* and Mary and Tom Poppendieck's *Lean Software Development. An Agile Toolkit.* Addison-Wesley, 2003.

[2]C.R. Jakobsen, personal communication.

challenges to the goal of ensuring close collaboration and, thus, success.

- *Offshore outsourcing* occurs when the domestic company outsources all or part of its software development to vendors offshore. Thus, the onshore and offshore companies are different companies, a situation that requires close attention to legal negotiation.[3]
- *Offshore insourcing* occurs when a company founds a subsidiary or buys a separate company located offshore. In such a case, only one company is involved but is itself distributed across the globe. Large multinational organizations, such as General Electric, Schlumberger, International Business Machines, or SAP (spell out??), typically pursue this strategy, primarily to gain market presence offshore.
- *Onshore outsourcing* occurs when one company carries out part of another company's development project. Both companies operate on the same shore, frequently in the same country or at least close by. Onshore outsourcing is not always considered to be global unless it shares many of the challenges of global development, such as cultural differences, which may interrupt communication, especially when due to discrepancies between corporate cultures. Perhaps the most significant challenge occurs when programmers, testers, database administrators, and other team members are not collocated.[4]
- *Nearshoring* can be combined with both insourcing as well as outsourcing. The major difference to the preceding settings is while both companies are located on the same shore they are not situated in the same country. For example a Californian

[3]According to B.B.M. Shao and J Smith-David, despite the logistical difficulties, offshore outsourcing is becoming increasingly popular. See "The Impact of Offshore Outsourcing on IT Workers in Developed Countries: Examining the Global Implications of Outsourcing for IT Workers," *Communications of the ACM*, Vol. 50, No. 2 (February 2007), pp. 89-94.

[4]Ibid. According to Shao and Smith-David, many outsourcing contracts are based on the strategy of onshore outsourcing.

based firm sourcing parts of the development activities out to Mexico.

## Customers and Distance

Communication *between customers and developers* is similarly important and challenging, as it is *among developers.* Customers must communicate requirements to developers, who, in turn, must fully understand the customers' needs in order to translate them into product functionality. Doing so is difficult even when customers and developers are collocated. It grows more difficult when physical distance and cultural and linguistic differences exist.

When requirements are unclear, developers can gain a better understanding by means of short feedback loops to clarify requirements verbally or to present an early version of the desired product or system. How successful the feedback loops are can depend on the physical distance between customers and developers—particularly when software must work on various platforms and systems, or display in different languages.[5]

Effective, accurate communication among customers and developers plays a major role in a project's success not only during development but also during planning, taking into account different legal requirements and diverse cultural backgrounds.

## Centrally Coordinated or Globally Integrated

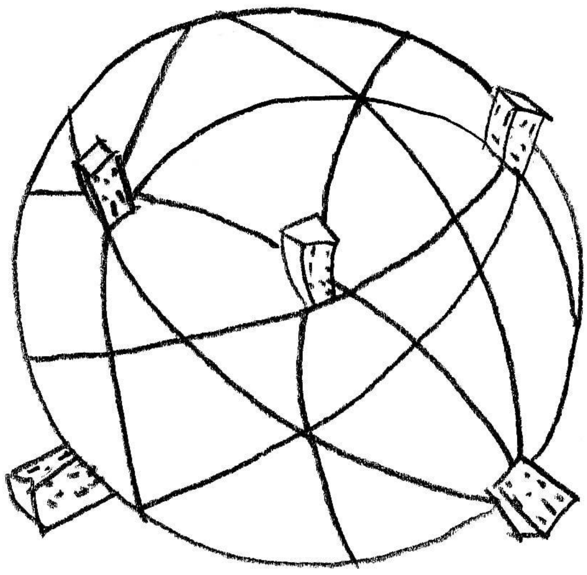
In his book *Global Software Teams,* Erran Carmel of American University identifies three evolutionary stages of the global development process. As Carmel sees it, at Stage I, the entire development process takes place at one location, and thus is not a global effort. At Stage II, development takes place at multiple

[5]Thanks to Rachel Davies for pointing this out.

sites, all of which are centrally coordinated and controlled by one headquarters. At Stage III, remote sites are self-directed, with coordination between sites operating like a network. Carmel posits that most software development companies function at Stage II, and only a few companies ever prepare to move to Stage III, in which ". . . various remote development sites assume greater responsibility for a range of tasks and coordinate some activities among themselves without funneling all decisions through headquarters."[6]

Many development efforts even run aground due to company policies that prevent them from advancing to Stage III. These efforts are limited to always assembling the higher management of a product at one location, typically at corporate headquarters.

[6]E. Carmel, *Global Software Teams: Collaborating Across Borders and Time Zones* (Englewood Cliffs, N.J.: Prentice Hall, 1999), p. 138.



**Global Integration**

## Overcoming the Distance

As now should be evident, distributed development involves much more than just finding (low-cost) experts somewhere. Rather, in order to successfully implement distributed development, the factors discussed above—multiple work sites, large projects, different companies drawn together, and distant customers, must all be addressed. The key to successful distributed development is establishing proximity despite all forms of distance.

## 1.2 Understanding Agility

A historic marker indicating that agile methods no longer would be considered mere hype or a fringe movement was Scott Adams' *Dilbert* comic strip on agility. With every passing year, agile concepts have become more firmly entrenched in mainstream business and, today, are largely accepted in the modern market. Of course, while noting the movement of agile methods from the realm of fringe, Adams also exposes typical misunderstandings, ill-formed expectations, and downright strange interpretations that some think still pervade the agile approach.[7]

Agility has come into its own as a value system defined by the Agile Manifesto.[8] Based on twelve principles created to ensure the value system,[9] the Agile Manifesto demonstrates that there is more to agile development than just one specific methodology, such as Extreme Programming[10] or Scrum.[11] The first value stated in the manifesto favors "individuals and interactions over processes and tools." The *processes* referenced in this first value statement include also agile development processes, which means teams must ensure that their development process supports their needs in the best way possible. Using the principles in the manifesto, teams can find guidance on how to modify and adjust their development processes to best support their needs.

### Core Value Pair Statements

The values expressed in the Agile Manifesto apply to all agile projects, superceding guidelines of any specific agile process. The

[7]See S. Adams, *Dilbert* (http://www.dilbert.com/).

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

[9]For an analysis of the Agile Manifesto, see A. Cockburn's *Agile Software Development: The Cooperative Game.* (Reading, Mass.: Addison-Wesley, 2nd edition. 2006). For information on agile development for large projects, see my book *Agile Software Development in the Large.*

[10]For XP, see http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap.

[11]For Scrum, see http://www.controlchaos.com and http://www.scrumalliance.org.

core of the manifesto compares in four statements two values and argues that although each value provides a value in general, the first value is more important than the second and that the latter half of the each statement is only valid if it supports the former.

Value Pair Statement #1,"Individuals and interactions over processes and tools," highlights the idea that it is always the people involved in a project and how they collaborate that determine a project's success or failure. The manifesto does not devalue processes and tools (otherwise, we wouldn't talk about processes, and the agile community wouldn't have created tools such as unit testing frameworks, integration and configuration management tools, and others), but if individuals don't work together as a team, the best tools and processes won't help the project succeed.

Value Pair Statement #2, "Working software over comprehensive documentation." is perhaps the most often misunderstood of the four statements. People unfamiliar with agile development may mistakenly believe agile projects don't document, or even disdain documentation. Not so. In the same way that processes and tools play a major role in successful development, documentation also plays a major role. However, this value comparison expresses that working software is the critical success factor for any development effort. Documentation might be needed to support or to understand the working software, but it can't and shouldn't be an end in itself
.

Value Pair Statement #3,"Customer collaboration over contract negotiation," emphasizes that although you need a contract, it can never be a substitute for a good relationship with your customer. In order to deliver a satisfactory product, involve customers regularly throughout the development process.

Value Pair Statement #4, "Responding to change over following a plan." advocates the importance of reacting to changes (especially in terms of requirements changes), rather than sticking to an inappropriate or obsolete plan. We accept that both the customer and the project team will learn over time, and we want to acknowledge this learning and incorporate it into the development effort. If the finished product delivers what the customer and we planned for before confronting changes and disregards anything we learned during development, the product will be a failure, even if it fulfills a contract.

The agile value system accommodates collocation as well as distributed software development. Later in this chapter, I examine implications of agile principles regarding globally distributed projects.

## Systemic Approach

Agile development promotes a systemic approach that is supported by a closed-loop routine of planning, doing (or performing), inspecting (or analyzing), and adapting, as follows:

- In *Planning,* plan immediate activities (having broken down a development project into deliverable chunks, begin planning for the first task). This is most often short-term planning, focusing on the next iteration, but it can also be long term, such as planning the next release.
- In *Doing,* perform activities planned in the first step.
- In *Inspecting,* analyze the performance of the activities planned in the planning step. Did all work as planned? Was there a specific process that worked well and would be appropriate to repeat in the future? Did a specific process or plan fail or require adjustments for the future?
- In *Adapting,* determine what kinds of adjustments the previous inspection step revealed are needed in order to improve development? In this step, decide necessary actions for the following iteration.

The last step in this closed-loop routine provides input for the first step in the next round, and so on.

## Risk Reduction

The goal of an agile project is not only to deliver a product at the end of the project's lifetime (also called a deadline), but as well to deliver early and regularly. In order to do so, we divide the project's lifetime into development cycles. A bigger cycle that produces much functionality (sometimes called a feature pack) is called a release. Within that we use a smaller cycle to organize work in smaller chunks, and to deliver smaller functionalities. This smaller cycle is called an iteration.[12] Both a release and iteration lead to a delivery or a potentially shippable product.

A tremendous advantage of agile development is risk reduction through high visibility and transparency. By developing iterations of a working system, receiving regular feedback from the customer and from tests, and with tangible progress, you have access to the actual status of the project. Knowing the actual status of the project in turn enables you to make decisions regarding further deliverables and necessary actions. For example, if you encounter that the system does not fully satisfy the customer and it can't be turned in the right direction, you have the advantage of being able to stop the project early, before all the money has been spent.

## The Productivity Myth

Another common, and misguided, argument is that following an agile approach will greatly increase a development team's productivity compared to other approaches. While this can be true, it is not always necessarily so. Agile development guides a team to deliver a working system frequently—"frequently" meaning in iterations lasting one to four weeks. A "working system," on the other hand, is defined by the customer's evaluation of usability. Thus, by providing a working, usable system periodically, say,

[12]In Scrum, "an iteration" is called "a sprint." I personally do not like that term because, for me, it connotes frantic, unreserved effort. Iterations should involve adequate resources so that teams are not racing to finish.

every two weeks, an agile team ensures maximum business value for its customer.

Therefore, following this approach your customer might decide to go into production with the system earlier. This will give your customer a market advantage. However, it does not necessarily mean that the project as a whole is finished −meaning all required features are implemented− earlier.

## More Than Practices

Agility is more than a collection of practices. Every so often, I hear people mixing up specific practices with agility. Practices—for example, Extreme Programming's pair programming or test-driven development—are a great means to preserve the agile value system; however, these practices are not the value system itself. For instance, you can successfully apply pair programming and use a linear (or waterfall) development approach.

## Neither Chaotic Nor Undisciplined

Many people consider the agile approach to be an undisciplined approach. Some regard agile as an ad-hoc approach that doesn't require any planning, one in which people act independently according to whim. Sometimes, the agile label is used as an excuse for lack of preparation. For example, if a person has to conduct a workshop or deliver a talk and doesn't prepare material, his or her presentation will consequently follow an ad-hoc approach. This person might argue that the approach used is agile, and therefore doesn't require preparation or planning. Instead, absolutely the opposite is true: Agility requires a lot of planning, even more planning than a linear approach. As Lise B. Hvatum states, "Agile is highly disciplined and more difficult, requires more maturity, than waterfall."[13]

[13]L.B. Hvatum, personal communication.

The reality is, agile requires and embraces planning. In agile development, the artifact of a plan is not overly important; the activity of planning, however, is essential. Jakobsen contrasts a choice between an old management style—for example, Taylorism, where managers dictate procedure—and an innovative management style—such as Lean Jidoka[14], based on trust, respect, empowerment, and belief that it is the people who use a process who are best able to improve it.[15]

Improving processes means changing your original plan, and preparing for future re-planning to utilize what you learn as development occurs.

## 1.3 Agile Principles Influencing Distributed Projects

Listed below are twelve principles of the Agile Manifesto, annotated in terms of their impact on distributed development and direct implementation of an agile approach.

*Satisfy the customer through early and continuous delivery of valuable software:* Early and continuous delivery is only feasible if all distributed project sites work in concert and take into account customers' wishes.

*Welcome changing requirements, even late in development:* Communicating requirements changes and their implications requires considerable coordination effort across different sites, but it is not more difficult in a global setting than in a local setting if people on the project are accustomed to pulling together toward a common goal.

*Deliver working software frequently:* To deliver working software at frequent iterations, the work done by all sites must be carefully

[14]Lean Jidoka requires all team members to be responsible for improving the process (immediately) as soon as the quality of the outcome decreases.

[15]C.R. Jakobsen, personal communication.

integrated. The effort required for teams to deliver a smooth build and integration is considerable even when teams are collocated; it is all the more so for a distributed project.

*Business people and developers work together:* Regardless of distance between sites, language differences, or cultural disparities, all project members must be fully aware of customer needs, and must make every effort to incorporate customer feedback in the development process.

*Trust motivated individuals:* Trust generally is built by proximity, a default obstacle in a distributed setting. The sense of closeness, though, must be fostered so as to bind teammates together despite physical distance.

*Face-to-face conversation:* Because direct, face-to-face conversation is one of the best ways for people to communicate their shared requirements and goals, periodically set aside a time, place, and technology to facilitate effective communication.

*Working software is the primary measure of progress:* In a distributed setting, making software work over different sites is much more difficult than when everyone is collocated. The major challenge is to ensure the joint effort of the different sites in order not to have several systems but, rather, one coherent, running system in place.

*Promote sustainable development:* This principle acknowledges the fact that people working too much overtime tend to burn out, adversely affecting the quality of a system. This is true for both distributed and collocated teams, but there is added difficulty on distributed, global projects which have people working at odd or irregular hours in order to communicate and collaborate. The time and physical effort people spend traveling between different sites also can negate their effort to build relationships, making people on distributed projects more susceptible to burn-out than are folks on collocated teams.

*Continuous attention to technical excellence and good design:* Some projects are negligent in establishing quality assurance at all sites. Assuming that continuous attention to quality is every project's goal, ensure that all sites and all project members work toward attaining it. Additional education may be needed to bring all staff at all sites up to snuff in such areas as testing, refactoring, quality metrics, or other skills.

*Simplicity is essential:* In distributed settings, project members sometimes develop a general, one-size-fits-all framework for the system in advance of beginning development work because they believe it will ease the developing business functionality later. However, such a framework generally is disconnected from and irrelevant to the customer's actual business requirements and thus doesn't support the domain. Such a framework introduces more complexity and compromises developing business functionality.

*Self-organizing teams:* Physical distance between sites can be particularly challenging to self-organizing teams because distance makes it harder for people to know and trust one another.[16] So a smell[17] for mistrust is if you're using a more command-and-control style of "collaboration" instead of enable the teams to self-organize. Trust is essential on globally dispersed or distributed teams, whose members may need to be educated about taking responsibility and self-organizing—especially in regard to concepts that contradict their culture.

*Team reflection and adjustment:* Here is a direct connection to the first value pair of the core of the Agile Manifesto, which values individuals and interactions over processes and tools. At first glance, this is not necessarily different in a distributed setting than it is in a collocated setting: The idea is, allow team members to reflect on how they're progressing to enable them to improve over time. The challenge, however, is to regularly promote reflections across all sites to improve cooperation and federation.

[16]For more on the concept "trust needs touch," see C. Handy, *Trust and the Virtual Organization* (Boston: Harvard Business Review, 1995), Vol. 73, No. 3, pp. 40-50.

[17]A smell is a sign for or a hint to a problem.

Globally distributed projects face the major challenge of how to organize iterations and releases across different sites and still ensure that something functional is delivered at the end of the development cycle. Moreover, the real challenge is not the organization of the work but the integration of the distributed development effort into one working system. Integration and build across teams and sites is essential.

## 1.4 Summary

There are as many assumptions and misconceptions about global development as there are about agile development. The implications of global development are that several development sites, often spread over several countries, are involved; that development is typically performed by several teams and thus large projects; that even a single team can be distributed across multiple sites (a dispersed team); that multiple companies can be involved; and that customers can be located far away from developers.

Agile development is more than just a specific methodology or collection of defined practices. Culture, values, and beliefs highly influence success in creating trust, collaboration, and a shared vision. One of the significant barriers in distributed projects to overcome is slow feedback due to all types of distances.