EXPERIENCE FROM AGILE ADOPTION IN DISTRIBUTED ENVIRONMENT

Jaroslav Procházka
Ostravská univerzita, Jaroslav.Prochazka@osu.cz

ABSTRACT:
There exist a lot of papers about agile development, its practices and comparison with waterfall method in proceedings, books and Internet. Lots of them are only repeating the principles, techniques or some specifics. This paper rather focuses on and summarizes experience with small as well as large global sourced projects trying to run agile. We summarize main reasons led us to develop agile and also typical impediments and pitfalls during introducing agile way of working. Finally, we outline some techniques that work really well in our environment.

KEY WORDS:
Agile development, agile techniques, experience.

1. INTRODUCTION

But to make short introduction for not familiar readers, we pinpoint main differences between agile and rigorous waterfall based model. Readers familiar with agile principles can skip this part. Agile development comprises several methods or frameworks, namely Scrum, XP, Lean Software Development, Feature Driven Development, Crystal methods and others. People working on these methods met in 2001 and form so called Agile Manifesto [1] defining principles and values common for these methods. Basics are:

- Individuals and communication are more then processes and tools.
- Working software is better than tons of documentation.
- Customer should tightly collaborate with the development team.
- Respond to change is more significant than follow the plan.

Figure 1. Agile manifesto

In Agile Manifesto, you can see stressed communication and cooperation among all people (business and IT, within IT teams) and self-managed teams, where team itself make decisions (no managers) to reach the final deliverable product. Next, it is the focus on continuous delivery of valuable software and software as a primary measure of progress, not design or requirement documents, which do not bring business value to the customer. Last but not least is simplicity (avoiding the waste, e.g. extra features) in development and continuous learning during the lifecycle. Of course, we can't forget change tolerance and taking the requirements change as an advantage for our customer, not as an impediment.

This can be taken as a short introduction into agile methods. Agile approaches occur as a response to the waterfall and spiral based models that generate problems in reality. Just to be sure that we are not blaming this model, we say, waterfall might work in environment where requirements are stable, problem domain is really known, team is skilled and familiar with technology, changes are small not affecting the architecture. Next chapter summarizes the problems that led our teams to think about another approach to software development.

2. MAIN PROBLEMS

As we have mentioned in the introduction, there are several issues dealing with the old fashion development model, that led us to thing of another way, how to develop software. These are typical issues we have experienced in our projects:

- Never used features and required changes in these – effort needed to develop and deliver features and changes.

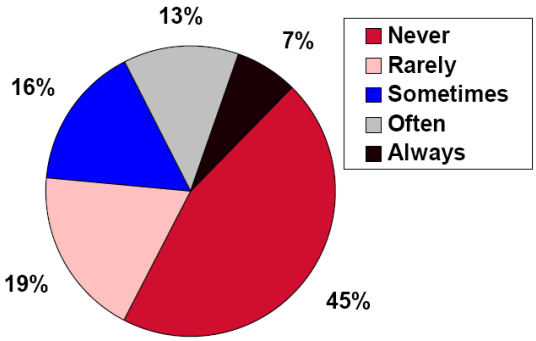

Figure 2. Features and their usage (source: [2])

- Projects haven't met agreed time and budget – a few projects on time and on budget; if so, then with a poor quality.
- Quality issues – lot of faults found by customer after release, poor performance or other quality issues.
- Misunderstood requirements – self-interpreted requirements, functionality is implemented in different way customer expects.
- Bad estimates for the whole project, for given phases (mainly the 2nd part of lifecycle: coding, integration and testing), for maintenance issues.
- Overloaded people – lot of things done mainly in the 2nd part of the development life-cycle, no time to solve structural issues – lot of workarounds introduced.
- Unpredictable events occur – not enough time to solve them, or too important that impacts architecture, integration, requirements and testing. Discovered late in the project.
- Squeezed time (or no time) of non-functional testing at the end of the project to save time (to deliver on time).

These are the issues leading us to think of better, more effective and efficient development model based on iterations and stressing communication. But there was one more outstanding topic also really important: customers really want agile way of doing software! Now we know the starting triggers for agile development in our projects. Let's start the discussion of possibilities in small Czech businesses.

3. SMALL CZECH BUSINESSES

For small Czech businesses is implementation of agile methods and practices easier than for larger ones. The reasons are following:

- Smaller and co-located teams – people are mostly in the same room, know each other personally, known their skills and previous work.
- Team members believe the champion that brings new way of working, if he or she is senior team member (skilled architect, PM or company's visionary).
- Communication is mostly not a problem – people communicate.
- Informal and valuable communication channels – informing each other, sharing knowledge, solving problems during the informal "hallway" communication.
- Smaller projects – 3 to 9 people working on project.
- Teams have better contact with management – management is more informed, managers knows well team members and can be easier convinced.
- Cross-functional teams – team members play a lot of roles (analyst, designer, developer, tester), each team member does everything needed to achieve the target.

Basically, these Small Business' teams already uses agile techniques, but informally without knowing it. Typical example of such used technique is pair programming for knowledge sharing and problem solving, self managed and cross-functional teams – team have not only responsibility but also authority to make decisions, daily meetings to synchronize the whole team (but also informal). But in this case is much harder to involve customer, IT companies usually develops systems for Small and Middle Business (SMB) and SMB employees have usually a lot of work and are critical, so cannot be able fulltime onsite for development team.

4. LARGE GLOBAL SOURCING PROJECTS

Problems mentioned above are common in both, small and large projects. In distributed environment occur also other problems because of geographically distributed teams and remote customer. Following list summarizes additional issues in global sourcing model:

- Harder communication – the most important topic, distributed teams need to communicate using expensive channels (phone calls, tele-conferences, video-conferences or travelling), depersonalized communication.
- Subjective requirements interpretation – caused by hard or no access to the customer and/or future applications users.
- Functional division of application – onsite team defines requirements and does analysis; offsite team codes and performs testing (communication barrier => loss of information).

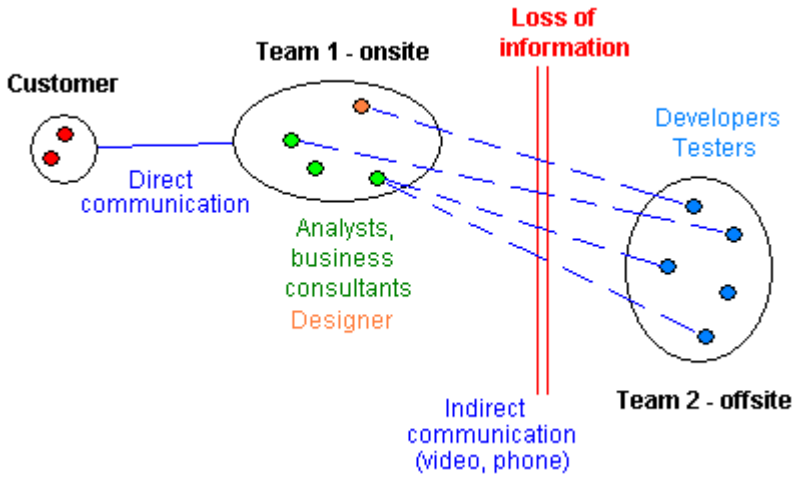

Figure 3. Communication in global sourcing

- Micro-management – onsite tendency to manage offsite teams (detail plans and steps from onsite managers), thinks stupid monkeys are offsite.
- Waterfall thinking – not existing cross-functional teams, analyst creates analysis document and "throw over the wall"; designer creates design document based on this, etc. (loss of information, knowledge, experience, common goal).

These are only the most visible issues we are experienced with in the global sourcing model (sometimes with common root cause). The most important issue and the root cause of the rest of the topics in the list is communication between distributed team members.

5. COMMON QUESTIONS, PITFALLS

There go a lot of stories and misconceptions around about agile approaches and techniques. The one is that there is no documentation, then no analysis is done, it is suitable only for green field development and small, co-located teams etc. This is not true! The main problem is trying to understand new way of development with the old-fashioned mindset. The focus is

not on the documentation but rather on activities that need to be done and the form of capturing information is not so important. Really famous tools for this case are whiteboards, flipcharts, post-it notes. Photos of drawings can be stored in the repository and serve as a developer's documentation. This can be accurate for the small and skilled team.

Second, analysis and design is done during every iteration, but only for the scenarios that are in the scope of given iteration. We do not focus on analysis of the whole system to avoid analysis paralysis. Rather, we choose only a piece of the system, so called scenario and completely develop chosen scenario, see following whiteboard examples of architecture model and storyboard of one scenario.

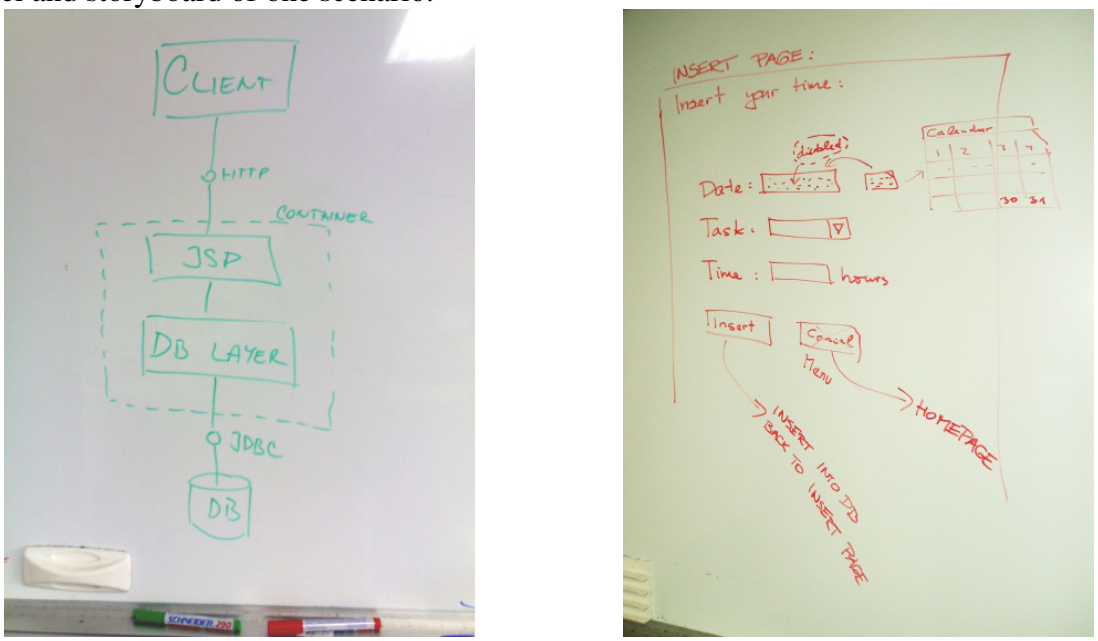

Figure 4. Candidate architecture (left) and story board (right)

Of course, also software tools are used but only in the case, where it is effective. For the large distributed project is needed to use issue tracking tool like Jira, modelling tools like IBM Rational modeller, building tool like Ant, Maven or team Wiki pages. The critical automated part of the process is building and continuous integration. There is the use of software tools must, because it is done very often. Daily builds with performed test suites gives the developers quick and precise feedback.

6. EXPERIENCED PROBLEMS

In this chapter, we outline the main problematic parts during agile adoption from our experience. As this experience is based on particular environment and working culture, for different organization can occur quite different issues. So take the following list only as an example of possible problems when adopting some agile principles and techniques, not as a full list to follow.

Fundamental issue – mindset change and mentality

People try to perform iterative approaches and techniques with the old mindset. They say: "Of course we do iterative development." The result is 1st iteration requirements gathering and analysis, 2nd iteration design, 3rd iteration development, etc. But agile development is quite different:

- Each iteration produces executable and tested build.

- All the disciplines are done in every iteration (analysis, design, testing, development, project management).
- We do scope management (see picture below); changes are welcome.
- We do only brief plan (road map) at the beginning, no huge upfront planning; detailed planning is done only for given iteration (much shorter period).
- Teams are self-managed and cross-functional.
- And many others.

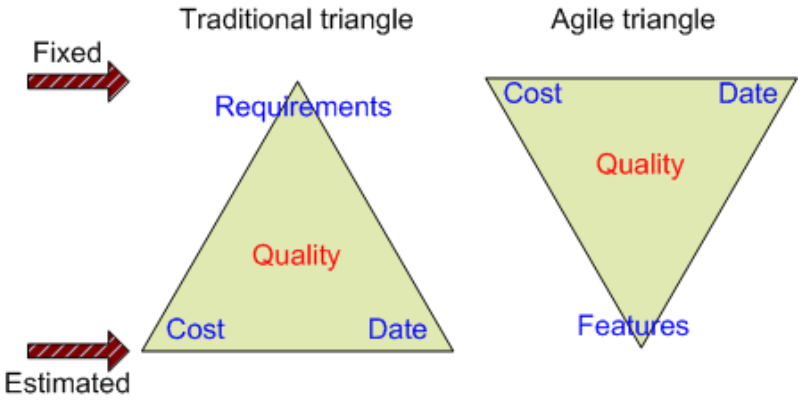

Figure 5. Traditional and agile triangle of quality

Traditional approaches gather all possible requirements upfront and try to estimate cost and the date of delivery for the whole projects. This is very hard because we do not know the risks and unpredictable events that can occur during the project. All these factors might change the estimates dramatically. On the other hand, agile approaches agree the cost and delivery date and try to implement as many features as possible, starting with the most important ones for the business. Customer cannot define all three parameters. Team must define one of these quality triangle parameters to assure quality of the overall product!

Basic presumption is also mentality of the developers and the customer. A lot of agile methods have come from US. Czech programmers do not have courage to throw away written code, rather plays and refactors. US programmers, throws it away and starts from the scratch. Next, customer representative (future user) have to be available for the developers during the development, this could be also problematic.

Understanding of iterative development, planning objectives and assessing the results of iterations are not easy. We meet every day people telling us opinions like iteration can be repeated; iteration can be extended; testing is done at the end of project; results of the iterations do not need to be verified, etc.

Communication

Communication is probably the most important topic of distributed agile development. First, the team has to know and share the vision of the system (project). The whole team should agree on vision, if not, people will not perform at their best. Scrum talks about so called self-managed teams. Such teams do not have only responsibility, but also authority to make decision. There is no manager giving tasks, team itself agrees on and each member takes tasks that need to be done. Such teams are also cross-functional, analysts do test cases or test scripts as well; team members cooperate during the whole development.

For knowledge and information spreading and for team commitment are important also daily meetings. Daily meetings take only 15 minutes and are performed every day in the morning.

The topic is to say to colleagues what I did yesterday, what I am going to do today and if I have any problems, impediments. If some problems occur, another meeting is planned with the only people needed.

As architecture is very important for every system, agile development approaches define some ways of communication. As said before, Scrum has daily synchronization meeting. Rational Unified Process (RUP) has communication around architecture

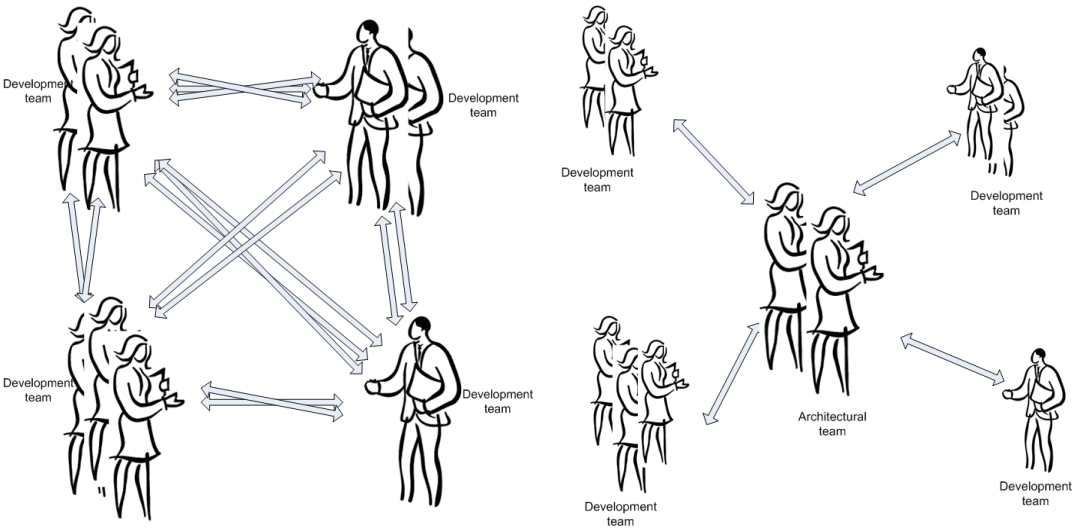

Figure 6. RUP approach - communication around the architecture (left part is common way; right part depicts architectural team)

Second way, how to spread the knowledge of requirements and architecture in distributed development is to have the role of requirements interpreter offsite. Man playing this role knows customer needs and usually was member of onsite team in the preliminary phase of the project. Another approach is so called gate-keeper. This is usually team architect (skilled, senior designer) and only these gate-keepers communicate among the teams, not particular team members. The last important thing also affecting communication is division of application into parts. It is not good to functionally (by roles) divide application that means analyst team, design team, development team, etc. It is better to divide application according the stories (functionality) and completely develop the stories. This has preconditions: strictly defined interfaces among subsystems and often synchronization and integration of the code.

Commitment

The management commitment is also important for agile projects. We need support from management and sales people (must understand “agile business model”). If contract says all requirements are defined and detailed plan created upfront and frozen and developers have no contact with the customer during the development, we are not able to do agile project (at least not a pure, effective one). The team member’s commitment in agile approaches is built in. In XP team member need to commit for values, in Scrum exists team commitment on sprint backlog.

Tracking the progress, evaluation criteria

Agile thinking changes also tracking the progress. We do not focus on plans and what is done from the plan, because it is often out of reality. Rather we focus on remaining work (e.g. Burn-down charts from Scrum). Definition of done is important. This definition is strict, e.g.:

story's implemented, 100% of unit tests passed and user has accepted. To know more about this, see [4] and [5].

Where and how to start

This question is perhaps the most frequent one in our life. So, to give short answer, that is not still the same and is not easy, follow these steps:

- Involve mentor into the project from the beginning.
- Start with a small skilled team, known domain or/and technology.
- Do an analysis/assessment with the mentor and start with the poorest part, than perform it again (various kinds of tools can be used: questionnaire, theory of constraint, CMMI).
- Do not implement all techniques at once (big bang), implement them iteratively (same as the software)
- OpenUP [8] is a good starting point.

7. CONCLUSION

This paper summarizes our experience from different projects, environment and problem domain. Iterative and incremental development with agile principles and techniques is suitable almost for all types of projects: new development, maintenance, combinations of both. Introduced techniques work well in all these types. But it is important to stress, that to start with it is not really easy. Involvement of the skilled mentor should be one of the critical things as well as customer and team involvement and commitment. Some other hints and approaches how to start with agile can be found in [7].

8. REFERENCES

[1] Agile manifesto. Available on [http://agilemanifesto.org]

[2] The Standish Group research: Chaos Report. 2002. Available on [http://www.projectsmart.co.uk/docs/chaos-report.pdf]

[3] Poppendieck, M.: *Implementing Lean Software Development*. AW. 2006.

[4] Cohn, M.: *Agile Estimating and Planning*. Prentice Hall. 2005.

[5] Schwaber, K.: Agile Project Management with Scrum. Microsoft Press. 2004.

[6] Bergstrom, S., Raberg, L.: Adopting the Rational Unified Process. AW. 2004.

[7] Kruchten, P.: *Going over the waterfall with the RUP*. 26 Apr 2004. Available on the Rational Edge: [http://www.ibm.com/developerworks/rational/library/4626.html]

[8] OpenUP. Available on [http://www.eclipse.org/epf/downloads/openup/openup_downloads.php]