The Top 13 Organization Challenges of Agile Development – and a Solution to Each

Executive Summary

Dealing with challenges is inherent in any development approach, but Agile methods often expose problems that were previously ignored or otherwise invisible to the organization. IT’s first reaction may be to address the technical challenges, but, in reality, the biggest impediments of Agile adoption are organizational, not technical. According to the “State of Agile” survey by Version One, the three greatest barriers to further Agile adoption are ability to change organizational culture (45%); general resistance to change (44%); and personnel with the necessary Agile experience (42%).

While organizational challenges aren’t simple to solve, they are somewhat predictable. Typically, they fall into one of these six categories:

- **People**: Organizations will always have people issues, but the introduction of Agile methods increases the rate of change and exposes many problems.

- **Process**: Agile methods are very process-light, but a lot of project teams have difficulty keeping it simple.

- **Technology**: Because of corporate policies, it can be difficult to procure the software that is needed.

- **Teams**: On the whole, teams seldom present impediments to Agile adoption, but there can be friction between teams.

- **Management**: It can be difficult to convince management for the need for Agile and get them to publically promote this.

- **Culture**: There can be a myriad of cultural challenges, but one of the biggest issues is aligning the way individuals are rewarded with the work they are doing.

If organizations want to successfully transition to Agile development, they need to be aware of and address these challenges. The longer these issues go unresolved, the more difficult they will be to fix. This white paper walks you through the primary challenges you will face as you adopt Agile development practices, and provides recommendations on what you can do to solve them.

The more entrenched an organization becomes with Agile development, the more people the challenges will impact.

**Challenge 1: Team members are overly-specialized, which requires work to be handed off several times.**

Overly specialized skill-sets require work to change hands multiple times. Handing-off partially completed work is very expensive, and it is desirable to keep this to a minimum. For example, an analyst will describe the problem before handing-off to a DBA, who will work with the database before handing-off to a developer, who write the functionality before handing-off to a tester.

To minimize hand-offs, encourage pairing at all stages and ensure that the necessary hand-offs are informal. Encouraging the team to tackle small chunks of work also helps because it forces them to communicate more frequently.

**Challenge 2: Lack of ownership by the team.**

Team members who are accustomed to receiving instruction often have difficulty engaging with Agile projects initially because they are waiting to be told what to do. To help individual team members take ownership, teams need to communicate openly. If you are having this issue, try some of these approaches:

- Let the team work directly with the customer
- Encourage team members to own and solve their own problems instead of letting organizations control technical conversations.
- Solicit input from different members of the team.
Challenge 3: Some team members refuse to interact with the team.

It is not uncommon for some team members, such as architects, developers and testers, to refuse to participate in the Agile process. Their justifications are many—they have too many prior commitments, their work cannot be broken down in a fashion that’s amenable to Agile software development, their time is too important, they work better as an individual—but these are nothing but excuses. Regardless of the reason, the end result is that they are deliberately putting themselves outside the team.

The team has to make its best effort to resolve these issues and to work with others who are attempting to separate themselves. Every member of the team needs to contribute, and if one or two members of the team are putting themselves above the team, this needs to be quickly addressed by management. Oftentimes, this results in the quick remove of the individual(s).

Challenge 4: It can be difficult to convince management for the need for a new development approach.

Educating a new client on how to build an Agile organization can be very slow and difficult work. To be successful, you need to be able to speak persuasively at many different levels of the organization.

The first step in educating an organization about Agile development is speaking to the team or to individual members of the team. However, you also need to consider a wider audience, including functional managers, the PMO and HR. Failure to address this wider audience can hobble the transition to Agile.

Agile teams also make the common assumption that management can see the positive changes and automatically understand the value of Agile methods. But management is no different from everyone else: they require education, coaxing and convincing to recognize that there is a better way to develop software.

It is important to note, though, that the topics of conversation differ between management and developers. For instance, instead of discussing Test Driven Development (TDD), discuss Agile metrics (burndown graphs over actual developer hours), or instead of discussing Pair Programming, discuss the need for collocation. In addition, when working with management, pay attention to both Agile metrics (or the lack, there-of), and adaptive planning over predictive planning. The Agile approach to both of these is counter intuitive for most classically-trained managers and requires constant reinforcing.
Challenge 5: Senior management is giving mixed signals regarding their support for Agile.

Nothing cripples an Agile project more than a few carelessly chosen words from senior management. Regardless of their own perspective, senior management needs to decide that they are committed to Agile methods before introducing it, and, once adopted, they are obligated to publicly support the effort.

While it can be difficult to control the actions of senior management, Agile teams and customers must portray these methods in a positive light. They must frequently communicate with senior management to clearly articulate their successes and challenges, and they should share their experiences with the wider organization so their achievements are acknowledged.

Challenge 6: No single Product Owner can be identified.

It can be difficult to identify a single Product Owner when there is a large separation between the business units and the software development departments, or several different groups have an equal interest in the success of a project.

In the situation where a single customer cannot be identified, pinpoint the project sponsor, who is the single person who ultimately approves the funding of the project. By working with the sponsor, and clearly articulating the need for a single business representative on the project team, this situation can usually be quickly resolved.

In the situation where several different groups have an equal interest in the success of a project, the team still needs a single representative who is willing to work with each of the different groups and prioritize accordingly. Again, work with the project sponsor to provide the path to a solution.

Challenge 7: Management wants to combine elements of RUP and Agile.

It’s common for management to want to combine the “best” elements of RUP and Agile, but the results are disadvantageous for both methodologies without the advantages of either. There are several different approaches to doing this; RUP management process combined with XP engineering practices is the most common.
While management often wants to combine methodologies so they can continue to provide familiar reports to senior management or business partners without having to re-educate them, this approach underestimates those receiving the reports, and does the organization a disservice. You can get better results by educating management and business partners while adopting a single approach, whether it be RUP or Agile.

**Challenge 8: The ScrumMaster refuses to protect the team.**

The ScrumMaster role is essential to maintaining healthy communications between team members and protecting the team. It takes a lot of courage to report problems to senior management and to make changes that others in the organization may view as a direct threat to their positions. At some point, even the best ScrumMasters will have their credibility questioned.

If the ScrumMaster is ineffective at protecting them team, this person needs to be replaced. To find a new ScrumMaster, request volunteers or ask the team who they would like to represent them.

**Challenge 9: You do not have a reliable build system and processes.**

The source control, build and testing of software is an intimate part of any software development team; it simply makes sense for this process to be efficient as possible. For instance, if it takes 2 hours for a developer to check in code and do a build, it would be better if this process could be automated and the developer could use that time in a more constructive manner.

Clearly, if the team is building and releasing code many times per iteration, any effort expended to create a reliable build and testing framework will be worthwhile.

Implementing Test Driven Development (TDD) and Continuous Integration (CI) are good first steps. The tools for both of these practices are common and (often) free.

**Challenge 10: QA issues are not addressed.**

With traditional development methodologies, testing is usually addressed at the end of the cycle. So, it’s not surprising when Agile teams do not pay appropriate attention to defects and code quality. However, code quality will quickly become a problem with repeated iterations, especially if the team is trying to build new functionality on top of defect-laden code. In addition, delaying testing until the later stages of a project will result in late discovery of unanticipated side-effects.
Producing high-quality code from the onset of the project should be the ultimate goal of every project. You can achieve this by implementing the Extreme Programming engineering practices, such as Test Driven Development, Continuous Integration, Refactoring and Pair Programming (see the sidebar for a description of each). It is only by addressing quality issues early and proactively that the fully nature of the software will be known.

**Agile Engineering Practices**

Although Scrum does not explicitly lay out engineering practices, teams find it difficult to deliver potentially shippable product increment every single Sprint without adopting the Extreme Programming (XP) engineering practices.

**Typical XP engineering practices:**

- **Test Driven Development (TDD):** Unit tests are written in alternation with the code. It forces developers to more carefully consider the design and usage of their code. The typical flow of a TDD session is to: 1) add a test; 2) get it to fail; 3) write code to pass the test; and 4) remove duplication (see Refactoring below).

- **Continuous Integration (CI):** Martin Fowler provides an excellent description of Continuous Integration and the benefits that it provides can be found on website. Here is a brief description, but check out his website for more information: [http://martinfowler.com/articles/continuousIntegration.html](http://martinfowler.com/articles/continuousIntegration.html) “Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.”

- **Pair Programming:** This is a practice in which two programmers participate in writing some code by sharing a single workstation and keyboard. While one person is working on the code, the other is actively reviewing that code. This approach allows the driver to focus on the tactical aspect of writing code while the observer considers the strategic direction. The pairs change roles frequently.

- **Refactoring:** This is the process of making the code cleaner and simpler without changing its functionality (i.e. it is not rewriting the code). A single refactoring is generally a very small change, such as renaming a variable, or removing code duplication. However, the cumulative effects of many small refactorings can produce a significant restructuring of the underlying code.

---

2 [http://martinfowler.com/articles/continuousIntegration.html](http://martinfowler.com/articles/continuousIntegration.html)
Challenge 11: External parties have mandated ineffective tools.

Large organizations often feel the need to control the number and types of tools used by project teams, and often they have a very good reason to do this, such as licensing, IP and IP rights management, and security.

But when the authorization process or the tools themselves act as an impediment to the team making progress, this becomes an issue. If a tool doesn’t meet the developers’ needs, they won’t use it. Or if the authorization process for introducing a Continuous Integration tool, such as CruiseControl, takes more than a few weeks, then what does this say about the organizations attitude towards software quality?

At this point, the ScrumMaster and project team need to address the issue with senior management. They need to explain why certain tools are relevant and need to be supported, and they need to make management understand that having the right tools can make the difference between a good enough product and a high-quality product.

Challenge 12: Friction can exist between teams.

As a general rule, teams seldom present impediments to Agile adoption because most Agile frameworks strongly advocate close, cohesive team work. Initially, the learning process may be slow, but this is the nature of change rather than an impediment.

The most likely source of friction is between different teams rather than within a team. There is the possibility of friction in any situation when one team is dependent upon another or where two different teams are using different methodologies; Agile vs. RUP, for example.

In both these situations, the standard approach is to prioritize dependant functionality early and to code to an agreed upon interface, but this is overly simplistic. Instead, teams need to recognize that there is a dynamic relationship between dependant teams that needs to be actively and continuously improved. Failing to acknowledge the risks involved in dependant projects can quickly lead to confusion and impact the team performance.
Challenge 13: The way company rewards employees is in conflict with the “values” of Agile development.

In an environment where performance is determined by specialization of knowledge, the promotions and compensation models reward the compartmentalization of knowledge, which is in direct conflict with the Agile model. There are two reasons why this occurs.

The first is immediately obvious. If Alice is rewarded for her understanding of the security system, for example, then she is likely to continue doing this until the reward mechanism is changed.

The second point is not nearly as obvious, but it is a reaction to increased specialization. In order to circulate knowledge and information, groups are formed where the individuals share a common function. Typically this results in an organization where there are groups for analysts, architects, developers, and testers, etc. Project teams are then composed by selecting individuals from each of the different groups. Ironically, this functional grouping of people serves to further segregate of information.

Agile methodologies break down these arbitrary boundaries on a project-by-project basis by encouraging cross-functional teams. Long-term solutions are dependent on rewarding teamwork and breadth of understanding.

It’s not an issue that lends itself to a simple answer because it depends so much on the exact context. The trite answer is to change the system for developing software, but that’s so high-level as to be meaningless. While there is no single or easy solution, some ideas include having team goals and bonuses, eliminating individual performance reviews, and allowing teams the opportunity to make mistakes.

Conclusion

Successfully introducing Agile methods into an organization is not a simple task. It takes many different skills, both technical and political. The physical and political environment that you’re work within requires as much attention as the technical problems that you’re trying to solve.

The main traits that are needed for successful Agile adoption is willingness to listen and to change accordingly. Many organizations will insist they listen to their employees and that they have an Agile culture. Although this may be true to a certain extent, the real test always comes when the Agile values conflict with the organizational values. Provided that an organization is willing to listen and change then all these issues will be eventually resolved; it will only be a question of time and effort.
If you are currently considering Agile development or if you are struggling with challenges along the way, consider signing up for a course:
http://www.scrumology.com/courses-offered/.

About Scrumology

Scrumology Pty Ltd is a private held boutique consultancy that specializes in Scrum Training, Coaching and consulting. In addition to teaching public courses worldwide, we have facilitated and coached at some of the world largest companies including: Microsoft, Oracle, CapitalOne Financial, Nationwide Insurance, Getty Images, Sony and Expedia. Started in February 2009, Scrumology is lead by Kane Mar.

Kane Mar has been developing, coaching and leading software projects for the last 20 years. He has been an active member of the Agile software development community since 2001, when he first had the opportunity of working with Ken Schwaber. Mar became one of the first 30 Certified Scrum Trainers in 2006, and one of the very first Certified Scrum Coaches in 2007. He has training and coached software teams throughout North America and Northern Europe. Prior to joining the Agile and Scrum communities Mar was a Rational Unified Process (RUP) and waterfall process Project Manager working extensively with PriceWaterhouse’s SMM methodology.

For more information about Scrumology, visit our website: http://www.scrumology.com/.