Accepted for the ICGSE Workshop on Global Software Engineering for Agile Teams,
Porto Alegre, Brazil, August 27, 2012

Configuration Management – Mother's little helper for Global Agile Projects?

Lars Bendix
Department of Computer Science
Lund University
Lund, Sweden

Christian Pendleton
Softhouse Consulting
Malmö, Sweden

Abstract—**There are many good reasons for turning co-located projects distributed, likewise there are many good reasons for turning traditional projects agile. In both cases there are many obstacles to overcome and pitfalls to avoid and the combination of agile and distributed does not make this situation any better. In general Configuration Management works as the infrastructure of any software project and its concepts and principles have to be implemented in different ways depending on the specific context. How to adapt Configuration Management for an agile context is well understood, how to adapt it for distribution is less understood – and what changes and how to fix that when a co-located agile team goes distributed is unclear. In this position paper, we draw on our experience from academic and industrial agile projects to give our opinions on what to look out for and possibilities for solutions.**

Keywords-configuration management; agile development; global distribution; tools and techniques; challenges

I. INTRODUCTION

Agile development methods seem to have had quite some success in recent years, but agile has also become a buzzword that is used and abused in many contexts. Agile often seems to be used as a quick fix to any problem. Lately, the idea of using agile on distributed projects has surfaced though it may seem a contradiction since agile methods are built on face-to-face communication [19]. In a review of reviews of global agile research it is concluded that there is an increasing interest in the research world (measured on number of publications) for both distribution and agile – and that there is the need for more research that covers both [11].

Both authors of this paper have more than a decade of experience with agile development (XP and Scrum) in academic as well as in industrial contexts. However, they have little or marginal experience with distributed development and were curious to look into how agile changes under distribution. Furthermore, both authors are experts in Configuration Management (CM) and were particularly interested in understanding how the practice of CM will change from co-located agile to distributed agile – and what new things CM could offer to provide support in a distributed agile setting.

To be able to understand and distinguish the different scenario we were working with we created the mental picture shown in figure 1.

What most people are probably interested in is the direct transition from co-located traditional to distributed agile development (the dotted line). However, in our opinion that is probably a dream as it would be making too big a leap in one single step.

The transition from co-located traditional to co-located agile is well understood and equivalent to simply adopting an agile development method [4]. The implications for the practice of CM are also well understood [5].

The transition from co-located traditional to distributed traditional development is the core focus of the International Conference on Global Software Engineering and is well investigated even if there are still aspects that may not be so well understood.

The transition from distributed traditional to distributed agile development and the transition from co-located agile to distributed agile development are clearly within the scope of this workshop. In this position paper, we focus our attention on the transition from co-located to distributed agile. In particular our interest is the implications for the practice of CM, but we will also touch some of the more general problems of going distributed on agile projects.

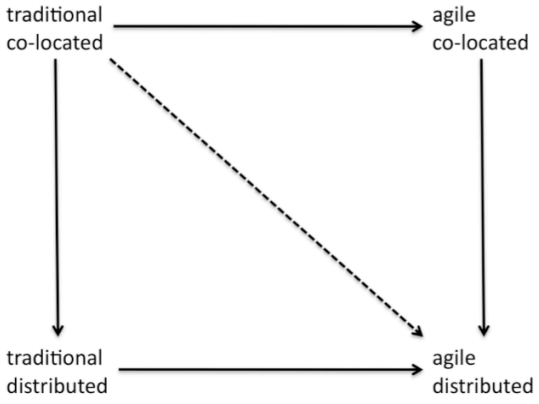

Figure 1. Moving to global agile.

In the following, we first have a short discussion of what we intend by the terms "agile" and "distributed" and a quick introduction to the area of CM. Next, we look at agile practices and analyse how their implementation will change when going distributed and how CM can support their implementation; followed by a discussion of how our proposals relate to existing work. Finally, we draw our main conclusions on how CM can support the transition from co-located agile to distributed agile development and sketch some future topics to investigate and discuss further.

II. AGILE AND DISTRIBUTION

Agile and distributed are concepts with many and vague definitions. We give a short discussion of what we intend when we use the terms in this paper. Neither of the concepts is "binary", so there are different shades or degrees of being agile and distributed.

A. Agile

What does it mean that a project team is agile? The agile manifesto [17] is rather vague and very open for interpretations. Even one of the founding fathers of the agile movement, Kent Beck, has changed his mind over the years. In the first edition of his book on Extreme Programming [3], he was rather rigid and required people to carry out all the prescribed practices in order to be considered agile. In the second edition of his book [4], he has "softened" somewhat. He has extended and refined the number of practices and divided them up into primary practices, that can be useful on their own and in any context (like Pair Programming) and corollary practices, that should not be used unless the relevant primary practices are in place first (like Shared Code, where Continuous Integration needs to be in place first). He also places much more importance on the five fundamental values on which he has built Extreme Programming: feedback, communication, simplicity, courage and respect.

In this paper we take "agile" to mean that the five values of Extreme Programming are followed. In particular that feedback is sought very frequently to obtain information on which decisions of possible changes of direction could be based. This definition includes not only Extreme Programming practices, but also Scrum practices and other practices or processes that follow these values.

B. Distributed

When is a project team being distributed? Does it have to be a situation similar to that of Open Source Software projects where the team consists of individuals scattered all over the globe? Or is it sufficient that the team consists of two groups, one located in Boston and the other in Tokyo? And would it also be distributed if the teams were located in adjacent rooms? Research works with three overall types of causes for distribution: physical distance, time-zone differences and cultural differences [16]. In this paper, we will primarily focus on the first two types. According to Piri et al. [13] distribution really makes a difference. In particular on a number of parameters like: Team Trust, Communication, Coordination, Mutual Support, Effectiveness (Quality), and Resourcing.

However, distribution can also be even more subtle than that. In the context of agile methods, which are based on quick and easy communication, a "distribution" effect can be caused by impediments for such communication. In one occasion we discovered that a team did not really use the information on the storyboard except at the stand-up meetings. It turned out that the reason was that when they worked they all sat with their backs to the storyboard and thus rarely looked at it.

So, in this paper we will take "distributed" to mean everything that causes obstacles for the quick and easy communication of information within a project team.

III. A QUICK INTRODUCTION TO CONFIGURATION MANAGEMENT

This section is a quick introduction to the most important concepts and principles in CM. It will give an alternative picture to the CM knowledgeable readers and make it easier for the CM-uninitiated reader to follow our subsequent discussions and reasoning.

A. Traditional Configuration Management

Traditionally CM is looked at as process biased and aiming at regulating the interface between the customer and the company and enabling the management of a development project. It consists of four activities: Configuration Identification, Configuration Control, Configuration Status Accounting and Configuration Audit [12].

The *Configuration Identification* activity makes sure that important artefacts of a project are put under configuration management, whereas the *Configuration Status Accounting* activity reports on the status of artefacts and development. One way to look at this is that the first activity is the definition of a database. What objects should be put into it, what information should be recorded about the objects, how should they be structured and what relations should exist between them. The second activity then becomes queries on this database of project artefacts, both standard queries and ad-hoc queries to provide information from the collected data. *Configuration Control* has focus on managing changes to project artefacts. It ranges from collecting relevant information about Change Requests over their treatment on the Change Control Board – that decides which Change Requests to accept, reject or defer – through the process of following them through to their closure. The decision part of this process has a "control" aspect to it, whereas the rest of the process can be seen as collection and provision of information. The purpose of the *Configuration Audit* activity is to make sure that a project is ready to deliver what is has promised. Much like when you see an airline pilot walking around his airplane before leaving, making sure that the engines are still there, that there is not ice on the fuselage and so on. Had the pilot had sensors – and trusted them – he

would not have to walk around to gather his information. Likewise, software development teams that do not have "sensors" (like valid measurements on release quality) – or do not trust them – will do Configuration Audits before releasing their products.

B. Team-oriented Configuration Management

A different view of CM sees it as tool biased and aimed at helping a team of developers to co-ordinate their parallel work and to handle the day-to-day and minute-by-minute evolution of the software inside the development team.

Wayne Babich [2] identified three fundamental problems that he had seen happen in the coordination of individual people's work in a team: shared data, simultaneous update and double maintenance. The *shared data* problem happens in situations where a problem is caused by the changes of other people – changes that we were not aware of. The *simultaneous update* problem happens when someone accidently overwrites and removes someone else's change. The *double maintenance* problem happens when something is copied – and changes are made to one of the copies. In order to keep the two copies identical the exact same change has to be made in the other copy too. We can never hope to completely eliminate these problems, but by clever use of processes and tools we can hope to be able to manage them. Peter Feiler [8] distilled work models of version control tools. They introduce the concept of a *workspace* to isolate people from other people's changes and unexpected Shared Data problems. Tools perform *concurrency checks* to avoid the Simultaneous Update problem by not allowing a commit to the repository if someone else has already committed a change. However, there are still subtle ways in which other people's changes can accidentally be removed. *Merge functionality* helps manage the Double Maintenance problem by automatically integrating the changes in one copy into another copy. For the synchronization of people's work early tools provided *locking mechanisms* to stop parallel work on the same components, but gradually moved towards a more relaxed model that allows parallel work since there is tool support to merge possible parallel changes. The *transaction model* changed from a very simplistic model where people were focused on single files that were committed one at a time towards the concept of logical changes where a set of changes are committed in one atomic operation.

IV. From Co-located to Distributed Agile

When going from co-located to distributed agile it will affect the way the team is working. Some practices (like Pair Programming) will be invariant to distribution, while other practices (like Sprint Planning or Continuous Integration) will have to be implemented differently. Furthermore, there may be some practices that cannot be turned distributed (like Sit Together), just like there may be the need for additional practices to take care of aspects pertaining to the distribution. In the following, we will present a small selection of the practices – CM-related as well as CM-enhanced – that we have looked at and analysed.

A. CM-related practices

Continuous Integration: Continuous integration is the response to the Double Maintenance problem [20]. CM can set up different strategies that allow the team flexibility in how often and how tight integration should be. When the integration runs smooth there is very little impact from distribution. The only thing being that it is more difficult to know when there is something new to integrate. CM can easily set up mechanisms to automatically notify everyone on the team. When there are problems with an integration the developer has to communicate with the other parties involved in the problem.

Ten-minute Build: Build automation is one of the services that CM is able to provide [20]. Like Continuous Integration, it is only in the case of problems that developers will have to communicate and will notice any difference in the change from a co-located to a distributed setup. Ten-minute Build is a "pre-requisite" for Test-Driven Development since it builds on it being easy and fast to build and unit-test the code [20].

Frequent Releases: If Continuous Integration and Ten-minute Builds are in place, it becomes quite easy to implement Frequent Releases. CM can provide means for doing Configuration Audits before release. Likewise there are version control tools that allow the introduction or exclusion of a story at the last moment by advanced use of transaction packages.

Sprint Planning: Sprint planning is a team activity in Scrum. The product owner may participate, but if the backlog items are prepared in advance, it is usually not required. Since we are dealing with the backlog items here, traceability can be boosted by proper routines for identification (same as for backlog handling). The way of conducting sprint planning will have to change dramatically if the team is distributed. Common sprint planning practices, like planning poker, will suffer from communication over some kind of media. It is hard to say that any CM practice can solve those problems though.

B. CM-enhanced practices

Backlog Handling: The backlog handling in Scrum is really the responsibility of one person – the Product Owner [21]. The need for CM support in the operational work is therefore quite low, neither is the impact of distribution very severe. But a backlog that is not visible to those who use it will not give value to the team. In a small, co-located team one can often do with just a bunch of cards (like in XP) that are sorted in priority order. When the team grows or goes distributed there will be the need for more controlled procedures for the backlog handling and a few CM practices will come to use. Publishing the backlog in a controlled way will make everyone in the team receive the changes simultaneously and will decrease confusion when communicating about the backlog. This kind of publishing is actually a baseline adapted to agile values. To make traceability from backlog items, through development, to

deliveries possible configuration identification is needed. If we add distribution to the scenario and work with several teams from different sites where backlog items can have dependencies between the teams, identification and traceability is crucial.

Informative Workspaces: A main objective of CM is to collect and present information. When information is digitalized and put online it will also give the team flexibility to communicate in ways that are asynchronous, public and persistent instead of relying on the normal verbal communication in a co-located setup – and it means that the team will be less affected by distribution. The digitalization also gives possibilities for capturing more information, automating some steps in processes and quality control of data. Finally, it is much easier to search and mine digital data.

Daily Scrum: The daily scrum is the short daily meeting dealing with the tasks within the current sprint. In a non-distributed environment, CM involvement is limited to possible traceability of tasks vs. backlog items and trouble reports – i.e. identification. If the organization is distributed with different teams on different sites, it is good to consider how to increase transparency towards teams on other sites since it is difficult to participate in the scrum of another team when there are dependencies in the sprint planning that needs to be handled. There are several tools on the market to support this, but once you go digital with the scrum board you most likely will want to integrate with other tools like version control tools and issue management tools. If the team itself is distributed over several sites, the digitalization of the scrum board soon becomes necessary and even more central in the daily work of the team. Again – the more distributed, the more benefits from routines for identification and publishing to make communication about the backlog items and the tasks easier and straight forward

Burndown Chart: The burndown chart is the major "in-sprint-follow-up tool" in Scrum. It shows the progress during the sprint and is handled by the team. Tools for computerized scrum boards almost always include a function for the burndown chart. There is little direct relation with CM here.

V. Discussion

There are many reports on difficulties and failures with blending agile and distributed. However, there are also a few studies that report success [6, 14]. Both of which show the importance of understanding the agile philosophy and also have strong aspects of CM without explicitly mentioning it.

It is important not to create silos in the distribution of work [15] but work in a more flexible network of changing connections like [7]. CM can help support the move from distributed teams to distributed individuals.

Gupta et al. [10] report on the lack of conclusive evidence of collaboration tools. They found that people rarely communicated and collaborated in planned ways but most often made ad-hoc decisions. This may indicate that a collaboration framework that is based on asynchronous collaboration through a knowledge base could be a solution rather than traditional collaboration tools there the benefits have not been established [18]. CM could provide at least part of such a more flexible framework.

One of the differences between co-located and distributed projects pointed out by Piri et al. [13] is the lack of trust in the distributed setup. Trust is very important for agile teams since it is strictly related to the values of courage and respect. One way to build trust in a team is to increase information and awareness about what is going on in the project. As an information providing service CM offers many possibilities to give support for that even on a distributed project.

From "prior experience" reported, it seems that most of the time when people have problems in going from traditional distributed to agile distributed it is because they lack even the most elementary understanding of what agile is – or because they think they are agile just because they do a couple of practices or skip the agile principles under pressure [9].

Alyahya et al. [1] deal with how to manage progress on distributed agile projects. The problem is that distribution makes it hard to create awareness about what changes go on and at what point they are. This affects the developers' understanding of development progress. They propose a holistic approach to manage the development progress. They want to monitor Unit Testing, Acceptance Testing, Continuous Integration and Source Code Versioning. This is quite similar to what happens in the Configuration Control activity after a Change Request has been approved and assigned. They state that it can be a problem that developers can forget to update the status when it has to be done manually, but following traditional CM standards such misses will be caught by the Configuration Audit. Finally, the overall awareness they aim at with their proposal is little more than what is (or could be) provided by traditional Configuration Status Accounting.

VI. Conclusions

It seems a contradiction that a method like "agile" that is so profoundly rooted in quick, easy and high-bandwidth communication should be proposed as the "fix" to problems with distribution. Our impression is that it is the "extremely frequent feedback" part of "agile" that works wonders. So, we have to find "fixes" for the loss of communication we experience when we go from co-located to distributed agile.

CM may not be a "little yellow pill" but if "you go running for the shelter of a Mother's little helper" it will certainly "help you on your way, get you through your busy day". CM provides support today on traditional co-located projects where focus often is on the control aspects of CM. However, CM could be even more helpful on agile distributed projects and since such projects thrive on seeking and consuming information there will have to be a stronger focus on that aspect of CM.

ACKNOWLEDGEMENT

We would like to thank Marc Girod, Ericsson, Ireland, for comments, ideas and discussions – and The Rolling Stones for inspiration for the title of the paper.

REFERENCES

[1] S. Alyahya, W. K. Ivins, W. A. Gray: "Co-ordination Support for Managing Progress of Distributed Agile Projects", in Proceedings of the First Workshop on Global Software Engineering for Agile Teams, Helsinki, Finland, August 15, 2011.

[2] W. A. Babich: "Software Configuration Management – Coordination for Team Productivity", Addison-Wesley Publishing Company, 1986.

[3] K. Beck: "Extreme Programming Explained: Embrace Change", Addison-Wesley Professional, 1999.

[4] K. Beck: "Extreme Programming Explained: Embrace Change (2nd edition)", Addison-Wesley Professional, 2004.

[5] L. Bendix, T. Ekman: "Software Configuration Management in Agile Development", in I. G. Stamelos, P. Sfetsos (eds.) "Agile Software Development Quality Assurance", IGI Global, 2007.

[6] S. Berczuk: "Back to Basics: The Role of Agile Principles in Success with an Distributed Scrum Team", in Proceedings of AGILE2007, 2007.

[7] S. Datta, R. Sindhgatta, B. Sengupta: "Evolution of Developer Collaboration on the Jazz Platform – A Study of a Large Scale Agile Project", in Proceedings of ISEC, 2011.

[8] P. H. Feiler: "Configuration Management Models in Commercial Environments", Technical Report, CMU/SEI-91-TR-7, Carnegie-Mellon University, Pennsylvania, March 1991.

[9] M. Giblin, P. Brennan, C. Exton: "Introducing Agile Methods in a Large Software Development Team: The Developers Changing Perspective", in Proceedings of XP2010, 2010.

[10] M. Gupta, J. Fernandez: "How Globally Distributed Software Teams Can Improve their Collaboration Effectiveness", in Proceedings of ICGSE, 2011.

[11] G. K. Hanssen, D. Smite, N. B. Moe: "Signs of Agile Trends in Global Software Engineering Research: A Tertiary Study", in Proceedings of the First Workshop on Global Software Engineering for Agile Teams, Helsinki, Finland, August 15, 2011.

[12] A. Leon: "Software Configuration Management Handbook", (second edition), Artech House, 2005.

[13] A. Piri, T. Niinimäki: "Does distribution make any difference?", in Proceedings of the First Workshop on Global Software Engineering for Agile Teams, Helsinki, Finland, August 15, 2011.

[14] B. Ramesh, L. Cao, K. Mohan, P. Xu: "Can Distributed Software Development Be Agile?", Communications of the ACM, October 2006.

[15] S. V. Shrivastava, H. Date: "Distributed Agile Software Development: A Review", Journal of Computer Science and Engineering, May 2010.

[16] F. Q. B. da Silva, C. Costa, A. C. C. França, and R. Prikladnicki: "Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review", in Proceedings of the 5th International Conference on Global Software Engineering, Princeton, New Jersey, August 23-26, 2010.

[17] Various authors: "Manifesto for Agile Software Development", agilemanifesto.org, 2001.

[18] J. Whitehead: "Collaboration in Software Engineering: A Roadmap", in Proceedings of FoSE, 2007.

[19] "First Workshop on Global Software Engineering for Agile Teams (Globagile)", www.inf.pucrs.br/munddos/globagile/, Helsinki, Finland, August 15, 2011.

[20] U. Asklund, L. Bendix, T. Ekman: "Software Configuration Management Practices for eXtreme Programming Teams", in Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques - NWPER'2004, Turku, Finland, August 17-19, 2004.

[21] K. Schwaber, M. Beedle: "Agile Software Development with Scrum", Prentice Hall, 2001.