Configuring Hybrid Agile-Traditional Software Processes

Adam Geras[1], Michael Smith[2], and James Miller[3]

[1] Ideaca Knowledge Services, Calgary, Alberta, Canada
adam.geras@ideaca.com
[2] Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada
smithmr@ucalgary.ca
[3] Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada
jm@ece.ualberta.ca

Abstract. The traditional versus agile project debate is most often represented in terms of polar positions of the life cycle – the process is either traditional or agile, waterfall or highly iterative. This may be effective in intellectual discussions, but it is highly unlikely to be useful to practitioners, especially those practitioners that are facing traditional project pressures and trying to find the "home ground" for their situation that will increase the likelihood that they will succeed. In this paper, we discuss extensions to Boehm and Turner's five dimensions for determining a project's "home ground" – that is, the process configuration that might best fit the situation at hand. We have added dimensions to the basic framework provided by Boehm and Turner and have considered the process configuration question as a process itself and increased its scope to include both management and engineering key practice areas.

1 Introduction

As agile processes enter the mainstream, it is becoming increasingly clear that many organizations will attempt at least some, if not all, agile practices, especially given the increasing pressure on software development organizations to be adaptable [1]. Boehm and Turner specify the dimensions of method selection as "criticality, size, personnel, dynamism and culture" [2]. In this paper, we first evaluate, by drawing upon both personal expertise and knowledge provided by a number of project managers, the re-categorization of software process determinants into two broad categories: customer/ developer concerns, and product/environment concerns. Then we will describe a process for configuring hybrid agile-traditional software that uses those determinants. By characterizing the customer/developer and the product/ environment, we are enabling a software process that is discovered and applied based on its context – a context-driven software process.

2 Software Process Determinants

In this section, we will describe the software process determinants used and the categories into which these are placed, as sh own in the Kiviat charts for the customer/ developer profile (Fig. 1) and the product/environment profile (Fig. 2).

P. Abrahamsson, M. Marchesi, and G. Succi (Eds.): XP 2006, LNCS 4044, pp. 104 – 113, 2006.

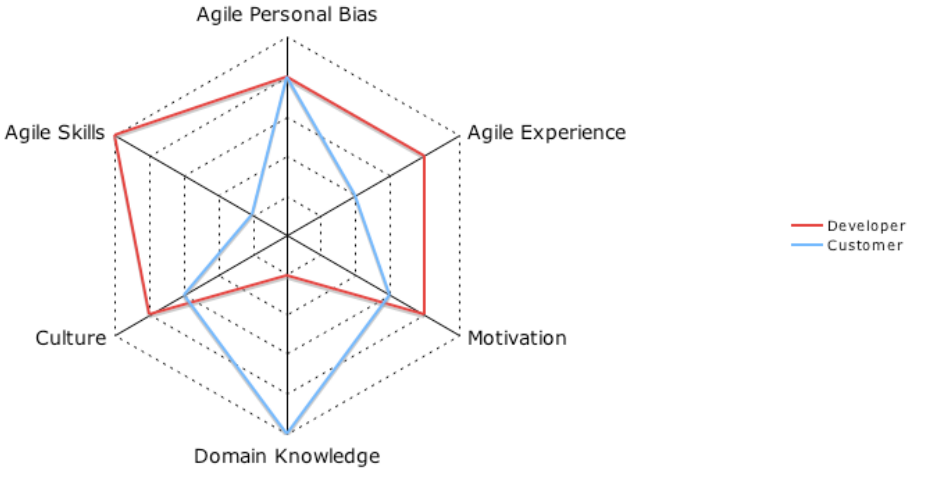

Fig. 1. The Customer / Developer Profile - The intent is to create one profile for each of the customer and developer, so that any highlighted distinctions can be addressed as risks and deviations from the ideal agile or ideal traditional home grounds can be assessed

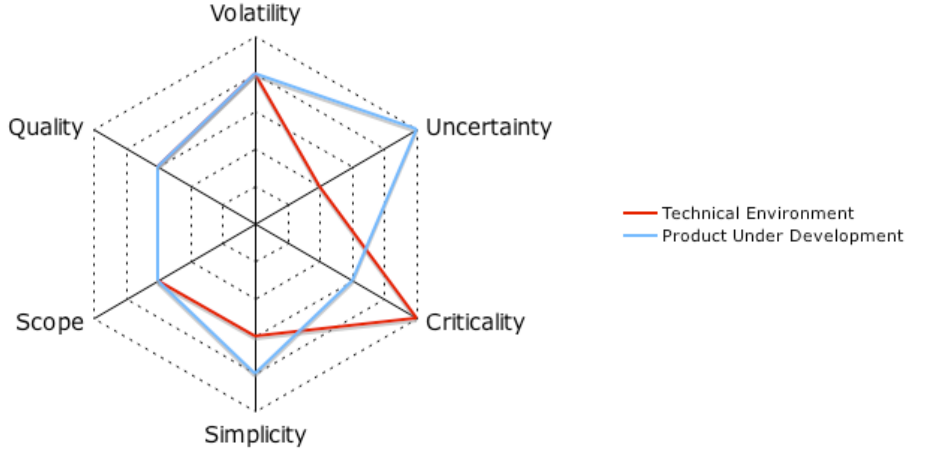

Fig. 2. The Product / Environment Profile - The intent is to create one profile for each of the product and the technical environment it will ultimately operate in, so that again any highlighted distinctions can be addressed as risks

2.1 The Customer/Developer Profile

The process determinants in the customer/developer profile describe the customer and developer in terms of their culture and values, skill, and history. Illustrative of the importance of the customer, most adaptive processes rely on user involvement as a key principle. Dynamic Systems Delivery Method (DSDM) uses "Ambassador User" and "Advisor User" roles as the archetype of all customers on the project [3] to again

signify that user involvement is a key to the success of the project. Similarly, Highsmith states that agility involves much more than reducing documentation or being lean - it's also about working collaboratively [4]. Cockburn incorporates the customer profile as part of the "personal anchors" in [5].

Boehm similarly includes the customer profile as part of the "culture" dimension in [2]. Both Boehm and Cockburn cite the culture of the customer organization as possibly distinct from the culture of the supplier organization; reinforcing the point that identifying and characterizing the culture of the customer is just as important as understanding the culture of the supplier. Boehm acknowledges that the customer representative becomes the primary stress point for agile methods [2], a point that highlights the relevance of the customer's *domain knowledge* as a process determinant. He further characterizes the importance of the customer relationship but unfortunately, his argument inappropriately boils down to talk of contracts and customer relationships that are characterized by formal agreements. The "human" side of the customer relationship should also be considered significant! The people fulfilling the customer/end-user role may not be in their comfort zone when working on the project, and preparing for their potential reactions to unforeseen events may prove fruitful in maintaining progress. They are, after all, domain experts and not necessarily software project experts. Hence, our primary customer profile process determinants, differ from Boehm and Turner, and are illustrated in Fig. 1.

The customer's *agile personal bias* indicates their particular experiences in previous software projects. If the projects were successful, then there may be a personal or even collective bias towards project styles and techniques that were successful. Even without this history, the customer's *agile project experience* level will also partly drive the determination of the optimal project style. For example, given a customer that has successfully accepted software in the past, the team may approach them to be more highly involved in decision-making. A customer that has less experience and tends to 'panic' at the slightest sign of trouble may be treated differently. On one project we witnessed the team instituting an additional testing level to shield the customer from the daily builds to counter the customer's panic that ensued from finding cosmetic errors.

Some projects also reported that customer availability is a limiting factor on their ability to use agile processes. Getting timely feedback is critical on the project, but sometimes it just isn't logistically or politically possible to have the customer/end users available as full-time members of the team. In many organizations, for example, the customer still has their regular, full-time job, alongside participating as the key user representative on the project. Both agile and non-agile processes would benefit from a high level of user involvement so a low ranking on this dimension should be treated as a risk on the project and an appropriate response designed-into the process. Availability is considered part of the '*motivation'* axis on the chart in Fig. 1.

A customer's personal style also plays an important part in determining an optimal software process. We have called this '*culture*' on the chart. People and organizations that struggling with accommodating or embracing change will find working with an agile method difficult. Similarly, if they have trouble with ambiguity then a development style that involves discovery (iterative and evolutionary) again might not work for them. This interpretation of 'culture' is identical to [2] except we apply it to the individuals and the organization separately. We have placed significance on the

separation of these two criteria based on experience with agile teams within non-agile organizations. In observing the behavior of the agile team and its customer within the non-agile environment, we concluded that the individuals on the team drove much of the agility despite the non-agile surroundings. We found agile projects thriving within non-agile organizations, and therefore concluded that making generalizations about a given organization does not serve software process configuration well.

The corporate and IT culture of the organization will also play a part in setting the software process. In many organizations, the funding for projects is based on a satisfactory (and approved) business case. This funding model is a precursor to a fixed-price, fixed-scope contract, even if the developer organization is an internal one. This type of contract makes agile development difficult (not impossible, but difficult) since the primary lever of control – variable scope – is less available. Similarly, IT culture may end as constraints on the project processes. Agile projects rely on multiple releases to achieve shorter time-to-benefits periods and to give the development team early feedback. If there is a rigorous environment change control process that any changes to production have to go through, then there may be some tension between the project and the organizations that enforce the change mechanism. The interactions between the project team and other IT organizations have to be considered in the configuration of the process. If not, the likelihood of creating an adequate development process decreases.

Finally, we have included a dimension on *agile skills*. In our experience, the agile project places significant technical demands on the people fulfilling the customer role. This is particularly notable in the areas of requirements management, change management, and testing. All modern software processes require user involvement, but some of the agile methods – extreme programming in particular – makes them part of the team with specific responsibilities for prioritizing user stories (requirements) and for developing and running customer tests. This dimension is not so much an assessment of the customer's ability to use computers, as much as it is an assessment of their skills in the agile practices that they must use to drive the project. If the customer were more familiar with and skilled in agile project practices, then they would receive a ranking on the periphery of the dimension. As with the customer agile skill assessment, the developer's agile skills are again not assessed from good to bad, but instead ranked based on their experience with an agile toolset and techniques (xUnit, FIT, refactoring, pair-programming, etc.). The ranking should reflect the developer's comfort with the agile practices and associated tools. If a team is not familiar with refactoring and test-driven development, then asking them to use these techniques to design and deliver a mission-critical system will not be optimal!

2.2 The Product/Environment Profile

The product profile is illustrated in Fig. 2 and shows a number of dimensions that are identical to what Boehm and Turner used in [2], specifically dynamism (relabeled as *volatility* in our figure), *scope*, and *criticality*. We have re-labeled the rating scales for complexity as '*simplicity*' to reflect the agile axiom for "keeping things as simple as possible" [6]. From the product perspective, the simplicity rating should reflect the amount of simplicity that the team can get away with and still deliver an adequate solution. The environment perspective, on the other hand,

should reflect the simplicity of the technical environment that the product will have to operate in. For example, certain systems may have low computational complexity but be required to inter-operate with several existing systems, making it's architecture more complex.

In relating Boehm's "dynamism" dimension to project managers, they often break it down further into two concerns –volatility and uncertainty. Volatility represents an assessment of the extent of changes that may appear over the course of the project. Uncertainty is related to other changes, such as architectural uncertainty, manpower issues or changes in the business climate.

In summary, we have found it beneficial to characterize the customer, the developers, the product and the technical environment by creating two profiles and then suggesting use of the resulting chart shapes to devise an appropriate starting software process for projects. The determinants presented here are examples of what could be done in any given setting – the actual choice of dimensions would be left to the person or organization performing the process configuration. With the profiles prepared, the process configuration can occur as part of a workshop at the beginning of the project. A proposed process for completing the configuration is presented in the next section.

3 Proposed Configuration Process

A person or team that has above-average communication and analysis skills is needed to complete the process configuration. Much of the information to be collected in order to construct the profiles is not readily available – it will take a number of interviews and a healthy dose of interviewing skill to be able to accurately assess many of the dimensions. In particular, the profile dimensions related to project histories and experience level. Few people want to talk about previous projects that have gone badly, even in project retrospectives. To obtain this information early in a new project may therefore require advanced communication skills.

In this section, we will discuss when a software process should be configured and the following steps in detail: *profiling* the project context, *aligning* the key practice areas with the profiles, *preparing* the team for the project, *running* the project, and then *checking* the configuration at regular points throughout the project. Configuring a process to suit a project is one of the highlights of Cockburn's work in [7]. Essentially what the process 'configurator' is seeking is a set of levers that can be adjusted, ultimately creating an initial process that the team can use as a starting point. The inputs to setting the 'levers' are the profiles discussed above.

3.1 Step 1 – Profiling

This step consists of conducting the necessary interviews, workshops, meetings, so that the customer, developer, and product profiles can be built. This may be difficult for a number of reasons. First, the customer may not be readily available for profiling. In competitive bidding, for example, the suppliers have to somehow envision the customer and product profiles based on the information they are given in the Request for Proposal (RFP).

In other situations, the profiling step can easily be incorporated into the existing scoping and requirements identification steps. Workshops set up to craft the first-stage business models and any other information-gathering sessions that are conducted can include considerations for gathering the profile information. Another guideline is that anything that would be useful for estimating is also useful for process configuration – especially in terms of risk. In this paper, we've avoided mentioning risk since its role in process configuration is well described in [2]. The intent there is that any dimension is a risk if it gets assessed as outside of the "home ground" that is ultimately chosen for the project.

3.2 Step 2 – Aligning

Aligning the process to the profile has been simplified in [2], making it sound like a simple binary decision between plan-driven and agile, and that anything in between can be handled as a risk. This warrants further discussion, and to handle that discussion in meaningful pieces, we have to break the project activities down further. The goal is to define a set of 'levers' that can be adjusted to define a process that will deliver a desirable product, and there are many aspects of that ecosystem that can be tailored and adapted. It's not an "all or nothing" decision. There is even a strong argument for suggesting that much of the future enterprise development will be done using hybrid agile and plan-driven methods [7].

We propose using some of the Capability Maturity Model (CMM[sm]) Key Practice Areas (KPA) as the basis for identifying project activities that can act as the 'levers' for configuring the software process. The CMM KPA's that are organizationally-focused (technology change management, process change management, organization process focus, organization process definition, and training program) are excluded given that we're configuring a process for a project, not an organization. Similarly, defect prevention and software quality management are excluded on the basis that they don't have agile and plan-driven extremes, unlike other engineering-related KPA (product engineering and peer reviews). We have also added iteration duration to the list given that we have witnessed organizations use it as a benchmark of agility for their active projects. The set of activities that we have used is listed in Table 1.

Table 1. Project activities based on the CMM Key Practice Areas (KPA) can be used as 'levers' that the team or 'process configurator' can adjust to match the project context

Project Activity	**More Agile**	**Less Agile**
Iteration Duration	2 weeks or less	8 weeks or longer
Requirements Management	User stories on cards	Use case descriptions
Software Project Planning	Entire team involved	PM/Tech Lead involved
Software Project Tracking	Burn-down charts, tests	Earned Value
Software Quality Assurance	Entire team involved	Separate team
Software Configuration Mgmt	Continuous integration	Periodic integration
Peer Reviews	Pair Programming	Formal Inspection
Product Engineering	Test-driven	Test-last

Iteration duration was added to the list because of its impact on the overall approach taken to the project. Shorter iterations imply a more advanced level of

agility. Not all agile teams are able to sustain short-duration iterations such as 1 week. For most teams, even 4-week iteration durations are challenging at first, until the team gains some practice and establishes an increasing number of lean techniques.

Change management and release management were mentioned as primary concerns for many project teams. In terms of the KPA, change management fits mostly in *Requirements Management* and release management fits mostly in *Software Configuration Management*. There are significant differences between "low ceremony" and "high ceremony" change/release management. The high influencers for change/ release management are going to be the cultural dimensions and the developer technical skills. A low-ceremony change management approach would use more face-to-face conversations to describe changes and a prioritized feature list to maintain the order of new and changed features as compared to the existing backlog. A high-ceremony change management approach would involve completing a change request form and basically instantiating a workflow to qualify, approve, schedule, and assign the change. There may even be monetary compensation involved for making approved changes in a high-ceremony change management approach. Developer skills are a high-influencer because of the extent that agile teams automate the build process – some teams implement the agile practice of "continuous integration" using solutions such as Cruise Control that creates a new build and runs associated tests on every source code check-in event. However, these approaches have a steep learning curve.

The first activity to be aligned to the profiles is iteration duration. This assumes that at least some form of iteration is going to be used, a relatively safe assumption. Few organizations are willing to plan for a completely non-iterative project. Instead, the question has really turned into a debate over the length of the iteration more than a decision to develop iteratively or otherwise. To align the iteration duration with the profiles, the first step is to look at the 'high influencers' – that is, the profile dimensions that influence the iteration duration the most.

The high-influencers for iteration duration are probably customer bias, customer motivation, culture, customer agile skills, developer bias, developer agile skills, volatility, uncertainty, and criticality. You can use either another Kiviat Diagram or a weighted ranking to determine the final outcome. As Boehm suggests, if any of these dimensions fall outside the stated decision, then they can be handled as risks [2]. Once the high-influencers are identified and ranked, then the optimal iteration length can be derived from the rankings. The underlying assumption here is that you decrease the length of the iterations if you can, to a minimum of 1-2 weeks.

Once iteration duration is configured then the other KPA can be configured using similar steps. The Change and Release Management KPA are closely related to iteration duration, so it might be appropriate to configure them next, but at the end of this step, all of the KPA should be addressed holistically to ensure that the proposed configuration of each one of them is appropriate – again bringing up connotations of Highsmith's ecosystem [9,10].

The ecosystem approach to the practices within a team is particularly acute in considering the Product Engineering KPA. In this area, requirements analysis, design, construction and testing are all considered as related activities. Taken together, the activities could implement a test-driven development, or a highly iterative test-last method. To establish a thriving ecosystem, the configuration of the product engineering practices then has to be integrated with the other KPA, in particular the Quality

Assurance, Software Configuration Management, and Project Planning and Tracking/ Oversight KPA. The high influencers on the Product Engineering KPA are the developer technical skills, the developer experience level, the developer's domain knowledge, and the volatility and uncertainty associated with the product.

In conclusion of the Aligning step, the team should have a shared vision of the development workflow. It might even make sense to informally model this workflow and display it publicly so that the team has an easily accessible depiction of the workflow to discuss. The underlying sensibility of the workflow is "practice makes perfect" – so that once development begins, the team can start practicing the intended workflow and over time – get better and better at it, fine-tuning it as the project proceeds.

3.3 Step 3 – Preparing

Even if the team actively participated in the process configuration profiling and aligning steps, they may still need to be prepared in order to make the envisioned process a reality. The best way to complete this preparation is by running Iteration 0 – an iteration that delivers nothing of value to the customer but allows the team some practice time. This is especially critical if all the members of the team are not familiar with all the underlying tools. The length of Iteration 0 does not have to conform to the same length as the rest of the development iterations - if it extends beyond two weeks, there is probably something else going on other than preparation.

3.4 Step 4 – Running

Once prepared, the development iterations can be launched, and the team can start performing the activities that comprise the envisioned workflow. As the team completes the workflow, their progress should be measured in an unobtrusive manner in order to feed the next phase, checking.

3.5 Step 5 – Checking

Checking is confirming that the current process and development workflow is optimal. This should be done periodically, probably at a greater frequency than iteration cycles (especially, if the iteration length is longer than 4 weeks). Checking enables the entire team to assess the earlier rankings and to fine-tune the development workflow and project technical processes as required.

3.6 Challenges

Configuring the process in this manner is difficult for a number of reasons, but the greatest danger comes from not knowing the individuals that will comprise the team at configuration time. Many software development organizations don't make explicit resource plans until after the project is confirmed. In competitive bidding situations, for example, the project configuration is done and offered to the customer as part of the bid process. Placing personnel on the team then has to be done with the promised software process in mind. In addition, the development team may not meet the individuals that will ultimately fulfill the customer role on the project until the project is launched. This will make tailoring the process for their personal bias impossible.

Other challenges to this process are on the relative uncertainty over the influence of certain dimensions. Take the 'Motivation' dimension of the developer profile as an example – there are sure to be differing opinions on how to deal with this. Some will say that an agile approach is better for dealing with this since then the effects of the low motivation (poor productivity) would be noticeable sooner. Others will say that the 'empowered teams' of agile is less likely to be effective when the team members have little motivation. This is just an example – the point is that the influencing dimensions and their effects are probably not universally applicable.

4 Future Work

The proposed configuration process is being used in an industrial setting in two ways – to configure projects as outlined here, and to help existing teams create a test strategy that matches the existing project context. A qualitative analysis of these projects will follow pending ethics and the participating companies' approvals.

5 Summary

Boehm and Turner specified five dimensions – size criticality, dynamism, personnel, and culture as the keys to finding a project's "home ground" [2]. This home ground represents the optimal balance between agile and plan-driven processes, with the exceptions being managed as risks. This approach is an exceptional contribution to the notion of tailoring the software process to match the project context. In this paper, we have extended the tailoring process in two ways – by first articulating dimensions of more resolution and second by proposing a process for conducting the configuration that considers the additional dimensions and the key practice areas that they might influence.

The underlying assumption is that hybrid projects are most likely to be the primary means that large organizations will be using to deliver working software to their users for the foreseeable future. Purely plan-driven processes are increasingly rare. Even if they are advertised, they are less likely to be followed to the letter. Even traditionally non-agile companies are starting to try out some aspect of agile software development. Based on this increasing need, a strong understanding of the relationship between the configuration criteria (dimensions) and the key practice areas is required. If we have this understanding, then we have a better grip on what sort of process might be optimal for any given customer/developer and product/environment combination.

References

[1] Correia, J. Recommendation for the Software Industry During Hard Times. Gartner Data-quest Report, June 6, 2002.

[2] Boehm, B. and R. Turner (2004). Balancing Agility and Discipline: A Guide for the Perplexed. Toronto, Addison-Wesley.

[3] DSDM Consortium. (2005). DSDM Lifecycle. DSDM Consortium. http://www.dsdm.org/tour/process.asp. Accessed 2005.

[4] Highsmith, J. (2005). AgileVsSelfAdapting. Alistair Cockburn.http://alistair.cockburn.us/crystal/wiki/AgileVsSelfAdapting. Accessed 2005.
[5] Cockburn, A. (2000). Just-In-Time Methodology Construction. Alistair Cockburn. http://alistair.cockburn.us/crystal/articles/jmc/justintimemethodologyconstruction.html. Accessed 2005.
[6] Beck, K. (1999). Extreme Programming Explained. Don Mills, Addison-Wesley Publishing Co.
[7] Cockburn, A. (1999). A Methodology Per Project. Alistair Cockburn. http://alistair.cockburn.us/crystal/articles/mpp/methodologyperproject.html. Accessed 2005.
[8] Barnett, L. and U. Narsu (2003). Planning Assumption: Best Practices for Agile Development. Cambridge, Mass., Forrester Research, Inc.
[9] Highsmith, J. (2004). Agile Project Management. Toronto, Addison-Wesley.
[10] Highsmith, J. A. (2000). Adaptive Software Development. New York, Dorset House Publishing Co., Inc.