Mix and Match – Making sense of the Best Practices jigsaw: Attainability

Paul Oldfield

(with help from Mike Cohn)

Mentors of Cally

Member of the Appropriate Process Movement

http://www.aptprocess.com/

Abstract

Not everyone favours mix and match approaches to agile methodologies. In some respects this attitude is quite correct, there is no need for another methodology consisting of pre-selected best practices. In other respects it is incorrect, existing methodologies are the result of such a mix and match effort, or alternatively advocate the selection of best practices as an ongoing process. Possibly of greater significance, those truly agile developers who decide how to do things as they proceed are, in effect, mixing and matching from their own repertoire of techniques and best practices, on the fly. Some of the ideas and techniques that get included into the mix may appear to have nothing to do with the development process but instead address, say, the environment or the social aspects of a team. Anything that helps delivery of working software is fair game for mix and match.

Copyright Note

Mix and Match – Making sense of the Best Practices jigsaw: Attainability

Not everyone favours mix and match approaches to agile methodologies. In some respects this attitude is quite correct, there is no need for another methodology consisting of pre-selected best practices. In other respects it is incorrect, existing methodologies are the result of such a mix and match effort, or alternatively advocate the selection of best practices as an ongoing process. Possibly of greater significance, those truly agile developers who decide how to do things as they proceed are, in effect, mixing and matching from their own repertoire of techniques and best practices, on the fly. Some of the ideas and techniques that get included into the mix may appear to have nothing to do with the development process but instead address, say, the environment or the social aspects of a team. Anything that helps delivery of working software is fair game for mix and match.

The question arises; how do we manage the mix and match process to arrive at an appropriate development process? We can start to answer this question by considering what would make a development process appropriate. There are three primary properties that such a process needs.
It must be Adequate – the process must be capable of producing the required products and meeting at least the most important subsidiary goals. A process that fails to detect and correct errors, for example, would produce a product that falls short of the requirements.
It must be Efficient – the total cost of enacting the process should not be any higher than it needs to be. A process that permits change, but only at great cost, is not efficient in cases where change is common and unavoidable.
It must be Attainable – the people available to enact the process must have the skills and ability to enact the process. A process that relies on refactoring must include people that can refactor and perform all the techniques that enable refactoring to be performed in safety.
It follows that any mix and match effort should produce a process that has these required properties. We may judge how good the resultant process is by how well it fits the required qualities, and thus we have a handle by which we can evaluate the mix and match efforts with a view to improvement. An inadequate or unattainable process will result in failure unless corrective action is taken. Of the processes that have these two required properties, we may compare their efficiency when selecting the most appropriate.

In this article I address the frequently neglected topic of Attainability and what consequences this has when considering the jigsaw of best practices.

Essentially all processes contain some parts that are defined. Even the most lightweight process, for example, may define when and where source code is

stored. A heavyweight process, on the other hand, may attempt to define every possible action taken by any developer. The best way to ensure that a mix-and-match process is attainable is to pay attention to the gaps between the defined parts of the process.

All development processes have gaps. This is a distinction between development processes and production processes. Unless a product can be described precisely, in advance, the process to build it cannot be described precisely. These gaps in a development process are important, they enable the fixed parts of process to flex to fit the product being developed. They allow the developer to choose how to bridge the gap.

An attainable process is one where the people available to enact the process can enact the predefined parts of the process, and can also bridge the gaps between these predefined parts.

At the extreme agile end of the agility spectrum, a process will be all gaps; there will be no predefined parts to the process and the developer will have complete discretion over how the work is done. At the other extreme of the spectrum, a process definer may attempt to eliminate the gaps entirely so that the work can be done by the almost completely unskilled, emulating the factory production line. However, software development is not a production environment; such efforts are doomed to failure.

How is it possible that the developer can make these decisions that are needed at the extreme agile end of the spectrum unsupported by the framework of a defined process? It is possible because the developer has his own internal framework, built up from a combination of experience and learned knowledge.

It follows that when a developer does not have this internal framework, where it is insufficient, incomplete or poorly constructed, then the developer will need outside help to bridge the gaps and inadequacies of his internal framework. It is also important to allow for internal frameworks that may be strong in some areas and weak in others. This is common where experienced developers gain all their experience in one environment.

There are various approaches toward providing support. The more agile approaches have the support coming from team members that have the relevant experience, or from mentors brought in to augment the team. Where there is a large shortfall and not enough available experience to make do, then it is time to add written, pre-defined elements of process.

Note that this is not the only reason to have written process; it may also be needed for coordination purposes. When there are differences in approach, and these differences risk undermining the work done by other people, then it is time to agree on elements of process to prevent conflict. Configuration management, for example, is one area where there is a broad consensus on the need for some agreed process.

The dilemma that a defined process attempts to solve is that the people defining the process hold the knowledge about process, while the people enacting the process are the people who hold the knowledge about the situation that determines what process would be appropriate. The ideal toward which all truly

agile approaches aspire is that the people enacting the process hold sufficient knowledge about process to define the process.

Before discussing how knowledge of attainability helps us select appropriate process,
I would like first to take a step back and think about the development process. In many respects, this is like any other product; the stakeholders have requirements that apply to process. A typical set of requirements may be the 200 plus goals of CMM; with suitable re-wording to remove non-agile assumptions, they may suit our purpose nicely. Yet if these are the requirements, the process that actually gets enacted is the design. All the agile principles and practices that we apply to design of software may also apply, perhaps with some modification, to design of process. As with all analogies, some parallels will be useful and instructive, some will not. Defining process 'just in time' is a very useful parallel; 'continuous integration' and 'refactoring' are parallels with little immediately visible benefit. This probably stems from the difference that process gets used and produces a product that never needs to be produced again, so that particular bit of process never needs to get used again. Of course, something remarkably similar may get used again, and it is this similarity that leads some people to believe there may be a possibility of designing process up front.

Let us consider the case where there is a shortage of experience, a lack of communication, and a lack of trust in the developers to make the right decisions with regard to process.
For inexperienced teams the process needs to leave fewer gaps, and the process described needs to be more in terms of solutions to problems. More feedback loops need to be placed in the process explicitly to detect those occasions where the process seems to be inadequate, where the goals are not being met by the developers following the defined process. Where problems are detected, a problem resolution strategy needs to be brought into play, bringing the available expertise to bear on the problem. Yet the process still has gaps, the developer still needs to use initiative.
Some people see these gaps as risks. A long series of process improvements may attempt to fill all the gaps each time a developer's initiative was misplaced. The resultant process would be so hidebound and inflexible that it would have the developers doing precisely what was asked even when this was known to be the wrong thing to do. Here, any success would be in spite of the process, where developers ignore the process and do what they think is right.

Now let us consider the case where there is considerable experience, good communication, and trust in the developers.
For those teams with a wealth of experience, the process can be defined initially as a set of goals. The team as a whole, and individuals within the team, can choose how the process they enact will meet the goals. Such a set of goals may be, for example, the aforementioned goals of CMM, suitably modified. Here, the stakeholders are defining the requirements for the development process rather than defining the design of the development process. We could attempt to fill

these requirements by defining a process up front, as in the example for inexperienced teams. Alternatively, we could design the process in an agile manner, much the same way as we would design a system in an agile manner. A typical agile manner may be to do some initial work up front to establish a direction for the process, then to let the detailed design of the process evolve as the needs dictate. The important thing is that the goals are met; the flexibility that comes through fewer constraints on how the goals are met gives the opportunity for an efficient solution to be found. Alternatively, it gives the opportunity for an inexperienced team to go astray.

Finally, let us consider the extreme situation, where every member of the team is highly experienced. Here the team may be trusted to establish their own list of goals and act on them without reference to any defined process at all. In all cases, they know what is the right thing, and do it. Where they don't have the relevant information or expertise they will seek it out, of their own volition.

There are various opinions on how to characterise a situation and its suitability and need for agility. DSDM, Appropriate Process, Crystal, and Boehm and Turner all have variants with a strong similarity; the DSDM view was shown in Agile Times issue 3. In general, there is a scale of suitability for agility, positions at one end being ideally suited for agile approaches, positions toward the other end requiring additional robustness of approach. It is useful to note that the different criteria have different impacts on agility. Some, such as the degree of change (alternatively familiarity with business, familiarity with technology) determine the need for agility, while others such as criticality determine the need for robustness. One of the criteria considered by all these approaches is that of the skills and ability of the available people. The more the team has in the way of appropriate skills, the greater is the degree of agility that can be attained.
Let us look at a few benchmark situations for an individual within a team. At the outset, we all start as beginners, with no techniques available for use, no experience in using them, and no ability to select appropriate techniques. Hopefully this situation has been improved before the student takes his first employment, but a person at this stage needs firm guidance in a technique, and mentoring to be able to use it at all. Effective use of the technique is still in the future.
With some experience, the apprentice developer gains the ability to use a few basic techniques without guidance, to enact those parts of the life cycle in which he is permitted to participate. For each case he will have a single technique available, and will be starting to learn a few wrinkles about how to apply it in different situations. There is still no opportunity to select between alternate techniques, because there are no known alternatives.
As the developer progresses through journeyman status, he will typically expand his competence to different areas of the life cycle. He may also learn alternative techniques. He may use these all the time, abandoning the old techniques, or he may start to choose one technique for one occasion, another for others. The developer now has some capacity to select process; he has the first opportunity to become agile with respect to process.

Eventually, the better journeyman developers may reach the stage of mastery, and a depth of understanding that allows them to shortcut the evaluation and selection of techniques, instead appearing to invent techniques precisely tailored to the situation as they go.

The learning of process can include learning a single technique for new areas of process, learning additional techniques for existing areas of process, and learning how to choose between alternative techniques where there are multiple known ways to achieve the current goal. To this, we may add learning the goals that are important when designing process.
When considering how much process to design in advance, we need to consider what gaps the developers can cope with given their current abilities, what gaps they can cope with given the aid of their team members, how to ensure this aid is given effectively, and what further support in learning about process should be given. In considering attainability, consider also the possibility of changing the membership of the team to enhance the overall attainability; say by adding people with relevant skills or buying in mentoring support. Consider also the rate of churn, and the possibility of losing the expertise that is currently available and that is built up as the project progresses.

One of the conclusions that is probably more startling to the traditional process mind-set is that as the team matures, process improvement should take the form of removing constraints and permitting more flexibility, rather than adding constraints to prevent recurrence of problems. A second conclusion that may surprise some of the agile adherents is that mix and match happens anyway. If it is not done up front by a process specialist, it will be done 'at the code face' by the experienced developer selecting techniques suitable to deal with the immediate problems, or by a huddle of experienced people trying to find a way to deal with the situation the project finds itself in.

A third conclusion is that if we are to shorten the time it takes to get individual team members to the stage where they can make sensible decisions with respect to the elements of process that are appropriate, then some time spent by them, explicitly considering what makes a process appropriate and how to select appropriate techniques and approaches, might be a sensible option. And finally, unless the developer can choose between at least two ways to achieve his goal, he cannot be agile. Where one has no choice, one cannot adapt to circumstances; one cannot be agile with respect to that goal.