Agile Logic
Deliver Better Software Faster

Refactoring the Development Process: Experiences with the Incremental Adoption of Agile Practices

Paul Hodgetts
Agile Logic, Inc.
phodgetts@agilelogic.com

Abstract

The goal of many current process improvement efforts is to become more agile by adopting an agile process. However, the results of several recent projects suggest that when attempting to become more agile, adopting an agile process is exactly the wrong thing to do!

In this experience report, I discuss my failures with wholesale process adoption and my successes using an incremental adoption strategy based on metric- and retrospection-driven feedback. Similar to refactoring practices for design and code, this strategy identifies "process smells," and targets the worst of them with specific agile practices drawn from several popular agile processes.

1. Introduction

When an organization has chosen to adopt an agile process, they have the option of adopting an entire process wholesale, that is, a set of practices en-mass, or of adopting only specific practices. Many process experts advocate wholesale process adoption, citing the need to experience the synergy of practices and the danger of omitting supporting practices. Others advocate an incremental approach to agile process adoption, citing the need to temper disruptive change.

Over the past four and a half years, as a team coach I have had the opportunity to assist several teams in adopting agile development approaches on nearly a dozen projects. Initially, I advocated a wholesale adoption approach, but found the success rate to be low. I then turned to an incremental adoption approach, and experienced a much greater degree of success. This experience report tells the stories of some specific adoption experiences, both with wholesale adoption as well as incremental adoption.

1.1. Subjects of the Experiences

Several teams are the subjects of the experiences described in this report. These teams are briefly described here. Unfortunately in some cases, contractual obligations prevent the identification of the specific organizations involved.

The Internet Start-Up. From 1999 through 2002, Escrow.com, a provider of business-to-business e-commerce solutions engaged in nearly a half-dozen significant development projects, most of which were conducted using agile processes. While some projects experienced notable successes [1], some experienced significant issues.

Escrow.com was notable in that they initiated a complete transition to agile development. All parts of the organization, from the developers to executive management received training and participated in the transition. Colocated working areas were designed and built, placing business experts, quality assurance, developers and management in close proximity.

Personnel were assigned to various product initiatives, each of which had a dedicated agile team of from four to eighteen members. Each team had its own business expert and development team. Most teams had at least one assigned QA engineer, although some were shared.

The Control Systems Manufacturer. In 2002, a manufacturer of control systems began a project intended to update their aging software for the control of maintenance systems in nuclear power facilities. Unfortunately, the initial attempt to adopt an agile process did not succeed, and the organization chose to pursue non-agile process alternatives for subsequent efforts.

The Control System Manufacturer chose to attempt a limited adoption effort. A small subset of the development group was assigned to the agile team. The

team chose to forego initial training, and did not utilize a common working area.

The team initially consisted of eight developers, two QA personnel and a product manager, although several team members were eventually shared with other, non-agile projects.

The Government Workflow Project. From 2002 through 2004, the government of a major California county initiated a project to automate the workflow of key business processes in the criminal justice system. After some initial adoption challenges, the project achieved notable successes using agile practices, most of which were incrementally adopted and refined.

The project was outsourced to a software development firm, but the project was conducted at the county's facilities. The team at first worked within the existing cubicle-based facilities, but within months had negotiated to combine a group of cubicles into an open work area within which the team was colocated. The developers were dedicated to the project, and worked full-time in the colocated space. The business experts and QA engineers were employees of the county, and were shared with other projects. They worked in the colocated area while assigned to the project.

The size of the development team varied from three to six developers. There were two business experts and one QA engineer assigned to the team by the county.

2. Wholesale Process Adoption

Following the advice of several agile process experts, on two occasions I attempted to lead a team in adopting a complete agile development process at once, including all required practices. In both cases, the results did not achieve the desired success, but for differing reasons.

The following sections describe two experiences with a wholesale process adoption approach.

2.1. Partial Objections and Overall Resistance

The Internet Start-Up was experiencing growing pains. After a year of rapid growth, the development team had expanded from four to eighteen members, and the product had grown from a single prototype to several products in various stages of their lifecycle. The team had started with virtually no organized process beyond the informal habits of the individual developers. As their informal process began to fail under the larger team size and increasing business demands, the team attempted to adopt practices based on the Unified Process, but found little relief.

Several senior developers had been studying agile processes, and had begun to experiment with agile practices. Their experiments showed promise, and their reputation with technical and corporate management enabled them to begin a wider adoption effort. These senior developers gained an understanding of agile practices, primarily those of Extreme Programming, as a result of their own study. The remaining developers were trained in XP practices through training courses and on-site coaching. The team's technical leaders and managers believed the entire team had sufficient understanding of XP practices to utilize them on the company's critical projects, and a decision was made to adopt the entire set of XP practices as the team's standard development process.

At first, some difficulties arose as the team attempted to utilize the set of XP practices, but these difficulties appeared normal for a typical adoption effort. However, after nearly eight weeks of XP development, persistent difficulties remained. Although the team had success with the set of planning practices, several key development practices, notably test-driven development and refactoring, were not producing satisfactory results. A closer examination revealed the difficulties were primarily occurring with a small number of developers. The team coaches and senior developers attempted to increase the level of training and mentoring with these developers. Although the developers appeared to understand the practices and acknowledged their ability to utilize them, the situation did not improve. It appeared when the developers utilized the practices on their own; they intentionally failed to execute the practices correctly.

The situation baffled the team leaders. With no outward expression of objections to the XP process, and no evidence of inability to understand the practices, it was not clear why these developers consistently failed to correctly execute the process. Eventually, it was necessary to remove these developers from the critical projects. They became increasingly dissatisfied with their role, and unfortunately chose to leave the company.

During one developer's exit interview, the technical manager learned an important lesson. The developer told the manager that while they believed they understood and were able to execute certain XP practices, they did not really believe those practices were the right way to develop software. Although they thought many of the XP practices were effective, their objection to the few with which they did not agree reduced their motivation and desire to execute the overall process.

In retrospect, the team leaders concluded they had not created an appropriate environment for the

adoption of agile practices. While they had provided sufficient training and mentoring for the team to understand and execute the practices, they had not created an open environment that allowed sufficient opportunity for the team to discuss and agree to their overall software development approach. This process would have allowed the team to collectively decide if an agile process was appropriate for their project and more importantly what practices they would adopt.

Had the team initially adopted the smaller subset of practices in which they all believed, it's likely they would have enjoyed early successes and been more willing to adopt further practices. Although it was unclear whether the few dissenting developers would have agreed to adopt further practices, an incremental approach would have earlier and much more clearly revealed the practices to which they objected, probably enabling the team to avoid the protracted problems that affected their ability to deliver.

2.2. Too Much to Learn at Once

The Control Systems Manufacturer launched a major effort to replace its aging control system. The team of eight developers, two QA personnel and a product manager previously used a heavyweight waterfall process, but had experienced major difficulties reaching successful project completion.

The team independently researched development processes and decided to adopt an agile process. Through their research, the team felt they had gained sufficient understanding of agile practices to easily adopt them. Despite the recommendations of their team coach, they chose not to start with any up-front training or preparation activities.

The team enthusiastically began the first two-week development iteration. They immediately realized they required some basic environmental support, such as a common working area, workstation configurations and an integration and build environment. At first, each new activity required some degree of learning, and the team frequently paused to conduct impromptu training sessions for new practices. The first iteration's progress was not encouraging, and they delivered almost no business value.

After some retrospection, the team felt their initial experiences were isolated to the first couple of iterations, and they would quickly ramp up their development velocity as they put the initial adoption overhead behind them. They proceeded to the second and then the third development iteration. But they still found their progress to be painfully slow. Although they concluded their learning curve was larger than they initially anticipated, they maintained hope the majority of the learning was behind them, and continued.

Two more iterations passed, and the project had now been ongoing for ten weeks. Recognizing that the project appeared to be experiencing difficulties, the technical manager for the group intervened. His review of the project revealed the project had delivered less than 20% of the business value projected in the team's original release plans. After a series of intense meetings, the manager concluded an agile process approach should no longer be pursued, and mandated that the team revert to their previous development process. The team was demoralized by the decision. Two key developers chose to leave the team soon after.

The team conducted one final retrospective, during which they candidly discussed the causes for their difficulties. They concluded that they severely underestimated the affect of concurrent preparation and learning activities on their ability to deliver business value. They had committed to a development velocity that was too large given the amount of preparation and learning needed, and had thus set unreasonable expectations with their manager and their organization.

They also concluded that given their inexperience with agile practices, it would likely have been appropriate to perform some initial preparation and training prior to attempting actual development. Although their coach had suggested an "iteration zero" to address these needs, the team did not fully understand the reasons behind that recommendation.

In retrospect, they agreed the overhead of the basic preparation and learning activities, combined with the complexities of the actual development of their control system, simply overwhelmed their capabilities. In the end, this resulted in both a slower rate of learning as well as a more reduced development velocity than would have been achieved if some of the activities would have been performed up-front in isolation. Despite their initial enthusiasm and best intentions, they were not able to overcome these difficulties.

3. Incremental Process Adoption

After a couple of painful failures with a wholesale adoption approach, I turned to exploring an incremental adoption approach, where the team would target a limited number of new practices to adopt over one or two iterations.

While my initial experiences with incremental adoption showed promise, the results were uneven and did not yield the rate of success I was after. Some practices would be enthusiastically adopted and

executed with much success. Some practices would be met with difficulty or resistance, resulting in half-hearted execution and poor results. While incremental adoption was working better, something still wasn't right.

The order in which practices were adopted was questioned. The teams had chosen which practices to adopt next based on several factors. We looked for practices we thought were easy to adopt given the project context. We looked for practices that everyone thought were a good idea. As coach, I often suggested practices based on my knowledge of how one supported the others. Once we decided which practice to adopt next, we began practicing it "for its own sake."

Kent Beck, the founder of Extreme Programming, talks about incrementally adopting XP "… one practice at a time, always addressing the most pressing problem for your team." [2] Although we weren't necessarily adopting XP each time, we explored this approach.

Finding "the most pressing problem" proved to be a challenge. We would identify problems such as "we can't roll into production on time." Problems like this generally had deeper root causes, for example, "we can't roll into production because we are finding last minute bugs that we have to fix." Even further analysis revealed the bugs occurred because either developer or acceptance testing was inadequate. As we reflected on the identified problems, we tried to distill them down to root causes. Inspired by Martin Fowler's book on refactoring [3], we named these root causes "Process Smells."

The following sections describe several experiences with an incremental process adoption approach. Each experience describes how the adoption of a specific agile practice was facilitated by an immediate pressing problem.

3.1. Test-Driven Development

The Internet Start-Up was actively chasing new customers in the spring of 2001. The market was extremely volatile and the business requirements were rapidly changing, necessitating rapid cycles with production releases occurring every few weeks. The team at the time consisted of eight developers, a QA engineer, and a product manager who was the business expert.

Although the team was very productive implementing new features, they faced a serious problem delivering releases to production. When a release candidate entered the quality assurance cycle, an unacceptable number of issues were found that delayed its release. Many of the issues were bugs, but a noticeable number were variances from expected behavior. Not only were the production releases being delayed, but the project metrics showed the size of the backlog was growing with new issues, deferred in the interest of just getting a release out the door.

The developers used common engineering testing practices that primarily consisted of performing manual unit tests during the integration of new code into the code base. QA employed after-the-fact testing strategies that largely consisted of learning the features as implemented, augmented with limited conversations with the product manager, and then devising and manually executing black box testing of the system. It was apparent to the team that the developer testing lacked sufficient coverage to adequately remove defects, and that the QA testing was uncovering basic requirements issues too late to correct them prior to release.

Although the team did not employ a specific retrospective practice, through informal discussions they resolved to improve their situation. They investigated and adopted both acceptance test-driven as well as unit test-driven practices to address both the defect as well as requirements issues.

The team's QA engineer joined the analysis sessions that occurred during iteration planning to learn the features earlier in the cycle, and develop a clearer understanding of the features directly from the business expert. She decided to write acceptance tests prior to development to provide clear specifications of correct feature behavior. Although it proved challenging to write acceptance tests ahead of time because of various GUI testing tool issues, she was able to able to write test outlines that provided sufficient completion criteria to eliminate the larger issues due to misunderstood requirements.

The developers adopted XP-style test-driven programming practices. It proved fairly easy to drive the majority of coding with unit tests, although advanced unit testing practices such as mock objects required several iterations to become effective. The developers also utilized QA's acceptance test outlines as completion criteria for the overall features. They required successful execution of all acceptance tests for a feature before the feature could be considered development complete.

Overall, adopting test-driven development required several iterations to show significant results. It was immediately apparent in the first iteration that the unit test-driven practices were significantly reducing defects caused by programming errors. Defect counts of 20-30 per iteration prior to adoption, were reduced to only a few per iteration. Adopting acceptance test-driven practices was a more difficult effort, requiring a

redefinition and reinvention of QA's role in the organization. The team's collaborative approach enabled QA and the business expert to develop together an approach that was simple yet sufficient to define the features and provide adequate acceptance criteria. Within four iterations, the acceptance test-driven approach resulted in a significant drop in new tasks generated each iteration to repair features that worked correctly but implemented undesired behavior. Additionally, the crisper definition of new features facilitated by the acceptance tests often resulted in the avoidance of unnecessary work.

The adoption of test-driven practices allowed the team to address a very real problem that was preventing them from delivering production releases. The motivation provided by their desire to succeed allowed them to learn agile practices that were not previously considered valuable.

3.2. Small Batch Size Development

The Government Workflow Project was well into the development of their second major workflow when they experienced frustrating velocity fluctuations and inconsistent completion of features. The team tracked their development velocity for each iteration in terms of feature points completed. The project metrics showed the iteration velocity dropped significantly when scheduled features could not be fully completed and spiked higher when partially completed features from the prior iteration were finished.

At first, the team questioned why their velocity was inconsistent, and searched for reasons their productivity would fluctuate. However, they discovered that the problem was not an inconsistent rate of development. In reality, the team was quite productive in terms of producing functionality each day. What the team noticed was that their feature estimates were often inaccurate. Further, the estimation inaccuracies were not systemic, that is, the team could not simply apply an adjustment factor to compensate.

The team typically performed feature estimation when planning releases that encompassed around ten iterations spanning several months in duration. In order to accomplish the estimation and planning work in a reasonable amount of time, the amount of analysis done prior to estimating was just enough to estimate the feature into general categories of large, medium and small, approximately corresponding to a week, half-week and day's duration.

Through further retrospection, the team noticed that many features were sized such that the feature estimate was very close to the iteration length of one week. This allowed little leeway in the event the feature required additional time to complete, resulting in the inability to complete the feature within the iteration. Large features were also difficult to spread across multiple developers in parallel, resulting in an inefficient use of the team's resources to accomplish the high priority features.

For the next major feature release, the team resolved to break down larger features into smaller-sized ones, a practice recommended by many agile process experts. They decided that they would target a maximum feature size of a half-iteration, and further break down any features that exceeded this limit. In order to accomplish this, additional understanding of each feature was required, and thus the team needed to perform more analysis work at planning time.

Initially, the higher degree of up-front analysis was felt to be very non-agile, and the team was concerned. However, over time the team learned how to perform just enough analysis, approximately a quarter to half-hour for each day's worth of feature development, to gain sufficient feature understanding to drive finer-grained feature breakdowns. Not only did the developers learn to break the implementation of features down into smaller-sized stories, but the customer also learned to better judge the minimal portion of a feature that was needed to provide the most value in terms of workflow labor savings. This allowed smaller features to deliver more overall value to the project.

Despite the initial discomfort of longer and more detailed release planning sessions, adopting finer-grained feature breakdowns produced dramatic results. The project metrics now showed a much more consistent velocity from iteration to iteration. The smaller features were now easily completed within the iteration boundary, even if there were normal statistical variances from the estimates. Additionally, the smaller features were more readily distributed across the available team resources, increasing the ability of the team to opportunistically adjust work assignments to complete higher-priority features.

The team initially had the belief that performing additional up-front analysis was counter to agile principles. However, once they found the courage to try a more detailed planning approach, they significantly improved their ability to plan the project. In retrospect, the team realized this practice implements the "smaller batch size" principle of Lean Software Development [4], and in fact increased their agility.

3.3. Pair Programming

The Internet Start-Up's agile team was initially formed from an existing team organized around specialties. There were several web client developers, several middle-tier developers and a database developer. Prior to adopting an agile approach, developers seldom ventured outside their area of specialty. It was often quite a challenge for the team leads and project manager to organize and schedule work to ensure everyone remained busy and productive.

When an agile process was adopted, the team learned about the generalist approach favored by agile processes. The web client developers and the middle-tier developers largely embraced the concept and agreed to take on tasks from each other's specialties. The database work, however, was considered too specialized by the team, and they felt all database development should still be done by the single database developer.

As the iteration progressed, the ability to share the web client and middle-tier development across the team allowed the team to schedule more parallel features for development. This in turn allowed the product manager to target several of their highest priority features for simultaneous development, thereby more quickly meeting their overall business objectives.

Problems arose, however, when the product manager chose several features that each required significant database development. The database developer became overloaded and could not meet the needs of all the high priority features. This issue was apparent when examining the iteration plans against the overall product backlog. It showed lower-priority features being assigned to an iteration before higher priority features, simply because they required less database development.

The team needed to break loose the bottleneck around the single database developer. They considered hiring another database specialist, but funds were not available. Training an existing developer from scratch would take too long. The team coach suggested that the team try a pair programming approach to database development. The team had previously rejected the pair programming practice, and the suggestion met once again with resistance. The coach reminded the team that their inability to deliver the higher priority features was adversely affecting the overall business, and asked them to try a pair programming experiment for several iterations, and then reflect on the results. The team agreed.

At first, the bottleneck still existed because the single database developer needed to be one of the pair for each database task. However, much more quickly than originally thought, the pair was able to split out simpler database tasks to be worked on in parallel by a second pair. The specialist then paired with a new developer, further increasing the dissemination of the skills, while the former apprentice paired with a new developer, switching back to working with the specialist when they got in too deep. Over just a few iterations, many simpler database tasks no longer needed the specialist involved, and allowed a much more opportunistic scheduling of tasks to better accommodate the customer's priorities.

This team had originally objected to pair programming as too wasteful of resources, and too intrusive on each developer's ability to concentrate on the tasks at hand. Although the potential benefits of pair programming were discussed, the team had decided not to adopt the practice. Faced with a serious problem that affected their ability to deliver, the team reconsidered pair programming as a practice. Once they were able to directly connect a potential benefit of pair programming, cross-training that allowed better distribution of tasks, to an immediate problem they needed to solve, they developed the motivation and courage to attempt a practice they had previously resisted.

3.4. Metaphor and Refactoring

The development team on the Government Workflow Project was nearing completion of its first major release. The release had not gone well. At this juncture, the development team consisted of three senior and two junior developers.

Although there were several issues that had hindered their progress, their retrospectives had identified one issue in particular that they wanted to target for improvement. Through examining their tracking metrics, the team saw that any feature that required new web interface development invariably overran estimates in unpredictable ways. In fact, the team had become so frustrated with progress on some interface tasks, that they referred to some parts of the interface code as "black holes," and dreaded taking on tasks in those areas.

The team devoted a few side meetings to discussing the issue and reviewing these parts of the system. They realized that the design and implementation of the web interfaces had no underlying metaphor (an XP term for conceptual model) guiding it. Although individual developers had more or less adopted their own personal approaches, the team had no shared best practices and standards for the interface development. Their first step was to figure out what the underlying

metaphor needed to be, and an engineering task was scheduled to do so.

By studying industry practices and refactoring selected portions of the code base, the team was able to fairly quickly arrive at a core metaphor that could be evolved by further experience. Over just a few iterations, the metaphor drove the design into a framework within which new development could quickly proceed. This greatly improved the quality of new code and reduced the effort to implement new web interface features considerably. The new metaphor did not, however, magically cure the issues with existing interface code.

The team determined that more refactoring was necessary to avoid continued cost overruns when working with the existing interface code. The code base had accumulated a large amount of "debt" that needed to be "paid off." A concentrated refactoring effort was not possible if the team was to continue to deliver any reasonable amount of business value. So the team decided to spin off smaller refactoring tasks whenever a new feature was scheduled that touched existing interface code. The refactoring effort would increase the time to complete the feature, but would incrementally pay off a portion of the debt as well.

The team also resolved to take heed of the lesson, and to be more mindful of refactoring any new code added to the system to prevent debt from accumulating. A positive side effect of the effort was that the team developed better overall definitions of "goodness" (standards and best practices) to guide their choices in the future.

The team made considerable progress cleaning up the web interface code, and virtually eliminated creating new code that was not clean. The new metaphor worked very well, and inspired the team to create similar metaphors for other parts of the system. Although it took many iterations of incremental refactoring to achieve, work in this part of the system is now considered cool and fun.

3.5. Sufficient Analysis

The time came for the Government Workflow Project to once again plan for a new release cycle. Unfortunately, once again the release planning session did not proceed smoothly. The team of two business experts and three developers utilized an Extreme Programming-based process where the business experts were assumed to bring a set of prioritized features to each planning meeting.

However, as the planning session progressed the team had a great deal of difficulty finalizing a set of features for the release. The business experts could describe general features, but were unable to provide sufficient details. They often needed to conduct further investigation before they could provide the necessary information. The team was confused on how to proceed. They needed to complete their planning session in order to move on to development, but lacked sufficient definition of the features to break them down into meaningful development tasks. Instead they ended up creating a number of "analysis tasks" that were placeholders for continued feature discussions once the business experts had researched the pending issues, and moved on to develop the portion of the features that were sufficiently defined.

This approach quickly created additional problems. Since perhaps one-third to one-half of an iteration was left for further definition, the team lacked clear completion milestones for the iteration. With an uncertain amount of work represented by the skeletal features, it was common for features to be left partially completed at the end of each iteration. The lack of success milestones was negatively affecting the team's morale.

The team reflected on their process for developing feature definitions, and discovered they had unreasonable expectations of the business experts. The business team was built around domain experts. While they knew the business very well, they were not skilled in analyzing their business process and creating a specification for a software implementation to support it. The team concluded that the business team needed the expertise of a professional analyst to assist the domain experts.

A dedicated analyst was not available, but a couple of the developers had sufficient analysis skills to help. They dedicated a portion of their time each week to work with the business team. This allowed the business team to put sufficient forethought into the features prior to planning time, enabling the planning discussions to proceed quickly. Over just a few iterations, the developers were able to teach the business experts sufficient analysis skills that the amount of time required to coach the customer team gradually declined to only a few hours a week.

4. Retrospective

My initial experiences with wholesale agile process adoption were not very successful. In the case of the Internet Start-Up, the organization overcame the initial difficulties and went on to achieve much success with agile processes. In the case of the Control Systems Manufacturer, the initial difficulties were more than

the organization was willing to tolerate, and the overall agile adoption effort failed.

Looking back on these wholesale adoption efforts, several key mistakes can be identified. In the case of the Internet Start-Up, the adoption effort was initiated without sufficient consensus of the team. By mandating the adoption of an agile process despite the objections of several team members, management created an environment where outward dissention was discouraged, and instead produced more subtle dissention that significantly hindered the project's success.

In the case of the Control Systems Manufacturer, the adoption effort was initiated without substantial preparation, both in terms of training as well as environmental factors such as workspaces and tools. While I believe it is possible for a team to incrementally perform such preparation, in this case the team drastically underestimated the cost of such concurrent work and the resulting decrease it would have on the rate of delivery of product features. By creating unrealistic expectations with their release plans, they sowed the seeds for an unfortunate meltdown as the expectations were increasingly unmet.

If I were to attempt another wholesale adoption effort, I would certainly address these issues. I would ensure the team was deeply involved in the decision to adopt an agile process and had reached a unanimous decision to do so, perhaps using the McCarthy's Decider protocols [5]. I would also ensure that the team had adequately prepared for the adoption effort. My preference is to now conduct some initial training for the team as well as initial environmental set up prior to attempting actual feature development.

But even with these precautions, my opinion is that wholesale agile process adoption is a risky endeavor. The transition to an agile approach introduces quite a bit of disruptive change to an organization. Concentrating all the disruptive change into a short period of time can overwhelm an organization or produce significant discomfort and resistance.

My experiences with incremental agile process adoption have been very successful. After the initial difficulties at the Internet Start-Up, the team retreated a bit and chose a smaller set of core practices on which to focus, including basic planning practices, short iterations, pairing, frequent releases and as much testing as they could manage. The team felt these practices implemented the basic cycles and rhythm of an agile process, as well as provided enough feedback for improvement. The team continuously added, modified and sometimes removed practices over the course of nearly a dozen projects involving half-dozen separate teams.

On the Government Workflow Project, the initial team had adopted agile practices, but had experienced some difficulties. When I joined the project, the team was not delivering business value on a regular basis, and the customer had begun to lose confidence in the team. Similar to my prior incremental experiences, I coached the team to focus first on better iteration and release planning, and then on establishing their core rhythm through short iterations, delivering working software, retrospectives and continuous learning. The team continued to improve dramatically over the next year, eventually establishing a highly effective agile process that produced significant business value and satisfied customers.

Through these experiences, a key factor emerged that correlated to success when adopting agile practices. It seemed I had greater success with a targeted adoption approach. Because the team had identified very real issues they were trying to solve, the adoption of practices was highly motivated. Each practice had an identifiable purpose, rather than being adopted "just because we should." Over time, each team eventually employed a wide array of agile practices as they incrementally adopted them to solve issues. Although they may not have ended up doing a "complete" or "official" agile process, like XP or FDD, it would be difficult to argue they did not have an "agile process."

It is my hope that others may recognize these experiences in their situations and perhaps gain some new ideas and approaches to solve their immediate issues and successfully adopt agile processes.

5. References

[1] Paul Hodgetts and Denise Phillips, "Extreme Adoption Experiences of a B2B Start-Up", *Extreme Programming Perspectives*, Addison-Wesley, 2003, pp. 355-362.

[2] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000, p. 123.

[3] Martin Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999, pp. 75-76.

[4] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison-Wesley, 2003, pp. 77-81.

[5] Jim McCarthy and Michele McCarthy, *Software for Your Head: Core Protocols for Creating and Maintaining Shared Vision*, Addison-Wesley, 2002, pp. 117-147.

