Use of Agile Methods and Practices in the Philippines

Raymund Sison and Theresa Yang
College of Computer Studies
De La Salle University
 Manila, Philippines
sisonr@dlsu.edu.ph

Abstract

Agile methods are increasingly gaining attention in many developed countries; however, there is a dearth of empirical studies showing their successful use in developing nations. This needs to be addressed because most software offshore outsourcing destinations are in the developing world. This paper describes experiences in the use of agile methods and practices by software development firms in the Philippines, a developing country that has consistently appeared in lists of top software outsourcing destinations. The results tend to confirm those found in the literature concerning the positive impact of agile methods on software development, and how software organizations tend to adapt agile practices to suit their own organizational and project contexts. One case study in this paper also describes a particular approach to agile offshore software development.

1. Introduction

Agile methods such as XP [1] are gaining more and more attention in the United States [2] and Japan [3]. However, considering that the offshore outsourcing of software development and maintenance appears to have become a mainstream phenomenon [4], with U.S. companies “shifting tens of thousands of IT jobs to countries such as India, China, Russia, and the Philippines” [5, p.2], it would do well for us to begin studying whether and how agile methods and practices are being used in these outsourcing destinations, most of which are in the developing world. In this light, it is therefore interesting that of the more than 200 papers presented in the past XP conferences, for example, only a couple reported experiences using XP practices in developing countries [6, 7]. The first of these two reported on how the XP practice of test-driven development improved overall team performance in a software firm in China. The second reported on the use of XP and a few other related practices in a particular project of a start-up firm in Brazil. A related paper [8] reported on software practices in five countries of the ASEAN (Association of Southeast Asian Nations), namely, the Philippines, Malaysia, Singapore, Thailand and Vietnam. Although the said study focused on CMMI Level 2 processes, it alluded to the possibility that agile practices are being implemented more commonly than one would think in ASEAN countries.

This paper follows up on [8] by looking at whether agile practices are indeed being implemented more commonly than one would think in the Philippines, one of the five countries in the said study. Specifically, this paper reports on the results of two case studies on the use of agile practices in software development firms in the Philippines. The Philippines is a developing country that was reported by McKinsey [9] as having supplied $1.7B worth of offshore services to the global economy in 2003 (cf. India: $12.2B; China: $3.4B; Asia (excluding India and China): $2.3B; Latin America: $1.8B; Russia and Eastern Europe: $0.9B). A survey conducted by the Weissman Center for International Banking and the Piaas Group showed that the Philippines ranked only second to India as the top IT outsourcing destination among North American and European companies in 2004 [10].

In the rest of this paper, we briefly describe the qualitative research method used in this research and then report on and analyze the experiences of two software development firms in the country in the use of XP and other agile practices.

2. Research Design

The purpose of this study was two-fold: to determine which software development firms in the Philippines were using, or have used, agile methods or practices; and to obtain an understanding of these
firms’ experiences in the use of agile methods or practices. Specifically we wanted to know whether agile methodologies were used as prescribed in [1] and if not, how and why not.

To address the first objective, a list of 110 software firms in the Philippines was first created from two lists: (1) the Philippine IT Directory, an annual publication of IT firms in the country, and (2) the membership list of the Philippine Software Industry Association (PSIA), which represents the local IT industry in talks with government and in trade missions abroad. Next, every software firm in the combined list was contacted from March to June 2006. Of the 110 software firms that we tried to contact, only 51 were active and actually developed software. The rest were found to have closed or to be software resellers.

Each of the 51 active software development firms were then e-mailed a short questionnaire asking whether it used any agile methods or practices, and if so, to specify these methods or practices. The questionnaires also explained what agile methods were and, in an attachment, gave examples of agile methods and practices and hyperlinks to major sites on agile methods. On occasion, phone calls were also made for clarification. The 51 firms’ response rate was 100%.

Of the 51 firms, only two have used or were using agile methods. There were three other firms that we also considered studying. One of these had only just begun using agile practices, and so was excluded from the study. The other two alleged that they used agile practices; however, subsequent interviews revealed that they were not really using agile methods, or practices strongly associated with agile methods.

The two firms that used agile methods were both software SMEs (small and medium enterprises). SMEs make up 90% of the software firms in the Philippines [12]. This is not so different from the situation in other countries, including the U.S., in which small companies represent up to 85% of all software organizations [13].

Data from each of these two companies were collected and analyzed from July to December 2006 through interviews and e-mail discussions with managers, analysts/designers, developers, and testers; observation of these persons at work; and content analysis of documents, including software and published documents.

3. Results and Discussion

3.1. Case 1: Solutions Integrator

Case 1 involves a small (less than 100 employees) software firm, hereafter called C1. Founded in 1991, it is an integrator of IT solutions with clients in the Philippines, North America, and Asia. It was appraised at CMMI Level 5 in 2006.

In 2001, a leading Internet automotive marketing services company in North America signed up C1 through its U.S. office (hereafter called C1-US) to develop a baseline CRM site. The client company wanted the CRM site to be developed using XP. The project was successful in the sense that the client company was happy enough to ask C1 to develop a second CRM application, one that could be used by registered automotive dealers to deliver service reminders to car owners via e-mail, direct mail, and call centers.

Both CRM/XP projects were successful in the sense that the applications that were developed are now routinely being used by car owners and car dealers in North America.

Results of interviews with three of the original 12 XP team members who were still with the firm at the time of the study revealed that five XP practices (small releases, metaphor, simple design, 40-hour week, and coding standards) were more or less implemented as prescribed in [1]. However, the rest of the XP practices (on-site customer, pair programming, collective ownership, test-driven development, refactoring, and continuous integration) were not implemented exactly as prescribed in [1], and it is these “variants” that we will focus on in the rest of this subsection.

On-site customer was simply not feasible because there were two sites: U.S. (for analysis and design) and the Philippines (for development). Therefore, the planning game had to be conducted through email and teleconferencing, with C1-US taking the role of business and C1 taking the role of development. Nevertheless, because the releases were small, the client was able to provide frequent feedback to C1-US, which in turn, immediately relayed resulting requirements/design changes to C1. There were also several visits made by client representatives to C1 in the Philippines, and vice versa.

Pair programming was implemented at the beginning of the first project, with senior developers paired with junior developers. Moreover, pairing was dynamic to some extent, so that a developer was paired with several different developers throughout the project. This implementation was, according to interviews, greatly appreciated by everyone because the junior developers quickly learned from senior ones, and senior ones were quickly able to share their best practices as well as company standards with the junior ones. Moreover, this increased the camaraderie among
the members of the 12-person development team. As the need of the junior developers to learn from the senior developers diminished, however, the pairing began to taper off so that by the end of the project, most developers were actually working on their own.

Collective ownership of the code, as defined in [1], was practiced at the beginning of the first project. However, it was almost absent toward the end of the project.

C1 performed testing in the traditional (manual) way. From the interviews, no attempt seemed to have been made to automate unit tests. Nevertheless, all the interviewees seemed to have the impression that testing in this project was quite thorough. Testing was primarily scenario-based, and there were the usual unit, integration, and system tests. System tests were performed by three independent testers. C1-US assisted in user acceptance tests in the U.S.

Refactoring was performed whenever possible, but the researchers could not ascertain from the available data how frequently and how well this was performed. There was a management directive to always refactor if necessary. However, the developers who were interviewed could no longer remember the extent to which they performed refactoring. Moreover, there were no formal mechanisms to evaluate the quality of the refactorings.

Continuous integration was not practiced the way it was described in [1], in which code is integrated and tested after a few hours or after a day of development at most. Instead, integration was performed when units were completed.

Today, four years after the second project ended, the three members of the original XP team of developers who are still with C1 gave the following impressions about XP:

- XP increases the degree of interaction among team members, and this leads to a high degree of camaraderie.
- XP is effective in quickly producing products that meet customer requirements.
- The most salient or defining XP practice is pair programming.
- Daily stand-up meetings are an important XP practice.

Despite the very positive impressions about XP, particularly about its effectiveness in improving team camaraderie and in delivering quality products quickly, very few of its practices continue to be used today in C1. In particular, only small releases, metaphor, simple design, 40-hour week, and coding standards are implemented today. Today, what originally began as a tapered version of “pair programming” has been reduced to the practice of senior developers pairing with junior developers every now and then.

3.2 Discussion of Case 1

Analysis of qualitative data from C1 shows that the methodology that C1 and C1-US ended up using was not strictly XP. In particular, although small releases, metaphor, simple design, 40-hour week, and coding standards were more or less implemented as prescribed in [1], on-site customer, pair programming, collective ownership, test-driven development, refactoring, and continuous integration were not.

The first set of XP practices could have been implemented without any difficulty because they were not so novel to C1. For example, an incremental approach to software development, which C1 favors as indicated in its other projects, encourages small releases and simple design. Metaphors are a natural way for human beings to communicate. Coding standards and the 40-hour workweek were established early in C1. Because these practices were quite established in C1 before it undertook the CRM/XP project, we shall therefore call these XP practices “old” XP practices (of C1). The other practices were, in contrast, “new” practices, and it seems that novelty played an important role in their not being implemented in the way they should have (i.e., as described in [1]).

Among the “new” practices, on-site customer was not only novel to C1; it was also very difficult to implement given the offshore aspect of the case: the client company communicated primarily with C1-US, which handled the analysis and design of the CRM applications, while C1 wrote and tested code according to the design of C1-US. Nevertheless, because the releases were small, the client was able to provide frequent feedback to C1-US, which in turn, immediately relayed resulting requirements/design changes to C1.

The tapering off of pair programming toward the end of the first project would be due to what the team appears to have thought was the main benefit of pair programming: getting junior programmers up to speed. Therefore, once the junior programmers understood the metaphor, design, and standards quite well, pair programming, or the need for it, disappeared.

Collective ownership of the code was practiced at the beginning of the first project mainly, it seems, as an offshoot of the dynamic pairing that took place during this period. Thus, as pair programming disappeared and as developers eventually began to
work on their own, collective ownership likewise disappeared.

Although there was sufficient testing, the development was not test-driven in the sense that automated tests were not written before the code and could not, therefore, be a basis, together with simple design, for the writing of the code. On the surface (i.e., based on interviews), this would simply be due to the lack of tools for automated ASP/XML testing at that time. However, C1’s still not performing any automated tests today would indicate the company’s lack of belief or interest in test automation, rather than the lack of automation tools, as the more likely reason why the XP projects were not test-driven.

As mentioned earlier, it is not clear to what extent refactoring was performed in the XP projects. This could be due to several reasons. First, there were no automated test scripts, so verifying that refactorings did not introduce bugs could be a tedious and error-prone task. Second, development environments (IDEs) at that time did not support refactoring to the extent that they do now. Third, the pairing of senior and junior developers might have stopped before the junior developers could acquire, by observation of their seniors’ programming behavior, a sense of “smell” keen enough to sniff out bad smells in code. Fourth, there was limited to no collective ownership, and so there was limited opportunity to refactor the code of others. Fifth, Fowler’s refactoring bible [14] was inaccessible to the developers at that time. Sixth, the code was not quite object-oriented, so even if [14] were available, many of the refactoring techniques in it would not have been applicable.

Finally, integration was not continuous, but followed the traditional approach of integration testing. This would of course be due to the ownership not being collective. The absence of automated test scripts would also have served to make integration less continuous.

As regards the impressions of the developers about XP, the developers’ marked appreciation for the increased camaraderie among developers that resulted from the use of XP practices might be due to the high degree of collectivism (as opposed to individualism) that Hofstede [15] detected among programmers in the Philippines in his landmark study on effects of values on culture.

The developers’ notion that pair programming is the most salient practice of XP could be because the other practices were either “old” or perceived as “old”. Having two programmers sharing one computer, keyboard, and mouse, was, in contrast, to them quite new. In other words, it was pair programming, and the collective code ownership that resulted from dynamic pair programming, that was novel. This novelty could also explain why these two practices were not sustained throughout the two projects. We will return to this shortly.

It is not clear where the developers got the notion that daily stand-up meetings (a Scrum practice [11]) were an XP practice. It is possible that such brief meetings were associated with small releases. It is also possible that such daily meetings, which are more an agile project management technique rather than an agile software construction technique, appeared to C1, being CMM Level 2 appraised at that time (and therefore having a high degree of project management maturity), as a good way to manage a rapid development project. It is also possible that the notion that daily stand-up meetings are an XP practice might have been formed inadvertently as the senior designers tried to learn about XP from the Web way back in 2001.

Despite the non- or limited implementation of the “new” XP practices, or their not having been implemented as prescribed in [1], the two projects of C1 with the North American automotive marketing services company were successful in the sense that products met the client’s requirements and constraints (including time and budget). As Figure 1 illustrates, this could be due to a combination of factors. The first factor would be the successful use of the “old” XP practices (small releases, metaphor, simple design, 40-hour week, and coding standards), which happened to be sufficient for the project. The successful use of these “old” practices would, in turn, be due to the familiarity of the company with these “old” practices and the maturity of the senior developers (who would be at Cockburn Level 2 or 3 [16]) and project managers. The apparent sufficiency of these “old” practices would, in turn, be due to the relatively stable product architecture.

The second factor affecting overall project success would be the relatively low cost of offshore development. The third factor affecting project success would be the maturity of the company’s project management process, as indicated by the company’s being CMM Level 2 at that time. This and the cooperation of the client led, in turn and among others, to sufficient regular communication between C1 and its client.

However, despite the success of C1 in these XP projects and the very positive long-lasting impressions of the XP team (or what remains of it) about the XP process, very few XP practices continue to be used in
C1 today. Interestingly, only the XP practices we earlier labeled as “old” continue to be practiced to this day in C1.

Coding standards and the 40-hour workweek continue to be practiced because these were already established in the C1 when the XP projects came in. In addition, coding standards are very much in line with CMMI philosophy. C1 now uses an incremental iterative process similar to the Rational Unified Process (RUP), so releases are still not so big, and design starts as simple but eventually gets more and more elaborated, so refactoring, though still performed to some extent, is not performed as often as it would have been under XP.

That very few XP practices continue to be used today could be due to a combination of factors, as Figure 1 illustrates. The first would be the non-implementation or limited implementation of the “new” XP practices. This, in turn, would be due to the unfamiliarity of the developers with these practices, the lack of tools to support these practices, and the offshore aspect of the projects. These factors were already explained in detail earlier.

The second factor would be the offshore development. As discussed earlier, some practices had to be adapted to the realities of offshore development.

The third factor could be the company’s CMMI initiative. C1 was appraised at CMM Level 2 in 2000, CMMI Level 3 in 2003, and CMMI Level 5 in 2006.

Because these capability maturity models are very documentation-extensive and plan-driven (which are quite the opposite of the agile philosophy), it is not surprising that the “new” XP practices, which never took root in the first place, are no longer practiced in C1 today.

Of course, all this is not to say that C1 cannot use XP (or agile) again if it has to. As it has demonstrated in the past, it has the ability to develop software in an agile manner, producing products quickly that meet customer requirements. Related to this, it is interesting to note that anecdotal interviews with a number of local software firms’ senior developers or managers suggest that there is widespread perception among software firms that, although they might not be using agile methods at the moment, they are capable of doing so at any time should a customer demand it.

3.3 Case 2: Game Development Firm

Case 2 involves a third-party game developer, hereafter called C2, founded in 2001. In 2003, it released the first Filipino-made game, a single-player role-playing PC game, which won two international awards, and was eventually translated into Russian, German, and Chinese.


Figure 1. Factors affecting customer satisfaction and discontinuation of XP practices in C1. Solid lines denote positive impact; dashed lines denote negative impact.
The change to Scrum was motivated primarily by the need to (1) reduce overtime and (2) reduce rework.

Overtime is common in the game development business and initially, the company thought that this was primarily due to publishers/managers (rather than developers) determining deadlines. Rework, on the other hand, was due to the team’s usually discovering that a game design was not good enough (e.g., not interesting enough) only after 2-3 months of development, which was the amount of time that builds took in the past.

Transition to Scrum (or what the team understood of it) took place quickly. Essentially, the staff was briefed by the project manager about Scrum and how it could reduce overtime and rework, after which they were given articles about Scrum to read. They were then asked whether they would like to implement it, to which the staff unanimously agreed.

Three teams were formed, with 3-4 members each. One team worked on game levels, another worked on characters, and a third worked on the game engine. Each team was cross-functional and had one designer, one or two 3D modelers, and a shared programmer. The game engine team was composed solely of programmers.

Of the Scrum practices in [11], the staff seemed to quickly appreciate the (1) sprint and its subpractices, specifically, short duration (a month), small cross-functional teams which have the final say on their own sprint goals in terms of backlog, and sprint review meetings; and the (2) daily scrum meetings.

The short sprint duration (compared to the previous build duration of 2-3 months) and review meetings at the end of a sprint enabled the teams and publishers/managers to discover much earlier any weaknesses in the design of a game under development. The special creative process of game development actually seems to indicate a prototyping approach to development. The staff also thought that because the sprint teams had the final say on their own sprint goals, there would be less overtime, and the staff would be more motivated to meet the goals, which they themselves have set.

The short daily scrum meetings were the other Scrum practice that the staff quickly appreciated. These daily meetings allowed the Scrum master to track the progress of his/her team more frequently than before, and allowed staff to tell their teammates about any of their issues or problems, and receive responses quickly. The scrum meeting also helped keep the teams focused. The scrum meetings also increased team bonding and camaraderie. However, some scrum meetings went 15 or so minutes beyond the prescribed 15-minute duration. This in turn was viewed by some as decreasing their productivity to some extent.

There were some adaptations to the usual sprint practices. One adaptation was when components built by the teams were being integrated, all the teams were fused into one large team. Another adaptation was the occasional sharing of a specialist (e.g., an artist) by two teams. Still another adaptation was the “extension” of sprints by several days.

At the end of the study period, the staff perceived that productivity had indeed increased due to less rework, due in turn to the short sprints and the review meetings, and also due to problems being easy to raise during the daily scrum meetings. However, overtime remained a problem.

An unexpected but nevertheless positive result of using Scrum was improved teamwork. Because the members of a sprint team had the final say on what their sprint goals were, and because they presented the output of each sprint to the rest of the company, they had a greater sense of ownership of their goal and product and therefore strove hard and worked together to meet that goal and produce a product to be proud of.

Unlike in the previous case, it is still too early to say whether and how Scrum would eventually be effective in reducing overtime while continuing to reduce rework and strengthen teamwork in C2.

### 3.4 Discussion of Case 2

As shown in Figure 2, the use of Scrum by C2 has led to a couple of positive results so far. First, there is increased productivity due to reduced rework. It will be recalled that this was one of the two main reasons behind C2’s switching to Scrum from waterfall in the middle of pre-production. In particular, the shorter (30-day) releases enabled more frequent review meetings with stakeholders, during which feedback was immediately received. This, in turn, enabled C2 to detect and remove early those features that might not sell (based on stakeholder opinions), thereby reducing rework. In addition, the daily Scrum meetings enabled developers to raise concerns, ask questions, and get these concerns and questions resolved quickly.

An “unexpected” but positive result or side effect of C2’s using Scrum was improved teamwork and camaraderie. This result was unexpected not in the sense that Scrum could not be expected to improve camaraderie, but in the sense that when C2 switched to Scrum, it did not do so to improve teamwork and camaraderie. Nonetheless, all the interviewees were happy with this apparent side effect. As in C1, this happiness with increased camaraderie could be due to
the high degree of collectivism in Philippine culture, as suggested in [15].

As Figure 2 illustrates, the increased camaraderie could be due to the frequent Sprint review meetings, which would involve the whole team, if not during presentation, then at least during preparation. The daily Scrum meetings by the teams would also lead to improved teamwork and camaraderie due to the daily opportunity to communicate with other team members, and share problems and solutions. The team’s having the final say on its goals could also increase teamwork and camaraderie because of the amount of interaction among team members that goal determination would entail, and the degree of accountability of the team as a whole for the attainment of its goals.

However, the use of Scrum has not yet led to any reductions in overtime. As Figure 2 illustrates, this could be due to a combination of factors including: repeated underestimation by the Sprint team of the time they need to achieve their goal for a Sprint; Sprint meetings that last more than 15 minutes, thereby taking away time from actual development work; and the merging of the teams into one big team for integration, which appears to lead to communication difficulties within the big team.

### 4. Concluding Remarks

This paper examined the experiences of software development firms that have used or are using agile methods or practices in the Philippines. The results indicate, among others, that agile methods are effective in the rapid production of software that meets customer requirements. Agile methods were also found by the firms in the study to result in greater learning (through the tacit transmission of knowledge) and greater teamwork (due to constant healthy interaction). These results are consistent with the efficacy results documented in [17] and in all the empirical studies surveyed in [17]. Both firms in this study freely adapted agile practices to suit their respective cultural, organizational, and project-specific situations. Again, this is consistent with the results of empirical studies in the literature such as those of [17].

Some of the major lessons that can be learned from the case studies are:

1. An agile offshore project can succeed despite cultural differences between clients and service providers and the non-standard implementation of agile practices. Mitigating factors would include the service provider’s project management process
maturity, sufficient regular communication between client and service provider, and a relatively stable product architecture.

2. The increased camaraderie that agile methods produce (more as a side effect) can be more important than we think, especially in cultures where collectivism is high.

3. Although project management process maturity (or, roughly, CMM/CMMI Level 2) can be helpful in agile offshore development, CMMI-based maturity in engineering and process management (or CMM/CMMI Levels 3-5) can lead to the death of some agile practices in an organization. The experience of C1 (the first case study) seems to reinforce the intuition that it might be difficult for XP practices to co-exist with CMMI practices. However, recent attempts of methodologists (e.g., [18, 19]) might provide the solution to this issue.

The items we have listed as lessons learned suggest the following directions for future work:

1. Because agile offshoring has recently become an issue of global import [2], more studies are needed to investigate the factors affecting agile offshoring success. At this stage, such studies would be mainly qualitative, like this study.

2. It might be useful to study more carefully the impact, if any, of increased camaraderie (among members of a team) to software development, resulting from agile methods, especially in cultures where collectivism is high (e.g., in Asia).

3. It might be useful to investigate further whether project management process maturity is always helpful in agile offshore development, and whether and how higher CMMI levels (levels 3 to 5) are harmful to (the retention of) agile processes.

5. References


