Learning From Experience: The Analysis of an Extreme Programming Process *

Sara Shahzad
Institute for Software Technology,
Graz University of Technology, Graz, Austria
sshahzad@ist.tugraz.at

Abstract

A software development process is required not only to guide and structure the development activity but at the same time it should allow improvement in the process itself. Agile development methodologies are said to be flexible and adaptive to the context in which these are applied. This flexibility and adaptability makes experimentation with the process more plausible and opens new ways to learn by experience. This paper is based on the theoretical analysis of an Extreme Programming (XP) process deployed to develop a multimedia streaming application for mobile phones. It presents the steps taken to initiate the development process, its setup and the continuous review of the process in order to find the best suited way to perform the practices which are needed for a software development activity, as well as, workable for the development team. The paper gives a detailed account of the implementation, analysis and the adjustment of the individual XP practices in order to emphasize how the development team has learned to shape the XP process according to the organization and the project context in which it is applied.

1. Introduction

Software development is a dynamic process which is affected by the changes in the product market, as well as, in the user requirements. Agile development methodologies tend to tackle this problem by incorporating an iterative process to answer the continuous flow of user requirements throughout the development process. XP, which is a widely practiced agile methodology, asks for a customer on-site who works in close collaboration with the development team and manages the changing requirements into the development process. The inclusion of customer input in the routine development process is not an easy task, as it requires a lot of effort from the development team and flexibility in the process itself. Also, the setup of a software development process varies with the type of organization and the product that is being developed. This process adaptability must be paired with flexibility in the process which needs to be continuously molded according to the changing business demands. Keenan [12] defines different ways of process molding and tailoring which are termed as static process tailoring and dynamic process tailoring. Static tailoring is defined as the changes made in the process before starting the project and dynamic tailoring is the "continuous process adaptation" made during the software development. Process adaptation is quite ignored in the first version of XP [1], introduced by Kent Beck. However, in the later versions of the book ([2] [3]) it is said that the process should be adapted according to the requirements of the organization and the product. This opens the doors for XP process improvement which provides software development organizations with mechanisms to evaluate their existing processes to identify possibilities for improving, implementing improvements and also to evaluate the performed improvements [6] [17].

The rest of this paper defines the way the XP process is being tailored by the development team in order see how the process is improved iteratively to meet the project demands. Sections 2, 3 and 4 provide the conceptual background of process improvement, project and process setup, respectively. Section 5 gives the details of iteration-wise improvement process. A conclusion is given in the last section.

2. Process Tailoring for Process Improvement

Agile software development principles give a high value to the dynamic process tailoring during the on-going projects. Regular process retrospectives greatly help in this regard [13]. Agile principle of self-organization also dictates that enough flexibility exists within the process and the
development teams have enough authority to adjust the process according to the needs [9]. Release and iteration-wise agile retrospectives as part of the XP process help overcoming the shortcomings of traditional post-project retrospectives by opening a space for the dynamic process improvement [4]. These retrospectives provide a formal platform for discussing the problems and suggesting improvements in the process and incorporating them dynamically in the coming development cycles, which are not possible using traditional retrospectives that allow learning only from finished projects [14]. The process adjustment and improvement in this project is mostly based on continuous team reflections made on the process after iterations and releases. The team’s experience also shows that start of a new iteration or release is the most plausible check-point to introduce a change in the process.

3. Project and Research Context

The project under study is a multimedia streaming application development for large archives of AV content [11]. The project is a university-based research project which was started in May 2007. The project is expected to be completed in May 2010. The project team is composed of five developers, a business person as on-site customer, and a product manager. The developers and the customer are PhD students and the product manager is the supervisor of their PhD research [10]. The technical experience of the developers varies from programming experience and knowledge taken only from university courses to experience with professional software development. The product manager is devoting a substantial amount of time on the business side of the project as well as on the development process. The research has been focused on XP since the start of the project. XP methodology was deployed as full-implementation in order to experience all XP practices in the defined way. Accordingly, every effort was made by the product manager to have all required physical facilities to aid the XP environment setup. The team was provided with a spacious room with every possible communication aid like white boards and flip charts. Different possible room setting were tried out which could help facilitating pair programming, team meetings and also for individual seating places to provide a private place for every team member [10].

4 XP Process Setup and Background

Although it has been suggested by many experts that a step wise introduction of agile practices must be made and many agile practitioners like Eckstein [5] suggests implementing one technique at a time and addressing the most pressing problem first.

The team opted to use the “big bang” concept of methodology implementation for the said project. As the project was a “green-field” project and none of the team members had any prior experience of the XP practices, thus, they decided to use only XP, not mixed with any other development process, in order to experience it in “pure” form. The standard books of XP by Kent Beck [1] [2] [3] were used as initial guide for setting up the process details. The team used the read-discuss-and-learn loop to understand the process. No initial training was taken by any of the team members. It is felt by many practitioners that agile methodologies require less training [15] and practices like pair programming are a type of continuous training which is better than explicit training. Also, the application and implementation of agile practices can be learned by self training [15] and many successful teams have done it.

In the said project the initial process setup included the practices of planning, pair programming, daily stand-up meetings and small releases and iterations. Infrastructure setup was made for continuous integration and a coding standard was adopted when actual programming tasks were started. Different experience reports describing XP paradigm were also studied. The team also applied some of the practices in the way that “successful” teams had defined. One of the main ideas that were taken from another team’s experience was to pair-work in blocks of time [7]. Two-hour working sessions were defined for a whole eight-hour day. Hence each working day consisted of three two-hour pair-working sessions and breaks in between. This was somewhat like the “Pomodoro” approach [8] which proposes to work in 25-minute sessions and each session is followed by a five minutes break. The initial setup of some of the practices as defined in [10] is summarized below:

- The first release was taken as a one-month release with four iterations of one week each.
- It was a common decision that every task, including coding and non-coding tasks, will be done in pairs. Many managerial and infrastructure setup related tasks have been done by the team in the first release. The team has also worked on a scientific paper writing assignment which is also done in pairs.
- Three types of stories were identified by the team: Application, Science and business.
- The business person, also a permanent member of the team, was identified as the on-site customer.
- The work was done 40 hours per week.

5. Iteration-wise Improvement Process

Agile practitioners greatly advise agile project teams to reflect on and iteratively improve their behavior in a logical
Different Story Types: Initially the team was using planning stories: As the team proceeded with planning work for the planning process and also to save time release cycles. This helped to get rid of extra administrative procedures so the team decided to start with three months and then to three months. After a couple of two-month iterations instead of one week iterations to save the time spent every week on iteration planning.

5.1.1 Release and Iteration length

Release length has been changed from one to two months and then to three months. After a couple of two-month releases, when the planning process matured and estimates became better the team decided to start with three months release cycles. This helped to get rid of extra administrative work for the planning process and also to save time which was being spent in organizing release planning meetings which, at that time, were also attended by the delegates from the project partners. Similarly, it was decided to have two-week iterations instead of one week iterations to save the time spent every week on iteration planning.

5.1.2 User-Stories

The following items have been discussed and tackled regarding user stories since the initial process setup.

• Different Story Types: Initially the team was using three types of stories, that is, application related stories (called application stories), and business related stories (called business stories, for example, stories to prepare for a business meeting with project partners), and research/Science related stories (called science stories, for example, writing a technical paper for a conference). After some time another story type was defined called “miscellaneous” stories which included developer stories as well as management related tasks. Many problems were encountered due to this division and need of different story types. The major problem was due to different planning requirements for different story types. For example, science and business stories required too much time to complete and hence there was always a feeling that it was not possible to give suitable amount of time to actual development work in iterations. Also, estimating stories like writing a paper was quite illogical.

Now, with the consent of the whole team, only application stories are planned and estimated. Science related tasks and business related tasks and other managerial tasks are done whenever required but are not calculated towards iteration and release velocities. This means that these stories are not planned and estimated anymore. The developer stories (infrastructure stories and spikes) are defined but are attached with some application story.

• Planning Stories: As the team proceeded with planning and estimating different types of stories it was understood that planning non-technical stories, such as business related and science stories, was very difficult in the sense that such stories could not be estimated properly. Business and science stories could not be divided into smaller stories. Planning those big stories in any release and iteration meant that more than 50% of the working velocity was eaten up by them and sometimes virtually no time was left for application stories. As described above, the team first took out business stories from planning process. This solved a lot of planning and estimation problems and
the working velocity got better. Seeing this improvement, the team did the same with science and miscellaneous type stories. Developer stories are considered technical debt and are done along with application related stories when required.

- Story Estimation: From the beginning of the project the team has been estimating stories in real time (in days and in hours) and the time was calculated for pairs (as pair days and pair hours). Some of the developers were not comfortable with this type of estimation as due to the unpredictable nature of some stories it was impossible to estimate them for some logical time duration. Recently, the team has started estimating stories in story-points. It is done as a poker game. Cards bearing numbers 0, 1, 2, 3, 5, 8, 13 are used. A given story is estimated by all the developers simultaneously by showing up a card of his/her estimate. Close estimates by all developers result in a consensus and too much different estimates result in a discussion between the highest and lowest bidder. After discussion an agreement is reached and the story is re-estimated.

- Story Writing: Many changes have been made in the story card template and the way the stories are written. The team has understood that writing a story in a form that it is actually understandable and features all required ideas when the developers actually start working on the story is also a big task. It is not as trivial as writing a description of a task. Even a simple statement can introduce many misunderstandings between developers and the customer at the end. The same sentence is usually understood differently by people. This means that the customer has to remain with the developers all the time to clarify the actual sense of the story whenever demanded by any developer. This has been very difficult in the said project as the customer is also the business person and has to deal with the business side of the project. Also, generally, even if the customer is there all the time with an XP team it can become very cumbersome for the customer to explain story cards again and again throughout the release. After an year’s experience with story writing, now finally the team has shifted to the story writing template specified by [16] and finds it quite useful in getting rid of all the ambiguities and misunderstandings about stories during iterations. Although it has taken some time for the team to understand and accurately use the new story template but now that the team has understood it and is quite happy with this. The developers consider it as the best thing to clarify everything about a story at planning time.

- Story Size: In the beginning the team did not pay too much attention on the size of the story. Big stories have also been a source of bad planning and estimation. Recently, with the consent of the whole team and according to the suggestion given by some XP practitioners, it has been decided to write stories of considerably small sizes. Also, the stories are now written by the customer together with a developer one day before the planning day. Previously, when the team was strictly following "customers write and developers estimate" it was quite difficult for the customer to understand the depth of stories and extent of the detail needed to make it unambiguous and clear enough to convey the real matter. Also when the customer was putting up stories at the release planning day or adding stories at the iteration planning, many stories were starting a point of long discussions among the development team and the customer in order to understand and clarify the story. Sometime it meant re-writing each and every story. This usually resulted in anguished developers and depressed customer after the planning meetings. Now, the stories are being written together by customer and one of the developers and are written at least one day before actual planning day.

5.2 Fixed Time slots

Two hour blocks system was dropped after a couple of months of practice. Working in two-hour blocks was not a good experience. It was felt by the developers that it was tiring and too restrictive. Due to this reason the pair programming timings are made flexible to introduce more agility. Pairing times now depend upon the pair partners. Only lunch break is being held at the specified time. As there are only five developers so frequent pair switching is anyway not possible.

5.3 Pair Programming

The practice of pair programming has been applied since the beginning of the project. The team composed of 5 developers therefore one developer always has to work solo. The sole person works on either a spike task or any other task which is logical to be done alone. In the beginning all types of tasks, including administrative and science related tasks, were performed by pairs only. But now pair work is required only for programming and design related tasks as it has become clear with experience that pair working on non-technical tasks is not as productive as is expected.
5.4 Customer Role

Initially, customer role was not well defined. User stories were written both by developers and the customer. As developers also had a say in selecting the stories for release and iterations it lead to long discussions during planning meetings. Also, due to unclear definition of acceptance criteria by the customer it was difficult for the developers and the customers to understand when to mark stories as finished. Now, the customer is the only one to define and accept stories. Stories are marked as finished only when these are formally accepted by the customer.

5.5 Usability Engineer Role

Initially the development team was directly communicating with the usability engineer (UE). The developers could ask UE directly for usability related feedback on the user interface design mock-ups. The customer, as well as, the product manager were also contacting the UE regarding same or some other issues. It sometimes created confusion among the developers and the customer. Due to this reason, now, generally the customer and the product manager do all the correspondence with UE regarding the usability feedback. The UE and the developers may talk directly with each other if it becomes necessary on some technical grounds. Also, the developers and UE communicate for research related issues, for example, usability tests and user studies.

5.6 Test First Design

Only one of the developers had a previous experience of test driven development (TDD). Rest of the team took some time to get used to this type of coding habit. Some of the team members even had reservations in the beginning about the usefulness of the practice in some programming assignments. So it was not made a mandatory practice for the team. Recently, after attending a test-driven development workshop the team has got a new dimension of TDD and now every team member tries to use the practice in routine coding assignments.

6 Discussion and Conclusion

The adaptability of the XP process has helped the team to mold individual practices according to the requirements of the project. The reason behind taking an overview of the change and improvement made in the process since the beginning of the project is to find a pattern between the changes in the business requirements and their effects on the process. It is learned that a need to modify the implementation of a practice arises when it appears not to be helping the development process and this becomes very visible when some major change in the business requirement is to be accommodated. We have seen many changes in the implementation of planning and pair programming practices due to this issue.

Dynamic method tailoring and continuous retrospectives on the process go side by side. The notion of learning lessons from experience during a process cycle and the provision to tailor the process according to the change required as soon as it becomes logical is a great benefit to agile teams.

Experiences of other agile teams and practitioners have also helped in tailoring the process. Although, this report provides a purely theoretic picture of the teams’ experience, still this analysis may be sufficient for the general agreement on many facts of experience. For example, the “big bang” way of deploying all XP practices at once may lead to slow and complicated process adaptation for a team. As it happened in the said project that the team tried to master a lot of practices like planning different story types, pair programming, customer involvement, and test-first development at the same time. Each of these practices should be given enough time separately to let the developers and other team members take maximum benefit from them.

As we have also learned from the experience of other teams, our experience will also be of interest specially for those teams who are working under similar circumstances as being in academic setting and working on a research cum business project.

Also, we have understood that not only experience of developers counts towards the success of the project but also the experience of the XP customer as being an on-site customer also counts towards good and proper planning and hence in “on-time product delivery” by the team.

Finally, the analysis of the whole XP process discussed in this paper depicts that although XP is flexible and adaptable, it still requires a proper implementation of the practices to get the required benefit from them. This can be done only when the logic behind using particular practices is properly understood. The inherent flexibility and adaptability of the process makes room for experimentation and self-learning. Hence, provided with some basis guidance, it also presents a good opportunity for new software developers to learn working in teams and to understand the overall structure and requirements of the software development activity.
References


