Personal Extreme Programming–An Agile Process for Autonomous Developers

Article · January 2009

CITATIONS
19

READS
11,046

3 authors, including:

Iva Krasteva
Sofia University "St. Kliment Ohridski"
19 PUBLICATIONS **116** CITATIONS
SEE PROFILE

Sylvia Ilieva
Sofia University "St. Kliment Ohridski"
70 PUBLICATIONS **338** CITATIONS
SEE PROFILE

Personal Extreme Programming – An Agile Process for Autonomous Developers

Yani Dzhurov, Iva Krasteva, and Sylvia Ilieva

Faculty of Mathematics and Informatics, Sofia University,
5 James Bourchier Blvd, Sofia 1165, Bulgaria
yani.dzhurov@gmail.com, iva.krasteva@rila.bg, sylvia@acad.bg

Abstract. Despite the tendency of growing the size and complexity of the developed software, significant part of it is still developed by autonomous developers. The current research study proposes a modification of PSP which aims at lightening the software development process and making it easier to follow, while keeping the PSP basic principles. The new methodology is extended with proven efficient development practices from the Extreme Programming in order to support better project planning and product quality control. The paper presents the results of methodology adoption and compares it to ad-hock development.

Keywords: software process optimization, Personal Software Process, Extreme Programming, agile software development, autonomous development

1 Introduction

Autonomous developers are software engineers who implement software solutions without being part of a team. Such software engineers are also called free-lance developers or contractors (one-man companies). The model of autonomous developers was born as a result of IT outsourcing and off-shoring which started during the early 90s of the previous century. The major goal of outsourcing and off-shoring is to achieve better product quality with lower labor cost. IT services could be outsourced either to software companies or to independent autonomous developers. The latter are often in a better position to offer lower cost of labor by reducing company operational costs, but in the most cases are not able to guarantee good and competitive product quality.

Autonomous developers are facing big challenges in handling projects in time and with high product quality and minimizing failure. Preconditions for this are the lack of planning and quality control. Autonomous developers need a rigid development process to better organize their daily activities and control projects quality. Most of the current software development processes (both the traditional and agile ones) are mainly targeting team development and need to be customized when adopted for autonomous developers. Only the Personal Software Process is explicitly specified to be used by individual engineers. However, applying PSP correctly requires very good knowledge of the process specification which is quite extensive. Furthermore, PSP involves a lot of efforts to prepare and maintain the amount of documentation and data. Thus, deploying PSP in a real project development is very tough. Autonomous developers are not prone to investing resources and time in learning and implementing heavy

processes because this will delay delivery interval which will decrease their competitive advantage on the market.

The current research study proposes a modification of PSP which aims at lightening the software development process and making it easier to follow, while keeping the PSP basic principles. The new methodology is extended with proven efficient development practices from the Extreme Programming in order to support better project planning and product quality control. The main objective of the suggested methodology, named Personal eXtreme Programming (PXP), is to improve the performance and quality of autonomous engineers by automating daily developer activities and performing regular retrospections.

The paper consists of 5 sections. In the next section and overview of principles and practices of PXP is made. Section 3 presents phases that comprise the development process. How the new methodology was implemented in by an autonomous developer is described in section 4. The section also presents a comparative analysis of the results from adopting ad-hock development and PXP in two phases in one the same project. Section 5 concludes the paper.

2 Principles and Practices of Personal eXtreme Programming

Personal Extreme Programming (PXP) is a software development process designed to be applied by software engineers individually. PXP aims at lightening PSP by reducing the number of scripts being followed and the amount of data to be filled in the forms (for complete reference on PSP specification please refer to [1]). PXP keeps the basic principles of PSP but diminishes the amount of documentation and maintenance efforts. In addition, PXP introduces a subset of the XP development practices which are appropriated to be performed by autonomous developers. The PXP development process is iterative and applying its practices allows the developer to be more flexible and responsive to changes. The suggested methodology relies greatly on automation of a major part of daily developer work to improve programmer performance and shorten the delivery interval and time spent on software system support.

The PXP methodology is based on the following principles:

- PXP needs a disciplined approach- developers are responsible to follow the process and apply PXP practices
- Developers should measure, track and analyze their daily work
- Developer should learn from their performance variations and aim at improving the process based on the collected project data
- PXP involves continuous testing
- Defect fixing should happen in early development stage, when the cost of it is lower
- Developers should try to automate as much as possible of their daily work

PXP is accompanied by 14 development and organization practices. The methodology relies on its practices to ensure good product quality and accurate project planning. Some of the PSP practices are kept and others from XP are introduced to replace the formal and complex methods for planning, system design and its verification.

Six of the PXP practices are preserved from PSP as explained in [2]:

- Time Recording
- Defect Type Standard
- Defect Recording
- Size Measurement
- Process Improvement Proposal
- Code Reviews

Six of the PXP practices are proven effective development practices from Extreme Programming ([3] and [4]):

- Continuous Integration – PXP includes the practices of source control versioning, automated builds, automated test executions, and automated defect submitting.
- Simple Design
- Small releases
- Refactoring
- Test Driven Development
- Spike Solutions

One of the PXP practices, namely Coding Standard, is present in both PSP and XP. PXP suggests a new way to perform planning activities. In PXP the planning of a task is mainly based on reports from previous projects. For each functional requirement a set of technical task is defined. Each technical task is assigned a category – for example creation of unit test, business class implementation, SP creation, UI form design, etc. A task estimate is based on estimates of tasks of the same category in previous projects. Developers should use previously collected data from other projects in order to predict their performance – they should not get the average time for a task of a specific category, but take into consideration the trend of the values in the report. The assumption is that when a developer performs a task regularly at some point it become a routine that take less or equal time to perform as the last time the task was implemented.

3 Phases of Personal eXtreme Programming

PXP process is iterative and comprises a couple of inline iterations and cycles. Requirements and tasks planning are usually done for the whole project as usually requirements are specified in advance and stay stable during the implementation.

In case of requirements change task planning could be revised. Development iterations start with iteration initialization and end with iteration retrospective. During the whole process the developer maintains log files with information regarding tasks planning and actual duration, improvement suggestions and defects count and details. Figure 1 gives an overview of PXP process.

During the requirements phase a document with the functional and non-functional requirements for the system is created. This phase is optional – the requirements document could be generated in a meeting between the client and another employee of the organization the developer is working for, or could be acquired from another software engineer, in case of extension of already existing product.

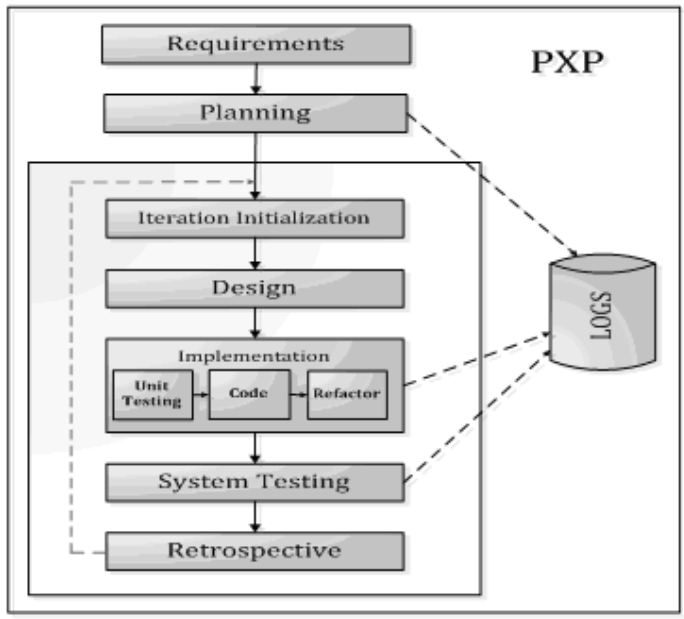

Fig. 1. PXP process phases.

In the planning phase the developer assemblies a set of tasks based on the requirements list document. Each task could be composed of smaller tasks that are categorized. Each small task is estimated based on planning data for tasks of the same type from previous projects (if previously collected data does not exist, then the developer must try to make best assumption about approximate time the task will cost). The sum of the estimates of child tasks is the estimate for the parent task. During this phase prior to task planning major design decisions are made – what programming language will be used, development framework, application model, etc.

Iteration Initialization indicates the beginning of each iteration. Iteration starts with tasks selection, which will be the focus of the iteration. Iteration length could vary from 1 to 3 weeks depending on the project scope. Each iteration could result both a release-candidate or in a released version of the product.

During the design phase the autonomous developer is modeling the system modules and classes that will be implemented in the ongoing iteration. The developer should aim to design the system to meet only the current client requirements without trying to make guesses what would be required in the future. The design method is selected by the developer, but it's recommended to use as simple as possible tools.

Implementation phase is where the actual code generation takes place. The developer implements all objects defined in the previous design phase and tests for them. This phase consists of three sub-phases- unit testing, code generation and code refactoring, executed in this order:

To exit the implementation phase the code should compile without any errors and all unit tests should pass successfully.

All features developed so far are tested during the system testing phase.

Developer should verify whether the implemented solution meets the initial project requirements. All found defects are recorded and fixed.

Retrospective marks the end of process iteration. An analysis of collected during the other phases data is being made. The developer must verify whether the estimated tasks time equals the actual one, find the reasons for potential delay in order to prevent under- or over estimates in future projects. The developer analyzes all process improvement proposals and, if needed, he could adapt any of the accompanying process practices. Changes addressing problems in the process and practices should happen in earliest stage possible in order to prevent project failures. This phase could end either in a release-candidate or in a product release. The 'Retrospective' phase could start a new iteration by moving in 'Iteration Initialization' phase or to mark the end of project development when all client requirements for the software system are met, and there are no remaining open defects.

4 Implementing Personal eXtreme Programming

In the current research Microsoft Visual Studio System (VSTS) was employed for the implementation of Personal Extreme Programming. VSTS is a platform of productive and integrated tools during the whole life cycle of software development. They improve the communication and collaboration in software development process, and allow extensibility by the team using it. Despite VSTS is mainly targeting development teams, autonomous developers could also take advantage of its powerful tools. VSTS was designed to fit the needs of the software engineer who is employing it by software development process templates. A process template consists of a structured set of xml documents that describe different aspects of the process (detailed specifications of a process template are available in [5]). For PXP a new process template was created. The process template contains information about the phases of the PXP process, specific for this development methodology work items (defects, tasks, scenarios, quality of service requirements, process improvement proposals), definitions of reports used during the planning phase, version control system settings and permissions, and associated to the project web portal for document storage. [6] gives an overview of VSTS built-in tools that support all accompanying PXP practices – tools for refactoring, automated code review, work items tracking, report viewing, automated remote builds and continuous integration, tests implementation and execution, software size and quality measurement, and tools for system design. A detailed description on implementation of PXP with VSTS refer to [7].

In order to study the extent to which PXP helps to optimize the development process, the methodology was adopted by an autonomous developer. A software project was developed in two phases – each of them planned to last 2 weeks. The first phase was developed with ad-hoc development, and the second with PXP.

During both phases the following process and project metrics are tracked and analyzed:

- Planning effort – the time spent on planning phase of the process
- Planning effectiveness – analysis of the ratio between planned and actual time spent on project tasks.

- Ratio between found defects and number of functional and non-functional requirements
- Unit Tests Code Coverage – the percent of code covered (executed) when unit tests are executed. This metrics allows tracking what part of the application could be tested in an automated way.

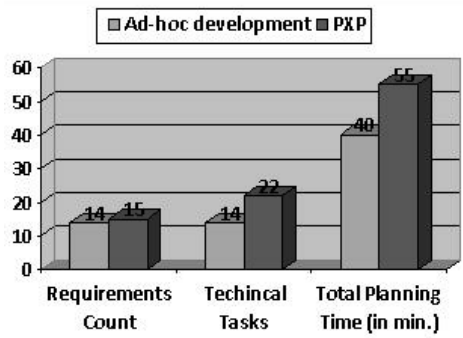

Fig. 2. Results from planning phase in ad-hoc development and PXP.

Figure 2 provides information about the planning effort in both phases. The planning phases cover approximately the same number of requirements for each development method (ad-hoc development - 14 requirements, and PXP – 15). Due the lack of previously collected data for PXP projects, the planning in both phases happens in relatively the same way. The main difference comes in the number of technical tasks being defined – PXP defines more fine grained tasks compared to ad-hoc development. Phase I defines one task for each requirement - 14, where as for phase II are define 22 tasks. In PXP the time spent on planning is 37.5% more than ad-hoc development, because the number of tasks to be estimated is bigger.

Table 1 gives an overview of the planning effectives in both project phases. Ad-hoc development does not include time tracking of planned tasks, so it's not possible to provide detailed information about it, but the total additional time that was spent on development – 16h which is 20% of the total planned time. In PXP time tracking is one of the main practices that facilitate project data collection so more detailed information is available – 5 tasks were finished before the planned time, 3 tasks were finished on time, and 14 were finished after the planned time. The actual development time spent in PXP was 67,5h when the planned was 58. The results shows that the planning effectiveness in both project phases is approximately the same – the reason for this is that the planning itself was performed in very similar way. PXP has slightly better planning effectiveness due to the greater number of tasks being defined – it is easier to better estimate a set of smaller tasks compared to a single big task (as is the case in ad-hoc development).

Table 1. Overview of planning effectiveness in ad-hoc development and PXP.

	Ad-hoc Development	**PXP**
Number of overestimated tasks	-	5
Number of correctly estimated tasks	-	3
Number of underestimated tasks	-	14
Total Planned Development Time (in hours)	80	58
Difference between actual and planned time	16h (20%)	9,5h (16,3%)

The number of defects found in the second project phase is significantly lower compared to the first one (Figure 3). PXP mainly focuses on test driven development which allows defects to be prevented during implementation phase of the process. In phase I of the project development only manual testing was performed and part of the defects was introduced during development of new features.

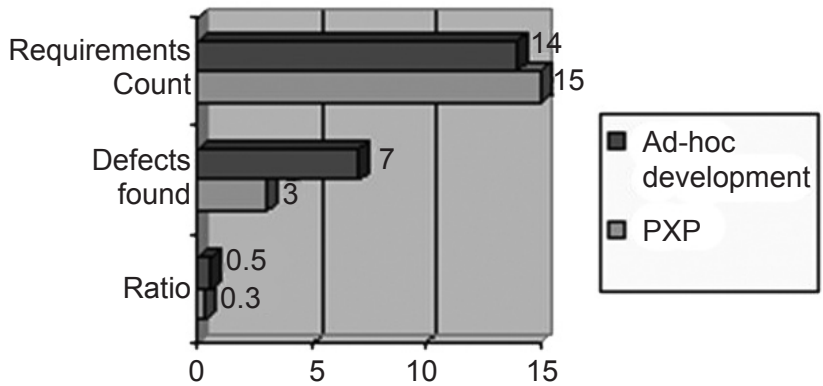

Fig. 3. Ratio between found defects and requirements in both project phases.

The last tracked project metrics is the percentage of code covered by unit tests. In ad-hoc development unit tests are not implemented so the code could be considered 0%. In PXP test driven development is one of the main practices being performed which causes unit tests to be implemented for all classes included in the project. In the second project phased developed with PXP the unit tests code coverage reaches 85.29%. The high level of code coverage allows easier system extensibility and maintainability with less manual testing performed.

5 Conclusion

The paper presents a new software process targeting autonomous developer that addresses their current needs and daily work problems, enhance the developed software systems quality and shorten the implementation time. PXP excludes from PSP the formal scripts, forms and design verification methods, and introduces some of the XP development practices. The implementation of the new process is supported by Microsoft Visual Studio Team System platform which gives opportunities for automating most of its activities. Furthermore, comparative analysis of development following ad-hock process and the PXP is carried in order to validate how PXP influences autonomous developer work. A comparison of the set of generated artifacts of both phases is made. Other monitored metrics of the process about project planning and product quality state that planning when PXP is applied for the very first time is as effective as planning in ad-hoc development. The reason for this is that there is no previously collected data about other project planning phases that could be used as a base for the current project. In the first project developed with PXP the time for planning is significantly more than in ad-hoc development (about 40% more) but this is an initial investment that will pay off immediately

in the next project development. Unlike planning, taking into account defect tracking PXP makes positives results even in the first project development. During the development of modules of approximately same scope, a decline in the number of defects found in PXP phase with about 50% compared to ad-hoc development is observed. PXP focus on test driven development leads to approximately 85% business logic code coverage by unit tests. In addition to this web UI and performance load tests are implemented and executed. All tests could be executed in an automated way, which aims to facilitate system extensibility and maintainability.

As future work, validation of the approach should be made by implementing it by autonomous developer who is experienced in PXP and has history data from previous implementations of the process. In this way, the developer could gain the whole potential of PXP planning practice. Furthermore, the developer could adapt his practices to his needs based on retrospectives and suggestions for improvements made in previous iterations and projects.

Acknowledgements. This work is partly funded by Bulgarian Ministry of Education and Science, NSF.

References

1. Software Engineering Institute, Carnegie Mellon, (August 2005) "The Personal Software Process – Body of Knowledge , v.1.0", http://www.sei.cmu.edu/tsp/tools/bok.html
2. Software Engineering Institute, Carnegie Mellon, (2006) "Self-Study PSP Materials", http://www.sei.cmu.edu/tsp/tools/student.html
3. Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta , (2002) "Agile Software Development Methods – Review and Analysis"
4. Kent Beck, (1999) "Extreme Programming Explained: Embrace Change", Addison-Wesley Professional
5. Jean-Luc David, Mickey GoussetandErik Gunvaldson, (2007) "Professional Team Foundation Server", Wrox Press
6. Richard Hundhausen, (2006) „Working with Microsoft Visual Studio 2005 Team System", Microsoft Press
7. Dzhurov Yani, (2008) „Deploying software development process for autonomous software developers with Microsoft Visual Studio Team System", Master thesis, Sofia University