Extreme Programming for a Single Person Team

Ravikant Agarwal
Computer Science & Software Engineering
107 Dunstan Hall
Auburn University, AL 36849
(334) 559-3545
agarwra@auburn.edu

David Umphress
Computer Science & Software Engineering
107 Dunstan Hall
Auburn University, AL 36849
(334) 844-6335
umphrda@auburn.edu

ABSTRACT
The purpose of this paper is to examine the features of Extreme Programming (XP) and determine how it can be applied to a single person team (i.e. a programmer who works mainly on his/her own as opposed to a programmer who works as a part of a well-integrated team).

A software development process for a single person team, Personal Extreme Programming (PXP) is created. Process script for PXP is explained and the core practices of XP are compared with those of PXP.

Keywords
Software Process, Extreme Programming, Personal Software Process, Personal Extreme Programming, Process Script

1. Introduction
Software development approaches have changed significantly throughout the last decade. There are several well-documented drawbacks of traditional "heavyweight" methodologies and as a response to that, a new group of methodologies have appeared in the last few years [5]. These are known as "lightweight" or "agile" methodologies.

The agile methodologies involve less documentation and are more code-oriented. They are more adaptive as compared to the other "heavyweight" methodologies. Several methodologies now march under the agile banner. They all share similar characteristics but also have significant differences. There are lots of Agile methods in use now days including Extreme Programming otherwise known as XP[2], Scrum[10], Cockburn's Crystal family[4], Feature Driven Development[8] and Dynamic Systems Development Method[9].

Extreme Programming (XP) is the most popular of the various types of "agile" software methodologies. XP preaches the values of community, simplicity, feedback and courage. It is a set of values, principles and practices for rapidly developing high-quality software that aim to provide the highest value for the customer in the fastest way possible. XP is extreme in the sense that it takes many well-known software development "best practices" to their logical extreme [3]. Extreme Programming (XP) was created in response to problem domains where requirements change frequently [11]. XP will be more effective than "heavyweight" methodologies in situations where customers are not sure about functions they want or are likely to change their mind every few months. XP is ideal for developing new types of software products and as such is a challenge for the developers. It is also suitable for projects where a system has to be developed within a strict time constraint.

Personal Software Process is a well-known software development approach and was designed with an aim to improve the quality and productivity of the personal work of individual software engineers [4]. Personal metrics such as lines of code produced per hour, errors per thousand lines of code, percent of time spent in each phase of the life cycle, etc. can be determined using PSP.

Our approach is based on combination of Extreme Programming (XP) and Personal Software Process (PSP) [6]. The PSP helps people to understand and improve their personal performance. It educates them to estimate and plan their work and to do this before committing to start doing the job. PSP, similarly to XP, focuses on building quality products and tracking quality from the initial development phase, instead of checking it right before delivery.

A complementary feature of PSP to XP is that PSP provides scripts that support each engineering activity and facilitate its correct completion. Another PSP feature that pairs the XP objectives is that it assists the software engineers in improving their performance.

2. Personal Extreme Programming (PXP): XP For A Single Person Team
XP focuses on situations involving small teams, where development is done in pairs. Many of XP practices require teamwork, e.g. having someone else reviewing your code. XP requires that all production code be written by two programmers who pair up and constantly review each other's work [7].

There are cases where pair programming is unattainable: where a single programmer is working on a project. As an independent consultant, you might not have a coding partner to pair with. For small or experimental projects, incurring the overhead of arranging pairing to carry out a task may be undesirable. To address these issues, we scale XP to produce a methodology that uses its practices, but in a form that an individual within a traditional project framework can use. The practices of XP are modified so that they can fit in a lone programmer situation and a software development process is created. We call our method PXP (Personal Extreme Programming).

ACM-SE '08, March 28–29, 2008, Auburn, AL, USA.

Table 1 shows the process script for the Personal Extreme Programming (PXP). User stories are written on cards. The card does not contain all the information that makes up the requirement. Instead, the card has just enough text to identify the requirement, and to remind everyone what the story is. The card is a token representing the requirement. It's used in planning. Notes are written on it, reflecting priority and cost. It's always handed to the programmers when the story is scheduled to be implemented, and given back to the customer when the story is complete.

Table 1: Personal Extreme Programming (PXP) Process Script

Entry		E1. Coding standard is available.
Tasks	**1. Planning**	P1. Produce or obtain a requirements statement.
		A. Write System Metaphor.
		B. Write User Stories
		P2. FOR all new User Stories
		A. Pick a User Story and break it into features.
		END
		P3. Create domain-driven design to reflect the domain model in your design.
		P4. Group features into meaningful groups (FeatureSets).1
		P5. Write acceptance tests for FeatureSets
		P5. Estimate FeatureSet size and required development time using ideal days.
		P7. Sort FeatureSet according to priority to create FeatureSetPriorityList.
		P8. Establish Iteration schedule.
	2. Development	D1. FOR all FeatureSets in the FeatureSetPriorityList
		A. Pick a FeatureSet to implement from top of the FeatureSetPriorityList.
		B. IF (change introduced in this FeatureSet == YES)
		1. Update the FeatureSet.
		2. Re-sort the FeatureSetPriorityList.
		C. ELSE
		1. Update design and create an iteration plan for the current FeatureSet.
		2. FOR all Features in the FeatureSet
		a. Pick a feature from the FeatureSet and break it into tasks.
		b. Sort tasks according to their priority to create a TaskPriorityList.
		c. FOR all task in the TaskPriorityList
		c1. Pick a task from the top of the TaskPriorityList.
		c2. Write unit tests for the task.
		c3. Write/Modify the code to implement the task.
		c4. Do code walkthrough.
		c5. Update the Development baseline and maintain versions in repository.
		c6. Compile and unit test the developed code.
		c7. IF (Unit test == Success) then
		i. Perform acceptance testing for code.
		ii. IF (acceptance test == Success) then
		a. Integrate into the Refactor baseline
		iii. ELSE
		Create a new task ("fix code ") as priority 1
		c8. ELSE
		Create a new task ("fix code ") as priority 1
		END
		END
		D. Perform integration test on the Refactor baseline
		E. IF (Integration test == Success) then
		1. Refactor the code in the Refactor baseline.
		2. Integrate into the Production baseline.
		F. ELSE
		1. Create a new task ("fix the Refactor baseline") as priority 1
		H. Perform acceptance testing for code in Production baseline.
		I. IF (acceptance test == Success) then
		1. Iteration release
		2. IF (New FeatureSet introduced == Yes) then
		a. Update FeatureSetPriorityList.
		J. ELSE

		1. Create a new task ("fix the Production baseline") as priority 1
		END
	3. Post Mortem	M1. Complete acceptance testing for code in Production baseline.
Exit		X1. Tested program

The code is developed in iterations and is released as versions. Each iteration starts with iteration planning and updating the designs. The developer maintains three baselines, namely development, refactor, and production baselines. After initial code development, the code is maintained by versions in the development baseline. Upon completion of the iteration, the code is integration tested and is moved to refactor baseline to be refactored. After refactoring is completed and acceptance tests are done, the code is moved to the production baseline for release.

3. Twelve Core Practices Compared

Table 2 shows the one to one mapping between XP [1] and PXP in terms of the twelve core practices followed by XP followers. Most of the core practices of XP are applicable to a single person team, while the others can be put into practice with a few modifications.

Table 2: Twelve Core Practices Compared

Twelve Core Practices	Extreme Programming (XP)	Personal Extreme Programming (PXP)
The Planning Game	The main activity in the planning game is the writing-estimation-prioritization back and forth negotiation of stories between programmers and customers. The customer and development teams get together to decide on what features of the required system will be of maximum value to the business.	If the developer is working for himself or knows enough to stand in for the customer, then one only needs to switch roles (role-playing) during planning. It is a good practice to write stories on cards, estimate, prioritize, and track when developing alone. Writing and using stories is a good way to break down a big project into smaller chunks that are easier to deal with and manage.
Small Releases	A simple system containing a useful set of features is put into production early and updated frequently in short cycles.	Early and often releases can be applied as easily with a lone programmer as with a team of developers.
Metaphor	Each project has a "system of names" and description which helps to guide the development process and communication between all parties.	A single person team can use and refine whatever metaphor proves best. Always choose a metaphor that helps you understand the parts of the system you are talking about.
Simple Design	The simplest design is always used to build the application as long as it meets the current business requirements. Do not worry about future requirements as requirements change with time anyway.	As a lone developer it is easy to maintain a simple design. A solution would be to use the high level design as a goal to work towards but develop an alternate design that is simpler and better but still delivers all the functionality.
Testing	Test-driven development is one of XP's main strengths. Software developed with XP is validated at all times. Before new features are added tests are written to verify the software. The software is then developed to pass these tests.	When writing a new module, programmers should write the interface first, then the unit test, and only then go on to implement the module. In a single person team test suits can be easily written and used. The developer should test and compile his code before putting it into refactor baseline. Until that point the code stays in the development baseline.
Refactoring	This is a technique for improving the design of an existing codebase. Its essence is applying a series of small behavior-preserving transformations that improve the structure of the code. Doing them in small steps reduces the risk of introducing errors.	As a lone programmer one can and should practice refactoring to the hilt except in situations where you don't have permission to change code you don't own. The refactored code should be integrated with the production baseline and tested.
Pair Programming	Programmers using XP are paired and write all production code using a single machine per pair. This helps the code to be constantly reviewed while being written. Pair Programming has proved to produce high quality code with little or no decrease in productivity.	When working as a lone programmer, benefits of pair programming are lost. You could ask a colleague to give you reminders often. Informal walkthroughs and test first philosophy can be applied.
Collective Code Ownership	All the code belongs to every member of the team, no single member of the team owns a piece of code and anyone can make changes to the codebase at any time. This encourages everyone to contribute new ideas to all segments of the project.	If you are the only developer in the project then there is no problem as you own all the source code. However, you will not be benefiting from other programmers' input. The developer can maintain versions of code in a repository and version-control techniques can be applied.
Continuous Integration	The aim of continuous integration is to prevent or reduce code from spreading from the main	If the developer is working alone with no other person making changes to the main codebase then there is no

	codebase; the more frequently code is integrated into the main codebase the less chances that there will be diversion. Software systems are built and integrated several times a day; at the very least all changes are integrated into the main codebase at least once a day. Each build is tested using the prepared test cases.	problem as the codebase serves as the linear record of one's work. Integration will not cause any conflicts and is trouble free but it is still quite easy to diverge from the main codebase the longer you work without integrating. Therefore continuous integration is still needed. The best way to develop is to work on a task, integrate and then move to the next task. This way divergence is kept to the minimum.
40-Hour Week	Programmers in an XP project normally adhere to a 40 hour working week in order to maintain productivity and avoid burn out.	All that is required to do this is to adhere to a 40-hour week and stop working for the day when you need to. This means stop working when no longer productive, are stressed or tired to reduce fatigue and keep you in excellent condition.
On-site Customer	One or more customers who will use the system being built are allocated to the development team. The customer helps to guide the development and is empowered to prioritize and state requirements or answer any questions the developers may have. This ensures that there is effective communication with the customer and as a result less documentation will be required.	If you are your own customer (at least initially) then mumbling to ones self is reasonable. If you have other customers then communication via email or phone will probably moderate this problem as long as the customers are open to communication.
Coding Standards	Everyone uses the same coding standards which make it easy to work in pairs and share ownership of all code. One should not be able to tell who worked on what code in an XP project.	As a single person team, how you choose to code is your coding standard. But some coding standard must be followed to maintain the consistency throughout the development.

4. PXP at Work

The PXP has not yet been applied to a large audience, but has been applied and tested to develop a tool that is capable of guiding a developer to develop a software project using PXP. The process script was followed by a single person team and the results showed considerable ease and efficiency in completing the project in estimated time.

Initially, the system metaphor was created, followed by writing the user stories needed for this tool. The user stories are shown in Figure 1. Then all the user stories were broken down into features. The features needed for this process tool are shown in Figure 2.

Upon creating features needed, domain-driven design to reflect the domain model of the project was created and the features were grouped into meaningful sets and were sorted by implementation priority to create a FeatureSetPriorityList (shown in Figure 3).

Iterative development cycle was used as recommended by the PXP process script and the FeatureSet with the highest priority was picked and iteration plan was created for the set. The FeatureSet was then broken down into tasks needed to implement the FeatureSet, and a TaskPriorityList was created (e.g. Figure 4), along with the unit test for each task.

1. Aid the developer to follow the PXP script.
2. Get the requirement statement from the customer.
3. Create domain-driven design.
4. Break user stories into features and group them into meaningful groups to create FeatureSets. Sort FeatureSets to create a FeatureSetPriorityList.
5. Check for any changes requested by the customer and update the lists.
6. Iteratively break the highest priority FeatureSet into tasks and create unit tests for tasks.
7. Develop FeatureSets by writing code for each task.
8. Perform code walkthrough.
9. Perform unit testing.
10. Perform integration testing.
11. Refactor the code and perform acceptance testing.
12. Trace from user stories to features, from features to FeatureSets, and from FeatureSets to tasks.
13. Maintain the three baselines (Development, Refactor, and Production).
14. Create new tasks as needed.

Figure 1: User Stories

1. Aid the developer to follow the PXP script.
2. Get the requirement statement from the customer
3. Create domain-driven design
4. Pick a User Story and break it into features
5. Group features into FeatureSets
6. Sort FeatureSets by priority to create a FeatureSetPriorityList
7. Check for any changes in user stories and features.
8. Update the FeatureSet and re-sort the FeatureSetPriorityList.
9. Break FeatureSet into Tasks.
10. Create unit tests for tasks.
11. Update design and create an iteration plan for the current FeatureSet.
12. Sort tasks by their priority to create TaskPriorityList.
13. Pick a task from the top of the TaskPriorityList and write/modify the code to implement the task
14. Perform code walkthrough.
15. Compile and unit test the developed code.
16. Perform integration test.
17. Refactor code in the Refactor baseline
18. Perform acceptance testing for Refactored code
19. Trace from user stories to features.
20. Trace from features to FeatureSets.
21. Trace from FeatureSets to tasks.
22. Integrate code into the Development baseline
23. Integrate code into the Refactor baseline
24. Integrate code into the Production baseline
25. Create a new task

Figure 2: Features for the user stories

1. Get the requirement statement from the customer.
   Pick a User Story and break it into features.
2. Group features into FeatureSets.
   Sort FeatureSets by priority to create a FeatureSetPriorityList.
3. Break FeatureSet into Tasks.
   Create unit tests for tasks..
4. Update design and create an iteration plan for the current FeatureSet.
   Sort tasks by their priority to create TaskPriorityList.
5. Pick a task from the top of the TaskPriorityList and write/modify the code to implement the task.
   Perform code walkthrough.
   Compile and unit test the developed code.
6. Integrate code into the Development baseline.
   Integrate code into the Refactor baseline.
   Integrate code into the Production baseline.
   Perform integration test.
7. Refactor code in the Refactor baseline.
   Perform acceptance testing for Refactored code.
8. Aid the developer to follow the PXP script.
9. Create domain-driven design.

Figure 3: FeatureSetPriorityList

1. Write System Metaphor
2. Write User Stories
3. Pick all user stories and break them into features

Figure 4: TaskPriorityList

The task with highest priority for development was then coded and unit tested, and saved in development baseline. The next task in the TaskPriorityList was picked for development.

The code was maintained by versions in development baseline. Upon completion of a FeatureSet (i.e. an iteration), the development baseline was refactored and integration tested and moved to the refactor baseline. Acceptance testing was then performed on the code in the refactor baseline and was moved to the production baseline upon successful test and was stored as a release version.

5. Conclusions

Personal Extreme Programming (PXP) is a software development process for a single person team. It is based on the values of Extreme Programming (XP) i.e. simplicity, communication, feedback, and courage. It works by keeping the important aspects of XP and refining the values so that they can fit in a lone programmer situation.

PXP can still be refined and improved. It is in the tradition of XP practitioners to vary XP to encompass whatever works. We hope that PXP inherits these pragmatic roots, as well. Giving up XP tenets like pair programming is not necessarily a tragedy. We still believe that following XP strictly is a more effective way to pursue multi-person projects. But we are also convinced that many of the XP practices and methods can be applied to individual work.

The PXP approach tries to balance between the "too heavy" and the "too light" methodologies. PXP will inject the right amount of rigor for the situation without overburdening the team with unnecessary bureaucracy.

6. References

[1] Akpata E., and Riha K., (2004), “Can Extreme Programming be used by a Lone Programmer?", Systems Integration 2004 .

[2] Beck K., (2000), “eXtreme Programming eXplained: Embrace Change”. Addison Wesley.

[3] Brewer J., Design J., (2001) “Extreme Programming FAQ”, http://www.jera.com/techinfo/xpfaq.html

[4] Cockburn A., (2004), “Crystal Clear: A Human-Powered Methodology for Small Teams (The Agile Software Development Series)”, Addison-Wesley Professional.

[5] Fowler, M., (2003), “The new methodology”, http://www.martinfowler.com/articles/newMethodology.html

[6] Humphrey W., (1996), “Introduction to the Personal Software Process”. Addison Wesley Longman.

[7] Jeffries R., (2000), “What is eXtreme Programming?” http://www.xprogramming.com/what_is_xp.htm

[8] Palmer, S.R., & Felsing, J.M. (2002). “A Practical Guide to Feature-Driven Development”, Prentice Hall.

[9] Rietmann, (2001), “DSDM in a bird’s eye view”, DSDM Consortium, p. 3-8.

[10] Schwaber K, Beedle M. (2001), “Agile Software Development with Scrum”, Prentice Hall.

[11] Wells D., (2003), “When Should Extreme Programming be used?” http://www.extremeprogramming.org/when.html