

PM WORLD TODAY – FEATURED PAPER – JUNE 2008

Extreme Programming for Better Results
on Software Development Projects

By Aleksandr Tarabykin

What is Extreme Programming?

Extreme Programming (XP) is a structured agile approach to software engineering prescribing a set of daily practices. It emphasizes the importance of communication with the customer, communication between team members, and development of automated testing plans over formal documentation.

XP is based on principles of rapid feedback, preferring simplicity (and rejection of planning for the future, scalability and reusability in advance), incremental changes and embracing the change in the requirements.

Why should I care about XP?

Researches conducted by many companies reflects improved performance, better quality of code (lower costly errors), better customer satisfaction and team morale when XP is implemented.

For instance, JP Morgan Chase study [4] reported that when XP is implemented, the total amount of defects in XP environment is averagely reduced by 63%, amount of critical defects goes down by 79% and number of features not working as designed is reduced by 38%. At the same time, the development time is reduced by 44-47%.

In 1999, a controlled experiment run by Robert Kessler at the University of Utah investigated the economics of pair programming. Advanced undergraduates in a Software Engineering course participated in the experiment. One third of the class coded class projects as they had for years – by themselves. The rest of the class completed their projects with a collaborative partner. The results of how much time the students spent on the assignments are shown below in Figure 1. After the initial adjustment period in the first program (the “jelling” assignment), together the pairs only spent about 15% more time on the program than the

individuals [6]. Development costs certainly do not double with pair programming! Significantly, the resulting code has about 15% fewer defects. (These results are statistically significant.)

Figure 2 shows the post-development test cases the students passed for each program – essentially the percentage of the instructor’s test cases passed.

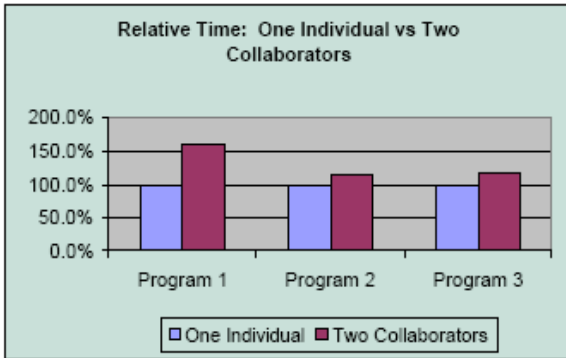


Figure 1: Programmer Time

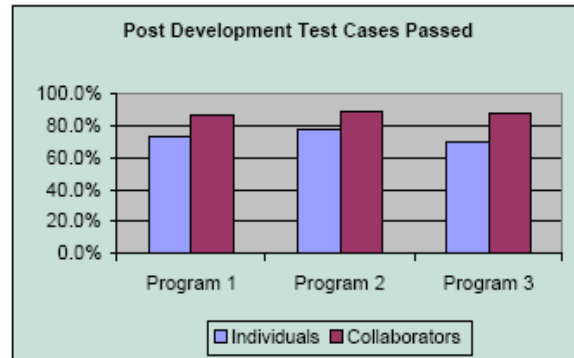


Figure 2: Code Defects

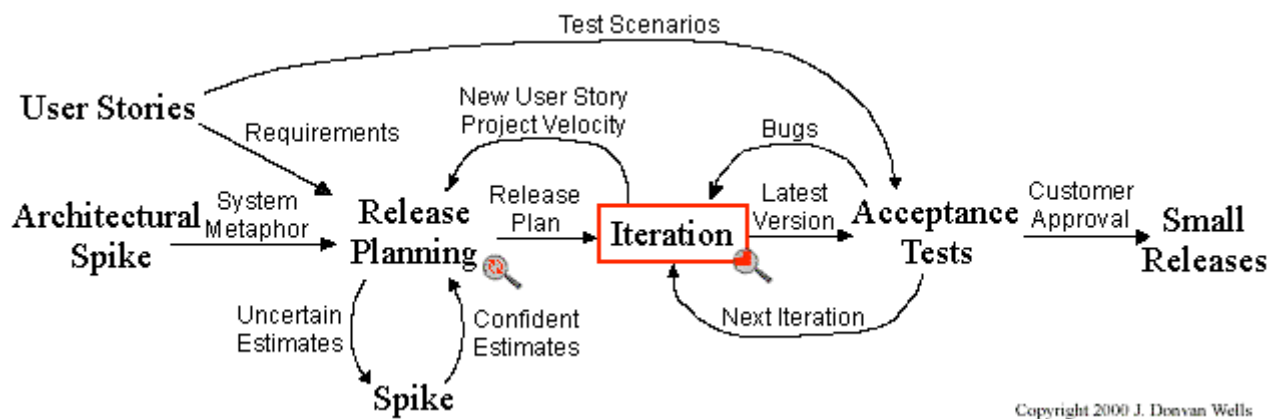
The initial 15% increase in code development expense is recovered in the reduction in defects, as the following example illustrates. Let a program of 50,000 lines of code (LOC) be developed by a group of individual programmers and by a group of collaborative programmers. At a typical rate of 50 LOC/hour, the individuals will develop this code in 1000 hours. It will take the pairs 15% longer or 1150 hours -- a cost of 150 hours. Based on representative statistics reported in [6], programmers inject 100 defects per thousand lines of code. A thorough development process removes approximately 70% of these defects. Therefore, the individuals would be expected to have 1,500 defects remaining in their program; collaborators would have 15% less or 1,275 – 225 less defects. In some organizations, developers’ code is passed to a test or quality assurance department, which finds and fixes many of the remaining defects. Typically, in systems test it takes between four and 16 hours per defect. Using a fairly conservative factor of 10 hours/defect, if testing finds these “extra” 225 defects they will spend 2,250 hours – fifteen times more than the collaborators “extra” 150 hours! If the program is sent directly to a customer instead of a test department, pair programming is even more favorable. Industry data reports that between 33 and 88 hours are spent on each defect found in the field. Using a fairly conservative factor of 40 hours/defect, if the customer is plagued by these “extra”

225 defects, field support will spend 9,000 hours – sixty times more than the collaborators “extra” time!

How does it work? What are the practices of XP?

ExtremeProgramming.Org proposes this simplified definition of XP Rules and Practices and matches them with 4 areas of Software Development Life Cycle:

a) Planning:



Copyright 2000 J. Donovan Wells

Figure 3. Planning workflow according to Extreme Programming.Org

- User stories are written (user requests a feature and drives development).
- Release planning creates the schedule (programmers estimate time required to develop each user story and users choose which stories are the most important for them)
- Make frequent small releases (releases must come every 1-3 weeks)
- The Project Velocity is measured (project velocity is a sum of estimates for the tasks/stories completed during the duration of a project)
- The project is divided into iterations (divide your project into a dozen of short iterations; keep iteration length constant; keep iteration deadlines seriously).
- Iteration planning starts each iteration (each iteration must start with choosing user stories to complete during the current iteration; it is against the rules to plan ahead).
- Move people around [to avoid serious knowledge loss and coding bottle necks]
- A stand-up meeting starts each day (everyone stands up in a circle to avoid long discussions)

c) Testing (comes before coding!):

- All code must have unit tests (a failure of a test gives authority to a developer to write any code, no code can be written unless a test is prepared)
- All code must pass all unit tests before it can be released (which allows collective code ownership and ensures new functionality doesn't break old one)
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published (Acceptance tests are black box system tests developed from a user story; each represents some expected result)

d) Coding:

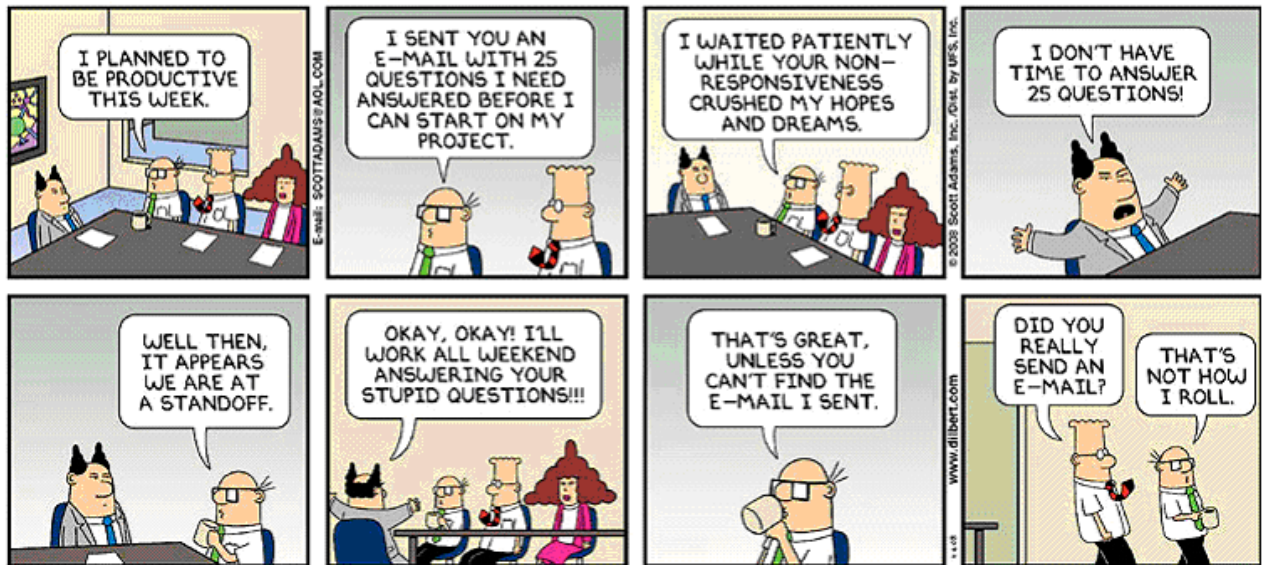
- The customer is always available (a customer must be available to the development team at all times)
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed (All code to be included in a production release is created by two people working together at a single computer)
- Only one pair integrates code at a time (to keep the system consistently robust at all times, all source code is released to the source code repository by taking turns)
- Integrate often (Developers should be integrating and releasing code into the code repository every few hours, whenever possible)
- Use collective code ownership (Collective Code Ownership encourages everyone to contribute new ideas to all segments of the project)
- Leave optimization till last (Make it work, make it right, then make it fast)
- No overtime (Working overtime sucks the spirit and motivation out of a team).

Is it practical enough in a rapid development world?

Most software development in the 1990s was shaped by two major influences: internally, object-oriented programming replaced procedural programming as the programming paradigm favored by some in the industry; externally, the rise of the Internet and the dot-com boom emphasized speed-to-market and company-growth as competitive business factors. Rapidly-changing requirements demanded shorter product life-cycles, and were often incompatible with

traditional methods of software development. So, XP emerged as a response to needs of modern era of software development and is actually tailored towards rapid development – its principles of simplicity, not over-designing things, not spending much time on writing formal documentation (but spending time on communication with the user and building test/acceptance tests), short-time releases are essence of real world requirements of modern IT industry.

Naturally XP as any other project management methodology has some managerial overhead and requires some discipline to follow it. But it is a light-weight methodology without other formalities than geared to the better quality of code and improved communications inside the team and between the team and a customer and therefore can be easier adopted by a small company which would be involved in a rapid development.



© Scott Adams, Inc./Dist. by UFS, Inc.

Figure 5. XP is not about losing time on those meetings

On the other hand, as Chaos report shows us, not having any methodology is costlier and lack of proper management and communication could lead to a project failure, as well as customer dissatisfaction, lack of confidence, and loss of potential contracts as a result of bad reputation which might be hard to overcome for a small company or a start-up.



Figure 6. Your customers would not want to be in this situation

Therefore, it's fair to say that a method allowing to avoid no Big Design Up Front, yet still allowing some structure and methodology into the project team sounds like an appealing and practical solution for a RAD development type of projects.

Does the word “extreme” suggests pushing software engineers more and how does it correlate with previous statements?

No, it doesn't. The idea was to take several proven and time-tested best practices of a successful software companies (or projects) and bring them to the extreme level. Here are some facts:

- the concept of writing tests first was used by NASA in 60s.
- the concept of bottom up development (from simplest blocks to complex), code refactoring and development in iterations were introduced by Leo Brodie in his book “Thinking Forth” published in 1984 [7]
- The ideas of reduced design time, collective code ownership, team rotation, pair programming, face-to-face work and reducing meetings were summarized in “Constantine on Peopeware” [8], but were obviously a summary of best practices and ideas which were already in the IT industry.

Software industry and big consulting companies which can afford rigorous following PM techniques already have a standard way of conducting a project (Waterfall method). Isn't strict following to its practices superior than following XP?

XP does not replace the need of 4 stages of Project Life Cycle. In fact, each stage of project life cycle is addressed by one or more technique or exercise of XP. XP project consists of several iterations and each iteration has to go through its initiation, planning, execution and closure. User stories create basis for project initiation, planning game undertaken in the beginning of each iteration creates project plan with its scope statement, time and budget management, acceptance tests are both parts of scope statement and once all of them are met, are reason for project closure. By going through several iterations, a team would learn lessons from previous iterations (and measuring project velocity is one of the mechanisms to assure that) more likely and would be more relevant than if a team closes a big monolithic project for a customer and probably would never have to deal with that customer again. So, learning phase also comes faster and delivers much needed experience when it is more valuable. If we take one step further and compare 9 areas of Project Management knowledge as defined in PMBOK, we would see that Extreme Programming practices each contribute to several areas of knowledge and complete set of XP addresses each area with its own lightweight approach (see Appendix A).

Comparing Waterfall Method vs XP itself we must notice that while certain projects and certain environments cannot avoid following the required methodology (by political reasons, or because of the established protocols), it has several big problems in it which brought agile techniques into existence. Big upfront costs, late testing, high level of pressure during the integration phase were the reasons why even big companies are constantly seeking for new ways to organize their development efforts.

Waterfall method is also considered to bear higher risk, since no integration is expected until the end of the project. With agile development in general and XP as one of them, customer would have a clearer picture and some working software with most desired features even if the project would not be completed on time. A good example of this can be a story of Chrysler Comprehensive Compensation System (C3). Chrysler was following traditional Waterfall method and after one year the project was on the edge of its cancellation with no software at all. Then

Kent Beck was hired to fix the system in 1996 and in 1997 his team delivered a working system which was actually able to pay 10000 of Chrysler employees. In 1998 Daimler took over Chrysler and early in 2000 the project was cancelled. This project is considered both success of XP methodology and its failure, but the project was definitely not on track before 1996, so from the pure mathematical point of view delivering 0 in 1 year is still less than delivering 10% of the project in 3 years.

Another project worth mentioning is Ford's Vehicle Cost and Profit System (VCAPS) project where waterfall method was used for 5 years to build a system. And after 5 years of development and as result of some hiring of new employees, some change in methodology was started as an underground effort. The team of XPers started first with applying automated tests which allowed reducing number of bug reports by 30% within the first 6 month and applying collective code ownership principles and frequent iterations reduced error reports by another 10% (all happened within the first year). Such significant reduction of bugs allowed the team to won management support to start system over. The new system with 80% of functionality (including unit-tests) was developed in 6 months. A good example of XP strength was that when a new requirement came to be included into both new version and old version of VCAPS, it took 3 days to incorporate new requirements into XP version (with the certainty that no old functionality is broken due to existence of automated tests) and it took 30 days for non-XP version to include new functionality. The project was officially cancelled before version 2 was completed, XP version of VCAPS (v1) replaced in production non-XP version and is still in production after 10 years with only 2 programmers left to maintain it. It is still scheduled to be eventually replaced by another cost and profit system, but the quality of VCAPS XP v.1 made it possible to still be there after 10 years of being in maintenance mode. Peter Foreman (VCAPS project team member) wrote about VCAPS project - *"This is a success story for XP. The goal of XP is to have customers happy with the software, managers happy with the cost, and developers happy to be on the project. We had all 3."*

How could Extreme Programming help us to achieve better results?

During Planning Phase of Software Development Life Cycle:

- Through collecting and processing user stories, a team gets important user feedback as to which features are the most desirable and customers get an idea about how much time it might take to implement requested feature.
- Release planning allows the team to implement the most important features first.
- Planned Iterations adds agility to the development process and allows the team to adjust to changing requirements. Having many releases also improves software integration, quality control and team morale.
- Measuring Project Velocity allows the team to come up with a good schedule and maintain about the same pace of development through many iterations.
- Standup meetings save time for development.

During Design Time

- Spike Solutions allows a team to attack technical challenges early.
- No functionality can be added early – which means we keep the system uncluttered with extra stuff you guess will be used later (of which only 10% would be actually used).
- Refactor requirement and allocating time for clean up at the end of each iteration helps us to keep our code clean, simple and healthy.

During Implementation

- Unit Test first rule requires to focus on input/output parameters and simplest solution to pass the test before doing any optimizations. It also creates a valuable documentation for other developers about how to use methods of the class; and provides a feedback to customers/project managers if the task is understood correctly by the developer.
- Collective Code Ownership improves overall quality of software, improve team flexibility and sharing knowledge of the whole system and boosts productivity.

During testing phase

- Ensures all code is tested with Acceptance Tests
- When a new bug is found, new test must be developed

Did you ever applied principles of XP to your own projects and how did you benefit from them?

Yes and no. XP is a set of principles and techniques evolving and clarifying over time and slightly different from situation, project length, team size and team members themselves. And so is my knowledge and application of XP, evolving with time. But as XP was built upon a solid foundation of best practices, I also used some of the XP methods without realizing they are part of XP. Let me tell you several stories illustrating different real situations.

Tale about “not using XP” and Swaps.

Before I would start writing about my personal experience with XP, I would like to share my deep regret that I did not know about XP before recent. I used to work for a financial software company and was assigned to the project of implementing Swap-contracts into our system. I did not heard about NUnit-testing methodology at the time, so every now and then (usually some big piece of code added) I would spend a day creating different swap-contract scenarios and running them manually. I would estimate that at least 20% of my time was lost because of need to test it manually and another 20% of time I would spend trying to fix the system when new functionality break some of old one.

Although we have not embraced agile approach, our team (me included) were using some agile and XP techniques. One of which would be short release cycles (not as short as XP would prescribe, but still we would be required to consistently bring small pieces of functionality each 3 months). Most remarkably we were doing a form of remote pair programming when an outside software consultant would be paired with an onboard software engineer taking turns in development due to different time zones. So, we maintained a shared body of knowledge which helped us to preserve the knowledge of every piece of code whenever either offshore or onboard developer had to leave. The one who stays with the company would keep a very valuable record of original product requirements, design decisions and reasons behind them and would help a new person to get on track much faster.

Tale about developing mobile label printing software and how unit test saved me from embarrassment to find the bug only at the very late stage.

Here is another recent story. I am currently developing a Mobil Label Printing software for a warehouse vendor and decided to use some techniques of XP (I couldn't use all of them because the project is supposed to be just 1 month long and I have only myself in the team). I decided (partially as a response to a criticism from my friend, experienced software engineer) to try "tests come first" approach and started implemented different methods and classes with delivering a unit test first. The application is really small and many methods seems to be trivial, like implementing simplest communication protocol and parsing XML config file for parameters. And since I am getting paid only for the final result, there is not any benefit for me to develop anything which is not required for successful closure of this project. But I decided to stick to the unit tests and time spent on NUnit tests saved me from 3 or 4 errors which might be more difficult to fix once the product is sent for external testing. One of those errors would very likely to be not noticed by the internal testing and could result in need to deploy the application to all mobile devices and could cost me loosing the customer.

Tale about this paper.

This paper is a result of reading and researches conducted as an individual mid-term project. The key word is project and writing the paper reflects stages of initiating, planning and execution of it. It did not seem right to me to make a paper about XP using the Waterfall approach. More importantly, I could not envision the big picture and start writing the project and as a result miss the deadline.

Then I tried to think small and figured a good way to apply it to the practice. I decided to break this paper into small stories or answers on specific queries and focus on answering one at a time. Whenever I come up with a good question (or hear a good objection from one of my friends, I would right it down to the end of this paper), but I wouldn't focus on it until its time come. I uploaded this paper often to swiki (so, whenever Jeff loses his patience with me, I would have something delivered). I don't know how my work would be graded yet, but I consider applying some of XP practices here a success, as without it I spent a lot of time delivering nothing at all.

Appendix A. Comparison between 9 areas of knowledge as defined in PMBOK and XP practices

	Integra-tion	Scope	Time	Cost	Quality	Human Resource	Communications	Risk	Procurement
Pair Prog			Immed. Peer Support		Immediate peer review	Cross-tem training			
Plannin' Game		Defines Scope	Defines Time	Defines Costs			Feedback to and from customer		
Test-Driven			Automated tests save time and cost		Tests insures quality				Acceptance tests assure proper closing once met
Onsite Customer	Project Initiation						Fast Commu-nications		
Continuous Integration					Ensures working product		Working product is available to customer	Reduces risks of failure at the very end	Released products are available for evaluation early
Refacto- -ring					Keeps code clean and easy to maintain				
Small Releases	Each Release is easier to initiate and plan	Estimates are more precise					Each release is an entry point to comm..	Custom-er gets working product even if the project is closed	Release is easier to evaluate and close. Feedback is used early.
Simple Design			Keeps time and costs of development under control					Focusing on simple design helps team creating huge and unuseful framewor-ks	
System Meta-phor			Developers intuitively understand code			Members of a team can understand and write code faster	Improved communicat-ions within team		

Collec. Code Owner					Internal quality control	Cross-team training			
Coding Conventions			Developers intuitively understand code		Code is consistent with best practices	Members of a team can understand and write code faster			
Sustain-able Pace		Measured velocity gives solid ground to estimate time to implement each feature			Well-rested team performs better and delivers higher quality code	Higher job satisfaction and less stress		A team has additional resource in case it cannot meet deadline	

References

During preparation of this paper I used these references, which are a great starting point to read about XP and IT industry in general:

1. Ward Cunningham Wiki-wiki Web <http://c2.com/cgi/wiki?XpFaq>
2. Extreme Programming FAQ by John Brewer and Jera Design <http://www.jera.com/techinfo/xpfaq.html>
3. Donovan J. Wells "Extreme Programming: A Gentle Introduction" <http://ExtremeProgramming.org>
4. "Extreme Programming (XP), Six Sigma, CMMI. How they can work together JP Morgan Chase case study" by Bob Jarvis and Stephen Gristock <http://www.sei.cmu.edu/cmmi/presentations/sepg05.presentations/jarvis-gristock.pdf>
5. History of Ford VCAPS project by team members <http://c2.com/cgi/wiki?VcapsProject>
6. Laurie Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries "Strengthening Case for Pair-Programming" <http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF>
7. Leo Brodie "Thinking Forth"
8. Larry Constantine "Constantine on Peopleware"

About the Author:**Aleks Tarabykin***Author*

Aleks Tarabykin is a software engineer. He completed his Bachelor and Master degrees in Computer Science at Tomsk State University (Russia) in 1997. He was working for TomskTelecom Corp and Contek Ltd in Russia and moved to Boston, USA in 2001. He was working for 6 years in Eagle Investment Systems, a fast growing company providing an enterprise-level system for Investment Accounting and Reporting for Mutual/Hedge Funds, Brokerage and Insurance Companies. Currently, Mr. Tarabykin attends Harvard Extension School and works towards earning his Master Degree in Management. He also works as an independent consultant helping several companies to set up and successfully operate offshore software development teams. He lives in the Greater Boston Area, MA, USA and can be reached with questions or comments regarding this article, ideas for new ones or business opportunities by email at sas@tarabykin.com.