# Extreme Programming, Too Extreme?

Kyungsub Steve Choi
and
Fadi. P. Deek

Information Systems Department
Guttenberg Information Technology Center, Suite 4400
323 Martin Luther King Jr. Blvd
Newark NJ, 07102
Phone: +1 (973) 596-3366
Fax: +1 (973) 596-5777
{ksc0984}, {fadi.deek}@njit.edu

Presenter: Kyungsub Steve Choi

Keywords: programming, extreme programming, pair programming, collaboration, agile software development

**Conf: SERP'02**

**Paper ID: 1031SP**

# Extreme Programming, Too Extreme?

Kyungsub Steve Choi and Fadi. P. Deek

Information Systems Department
New Jersey Institute of Technology

**Abstract**

*Extreme programming is one of the agile software development paradigms that is grabbing peoples' attention with its dramatic and anecdotal approach. Its value system is completely different than the traditional practices. Despite its kudos, there is a large significant body of concerns and oppositions. This paper shares some of collected opposition views and discusses them in an effort to understand the opposition's validity and further the understanding of extreme programming.*

Key words: extreme programming, pair programming, collaboration, programming, agile software development

## 1. Motive of this paper

In the last couple of years, the startling XP popularity increase has drawn more scrutiny. All published XP articles starts off with a few paragraphs of author(s) subjective views to XP, either pros or cons. Recently, a growing number of anti-XP voices are spotted in both literatures and the Internet newsgroup. Any of the active XP debates is easy to spot on web at anytime. Besides the opposition at some ignorance level, there are some earnest sentiments and legitimate concerns. This paper presents some of them in an effort to understand better on the opposition's validity and XP.

## 2. Opposing views to XP

The sources of the opposition views of this paper are gathered through a search in literatures, Internet newsgroups and other online discussion forums. As a nature of Internet newsgroup and online discussion forum, many postings were authored either anonymously or in pseudo-names. However, there were a many postings with valid personal name and professional affiliation, mostly from the XP authors and XPers. We have carefully evaluated and analyzed the posts from mostly the XP opposition or anti-XP posts. The list in this paper is a categorized collection of most common complaints that we have gathered from the sources. Because of the vast amount of complaint posts and their variety, the list is only a portion of them. The profile of typical online discussion participant is an experienced professional programmer who either did try XP or encountered XP indirectly through published literatures or in a managerial capacity. It was apparent that the legitimate posts were likely from the professionals who have actually have tried XP, and this was found to be very important in understanding XP.
The following paragraphs are described per opposition views.

***"XP is simply not a good method in producing high-reliability software."*** [4], [13]
A quote from the article [13] says "however, in order to successfully develop

software we have to limit the degrees of freedom our nice languages, tools, and personalities offer and select from all the practices and techniques available and formal quality models are useful guides." Everything about XP is issue. The XP itself is against all the good software practices. The C3 project [1], which is the showcase XP example, is misleading because the project was cancelled after being nearly four years in XP mode [13]. XP the "humanistic method" [15] is risky and unaccountable. It is true that the human factor is the key to success but in the same token, it can also be the key failure factor. The decades of hard work and caring minds have established many prudent software process principles and there is definitely a set of irreplaceable values, which cannot be undone by XP.

***"XP is not good in systems that haul many constraints and strong security requirements."*** [13], [19]

XP's concept is "do it simply as possible and do it fast." XP is not a good match for a project that carries a bag of many interacting constraints of other associated entities. If a project where the security and specified constraint requirements are more value than the project itself such as a military missile guiding system or a nuclear reactor controller system, then XP is definitely not the choice. The government regulated industries do not use XP at all as the features such as no documentation and no up-front design are completely unacceptable per government inspection which tells a lot about XP as a prudent software process method.

***"On-site customer will not work."*** [4], [19], [27], [28]

The first on-site customer of the first XP project, C3 [1] was burned out only few weeks into the project and was not adequately replaced [13]. To relieve the load a pair customer was tried, but the pair customers would not help either, as they don't speak in "one voice" [19]. Human memory is bad and should not be used as an accurate account. The requirements of software must be accurate and specific. On a more practical level, the on-site customer would likely be a junior in his organization, which means his answers to the programmers' questions may not be correct. No company would spare a senior staff to a project that requires on-site for months or even more than a year. Further more, who would take the job if the job requires nothing but answering random questions of programmers whole day? [4]

***"No documentation is insane."*** [4], [11], [13], [19], [27], [28]

XPers insist that the code itself is document. This is an oxymoron. The documents are mainly there to explain what all those thousands codes mean. "Considering maintenance and usage aspects, proposing "no documentation" is clearly a professional malpractice" [13]. Albeit exercising the good programming practices such as inserting full conforming code comments as possible, it's still not a substitute for the no-documentation. According to XP, a large stack of binders of documentation is no short-term value [3]. This is completely incorrect. Software documents are not only for the long-term value such as production maintenance, document itself is a part of software that is inseparable. How about the change control documentation? What got fixed, what's outstanding, what's on standby, and more. There is a definite short-term value. Exercising good programming practice such as inserting a long code comment, a description in the beginning, appropriate readable code comments for each significant code, indentations and "white spaces" for easy readability are what every programmer must adhere to regardless of documentation or no-documentation. In government-regulated industry, software is not complete software without its complete set of documentation. After raising this no-documentation concern, XPers have lowered their point to "optional or minimum documentation," but the true spirit of XP is no documentation. Also, there is no such thing as optional or minimum documentation. It's either you have it or not. Here, XP directly collides with the traditional software process value.

***"No up-front design will only frustrate the programmers if not give up."*** [4], [11], [13], [19]
Just like the no-documentation, XPers do not exercise any formal design stage. After collecting the index cards of user stories, immediately they start to code. A structured and careful design is again eliminated in favor of the spontaneous pair collaboration, integration and programming. Again, XPers explain code itself is design. For an outsider, this is a hard one to accept in programming. Programming without any up-front design has been epitomized as the programming ignorance. To many, this would be like a platoon going to a battle without any plan and plan as they battle, who would survive?

***"XP's pair programming is a double-edged sword."*** [4], [14], [17], [27], [28]
Pro XP articles are rushing to share the "good news" about XP, how great the pair programming is and its wholesome results [24], [25]. Is this expected always and in every time? The psychological aspect of pairing up with another individual warrants a careful preparation. It is true that a good pair can bring the synergy [2], [10] that XPers are boasting about, but on the other side of the coin it can also bring a disaster. Possible reasons for a failure can be in differences in personalities, work ethic, job satisfaction, personal agenda, the relationship with the paired-partner, cognitive approach in problem solving, and some unfortunate social issues such as politic and power management. In programming, "it takes a certain kind of programmer to want to work in harness with another. Many of us prefer to work for bursts of creative time in isolation…" [8] It's tough when one is trying to think and other is trying to talk. The protocol of pair programming is vague as well because there is none, at least not an official one yet. Some articles do suggest some general guidelines [24] but enforcement is unlikely. Many who are planning to try or have tried XP are mostly concerned about this pair programming [14], [17], [27], [28]. As the human nature has it and the "Internet time" as the primary driver for the business ecosystem, the time factor eventually forces the more experienced programmer to drive the keyboard as the less experienced programmer would "assist" in getting the job done quickly. This is what's happening to a typical pair that is told to exercise XP with a project deadline hanging over their shoulders [11], [27], [28].

## 3. Discussion on the views

***"XP is simply not a good method in producing high-reliability software."***
It's unfair to say that software developed by XP is not high-reliable software in general. Under what context and the purpose of software are important. The aggressive unit testing practice of XP has been praised even by the skeptics and some opponents [8], [27], [28]. Anecdotal as is, the XP results [25], are surprisingly high in quality. Despite the fact that XP process is very skeptic and questionable, the results are promising hence, we have this XP euphoria. There is an effort [18] to bring this strange and extreme method to more into software process practice mainstream. This paper is certainly too petty in even an attempt to discuss the reliability issue. However, the issue may be accepted accordingly to one's background. In software community, the programmers tend to favor the code-centric XP hence do not agree with the reliability issue, and in the other hand software inspectors and engineers

completely agree to the reliability problem [16].

***"XP is not good in systems that haul many constraints and strong security requirements."***

A project that carries many constraints and strong security requirements means it's an organizational critical system. Military software is a good example. When software deals with human lives it allows zero tolerance in the performance and safety. A quote from an online message [19] reads; "You're got to know when XP (or any of the agile methods) is appropriate. If you are building flight trajectory calculation for a missile targeting system, you shouldn't even be thinking about XP or agile. For God's sake, get those requirements down, in stone, simulated, have the engineers look them over, and so forth." For all life critical and organizational critical software, XP is not the right choice.

***"On-site customer will not work."***

Here, the opponents' view is well founded and this is echoed repeatedly in a number of posts [11], [19], [27], [28]. In many real life cases, it would be lucky to have any willing customer in a software development process. In the traditional way, only the requirement engineering has the customer's involvement, but in XP the customer is on-site and in the process from the scratch to delivery, and all the deliveries (releases). This imposes a very stressful job condition to the on-site customer. The on-site customer must make himself available to the programmers' questions at anytime and all the time, worse yet this is all he does the whole day. If known, undoubtedly no senior staff would want it. This is another reason why XP must be used in a small project or else the poor on-site customer will have a burnout just like the first on-site customer in first XP project, C3. Use it in a short small software development so that the stay of on-site customer is a trivial. Also, this is a good example of how important is the requirement engineering for the success of a project. On-site customer is most extreme case of requirement engineering and it will not work. There is a suggested alternative [13], but this will go as the field of requirement engineering advances.

***"No documentation is insane."***

After a massive opposition in this, XP lowered itself to "optional or minimum" or "as you wish" documentation. There are probably many reasons of why we should have the documentation but the XP's view to "only the long-term benefit" [3] documentation is a lesser value than the code, the end product and delivering the software in time. This view can be discussed, challenged, and rebutted endlessly between the opponents and XPers. The continuous pair review, pair integration, unit testing and the final customer acceptance test do not and can not fill the void left by the missing documents. Coding is definitely not an isolated activity. All associated activities are involved and interact with the coding activity and documentation is required in defining and structuring this. The documentation always has been the map in telling how much of progress is made and where to go from here. "Document appropriately" but keeping the XP pace is what the debate is.

***"No up-front design will only frustrate the programmers if not give up."***

This really challenges the way most programmers are used to think and approach before coding [7]. Without having any kind of architecture up front, it forces the programmers to "design as you code." This really turns off many experienced programmers [8], [11], [13], [27], [28]. One alternative is to provide a written minimal but concise high-level architecture would lessen the frustration and then the programmers may "design as you code" for the lower-level architecture.

To many of these questions, the XPers' attitude of "practice, practice, and more practice" certainly does not help to ease the issue.

***"XP's pair programming is a double-edged sword."***
Besides all the eyebrow raising features of XP, the pair programming is the most intriguing part. Ever since man made the computer, programming was and still is a one-man activity. It's one's world where he let go of his creativity mind. Enters XP, and now we need to "share." As XP itself is extreme, this certainly bills to it. Besides the goodness of knowledge transfer, pair pressure, and confidence builder, the pair programming can also introduce bad programming habits under the name of "do it this way, it's much easier." The senior programmer may influence the junior programmer on how to "cut corners" against the standards. Under the time pressure and with the project completion due date hanging over their shoulders, this is a likely scenario. Also in [24], the authors make a point of the partner catching "the most glaring errors, the errors that anyone else can see in an instant" [23] that the other partner just can't. What if both partners can't? The pair has been "synchronized" in their programming. They have worked long hours, maybe days, together and maybe they both just can't see the glaring errors? Despite some skepticism [8], [14], [17] and abhorrer [4], some accept it well [2] and the empirical studies are promising [1], [25]. A quote from [2] states "Sometimes if you're coding alone, you end up going off on the wrong thing for a while, ..If you're 'pair programming,' that doesn't happen, or it doesn't happen for very long...As soon as one person runs out of ideas, the other person just picks up on them." The general idea of pair programming is catching on, but practitioners are seeking more guidelines in pair programming [14], [17], [27], [28] Currently, there is no explicit guideline or a set of protocol for pair programming. A quote from [17] states "Pair programming is adopted easily and an enjoyable way to code. However, it is unclear what type of work not to do in pairs and how best to structure pair interaction." Simply pairing two programmers and expecting the result is certainly not achieving. In a pair, not only programming, but also review, integration, design, and tests occur. This adds more to the pairing process and protocol.

## 4. Concluding Remarks

In search of relevant empirical evidence of the opposition views, as expected, there was no such case as "complete failure." However, as XP is comprised of many useful techniques in one, we have seen cases where each of these techniques was either only partially adopted or not adopted. There were notable comments such as "the need for documentation was ingrained in the culture, so we expected concern over XP's lack of formal documentation." [9], "Lack of planning and initial design is now being attributed to an overabundance of code rework at each new deliverable" [14], and "the highest risks seem to be in the lack of a design phase or document, and in the extreme way in which "design for today" has to be adopted." [21] Just as there is no "complete failure of XP," there is no "complete successful use of XP" either.

XP, the humanistic approach and code-centric method, is viewed by many to be in an intermediary state [3], [20]. XP is not complete, it's at its beginning hence we have a large opposition body. Nevertheless, it's refreshing and exciting for the agile software methodologies [5], which will take us through the "Internet time". The XP's ultimate goal of "do little as possible for only the required" has thrown out many of what have been the essentials of good software process practices. XP is a pro-human method that strongly believes in the human ability as the ultimate key to success [15]. Ability to make sound judgments, "use appropriately", and keenly focused on the goals are all from human's. It appears that many who have tried XP, voice that the initial implementation was very difficult. Some of XP features were adopted and some were not, but at the end all have showed a positive feedback on the XP experience [9], [10], [12], [14]. The users are already customizing accordingly and adopting selectively of the XP features [9], [12]. Despite the "extremes," XP appears to be finding a niche in the ever rapidly evolving software process ground. A quote from [16] states: "A one-size-fits-all development process does not exist. Software projects vary wildly in technology, size, complexity, risk, criticality, regulatory, and cultural constraints, and many other key variables. .... there is a sweet spot where XP will flourish…" Again, the intent of this paper is not to issue any verdict on XP by any means. The purpose is simply to shed more light on the growing concerns and oppositions to XP and this certainly must continue. There is no right or wrong in XP today. These debates continuously reveal the nuggets of XP and we can expect to enjoy the "contribution" of XP later.

## Reference

[1] Anderson, A., Bettie, R., Beck, K. et al., Chrysler goes to "Extreme" in *Distributed computing*, Oct 1998, pp. 24-28

[2] Barnes, C., More programmers going "Extreme," *CNET News.com*, April 3, 2001, http://news.com.com/2100-1040-255167.html?legacy=cnet

[3] Beck, Kent. *Extreme programming explained: embrace change*, Reading, Mass.: Addison-Wesley, 2000.

[4] "The Case Against Extreme Programming" website http://www.bad-managers.com/Features/xp/case_against_xp.shtml

[5] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2001

[6] Cohn, T.M., and Paul, R.C., "A Comparison of Requirements Engineering in Extreme Programming (XP) and Conventional Software Development Methodologies" in Seventh Americas Conference on Information Systems, 2001, pp. 1327-1331

[7] Fowler, M., "Is Design Dead?" in *Extreme Programming examined*, Succi, G., and Marchesi, M., Eds., Boston: Addison-Wesley, 2001.

[8] Glass, R.L., "Extreme programming: the good, the bad, and the bottom line" in *IEEE Software*, Vol. 18, Issue 6, Nov/Dec 2001, pp. 112 –111

[9] Grenning, J. "Launching extreme programming at a process-intensive company" in *IEEE Software*, Vol.18, Issue 6, Nov.-Dec. 2001, pp. 27 –33

[10] Haungs, J., "Pair programming on the C3 project" in *Computer*, Feb 2001, pp118-119

[11] Internet newsgroup; comp.software.extreme-programming

[12] Karlsson, E.-A.; Andersson, L.-G.; Leion, P. "Daily build and feature development in large distributed projects" in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 649 –658

[13] Keefer, G., Extreme programming considered harmful for reliable software development, 2002 http://www.avoca-vsm.com/Dateien-Download/ExtremeProgramming.pdf

[14] Kivi, J., Haydon, D., Hayes, J., Schneider, R., and Succi, G. "Extreme programming: a university team design experience" in *Electrical and Computer Engineering, 2000 Canadian Conference on*, Vol. 2, 2000, pp. 816 -820

[15] Martin, R.C. "eXtreme Programming development through dialog" in *IEEE Software*, Vol. 17, Issue 4, July-Aug. 2000, pp. 12 –13

[16] McCormick, M., "Technical opinion: Programming extremism" in Comm. of ACM. Vol. 44, Issue 6, June 2001, pp. 109-119

[17] Muller, M.M. and Tichy, W.F. "Case study: extreme programming in a university environment" in *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, 2001, pp. 537 –544

[18] Nawrocki, J., Walter, B. and Wojciechowski, A., "Toward maturity model for extreme programming" in Euromicro Conference, 2001. Proceedings. 27th , 2001, pp. 233 –239

[19] Online message forum on XP website: http://www.bad-managers.com/js.matt/com.maff.forum.ForumServlet?key=XPGen

[20] Riehle, D., "A Comparison of the value systems of adaptive software development and extreme programming: How methodologies may learn from each other" in *Extreme Programming examined*, Succi, G., and Marchesi, M., Eds., Boston: Addison-Wesley, 2001.

[21] Van Deursen, A., "A. Program comprehension risks and opportunities in extreme programming" in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, 2001, pp. 176 –185

[22] Viljamaa, J., Refactoring I – Basics and Motivation, from seminars on programming paradigms, Oct 2000, http://www.cs.helsinki.fi/u/jviljama/paradigms/paper.pdf

[23] Weinberg, G. M., *The psychology of computer programming*, Silver anniversary ed. New York : Dorset House Pub., 1998.

[24] Williams, L. A. and Kessler, R. R., "All I really need to know about pair programming I learned in kindergarten" in *Communications of the ACM,* Vol. 43, Issue 5, May 2000, pp. 108-114

[25] Williams, L., Kessler, R.R., Cunningham, W., and Jeffries, R., "Strengthening the case for pair programming" in *IEEE Software*, Vol.17, Issue 4, July-Aug. 2000, pp.19 –25

[26] XP website http://www.extremeprogramming.org

[27] Yahoo XP user group website A: http://groups.yahoo.com/group/xpusergroups/

[28] Yahoo XP user group website B: http://groups.yahoo.com/group/extremeprogramming/