Perceptions of Extreme Programming: An Exploratory Study

Vojislav B. Mišić
University of Manitoba
Wiinipeg, Manitoba, Canada
e-mail: vmisic@cs.umanitoba.ca

Abstract

Extreme Programming (XP) is probably the best known and (arguably) the most controversial of the so-called agile software development methodologies. The paper presents the main findings of a small pilot survey about user perceptions of XP.

Introduction

For many years, the preferred solution to the problems of software development was (and, for some people, still is) to search for newer technologies that should make people more productive. Such technologies include structured programming in mid-seventies [SMC74, DM78]; object-oriented technology in mid-eighties [Mey97, JCJO94]; patterns [GHJV95, Fow97], formal methods [Jon90, Win90], components [Szy98], and others in mid-nineties. All of these attempts were—at best—only partially successful: while significant progress has been made in terms of productivity, the basic problems of software development remain essentially unchallenged. According to the 2001 update to The Standish Group's CHAOS report, still more than half of all projects either fail or are not finished on time and within budget [Sta01].

Recently, support has begun to gather for an alternative approach that emphasizes the fact that software development is done by humans and, consequently, strives to tailor the development process to match the way humans work. This approach is the foundation upon which a number of so-called agile development methodologies [Lar03] have been introduced in the last decade. Among the agile methodologies, Extreme Programming is probably the best known and, arguably, the most controversial example. Extreme Programming (XP) was devised and subsequently refined as a lightweight approach to software development [BA04]. It is technology-neutral insofar as it does not prescribe the use of any particular programming language, development tool, hardware/software platform, or modeling paradigm. Instead, XP focuses on the manner in which human programmers work and collaborate, and utilizes this knowledge to organize the programming process most effectively.

XP is typically met with extreme responses. Some people—mostly practitioners—hail it as a viable solution that has the potential to cope with some, if not all, of the problems that seemed to be an inseparable component of the software development process [Coc02]. Others (again, mostly practitioners) have been staunchly opposed, considering it as little more than hacking in a clever disguise [Rak01]. A notable minority have tried to find a middle road, looking for the right balance between agility and discipline [BT04]. A number of academics have tried to investigate whether XP and other agile practices really work, and if so, under which conditions [Fow99, WK02].

A growing body of evidence suggests that these methodologies have the potential to cope with some of the problems mentioned above. However, if this potential is to be further developed, two main issues have to be addressed. From the technical viewpoint, we need to fully develop the body of best practices, procedures, rules, and guidelines – in other words, to specify the methodology in as much detail as necessary. From the social and managerial viewpoint, we need to investigate the perception of such methodologies, as well as other factors that might facilitate or impede their adoption and subsequent use in practice. Adoption or rejection of Extreme Programming depends on the perceptions and views held by software developers – practitioners as well as organizations. It is this latter view that is the focus of the work reported here.

In order to find out more about those perceptions and views, we have created a small online survey. The paper discusses, in sequence, the demographic profile of the respondents, findings on on general perception of XP, and more details on the perceptions of XP core principles and practices. Main tenets of XP are discussed in a short Appendix, followed by the survey instrument.

Demographic profile

The invitation to take the survey has been disseminated through several channels, and the introductory comments invited the addressees to fill in the survey, but also to forward it to interested colleagues, team leaders, managers, and others. The invitation resulted in a total of 85 responses: 46 were obtained through University of Manitoba; 28 were obtained from the Greater Toronto Area; and 11 more were received through an open call on a web site. The first group of responses may have included a substantial number of final year student who are currently participating, or have recently participated, in the co-op program; since the survey was anonymous, exact details about whether the respondents were students or not was not available.

Table 1: Respondents' profile: experience.

Age (years) Experience (years)	Number	Percent
less than 5	27	31.8
5 to 10	25	29.4
10 to 15	15	17.6
over 15	18	21.2

Basic information about different demographic aspects of respondents' profile is given in the following, starting with the data about respondents' age and experience given in Table 1. While

the observed sample is somewhat skewed toward younger and less experienced, a substantial fraction (more than a third) of respondents reported having over 10 years of software development experience. Furthermore, 80 respondents (94.1%) reported having programming experience in capacities such as analyst, programmer, and tester, while 41 respondents (47.7%) reported experience in some kind of group or team management role; a number of respondents have performed other tasks, e.g., architecture design and customer liaison.

Table 2: Respondents' profile: maximum team size in which the respondents have participated.

Maximum team size	Number	Percent
Up to 5 persons	33	38.8
5 to 10 persons	16	18.8
10 to 20 persons	17	20.0
more than 20 persons	16	18.8

As shown in Table 2, almost half of the respondents have had experience with small teams (up to five persons) only, while the other half of the answers are evenly divided between the other options offered. Such distribution could have been expected, given the distribution of age and experience of the respondents' sample.

We have also asked about the respondents' experience with XP or other agile methodologies. About 60.5% of the respondents have used XP to some extent, with about 16.5% of them reporting to have extensive experience with it; slightly less than one-quarter of respondents have no experience with XP; and a small number of answers indicate experience with other agile methodologies. (We did not ask which methodologies in particular, since the survey was focused on XP.) Although our sample is limited in size and, possibly, somewhat biased toward XP users, these numbers still indicate that many developers are using, or at least experimenting with, XP as a viable alternative to the traditional approach to software development.

Table 3: Respondents' profile: experience with XP.

Type of experience	Number	Percent
no experience with XP	20	23.5
some experience with XP	46	54.1
extensive experience with XP	14	16.5
other agile methodologies	5	5.9

Altogether, it appears that the respondents' sample was fairly wide in scope, despite its relatively small size.

General perceptions of XP

The second group of questions tried to elicit the respondents' views on XP in general through a list of questions, labeled G.1 to G.8, given in the Appendix. For all of those questions, answers were offered ranging from 1 (corresponding to 'strongly disagree') to 5 (corresponding to 'strongly agree'), with 3 corresponding to 'neutral'.

The questions and the descriptive statistics of the answers, together with the results of the one-sample t-test against the 'neutral' value of 3, are shown in Table 4. Some, but not all, of the mean values deviated significantly from the value of 3, with $\alpha \leq 0.05$. The respondents feel that XP gives results and that it does not require special people to succeed, which is encouraging for the proponents of XP. They also feel that XP must be adapted to the local environment and that not all practices in XP are equally useful. The survey did include questions about the relevance of specific practices; the answers are discussed in the next section. It is interesting (but not entirely unexpected) that developers seem to like XP because it originates from practice; this point is elaborated in more detail below.

Furthermore, the respondents did not feel that the switch to XP is conditioned on customers' acceptance, nor that it is conditioned on the management approval either. These are important findings which should be taken into account when constructing the theoretical model for the acceptance of XP and, possibly, other agile methodologies as well.

In order to identify significant differences within the respondents' sample, we have used the independent sample t-test and ANOVA with corresponding demographic variables as independent factors. Only a couple of such cases were found, as shown in Table 5.

We have also plotted the mean values within groups, for a number of independent factors. Some of those plots, with age and experience with XP as independent variables, are shown in Fig. 1, from which a number of observations may be made.

First, some of the answers demonstrate a clear trend with increasing respondents' age, as shown in Fig. 1(a).

- Regarding the answers to the question G.3: although the difference in the means is not high, it is clear that the perception of 'radicalness' of change required by the switch to XP does decrease with respondents' age. This may come as a surprise, as the common perception is that older people are more conservative than the young, and hence would be inclined to perceive the same changes as more radical.

- A similar trend may be observed for the answers to the question G.7, where older respondents are found more likely to subscribe to the view of XP being popular on account of its origins in practice, rather than in academia (although this alternative was not explicitly mentioned). This trend may perhaps be attributed to the multitude of methodologies that have been proposed over the years, but failed to lead to significant improvements in the software development process [Wie98]. As most of those methodologies originate in academic research, older respondents may have developed a lack of trust in the proposals that come from academia. Instead, they tend to give preference to the solutions developed 'from the trenches'.

- Finally, this trend is most pronounced for the answers to

Table 4: General perceptions of XP.

Number	Question	Number of responses	Mean value	Standard deviation	Significance (α)
G.1	Management would switch to XP if the customers would accept it.	78	3.06	1.52	.711
G.2	Developers would switch to XP if the management would allow it.	72	3.18	1.24	.220
G.3	XP requires too radical a change.	72	2.83	1.18	.233
G.4	XP gives results.	75	3.89	1.38	.000*
G.5	Not all practices in XP are useful.	78	3.58	1.36	.000*
G.6	To be successful, XP needs to be adapted to local environment.	75	3.71	1.42	.000*
G.7	Developers like XP because it originates from practice.	72	3.47	1.20	.001*
G.8	XP requires special people to succeed.	72	2.56	1.17	.002*

Note: Asterisks denote significance values below $\alpha = 0.05$.

Table 5: Significant differences within the sample.

Number	Question	Factor	Significance (α)
G.1	Management would switch to XP if the customers would accept it.	age	.037*
G.4	XP gives results.	experience with XP	.001*
G.5	Not all practices in XP are useful.	age	.023*
G.5	Not all practices in XP are useful.	maximum team size participated in	.030*

Note: Asterisks denote significance values below $\alpha = 0.05$.

question G.5; note that this is the only question among those shown on the plot in Fig. 1(a) for which the differences between groups are statistically significant at $\alpha \leq 0.05$.

While the sample size is too small for any definitive conclusion to be drawn, the existence of clear trends is certainly interesting and deserves further research attention.

Second, an even more interesting observation stems from the means plots in Fig. 1(b). Informally, some of the questions from this group—G.3, G.4, G.6, G.8, and even G.7, albeit to a somewhat lesser extent—may be interpreted as a way to measure the respondents' 'empathy' towards XP. For most people, this 'empathy' increases with experience: the more people practice XP – or other agile methodologies, for that matter – the more they will tend to look favorably at it. Therefore, it might be expected that the group of respondents with no experience with XP should exhibit the lowest degree of 'empathy' towards XP. On the other hand, the highest degree of 'empathy' should be observed in the group of respondents with extensive experience with XP, but also in the group of respondents with experience in other agile methods.

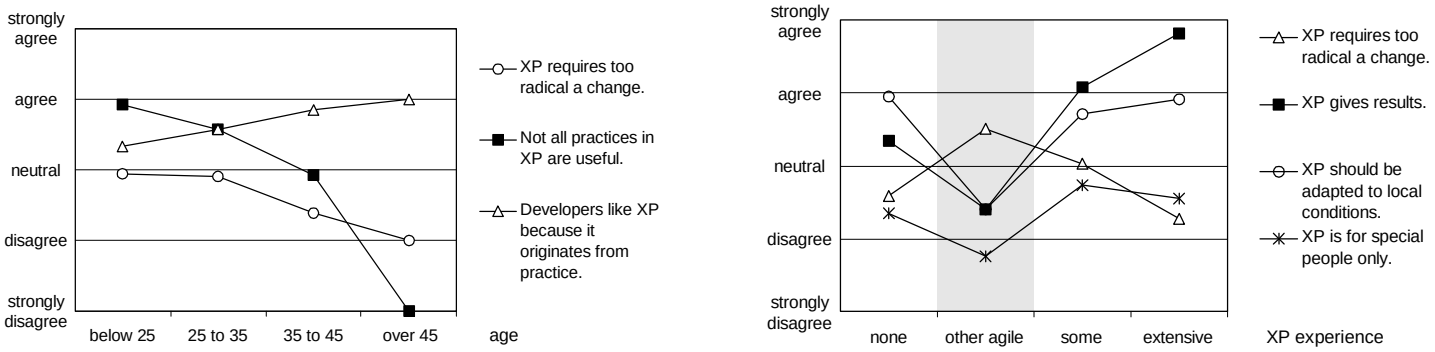
However, the data depicted in Fig. 1(b) paints an entirely different picture – the least favorable answers were given by the group of respondents who have experience with *other* agile methods. This holds for all of the above questions: G.4, G.6, G.7, *and* G.3.

One possible explanation for this counter-intuitive observation might be as follows. Namely, most other agile methodologies exhibit a more disciplined approach to software development than XP [Lar03]. Being well known to their practitioners, those methodologies also appear well ordered as well as effective and efficient. To them, the seemingly chaotic and orderless nature of XP makes it appear too radical, too difficult for normal people, and unlikely to give results, even when adapted to local conditions. On the other hand, people with no experience in XP or other agile methods may be more willing to experiment, if only out of curiosity.

An alternative explanation might be that the results are the consequence of the small size of the observed group – only five respondents out of 85 have reported experience with other agile methods, as shown in Table 3. But in either case, this observation probably deserves to be investigated in more detail.

XP core values and practices

The final part of the pilot study were four groups of questions that focused on XP core values and practices. For each of the four core values and twelve practices, respondents were asked to



(a) Grouping factor: age.

(b) Grouping factor: experience with XP.

Figure 1: Means plots for selected questions about general perception of XP.

assess its relevance to successful software development, as well as the degree of ease with which it can be followed and/or achieved in the actual development process.

Relevance The results can be summarized as follows. Regarding relevance, all four principles were found to be relevant through the t-test (against the value of 3 corresponding to 'neutral'), at the significance level of $\alpha \leq 0.05$.

Table 6: Relevance of XP core principles and practices.

core principle or practice	Relevance		
	Mean	SDev	α
communication	4.79	.412	.000*
embrace change	4.02	.874	.000*
feedback	4.54	.576	.000*
simplicity	4.08	.788	.000*
collective ownership	3.96	.727	.000*
coding standards	4.42	.785	.000*
continuous integration	4.36	.776	.000*
customer tests	4.28	1.11	.000*
metaphor	3.00	1.28	1.00
pair programming	4.02	.869	.000*
planning game	3.62	1.35	.002*
refactoring	4.36	.827	.000*
simple design	4.58	.538	.000*
small releases	4.06	1.04	.000*
sustained pace	4.24	.916	.000*
test-driven dev't	4.52	.814	.000*

Note: Asterisks denote $\alpha \leq 0.05$.

Communication and feedback were deemed more relevant, with mean values of 4.79 and 4.54, and standard deviations of 0.412 and 0.576, respectively. (The other two principles, simplicity and embracing change, were also found relevant but with slightly lower mean values.) It may be worth noting that both principles are predominantly of psycho-social nature, with little or no dependence on technology. This observation might be of interest to the designers of software engineering curricula where the emphasis is traditionally put on technology-related issues; perhaps more benefits would be obtained by emphasizing social and communication skills necessary for successful software development.

Similarly, most of the practices were found relevant. The practices of simple design and test-driven development led the way with mean values over 4.50. The only exception was metaphor, for which the mean was almost exactly 3; it is therefore not deemed (particularly) relevant, but it is not deemed irrelevant either.

The relevance of the practice of planning game had the second lowest mean value of only 3.62, which was somewhat unexpected. (Note that even this mean value is significantly different from the 'neutral' value of 3.) One possible explanation is that small team sizes tend to favor of frequent re-adjustments on the fly over formal planning sessions – even though the XP planning game is not very formal at all. Another explanation might be that the results are skewed due to the small sample size. Either way, this result deserves further investigation.

Difficulty Regarding the degree with which the core principles of XP are easy to achieve in practice, feedback and communication were deemed slightly easier to achieve than the other two.

Perhaps surprisingly, none of the practices were deemed too difficult to achieve: the highest mean was only 3.33, quite close to the 'neutral' value of 3. Simple design was deemed the most difficult, closely followed by pair programming, customer tests, and collective code ownership. Note that the last two practices are of predominantly social, or social and management-related categories, rather than technical; pair programming is a psycho-social category with some technical content; and simple design, although nominally of technical nature, could also be classified as a design category. Together, these results might be interpreted to mean that the four most difficult practices are only marginally technical in nature, which seems to support the observation on social-vs.-technical skills made earlier.

Interestingly enough, simple design was the most useful and the most difficult practice to achieve. In a certain way, this result complies well with the emergent design paradigm advocated by the authors of XP [BA04]. It may also mean that the results

Table 7: Difficulty of achieving XP core values and practices.

core principle or practice	Difficulty		
	Mean	SDev	α
communication	3.00	.863	1.00
embrace change	3.23	.831	.051
feedback	2.79	.800	.062
simplicity	3.33	.810	.005*
collective ownership	3.10	.863	.417
coding standards	2.50	.735	.000*
continuous integration	2.82	.873	.151
customer tests	3.08	.966	.561
metaphor	2.58	1.20	.017*
pair programming	3.18	.941	.182
planning game	2.48	.974	.000*
refactoring	2.90	.814	.389
simple design	3.32	.844	.010*
small releases	2.80	.833	.096
sustained pace	2.98	1.02	.890
test-driven dev't	3.02	1.00	.888

Note: Asterisks denote $\alpha \leq 0.05$.

obtained with this methodology might be improved (and, by extension, the overall methodology might become more successful) by placing more emphasis on design-related issues [Amb02].

Among the core practices, planning game was deemed the easiest one to achieve. This may be attributed to the simplicity of the exercise and the short time it requires. The second easiest practice was adherence to coding standards, possibly because of the availability of automated tools which may provide the necessary support.

On Metaphor Metaphor was found to be the third least difficult practice to achieve; note that it was also found to be not relevant. These findings might be explained as follows. Since metaphor is predominantly a design category, metaphor-related decisions are likely to be made by the persons with more experience and perhaps more seniority (programmers, architects, or interface designers). Younger and less experienced developers have fewer opportunities to participate in this process; in fact, metaphor-related decisions may often be simply handed over to them, ready-made and immutable. Consequently, younger developers may be inclined to equate the lack of participation with the lack of relevance, and to interpret the ease with which they were able to *obtain* metaphor-related decisions as the ease of actually *making* them. Note that our respondents' sample is somewhat skewed toward the younger and less experienced.

While this explanation is somewhat speculative, it is well known that design issues, in general, are difficult to master [Amb02]. Furthermore, younger developers often lack perspective and find it difficult to grasp the 'big picture', both with regard to the particular project and with regard to the software development process in general.

On a less abstract level, metaphor is intimately related to the user interface and its capabilities and conventions. In this case, the number of choices is effectively limited by the prevalence of graphical user interfaces such as Windows, MacOS, or various flavors of Linux GUI. Since there are fewer choices to be made, metaphor is seen as an easy and, thus, less important practice. Still, other explanations, including the possibility that the results are inaccurate due to the small sample size, cannot be ruled out.

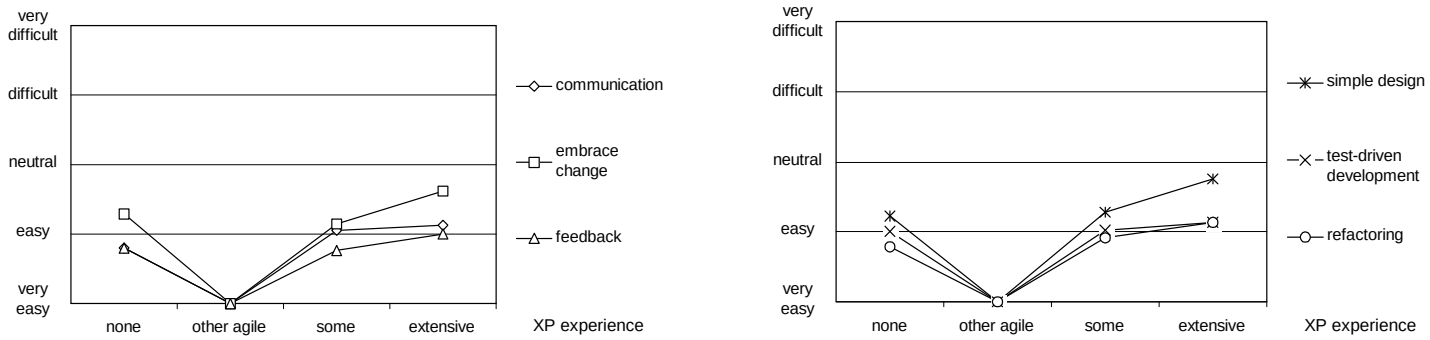
Experience with XP vs. difficulty of principles and practices The ANOVA procedure was applied to the relevance and difficulty of XP core principles and practices, using experience with XP used as the grouping factor. The differences between groups were found to be statistically significant in a small number of cases only, which is why we don't show them here. However, more discrepancies regarding the group of respondents that have experience with other agile methods were revealed, as shown in Fig. 2. That group has rated the principles and practices depicted in the diagrams as very easy to achieve, noticeably different form other groups. In view of similar observations discussed above, it is tempting to conclude that experience with other agile methods does affect the developers' opinions about XP. More work is needed to explain this issue in greater depth.

On customer involvement Customer involvement is known to be an important success factor, the lack of which is often cited as one of the most prominent reasons for project failure [Sta01]. As the survey focused exclusively on perceptions of XP, it did not attempt to directly assess the degree of customer involvement. However, several informal comments did indicate that customers do not always provide the level of support required by XP, which may critically affect the outcome of the project. Note also that the XP principles of communication and feedback heavily rely on customer interaction. Moreover, the practice of customers writing acceptance tests, or at least participating in this writing, is one aspect of customer involvement which is specific to XP. These and other issues related to customer involvement certainly deserve more attention in future studies.

Overall, the results of the study shows the authors of Extreme Programming were quite successful in pinpointing the main practices that might improve the quality and timeliness of the software development process. While all of the practices prescribed by XP are perceived as reasonably easy to achieve, it is worth noting that the most difficult ones are, in general, those with heavy social and management orientation, rather than those with dominant technology focus. This observation may serve as a useful reminder to managers and educators that software development, being a creative and labor-intensive process, must focus on human characteristics; technology, while necessary, nonetheless plays a subordinate, perhaps more of a supportive, role in this process.

What next?

From the methodological point of view, the study shows the authors of Extreme Programming were quite successful in pinpoint-



(a) Difficulty of selected principles.

(b) Difficulty of selected practices.

Figure 2: Means plots, grouping by experience with XP.

ing the main practices that might improve the quality and timeliness of the software development process. While all of the practices prescribed by XP are perceived as comparatively easy to achieve, it is worth noting that the most difficult ones are, in fact, those that depend the least on technology. Instead, they are either social and management-oriented, or focus on design-related issues.

This observation may serve as a useful reminder to managers and educators that software development, being a creative and labor-intensive process, must focus on human characteristics; technology, while necessary, nonetheless plays a subordinate, perhaps more of a supportive, role in this process.

Attention should also be given to some of the points highlighted in the study, most notably the perception of XP among practitioners of other agile technologies, in order to gain more insight into the mechanics of adoption of modern development methodologies.

Overall, the results of this study give valuable input (or should we say feedback?) about the perceptions of Extreme Programming. This should facilitate the development of a full fledged model to describe and predict the acceptance of Extreme Programming among practitioners, as well as the ways and approaches in which this acceptance might be facilitated.

References

[Amb02] Scott Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, New York, NY, 2002.

[BA04] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Boston, MA, 2nd edition, 2004.

[BT04] Barry Boehm and Richard Turner. *Balancing Agility and Discipline – a Guide for the Perplexed*. Addison-Wesley, Boston, MA, 2004.

[Coc02] Alistair Cockburn. *Agile Software Development*. Addison-Wesley, Boston, MA, 2002.

[DM78] Tom De Marco. *Structured Analysis and System Specification*. Yourdon Press, Englewood Cliffs, NJ, 1978.

[Fow97] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison Wesley Longman, Menlo Park, CA, 1997.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Menlo Park, CA, 1999.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Reading, MA, 1995.

[JCJO94] Ivar Jacobson, Magnus Christenston, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering*. ACM Press and Addison-Wesley, New York, 1994.

[Jon90] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, Hemel Hempstead, UK, second edition, 1990.

[Lar03] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, Menlo Park, CA, 2003.

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1997.

[Rak01] Steven R. Rakitin. Manifesto elicits cynicism. *Letters, Computer*, 34(12):4, December 2001.

[SMC74] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.

[Sta01] Extreme CHAOS. the 2001 update to the chaos report, The Standish Group, West Yarmouth, MA, 2001.

[Szy98] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Harlow, UK, 1998.

[Wie98] Karl E. Wiegers. Read my lips: No new models. *IEEE Software*, 15(6):10–13, September/October 1998.

[Win90] Jeannette M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–24, September 1990.

[WK02] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Professional, Menlo Park, CA, 2002.

On Extreme Programming

XP is based on the following core principles or values.

Precise, constructive and timely **communication** helps everyone understand what the project is about, what are its expected results, and how to proceed in order to obtain those results. This principle emphasizes the fact that software development is a labor-intensive process performed by teams of collaborating humans, and communication allows those teams to function effectively and efficiently.

Change is an inevitable part of software development. Anything can change – requirements, priorities, and designs alike – and many will change during the development of a single project. Therefore, the development process should be able to **embrace change**, i.e., accept changes, accommodate them, and still obtain the best possible results.

Feedback allows the communication to develop its full potential, in particular with respect to changes in requirements, priorities, designs, and other conditions that may affect the course of actions.

The goal is to find the simplest solution that can possibly work; in this manner, the impact of changes, and (in particular) the effort needed to cope with them, is thus minimized. Thus, **simplicity** actually translates into efficiency, but also into effectiveness, as the process is constantly reoriented toward the desired goal.

These principles or values are the basis for a number of practices, the most popular set of which are listed below (in alphabetical order).

1. Collective code ownership means that all members of the team should view the entire production and test code as their own. This feeling is facilitated by the absence of predefined roles within the team, as every pair can test, modify, and/or update any piece of code at any time.

2. Common coding standards should be adhered to by all members of the team, thus supporting collective code ownership and promoting understandability and maintainability.

3. Continuous integration means that a functional production system is available at (almost) all times. This improves testability of the code and helps catch errors faster; moreover, a working version is always available for inspection by, and/or delivery to, the customer.

4. Customer tests refer to acceptance tests which (ideally) should be written by the customers, or at least with direct customer involvement. (Other tests are written by the programming team.) Acceptance tests directly correspond to the requirements, and thus ultimately determine the success or failure of the project.

5. Metaphor refers to the common vision of the application and the way it operates; this vision evolves and should be embraced by the entire team, so as to improve the understanding of the project and maximize team productivity.

6. Pair programming means that all production code is (or should be) built by teams of two programmers working side by side at the same computer; one of them actually types at the keyboard while the other provides feedback. In this manner, production code is subject to continuous inspection, which should lead to improved quality and productivity.

7. Planning game refers to the procedure of creating short-term plans with heavy customer involvement, with the primary goal of delivering value to the customers while keeping the project manageable. The procedure itself is deliberately kept simple, thus allowing transparent and flexible handling of changing requirements and task priorities.

8. Refactoring refers to the process of frequent rearranging and restructuring production code without changing its externally observable functionality. These improvements facilitate understandability and maintainability of the design later on in the product lifecycle [Fow99].

9. Simple design is meant to facilitate the overall development process. By avoiding big design up front, the programming team can immediately start delivering functionality, while keeping the design flexible enough to cater for changing requirements.

10. Small (and, therefore, frequent) releases make it easier to cope with changing requirements and priorities while delivering software of increasing value to the customer; at the same time, they help keep the project under control.

11. Sustainable pace is required so that individual team members can always give their best in the long run. While occasional overtime cannot be avoided, it should not happen too often, as the overall work satisfaction and, by extension, productivity will suffer.

12. Test-driven development requires that tests are written before the code they are intended to test. The test suite evolves together with the production code, which is simply not allowed to deviate too much from the behavior prescribed by the specifications.

We note that different authors give somewhat different sets of values and practices; the list given above is a common denominator for most of those sets.

Survey questions

Questions about demographic profile were:

D.1 How much software development experience you have?

D.2 What kind of activities have you performed in software development?

D.3 What was the largest development team you have participated in?

D.4 What was the largest development team you have managed?

D.5 Have you had experience with XP?

Questions about general perception of XP were:

G.1 Management would switch to XP if the customers would accept it.

G.2 Developers would switch to XP if the management would allow it.

G.3 XP requires too radical a change.

G.4 XP gives good results.

G.5 Not all practices in XP are useful.

G.6 To be successful, XP needs to be adapted to local environment.

G.7 Developers like XP because it originates from practice.

G.8 XP requires special people to succeed.

The final group of questions focused on XP core values and practices. For each of the four core values and twelve practices outlined in Section , respondents were asked to assess its relevance to software development and the degree of ease (or difficulty) with which it can be achieved in practice.

For the last two groups of questions, the answers offered were using a Likert scale from 1 to 5, with 3 corresponding to ‘neutral’.