XP: Taking the psychology of programming to the eXtreme

Sallyann Bryant
IDEAS Laboratory
University of Sussex
S.Bryant@sussex.ac.uk

Keywords: Extreme Programming, XP, pair programming, metaphor, external representation

Abstract

Extreme Programming (XP) is a software development methodology which is growing in popularity and commercial use. Despite a number of published experience reports and a small number of studies, predominantly in an academic environment, our knowledge about how and why some aspects of it work is still in its infancy.

One major limitation of many of these studies is a failure to question why the practices of XP appear to work or fail. This paper reviews the research on Extreme Programming and suggests further work is required in order to ascertain how these practices fit into the framework of existing knowledge on the psychological aspects of programming.

Introduction

Much progress has been made in terms of our knowledge and understanding of the manner in which computer systems are both produced and used. Studies of subjects ranging from the use of metaphor in user interface design (Marcus, 1997) to the implications of collaborative systems (Crabtree, 2003) and from debugging Java programs (Romero et al. 2002) to using external representations in systems design (Petre, 2003) have allowed researchers to start to uncover the psychological aspects and processes underlying our behaviours when working on or with IT.

Relatively recently, Extreme Programming has been introduced as a new way of approaching the production of software. Extreme Programming is a form of 'agile development', defined by the Agile Alliance Manifesto (2004) as valuing:

> Individuals and interactions over processes and tools
>
> Working software over comprehensive documentation
>
> Customer collaboration over contract negotiation
>
> Responding to change over following a plan.

Beck (2000), who coined the phrase Extreme Programming, explains that XP consists of a set of 12 practices. Whilst some of these (e.g. the 40 hour week) are self-explanatory, and others (e.g. the planning game) are outside of the scope of this paper, the overall use of external representations, pair programming and system metaphor will be considered in more detail in terms of the studies which have taken place and the links they have to current knowledge regarding the psychology of programming.

External representations

Extreme Programming requires a development project to be defined as a number of 'stories'. These stories are written on a set of story cards, each of which is prioritised and those selected for development in the next iteration are split into tasks, each detailed on a task card. Both story cards and task cards typically consist of nothing more than a single sentence written on a ‘cue card’ and are discarded once they have been developed and integrated into the system.

Beyond these cards information regarding the system is communicated in two other ways, verbally via a system metaphor and/or representationally via the code itself. Thus the only three types of physical external representation prescribed in Extreme Programming are: story cards, task cards and the program code. This is in direct contrast with traditional system design, where a number of representations of the system design, such as structured diagrams of various kinds, are produced and maintained throughout the project, and indeed are in many cases retained after the system has ‘gone live’.

The projected benefits of the traditional structured diagrams include their helpfulness in seeing the ‘bigger picture’, providing a means of representing each level of abstraction in top-down design, structuring and helping to understand the problem, reducing cognitive load on the designer/developer, providing a common grammar between the developers and the customer in which to communicate the solution detail, allowing the representation to form a ‘contract’ between the user and the project about what will be produced and ensuring that a co-ordinated approach is taken by developers on large projects. However, Extreme Programming asserts that coding is an evolutionary endeavour and requirements usually change long before development is complete, thus rendering other external representations obsolete and costly to maintain. An XP approach would therefore seem to assume that between them the cards and code articulate the system at the only two necessary levels of abstraction, and that there is no need for the overhead of maintaining additional representations.

This poses a number of questions, two of which will be considered here: First, how do developers define and understand problems and produce solutions using only the code as representation? Second how can the user and programmers communicate about requirements? In short, how does Extreme Programming allow the project team to “ensure accurate and effective communication regarding a product that no-one can see” (Perry et al. 1994) without the use of diagrams?

Whilst a number of overviews of the use of external representations have been published (e.g. Scaife & Rogers, 1996) which are both insightful, thorough and useful, as far as the author is aware use of external representations within Extreme Programming has not been specifically studied so far. This might be because the adoption of particular forms of structured diagrams or alternative articulations of the system is not encouraged. Further studies are required to ascertain the extent to which formal and informal representations are actually used on extreme projects. The use of informal external representations may also prove a key factor in the successful communication of ideas and solving of complex problems within a programming pair and external representation of the system metaphor (see below), albeit informal and temporary, may help to provide a method of assisting in its successful use. Should alternative representations be found absent, the production and use of story and task cards are in themselves external representations, whose form, production and usage should warrant further investigation in order to assess how and when they assist the cognitive processes involved in the production of software which conforms to the users requirements.

Some interesting progress can be made in understanding the use of external representations in XP by considering the relevance of publications on the use of external representation in other areas of software development, then applying them to the Extreme Programming context. However there is a need for some more specific studies to take place. In particular, studies identifying the extent to which the external representations which are currently used may allow insight into the cognitive aspects of whether the identified, or indeed further, external representation might be desirable and why. This information could prove valuable not only in furthering our understanding of the usefulness and applicability of XP, but also in making decisions about the integration of extreme practices into more traditional approaches to software development.

Metaphor as architecture

Extreme Programming assumes the use of a system metaphor as a replacement for a more formal system architecture. This metaphor acts as a shared informal model. It should ideally take the form of a 'true metaphor', not directly related to the problem domain, for example using 'cookie cutter' to refer to instantiating an object. However, where this is not possible, the metaphor may be transferred directly from problem domain to programming domain. Such metaphors are known as 'naïve metaphors'. A common example would be the use of the term 'customer' to refer to something in both the real world and the system. For further examples of potential systems metaphors see Wake (2002).

Rather than being physically expressed, metaphors are used mainly to 'talk about' the system. In practice, the use of metaphor has been found to be sparse and often problematic as shown in the studies outlined in Figure 1. Only three of the thirteen studies considered lead to positive results regarding continued use of metaphors on all projects. The overwhelming majority of studies found problems either in the understanding or use of system metaphors. Nevertheless empirical work showing that experts maintain a mental model of the design in progress (e.g. Adelson & Soloway, 1988) would lead us to assume that the transformation of this working model into an expressible metaphor could prove invaluable in assisting collaboration on system development tasks. In addition, the many success stories in other arenas such as analogical problem solving (Gick & Holyoak, 1985) and user interface design (Preece et al., 1994) suggests that the appropriate use of true metaphor is highly desirable, especially when that use provides a means of exploring the problem space and applying knowledge from other areas to the problem at hand. In fact, Carrol and Thomas (1982) provide us with a scenario by which the use of metaphor may be understood at a cognitive level whereby the metaphor is entered into working memory, as a consequence of which general knowledge is retrieved from long term memory and the limited capacity of working memory forces consolidation of the two by creating associations.

Whilst in Extreme Programming system metaphors are not typically represented in an external form beyond verbalization, there may be some call to encourage doing so. External representations have been shown to lessen cognitive load on the designer (Suwa & Tverksy, 2002) and assist in managing complexity (Dogan & Nersessian, 2002). Similarly, Scaife & Rogers (1996) state that systems design is better and faster with diagrams. Perhaps even a simple representation of the system metaphor in a prominent place in the project space may be enough to encourage its consistent use and ensure improved co-ordination of development effort. The study by West (2002) in Figure 1 suggests the use of metaphor 'cartoons'. However to be consistent with the ethos of XP, care should be taken that the metaphor is not so complex that significant effort is required to keep the representation in sync with the evolving code. Therefore, any cartoon produced should be simple enough to avoid on-going modifications.

Despite being considered a 'last resort' in Extreme Programming literature, current research seems to show that naïve metaphors – those taken directly from the problem domain - are more often used than true metaphors. One theory might be that because metaphor use and its benefits are poorly understood it is hard to justify the investment required to come up with a relevant and suitable metaphor.

Whilst this is undesirable, in that it may more easily obscure differences in assumed knowledge or discourage full exploration of the problem domain beyond current practices, it may help to explain the lack of external metaphor representation. It is possible that this highlights a dependency between the form of metaphor articulation and the type of metaphor. As Hutchins (1995) questioned whether mental models are still required when an actor is directly interacting with the environment, so naïve metaphors with existing real world manifestations may lessen the importance of producing a physical metaphor representation. For example, why have a diagram of a customer, when you can interact with a real one on your very project team?!

Nevertheless encouraging the production of true metaphor and manifesting it using a physical representation may still help promote its production and adoption. This concept is similar to that of ‘focal images’, which in studies by Petre (2003) were seen to be both a useful and important aid in system design.

Study	Setting	Type	Size	For	Against	Outcome
West, D (2002)	Academic	Observation and presentations	>15 courses	Recall tripled. High design convergence.		Retained - use cartoons
Harrison (2003)	Commercial	Interviews	6 projects of < 20 people		Not used. No-one interested	Rejected
Rumpe, Schroder (2002)	Commercial	Survey	45 people		40% didn't use 68.9% had difficulties	N/A
Deias R (2002)	Commercial	Experience report	2 projects of 6/7 people		Could not make it work. Did not understand it.	Sketch architecture instead
Karlstrom D (2002)	Commercial / Academic	Experience report	1 project of 9 people		Too detailed & not properly maintained	Use simpler metaphor or cards & code instead
Lappo P (2002)	Academic	Experience report	1 course	More relevant with more experience.	No-one really understood it.	N/A
Tomayko (2003)	Academic	Observation and interviews	1 course of 35 students	Not very costly. Many thought helpful	Not considered very useful for any purpose	Students recommend further use
Herbsleb et al (2003)	Academic	Observation and interviews	1 course of 35 students	Little difficulty developing metaphors	Evidence of metaphor in 6 of 14 architectures. Relatively useless.	N/A
Sharp H & Robinson H (2003)	Commercial	Ethnographic	1 team of 10	Considered fundamental for shared vision	Metaphor integrity 'onerous but important'	Not stated
Lippert M et al (2003)	Commercial	Experience report	1 project (size unknown)	Use accepted	Finding the right metaphor is demanding	Use accepted
Johnson et al (2003)	Commercial	Experience report	1 project - team of 4		Used existing system instead	Naïve metaphor
Ambu et al (2003)	Commercial	Experience report	2 projects - one team of 10/one unspecified		Metaphor unused and hard to apply.	Metaphor not used
Becker et al (2003)	Academic	Experience report	3 projects of 8-12 students	Believe metaphors important	System too small - unnecessary	Not used on small projects

Fig 1 Studies of the use of metaphor in Extreme Programming

The studies considered seem to suggest that metaphors in the XP sense are rarely used, and when they are their use is fraught with difficulties. Similarly, literature on Extreme Programming assumes that the production of diagrammatical representations of the systems architecture, are an un-necessary overhead. This is in direct contradiction with literature on the psychology of programming, which has consistently found both metaphors and diagrams to be useful in problem understanding, representation and solving (e.g. Suwa & Tversky, 2002, Carrol & Thomas, 1982). As West (2002) mentions and the studies shown highlight, the system metaphor is currently considered the weak point of Extreme Programming. Perhaps with a more in depth investigation into its effective use this could be reversed.

Pair programming

Extreme Programming advocates programming in pairs. Once a task has been allocated, a programming pair proceed to develop together, taking it in turns to 'steer' at the terminal. Pairs are dynamic and can - in fact many say should - change between tasks in order to maximize the spread of knowledge about the system. As shown in Figure 2, the majority of studies which have taken place have either been in an academic environment or have provided experience reports from practitioners. The single ethnographic study (Sharp & Robinson, 2003) provides an insightful story of XP in a commercial environment, but does not assess pair programming from a cognitive perspective. Whilst studies have compared pair programming favourably with programming alone in terms of quality of software produced and side effects such as decreased 'tunnel vision' and positive 'pair pressure' (e.g. Williams et al. 2000), the addition of an extra programmer to review and make suggestions may lead to a number of problems.

Figure 2 also highlights some apparently contradictory results regarding the extent to which programmers enjoyed working in pairs. Whilst four of the studies found programmers enjoyed pairing, six other studies found either the opposite or a preference for selective use of pairing. The study by Dick & Zarnett (2002) showed that care is required regarding the suitability of developers to this approach. On an example project, they saw that the necessary role changes sometimes did not happen, which lead to one developer 'driving' all the time, and the other one drifting off. They therefore suggest that during interview, development candidates are assessed on the basis of communication, comfort in pair working, confidence and ability to compromise but without much evidence into the relevance of these particular characteristics to pair programming.

The author's own initial observation of just two pairs programming highlighted two very different approaches to pairing: a co-worker model and an apprenticeship model. This apprenticeship model accords well with suggestions by Williams et al. (2000) that pair programming may help provide an apprenticeship environment within a community of practice (Lave & Wenger, 1991). Here the apprentice learns through legitimate peripheral participation, that is, not only through 'doing' themselves, but also through 'seeing' those with more expertise perform some more advanced tasks in the surrounding area. Four of the studies in Figure 2 suggest pairing provides a good learning environment, however, this must be offset against the fact that during the apprenticeship period, particularly early on, there is only one experienced programmer working on the development, which may decrease the benefits of pair programming as, for example, the quality of code produced may not be suitably reviewed.

Thought also needs to be given to the effect of the apprenticeship on development time where deadlines are tight. In fact, the study by Ambu & Gianneschi (2003) found that pair programming was often seen as impractical when pressing deadlines loomed.

The same study raises a similar issue regarding training experts in Extreme Programming through its finding that programmers were sometime reluctant to pair once they were competent. As Dick & Zarnett (2002) mention, such a culture change may be more suited to individuals with particular working styles than others. However, as far the author knows of no empirical studies showing how this might be ascertained. That is, what characteristics or behaviours are found in successful pair programmers, which are not present in those who are less successful at pair programming.

Study	Setting	Type	Size	For	Against	Outcome
Williams et al (2000)	Academic	Timesheets, auto test	28 pairers 13 solo	40%+ faster. 96% enjoyed	15-60% slower	Benefits outweigh costs
Noll et al (2003)	Academic	Observation	4 teams of 6-8		Hoarded	Little difference
Harrison (2003)	Commercial	Interviews	6 projects		Not practical – remote	Voluntary
Benedicenti & Paranjape (2001)	Commercial & academic	Experience	12 participants	Morale & integration.		Not mandatory
Dick & Zarnett (2002)	Commercial	Experience			Not all are suited	Phase in. Interview traits.
Macias et al (2002)	Academic	Experience	80 students	Higher quality		XP higher quality
Rumpe (2002)	Commercial	Survey	45 developers		Some refused	93.3% use again
Deias et al (2002)	Commercial	Experience	15 people	Disciplined		Developers only
Pulugurtha et al (2002	Commercial	Experience	?	Improved comms.		Pair on other tasks too
Lappo (2002)	Academic	Experience	?		None took to it	
Sharp & Robinson (2003)	Commercial	Ethnographic study	8 developers	All paired. More comms		'Exposed pair' to manage comms
Tessem (2003)	Academic	Observation/ survey	6 developers	All positive. More quality	Exhausting. Communication	Frequent partner changes
Jensen & Zilmer (2003)	Commercial	Experience	1 project - 400 people		Less value if don't re-pair	
Heiberg et al (2003)	Academic	Experiment	110 students			No correlation NEO PI traits
Lui & Chan (2003)	Commercial	Experiment	15 developers	Helped new problems	No help on old problems	Work at rate of 'smarter guy'
Vanhanen et al (2003)	Commercial	Interviews	3 project		One project, used for debug	
Fuqua & Hammer (2003)	Commercial	Experience	2-4 programmers, 40 iterations	New members up to speed fast		
Johnson et al (2003)	Commercial	Experience	1 project (team of 4)	Behaviour improved		
Johnson & Johnson (2003)	Academic	Experience	?	> interaction Role models		
Ambu & Gianneschi (2003)	Commercial	Experience	10 programmers	Faster. Less bottlenecks.	Reluctant if Competent or tight deadlines.	
Becker-Pechau et al (2003)	Academic	Experience	3 teams of 12-18 students	Better integrated	Changing pairs slowed down	Use on critical tasks
Steimann et al (2003)	Academic	Experience	?		Only 1/3 of work in pairs	Alternative to co-location
Gittins et al (2001)	Commercial	Observation, interviews & survey	?	Did not pair all the time	Very taxing	
Cockburn & Williams (2001)	Academic	Interviews & experiments	?	Better speed quality & enjoyment	Development cost increase 15%	Phase pairing in

Figure 2 Studies of pair programming

Beck (2000) suggests several changes to the physical environment can encourage Extreme Programming, and that the best way to introduce it is to use it on a really tricky problem as a test. However, essentially one would imagine that the extent to which this pilot project team had 'bought into' the approach could directly affect the outcome of this test and that their attitude towards Extreme Programming might be quite sceptical considering how far XP seems from current 'best practice'. The author feels that further evidence concerning *why* pair programming might work and *how* it works best could be essential aids in providing a more factual basis for its introduction.

Another interesting aspect of pair programming not covered in the studies considered, is the collaborative use of a computer system, which was designed for a single person. Of course, collaborating over a single screen is nothing new – more than one person often work together to solve word-processing problems (for example, see Twidale, 2000). The provision of a suitable tool to support pair programming, potentially remotely, could prove instrumental in allowing the benefits of pair programming to be gained where a physically 'side by side' environment is not possible. For example, Harrison (2003) observes cases where developers who were 2000 miles apart wrote code individually (presumably as they lacked a suitable shared development environment) but debugged their code together using a shared desktop and Kircher et al. (2001) describe a project with members in India, Germany, Italy and the USA, but freely admit that they could not completely substitute close physical proximity. Beck (2000) emphasizes the importance of a supportive environment, which should encourage pair programming and project team interaction but also provide privacy as and when required, but appears to assume that this is not possible without close physical proximity. Further research is required to ascertain whether this is, indeed, the case.

Within Extreme Programming experience reports (see Figure 2) pair programming is generally considered a successful, useful and enjoyable approach, at least on a voluntary basis. However, sceptics have some reservations about the applicability of pair programming to every situation. Data from further studies could provide insight into pair programming in three ways: First, by helping to provide information regarding the suitability of particular individuals to pair programming by identifying characteristics of potentially successful pairers; Second, by providing information regarding behaviours and approaches which are observable in successful and less successful pairers, thus assisting in identifying training needs to improve pairing; Third, by assessing the suitability of remote pair programming given suitable tools. Further research could also provide empirical evidence regarding whether cross-pairing successful with less successful pairers can provide a more helpful learning experience than learning by 'going it alone' on successively harder tasks. If this is so then a case might be made for introducing pair programming into system development education for reasons other than the provision of a study group for research. In fact, one wonders whether pairing might work equally well in other disciplines and domains.

Observations

The work reviewed in this paper illustrates the level of interest in Extreme Programming and the wide number of studies that have taken place regarding this relatively new practice. While many studies have made progress in understanding what XP involves and how it might be implemented, a number of questions regarding its global application and the psychological aspects underlying its twelve practices remain unexplored. In particular, questions regarding the mechanisms by which between-programmer communication and programmer-user communication take place without the use of external representations seem key. The use and relevance of the system metaphor could also be more successfully discussed if these debates could be founded on a further understanding of when and how metaphors are produced, used and discarded on commercial IT projects. There are also many areas of pair programming which remain poorly understood – not least the attributes of successful pairs, the potential provision of a rich apprenticeship environment and whether pair programming can be facilitated by the provision of better tailored development tools, potentially remotely.

In order to begin to address some of these shortfalls the impact of study environment and methodology must be thoroughly considered. As stated by Curtis (1986), for example, the extrapolation of results from academic findings to a commercial setting may provide misleading conclusions. Similarly, whilst practitioners experience reports remain useful for advising how to introduce or practice XP, and highlighting some of the pitfalls to be avoided, they do not provide insight into the mechanisms employed, nor do they tend to take a standard approach to study design. Suitably disciplined ethnographic studies of extreme programmers 'in the wild' could assist in obtaining this insight into the cognitive aspects of Extreme Programming. One particularly fruitful approach might be to seek informed hypothesis through observational studies that could then form the basis of further empirical work.

Conclusion

The studies reviewed in this paper highlight an apparent gap between the psychology of programming and Extreme Programming. Whilst experience reports may prove useful in highlighting some of the potential problems with the introduction and on-going use of XP in a variety of companies and projects, there is a need for some over-arching understanding about the use of Extreme Programming practices on a psychological level. To the author's knowledge this work has not yet begun, and the handful of empirical studies which have taken place have done so in an academic environment, or have attempted to answer questions about whether or not a practice is appropriate, without taking any more than an educated guess at why this may be the case.

Further commercially based research into the cognitive aspects of Extreme Programming may not only assist in ensuring where, when and to whom these practices may be most applicable, but also what the highlights and shortfalls of Extreme Programming are, and how it might usefully be modified or extended. Such information is essential if companies and educational institutions are to make informed choices about their development methodological paradigm rather than taking a 'leap of faith' into the unknown.

Acknowledgements

The author would like to thank Professor Benedict du Boulay and Dr Pablo Romero for their invaluable advice and ideas and Genaro Robelledo-Mendes for his assistance and input.

References

Adelson B & Soloway E (1985). The role of domain experience in software design. *IEEE Transactions on Software Engineering*, 11 (11), p1351-1360.

Agile Manifesto (2004). www.agilemanifesto.org

Ambu W & Gianneschi F (2003).Extreme Programming at work. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering* in *Lecture Notes In Computer Science*, 2675, p.347-350.

Beck K (2000). *Extreme Programming Explained: Embrace Change*. Addison Wesley.

Becker-Pechau P, Breitling H, Lippert M and Schmolitzky A (2003). Teaching team work: An extreme week for first year programmers. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering. Lecture Notes in Computer Science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p360-362.

Benedicenti L, Paranjape R (2001). Using extreme programming for knowledge transfer. *Proceedings of the 2nd International Conference on eXtreme Programming and Agile Processes in Software Engineering.*

Carroll JM, Thomas JC (1982). Metaphor and the cognitive representation of computing systems. *IEEE Transactions on Systems, Man and Cybernetics*, 12 (2), p107-116.

Cockburn A, Williams L (2001). The costs and benefits of pair programming. *Extreme Programming Examined*. Addison Wesley.

Crabtree A (2003). *Designing Collaborative Systems: a Practical Guide to Ethnography*. Springer-Verlag, London.

Curtis, Bill (1986). By the way, did anyone study any real programmers? *Empirical Studies Of Programmers, First Workshop*. Soloway E & Iyengar S (Eds), p.256-261.

Deias R, Mugheddu G, Murru O (2002). Introducing XP in a start-up. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy.

Dick AJ, Zarnett B (2002). Paired programming and personality traits. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy.

Dogan F & Nersessian N (2002). Conceptual diagrams: representing ideas in design. *Diagrammatic Representation and Inference*, Hegarty M, Meyer B, Narayanan N H (Eds), p353-355.

Fuqua AM & Hammer JM (2003). Embracing change: An XP experience report. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering in Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p298-306.

Gick ML, Holyoak KJ (1985). Analogical problem solving. *Cognitive Psychology* 12(80), p306-356.

Gittins RG, Hope S, Williams I (2001). Qualitative studies of XP in a medium sized business. *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering*,Villasimius, Sardinia, Italy.

Harrison NB (2003). A study of Extreme Programming in a large company. www.research.avayalabs.com/techreport/ ALR-2003-039-paper.pdf

Heiberg S, Puus U, Salumaa P and Seeba A. *Pair-programming effect on developers' productivity. Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering*. Lecture Notes in computer science, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p.215-224.

Herbsleb, James; Root, David; & Tomayko, James. The eXtreme Programming (XP) metaphor and software architecture. (Technical Report CMU-CS-03-167) Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Hutchins E (1995). *Cognition in the Wild*. The MIT press, Cambridge, MA

Jenson B & Zilmer A (2003). Cross-continent development using scrum and XP. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering in Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p146-153.

Johnson S, Mao J, Nickell E, Smith I (2003). Extreme makeover: bending the rules to reduce risk rewriting complex systems. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering in Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p307-314.

Johnston A & Johnson CS (2003). Extreme Programming: A more musical approach to software development. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering*. Lecture Notes in computer science, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p325-327.

Karlstrom D (2002). Introducing extreme programming – an experience report. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*. Alghero, Italy.

Kircher M, Jain P, Corsaro A, Levine D (2001). Distributed extreme programming. *Proceedings of the 2nd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Cagliari, Sardinia, Italy.

Lappo P (2002). No pain, no XP: Observations on teaching and mentoring Extreme Programming to university students. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy.

Lave J & Wenger E (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.

Lippert M, Schmolitzky A and Zullighoven H (2003). Metaphor design spaces. *Proceedings of the 4th International conference in Extreme Progrmming and Agile Processes in Software Engineering*. Lecture Notes in computer science, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p.33-40.

Lui KM & Chan KCC (2003) When does a pair outperform two individuals? *Proceedings of the 4th International conference in Extreme Programming and Agile Processes. Software Engineering in Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p225-233.

Macias F (2002). Empirical experiments with XP. In *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*. Alghero, Italy.

Marcus A (1995). Metaphor design in user interfaces: how to manage expectation, surprise, comprehension and delight effectively. *Conference companion on Human Factors in Computing Systems*, p373-374.

Noll J & Atkinson D (2003). Comparing extreme programming to traditional development for student projects: a case study. *Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes .Software Engineering. Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p372-374.

Perry DE, Staudenmayer NA, Votta LG (1994). Understanding software development: processes, organizations and technologies. *IEEE Software*, 11(4), p36-45.

Petre, M. (2003). Team coordination through externalised mental imagery. *Proceedings of the co-located 15th Annual Psychology of Programming Interest Group Workshop and Empirical Assessment of Software Engineering Conference*. (Keele, April). Petre, M (Ed).

Preece, Rogers Y, Sharp H, Benyon D, Holland S & Carey T (1994). *Human-Computer Interaction*. Addison-Wesley, Wokingham, England.

Pulugurtha S, Neveu J-N, Lynch F (2002). Extreme programming in a customer services organization. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy.

Romero P, Cox R, du Boulay B and Lutz R (2002). Visual attention and representation switching during Java program debugging: a study using the Restricted Focus Viewer. *Lecture Notes in Artificial Intelligence*, 2317. Goos. G, Hartmanis, J & van Leeuwen J (Eds). Springer.

Rumpe B, Schroder A (2002). Quantitative survey on extreme programming projects. *Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy.

Scaife M & Rogers Y (1996). External cognition: how do graphical representations work? *Int J. Human Computer Studies*, 45, p185-213.

Sharp HC & Robinson HM (2003) An ethnography of XP practice, *Proceedings of the co-located 15th Annual Psychology of Programming Interest Group Workshop and Empirical Assessment of Software Engineering Conference*. (Keele, April). p15-27.

Steimann F, Gossner J & Muck T (2003). Filleting XP for educational purposes. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering. Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p.225-233.

Suwa M & Tversky B (2002). External representations contribute to the dynamic construction of ideas. *Diagrammatic Representation and Inference*, Hegarty M, Meyer B, Narayanan N H (Eds), p341-343.

Tessem B (2003). Experiences in learning XP practices: a qualitative study. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes in Software Engineering in Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds), p131-137.

Tomayko JE & Herbsleb J (2003). How useful is the metaphor component of agile methods? A preliminary study. (Technical Report CMU-CS-03-152) Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Twidale, M.B. (2000). Interfaces for supporting over-the-shoulder learning. *Proceedings of HICS 2000: The Fifth Annual Conference on Human Interaction with Complex Systems*. The Beckman Institute, University of Illinois at Urbana-Champaign, M. Benedict (Ed.), p33-37.

Vanhanen J, Jartti J, Kahkonen T (2003). Practical experiences of agility in the telecom industry. *Proceedings of the 4th International conference in Extreme Programming and Agile Processes. Software Engineering. Lecture Notes in computer science*, 2675, Goos G, Hartmanis J, van Leeuwen J (eds). P279-287.

Wake, W (2002). *Extreme Programming Explored*. Addison Wesley, NJ, USA

West D (2002). Metaphor, architecture and XP. *Proceedings of the 3rd International conference on Extreme Programming and agile processes*, Alghero, Italy.

Williams L, Kessler RR, Cunningham W, Jeffries R (2000). Strengthening the case for pair-programming. *IEEE Software*, 17(4), p19-25.