

UPGRADE is the European Online Magazine for the Information Technology Professional, published bimonthly at <http://www.upgrade-cepis.org/>

Publisher

UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by Novática (<http://www.ati.es/novatica/>) and Informatik/Informatique (<http://www.svifsi.ch/revue/>)

Chief Editors

François Louis Nicolet, Zurich <nicolet@acm.org>
Rafael Fernández Calvo, Madrid <rfoalvo@ati.es>

Editorial Board

Prof. Wolfried Stucky, CEPIS President
Gloria Nistal Rosique and
Rafael Fernández Calvo, ATI
Prof. Carl August Zehnder and François Louis Nicolet, SVI/FSI

English Editors: Mike Andersson, Richard Butchart, David Cash, Arthur Cook, Tracey Darch, Laura Davies, Nick Dunn, Rodney Fennemore, Hilary Green Roger Harris, Michael Hird, Jim Holder, Alasdair MacLeod, Pat Moody, Adam David Moss, Phil Parkin, Brian Robson

Cover page designed by Antonio Crespo Foix, © ATI 2002

Layout: Pascale Schürmann

E-mail addresses for editorial correspondence:
<nicolet@acm.org> and <rfoalvo@ati.es>

E-mail address for advertising correspondence:
<novatica@ati.es>

Copyright

© Novática and Informatik/Informatique. All rights reserved. Abstracting is permitted with credit to the source. For copying, reprint, or republication permission, write to the editors.

The opinions expressed by the authors are their exclusive responsibility.

eXtreme Programming

Guest Editor: Luis Fernández Sanz

Joint issue with NOVÁTICA and INFORMATIK/INFORMATIQUE

2 Presentation: eXtreme Programming

– Luis Fernández Sanz, Guest Editor

4 A new method of Software Development: eXtreme Programming

– César F. Acebal and Juan M. Cueva Lovelle

What is eXtreme Programming – also known as XP? The aim of this article is to answer that question, and to reveal the nature of this new method of software development to the uninitiated reader. We will try to be sufficiently informative so that you will all come away with some idea of the basic underlying principles, and for anyone who might want to delve deeper into the subject, we will provide suitable references.

9 Programming Extremism – Michael McCormick

The author reviews antecedents and experiences of the "agile" methodology of software development called eXtreme Programming, comparing it to other methodologies and pointing to its advantages and disadvantages from a pragmatic standpoint, depending on the kind of project it applies to. He draws the conclusion that it is necessary to stay away from "religious" positions about existing methodologies.

11 The Need for Speed: Automating Acceptance Testing in an eXtreme Programming Environment

– Lisa Crispin, Tip House and Carol Wade (Contributor)

How to focus acceptance testing for XP, how to design automated tests that are low-maintenance and self-verifying, how to apply the values of XP to test automation, and ways to gather metrics and provide useful reports.

18 Qualitative Studies of XP in a Medium Sized Business

– Robert Gittins, Sian Hope and Ifor Williams

Qualitative Research Methods are used to discover the effects of applying eXtreme Programming in a software development business environment. Problems dominating staff development, productivity and efficiency are parts of a complex human dimension uncovered in this approach. The interpretation and development of XP's "Rules and Practices" are reported, as well as the interlaced communication and human issues affecting the implementation of XP in a medium sized business.

23 XP and Software Engineering: an opinion – Luis Fernández Sanz

In this article, the author makes some reflections on certain specific aspects of eXtreme Programming as described in Kent Beck's book "eXtreme Programming explained. Embrace change". The analysis presented here is in relation to principles and techniques of software engineering.

27 XP in Complex Project Settings: Some Extensions

– Martin Lippert, Stefan Roock, Henning Wolf and Heinz Züllighoven

XP has one weakness when it comes to complex application domains or difficult situations at the customer's organization: the customer role does not reflect the different interests, skills and forces with which we are confronted in development projects. We propose splitting the customer role into a user and a client role. The user role is concerned with domain knowledge; the client role defines the strategic or business goals of a development project and controls its financial resources. It is the developers' task to integrate users and clients into a project that builds a system according to the users' requirements, while at the same time attain the goals set by the client.

Coming issue:
"Information Retrieval"

Qualitative Studies of XP in a Medium Sized Business

Robert Gittins, Sian Hope and Ifor Williams

Qualitative Research Methods are used to discover the effects of applying eXtreme Programming (XP) in a software development business environment. Problems dominating staff development, productivity and efficiency are parts of a complex human dimension uncovered in this approach. The interpretation and development of XP's "Rules and Practices" are reported, as well as the interlaced communication and human issues affecting the implementation of XP in a medium sized business. The paper considers the difficulties of applying XP in a changing software requirements environment, and reports on early deployment successes, failures and discoveries, and describes how management and staff adapted during this period of change. The paper examines the benefits of a flexible management approach to XP methodology, and records the experiences of both management and staff, as initial practices matured and new practices emerged.

Keywords: Extreme Programming, Qualitative methods, Software Methodology.

1 Related Work

Previous qualitative research [Seaman 99], [Sharp et al. 99], [Cockburn/Williams 02], has concentrated on non-judgmental reporting, with the intent of provoking discussion within the culture being studied by providing observations and evidence, collaborators deciding for themselves whether any changes were required. This fieldwork study follows the format of [Gittins/Bass 02], whereby the researcher is immersed for a period in the software developer team; thereby the active researcher becomes instrumental in the development and improvement of XP. [Seaman 99] describes an empirical study that addresses the issue of communication among members of a software development organization. [Sharp et al. 99] use combined *ethnography* and *discourse analysis*, to discover implicit assumptions, values and beliefs in a software manage-

ment system. [Cockburn/Williams 02] investigate "*The cost benefits of pair programming*". [Sharp et al. 00] describe a "cross-pollination" approach, to a deeper understanding of implicit values and beliefs.

XP developed recently from [Beck 00] and [Beck/Fowler 00], and more recently in [Jeffries et al. 00]. [Williams/Kessler 00] study lone and paired programmers, and [Williams et al. 00] the cost effectiveness of pairing.

2 The Study

Secure Trading, the focus of this paper, is a medium sized software company committed to implementing XP, and comprises a team of nine developers. Secure Trading decided to implement XP in a progressive manner, conscious of minimising disruption to the business process. Reference material from other companies, not specifically named in this paper, will only be used in general terms to highlight some typical problems facing established, and highly traditional companies,

Robert Gittins is final year PhD student at the Ada Lovelace Laboratory, University of Wales, BangorGwynedd, Wales, UK. His research explores the interfaces between the disciplinary experts who collaborate to develop approaches to developing commercial software development solutions, especially for distributed systems. Communication between these disciplinary domains, and the cooperative solution of conflicting design problems, are the key areas of his investigation. His research goal is to contribute strategies, methods or algorithms for novel software tools to support the design process. <r.g.gittins@informatics.bangor.ac.uk>

Sian Hope is a senior lecturer at the School of Informatics, University of Wales Bangor. Research interests are focused on contributing to enhancement of the discipline of Software Engineering by researching into practically applicable development methods which are firmly grounded on scientifically sound concepts. Fundamental

approaches are sought, which provide a bridge between theory and practice and which are aimed at producing engineering methods, tools and metrics for all phases of software development. Sian can be contacted at sian@informatics.bangor.ac.uk

Ifor Williams. After obtaining a degree in Computer Engineering from the University of Manchester Ifor went on to gain a PhD for research into computer architectures suitable for the efficient execution of object-oriented applications. This work resulted in the design of a machine (MUSHROOM) incorporating support for dynamic binding, object-based virtual memory, efficient garbage collection and targeted as a high-performance Smalltalk platform. Later he spent some time developing software for the medical diagnostics industry using prescribed conventional development processes before joining the rapidly changing.com world where XP proved to be invaluable. He can be contacted at ifor.williams@securetrading.com.

sensitive to their developer environment, and to the cost of disruption that change would incur on staff and production.

Secure Trading, had recently moved to larger offices. When research started, their involvement with XP consisted of some intermittent attempts at “pairing” developers. Their move presented opportunities for improving “pairing” proficiency, and the selective adoption of XP practices.

3 Qualitative Research Work

This research adopts some of the techniques historically developed in the Social Sciences [Gittins 02], *ethnography*, *qualitative interviews* and *discourse analyses*, an understanding of “grounded theory” was particularly important. Grounded theory can provide help in situations where little is known about a topic or problem area, or to generate new ideas in settings that have become static or stale. Developed by Barney Glaser and Anselm Strauss [Glaser/Strauss 67] in the 60s, grounded theory deals with the generation of *theory* from *data*. Researchers start with an area of interest, collect data, and allow relevant ideas to develop. Rigid pre-conceived ideas are seen to prevent the development of research. To capture relevant data, qualitative research techniques are employed [Gittins/Bass 02] that include the immersion of the researcher within the developer environment, qualitative data analyses, guided interviews, and questionnaires.

3.1 Qualitative Data

Qualitative evaluation allows the researcher to study selective issues in detail, without the pre-determined constraints of “categorised” analyses. The researcher is instrumental in the gathering of data from open-ended questions. Direct quotations are the basic source of raw materials, revealing the respondent’s depth of concern. This contrasts with the statistical features of quantitative methods, recognised by their encumbrance of predetermined procedures.

3.2 Qualitative Interviews

[Patton] suggests three basic approaches to collecting qualitative data through interviews that are open-ended. The three approaches are distinguished by the extent to which the questions are standardised and predetermined, each approach having strengths and weaknesses, dependant upon the purpose of the interview:

1) “*Informal conversational*” interviews, are a spontaneous flow of questions where the subject may not realise that the questions are being monitored. 2) The “*General interview guide*” approach, adopted extensively for this study, predetermines a set of issues to be explored. 3) The “*Standardised open-ended interview*” pursues the subject through a set of fixed questions that may be used on a number of occasions, with different subjects.

In a series of interviews, data was collected using “Informal conversation” and verbatim transcripts taken from “General guided interviews”.

3.3 Questionnaires

In an extensive questionnaire consideration was given to the “Rules and Practices” of XP. Questions targeted the software development process, XP practices, and both managerial and behavioural effectiveness. Behavioural questions were based upon Herzberg’s “Hygiene and Motivation Factors” [Herzberg 74]. Ample provision was provided for open comments on each of the topics, and a developer floor plan provided for a respondent to suggest improvements to the work area. Repeating the questionnaire at three monthly intervals will help research and management by matching the maturing XP practices, as they progress, against developer responses.

4 Rules and Practices

4.1 Pair Programming

(See [Beck 00], [Beck/Fowler 00]) XP advances what has been reported for some time [Cockburn/Williams 02], [Williams/Kessler 00], [Williams et al. 00]; Two programmers working together generate an increased volume of superior code, compared with the same two programmers working separately. Secure Trading management, discussed the implementation of “Pairing” with the development team, who unanimously agreed to “buy-in” to the practice. The first questionnaire showed some of the team were unhappy with pairing. 28% of developers preferred to work independently, 57% didn’t think they could work with everyone, and 57% stated that pair programmers should spend on average 50% of their time alone. XP practices recommend no more than 25% of a conditional 40-hour week be paired. Two developers summed up the team’s early attitude to pair programming: “*I feel that pair programming can be very taxing at times, although I can see the benefits of doing it some of the time.*”

“*Not everyone makes an ideal pair. It only really works if the pair is reasonably evenly matched. If one person is quiet, and doesn’t contribute, their presence is wasted. Also, if a person is really disorganised and doesn’t work in a cooperative way, the frustration can (disturb) the other participant!*”

Developers estimated that they spent approximately 30% of their time pairing, with partner changes occurring only upon task completion, changes being agreed and established *ad hoc*. Frequent partner swapping, and partner mixing, commands great merit in XP. Pairing practices matured with the introduction of a team “Coach” and later a “Tracker” [Beck 00]. Maintenance tasks were another problem which routinely disrupted pairing. Here control was reviewed and tasks better ordered to minimise this problem. In time, the impact of pairing activity upon developers will translate into evidence, returned in the periodic questionnaire reviews, and in the timeliness and quality of code produced.

4.2 Planning Games

(See [Jeffries et al. 00]) Planning games were introduced soon after pairing practices were established. The “customer” duly chooses between having more stories, requiring more time; against a shorter release, with less scope. Customers are not permitted to estimate story or task duration in XP and

developers are not permitted to choose story and task priority. Where a story is too complex or uncertain to estimate, a “Spike” is created. Spike solutions provide answers to complex and risky stories. Secure Trading succeeded well in developing Planning games, utilising “Spike solutions” by logging a “spike” as a fully referenced story to quickly attack the problem, reducing a complex, inestimable story to a simple, and easily understood, group of stories. Results were very effective; “spike solutions” proved easy to develop and derived estimates for completion proved consistently accurate. It was common practice to have the essential elements of both *iteration* and *release* Planning games combined into one meeting. This practice worked for them in the context of the jobs they were planning.

4.3 Client On-site

(See [Beck 00]) Secure Trading rarely had this luxury. When required the “Client” role was undertaken by a client’s representative, co-opted from the Customer services department by staff who had worked closely with the client and were able to accept that responsibility. Developer Manager: *“The inclusion of a representative from Customer services has proven to be hugely beneficial, providing immediate feedback of the system’s successes and failures on a day-to-day basis.”*

4.4 Communication

(See [Beck 00]) A great deal of attention is necessary in providing an XP environment in keeping with the practices to support XP. Key factors in communication are: the use of white boards, positioning and sharing of desk facilities to facilitate pair programmers, “stand-up” meetings, developers “buying-in” to the concepts of the “rules and practices” of XP, and “collective code ownership”. Interviews and questionnaires revealed many areas of concern among developers. For example, 86% of developers disagreed that meetings were well organized; *“Agreements at meetings are not set in concrete”* and, *“Confidence is lost with meeting procedures, when agreed action or tasks are later allowed to be interpreted freely by different parties.”* Management were quick to address these concerns by concentrating on the development of XP story card practices. Developers were encouraged to agree, and finalise with the client, the task description and duration estimates at timely Planning Game meetings. Story cards were fully referenced and signed by the accepting developer, thereby becoming the responsibility of the initiating developer until completion. Only the responsible creator of a Card was authorized to amend it.

The use and placement of White boards is said to be an essential supporting means of good communication in XP practices [Beck 00]. Mobile whiteboards were introduced by Secure Trading soon after pair programming practices gained momentum and used to record the story details agreed at Planning Game meetings. At one point, story cards were physically stuck to the boards in prioritised order with adjacent notes written on the board. This proved unpopular and developed into cards being retained but not stuck on the white board. Stories were written on the boards. Referenced stories

contained ownership, estimation, as well as iteration and priority, which were displayed in columned format. On completion, the owner added the actual task duration. The information served to improve personal proficiency in estimation and in providing feedback towards establishing project “velocity” data, for future Planning Game meetings.

Stand-up meetings promote communication throughout the team. Secure Trading introduced this practice from day one. At ten o’clock every morning, a meeting allowed everyone to briefly state (standing promotes brevity) their work for the day, and discuss problems arising from the previous days activity. Anyone was free to comment, offer advice or volunteer co-operation. The benefits of adopting stand-up meetings were far-reaching and seen by developers and management as an effective way to broadcast activities, share knowledge and encourage collaboration amongst and between team members and management. Secure Trading meetings tended to degrade when reports migrated to topics of yesterday’s activity, rather than those planned for the day. This activity persists and may remain or need to be resolved and modified as their particular brand of XP develops.

4.5 Simple Design

Beck [Beck 00] summarises simple design in *“Say everything once and only once.”* However a comment by one developer interviewed revealed a common concern, *“Sometimes, it is a bit too simplistic, and issues seem to be avoided”*. XP states that it is important to produce a simple system quickly, and that “Small Releases” are necessary to gain feedback from the client. Secure Trading didn’t see themselves in a position to implement this practice so early in their XP programme. XP allows companies to cherry-pick those practices they regard suitable for implementation, in the order they see fit.

4.6 Tests

Unit tests are written in XP before main code and give an early and clear understanding of what the program must do. This provides a more realistic scenario, as opposed to “after-the-code testing,” that could, for many reasons, neatly match completed code. Time is saved both at the start of coding, and again at the end of development. Latent resistance to early unit testing became manifest, when the perceived closeness of a deadline loomed. This activity is perhaps the hardest to implement and requires commitment from developers. An early questionnaire revealed that 71% of Secure Trading developers regarded unit-testing practices in general to be “very poor”. Developer Manager on early introduction of unit testing: *“If you already have a large complex system, it is difficult to determine to what extent testing infrastructure is to be retrospective-ly applied. This is the most difficult aspect in our experience. Starting from scratch it is much easier to make stories and code testable.”*

4.7 Refactoring

(See [Fowler 99]). *“The process of improving the code’s structure while preserving its function.”* The use and reuse of old code is deemed costly, often because developers are afraid

they will break the software. XP indicates that refactoring throughout the project life cycle saves time and improves quality. Refactoring reinforces simplicity by its action in keeping code clean and reducing complexity. Secure Trading had not developed refactoring activities in line with XP at that time. Many developers expressed concern with refactoring, more commonly reported by traditional companies: "... with more people, we could spend more time refactoring and improving the quality of our existing code base." The questionnaire revealed that 45% of developers considered refactoring sporadic or very poor.

4.8 Collective Code Ownership

(See [Beck 00], [Beck/Fowler 00]). This concept states "Every programmer improves any code anywhere in the system at any time if they see the opportunity." Collective code ownership has many merits: It prevents complex code entering the system, developed from the practice that anyone can look at code and simplify it. It may sound contentious, but XP Test procedures should prevent poor code entering the system. Collective Code Ownership also spreads knowledge of the system around the team. Secure Trading experienced growing pains in developing this principle, revealed by the comments of two developers: "I have conflicting interests in collective code ownership. I think it is very good when it works, but there are times when some code I have written seems to just get worse when others have been working on it."

"I like the idea of collective code ownership, but in practice I feel that I own, am responsible for, some bits of code." From the traditional perspective of individual ownership, it will be important to record how attitudes change, as XP practices mature.

4.9 Metaphor

A metaphor in XP is a simple shared story to encompass and explain what the application is "like", communicating a mental image, so that everyone involved can grasp the essence of the project in a term universally understood. This may seem to be a relatively easy, or lightweight, activity to adopt. However, the value of this practice was not immediately evident to developers, early difficulties developing and applying suitable metaphors were experienced and this practice was reluctantly abandoned for future consideration.

5 Companies Starting from Scratch

Long established and traditional companies, considering adopting XP, have, unlike Secure Trading, many more difficulties to overcome. They mostly comprise traditional teams of developers, who are comfortably established, working in small offices, in prohibitively cloistered environments. Management is often aware that legacy software in circulation is in the "ownership" of one or two heroic developers, at the cutting edge of their business. Some teams were reported as badly under-performing and in some circumstances management had resorted to consultants to resolve their problems with no significant success reported. Often with great reluctance, management

allowed the research team to visit developer offices. Tension was evidently high. In these companies, "Risks" [Beck 00] are high, quality is compromised, communication difficult, and control largely ineffective. There are other considerations when starting from scratch; The Secure Trading developer manager reflecting upon attempts at implementing XP in his early projects stated: "One of the key "discoveries" has been the relative ease to which XP has been employed on an all-new project, and the difficulty in applying XP retrospectively on an established system."

6 Conclusions

A combination of qualitative and quantitative methods has helped identify uncertainties in applying XP practices in a medium sized software development company. How particularly one company interpreted and developed their *brand* of XP, moulded from their successes and failures. Successes in such areas as the use and development of "spike solutions", and Customer role-play within "Planning Game" activity, and from failures, as in developer reluctance to "buying-in" to "collective code ownership", and the difficulties of implementing the practice of "simple design", and in the use of "metaphors". Partial success was seen in "Pair programming", that having posed early problems, showed improvement in maturity. Future work will monitor the complex factors in the development of XP within small and growing companies at various levels of maturity. By acknowledging the characteristic unsharp boundaries of qualitative data sets, future work will investigate the use of fuzzy logic for data analyses.

Acknowledgement

This paper acknowledges the funding and support of the EPSRC (Award No. 99300131).

References

- [Beck 00]
K. Beck: "Extreme Programming Explained: Embrace change". Addison Wesley. 2000
- [Beck/Fowler 00]
K. Beck and M. Fowler: "Planning Extreme Programming". Addison Wesley 2000.
- [Cockburn/Williams 02]
A. Cockburn and L. Williams: "The cost benefits of pairprogramming".
<http://members.aol.com/humansandt/papers/pairprogrammingcostbene/pairprogrammingcostbene.htm>.
- [Fowler 99]
M. Fowler: "Refactoring: Improving the design of existing code", Addison Wesley. July 1999.
- [Gittins 02]
R. G. Gittins: "Qualitative Research: An investigation into methods and concepts in qualitative research". Technical Paper: via <http://www.sesi.informatics.bangor.ac.uk/english/home/research/technical-reports/sesi-020.htm>
- [Gittins/Bass 02]
R. G. Gittins and M. J. Bass: "Qualitative Research Fieldwork: An empirical study of software development in a small company, using guided interview techniques", Technical Paper: via <http://www.sesi.informatics.bangor.ac.uk/english/home/research/technical-reports/sesi-021.htm>

[Glaser/Strauss 67]

B. G. Glaser and A. L. Strauss: "The discovery of grounded theory: strategies of qualitative research" Chicago: Aldine Publications. 1967

[Herzberg 74]

F. Herzberg: "Work and the Nature of Man", Granada Publications Ltd. 1974

[Jeffries et al. 00]

R. Jeffries, A. Anderson and C. Hendrickson: "Extreme Programming Installed". Addison Wesley 2000.

[Patton]

M. Q. Patton: "Qualitative Evaluation and Research Methods" (2nd Edit.). SAGE Publications

[Seaman 99]

C. B. Seaman: "Qualitative methods in empirical studies of software engineering", IEEE Trnsctns on Software Engineering, Vol.25 (4):557-572 Jul/Aug 99.

[Sharp et al. 99]

H. Sharp, M. Woodman, F. Hovenden and H. Robinson: "The role of 'culture' in successful software process improvement." EUROMICRO:1999 Vol.2, p17.

[Sharp et al. 00]

IEEE Computer Society. H. Sharp, H. Robinson and M. Woodman: "Software Engineering: Community and Culture". IEEE Software, Vol. 17, No.1, Jan /Feb2000

[Williams/Kessler 00]

L. A. Williams and R. R. Kessler: "All I Really Wanted to Know About Pair Programming I Learned in Kindergarten". Communications of the ACM. May 2000 Vol.43, No5.

[Williams et al. 00]

L. A. Williams, R. R. Kessler, W. Cunningham and R. Jeffries: "Strengthening the Case for PairProgramming". IEEE Software, Vol. 17, No. 4: July/August,2000,pp19-25.