

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. Wolffried 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"

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. Naturally the length of any technical article does not permit more than a brief introduction to any new method or technique, but 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.

Keywords: eXtreme Programming, XP, Software Development.

1 Introduction

XP is a new software development discipline which, amid much fanfare, has recently joined the welter of methods, techniques and methodologies that already exist. To be more precise: this is a lightweight method, as opposed to heavy-weight methods like Métrica. Before we go on, we'd like to make a clarification: in this article we refer to XP as a "method", contrary to the official IT tendency to apply the term "methodology" (science of methods) to what are no more than methods¹ or even mere graphic notations.

It could be said that XP was born "officially" five years ago in a project developed by *Kent Beck* at *Daimler Chrysler*, after he had worked for several years with *Ward Cunningham* in search of a new approach to the problem of software development which would make things simpler than the existing methods we were used to. For many people, XP is nothing more than common sense. Why then does it arouse such controversy, and why do some people adore it while others heap scorn on it? As *Kent Beck* suggests in his book [Beck 00], maybe it's because XP carries a series of common sense techniques and principles to extreme lengths. The most important of these techniques are:

- Code is constantly reviewed, by means of *pair programming* (two people per machine)
- Tests are made all the time, not only after every class (*unit tests*), but also by customers who need to check that the project is fulfilling their requirements (*functional tests*)
- Integration tests are always carried out before adding any new class to the project, or after modifying any existing one (*continuous integration*), making use of *testing frameworks*, such as *xUnit*
- We (re)design all the time (*refactoring*), always leaving code in the simplest possible state

- Iterations are radically shorter than is usual in other methods, so that we can benefit from feedback as often as possible.

By way of summary, and to wrap up this section and to get down to the nitty-gritty, I will leave you with this quote from *Beck's* aforementioned book.

"Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team members change. The problem isn't change, per se, because change is going to happen; the problem, rather, is the inability to cope with change when it comes."

2 The four variables

XP sets out four variables for any software project: *cost*, *time*, *quality* and *scope*.

It also specifies that of these four variables, only three of them can be established by parties outside the project (custom-

César Fernández Acebal received a degree in informatics engineering from Oviedo University. He worked as a teacher in Java and Web programming for students of higher professional education. Afterwards, he worked as technical director in a web site development company. He has combined these positions with a continuous educational activity related to Java, XML, Web development, etc. He is currently an IT architect of B2B 2000, an e-business company. His research interests include object-oriented programming and software engineering and agile software processes. He is a member of ATI, IEEE, Computer Society, ACM, etc. <acebal@ieee.org>

Juan Manuel Cueva Lovelle is a mining engineer from Oviedo Mining Engineers Technical School in 1983 (Oviedo University). He has the Ph. D. from Technical University of Madrid in 1990. From 1985 he is Professor at the Languages and Computers Systems Area in Oviedo University. ACM and IEEE voting member. His research interests include Object-Oriented technology, Language Processors, Human-Computer Interface, Object-Oriented Databases, Web Engineering, Object-Oriented Languages Design, Object-Oriented Programming Methodology, XML, WAP, Modelling Software with UML and Geographical Information Systems. <cueva@lsi.uniovi.es>

1. Ricardo Devis Botella. C++. STL, Plantillas, Excepciones, Roles y Objetos (Templates, Exceptions, Roles and Objects). Paraninfo, 1997. ISBN 84-283-2362-3

ers and project managers), while the value of the free variable will be established by the development team in accordance with the other three values. What is new about this? It's that normally customers and project managers considered it their job to pre-establish the value of *all* the variables: *"I want these requirements fulfilled by the first of next month, and you have this team to work with. Oh, and you know that quality is the number one priority!"*

Of course when this happens – and unfortunately it happens quite often – quality is the first thing to go out of the window. And this happens for a simple reason which is frequently ignored: no one is able to work well when they are put under a lot of pressure.

XP makes the four variables visible to everyone – programmers, customers and project managers –, so that the initial values can be juggled until the fourth value satisfies everybody (naturally, with the possibility of choosing different variables to control).

Also the four variables do not in fact bear such a close relation with one another as people often like to think. There is a well known saying that "nine women cannot make a baby in one month" which is applicable here. XP puts special stress on small development teams (ten or twelve people at most) which naturally can be increased if necessary, but not before, or the result will generally be the opposite of what was intended. However, a number of project managers seem to be unaware of this when they declare, puffed up with pride, that their project involves 150 people, as if it were a mark of prestige, something to add to their CV. It is good, however, to increase the cost of the project in matters such as faster machines, more specialists in certain areas or better offices for the development team.

With *quality* too, another strange phenomenon occurs: often, *increasing the quality means the project can be completed in less time*. The fact is that as soon as the development team gets used to doing intensive tests (and we will be coming to this point soon, as it's the corner stone of XP) and coding standards are being followed, gradually the project will start to progress much faster than it did before. The project's quality will still remain 100% assured – thanks to the tests – which in turn will instil greater confidence in the code and, therefore, greater ease in coping with change, without stress, and that will make people programme much faster... and so on.

The other face of the coin is the temptation to sacrifice the internal quality of the project – that which is perceived by the programmers – to reduce the delivery time of the project, trusting that the external quality – that which the customers perceive – will not be affected too greatly. However, this is a very short term bet, which tends to be an invitation for disaster, since it ignores the basic fact that *everyone works better when they are allowed to do a quality job*. Ignoring this will cause the team to get demoralised and, in the long term, the project will slow down, and much more *time* will be lost than ever could have been hoped to be saved by cutting down on quality.

With regard to the project's *scope*, it is a good idea to let this be the free variable, so that once the other three variables have been established, the development team should decide on the scope by means of:

- The estimation of the tasks to perform to satisfy the customer's requirements.
- The implementation of the most important requirements first, so that at any given time the project has as much functionality as possible.

3 The cost of change

Although we cannot go into any great depth on this subject here, we believe it is important to at least mention one of the most important and innovative suppositions that XP makes in contrast to most known methods. We are referring to the cost of change. It has always been considered a universal truth that the cost of change in the development of a project increased exponentially in time, as shown in figure 1.

XP claims that this curve is no longer valid, and that with a combination of good programming practices and technology it is possible to reverse the curve, as we show in figure 2.

Naturally, not everyone agrees with this supposition (and in Ron Jeffries web site [Jeffries] you can read several opinions to this effect). But in any event it is clear that if we decide to use XP as a software development process we should accept that this curve is valid.

The basic idea here is that instead of changing for change's sake, we will design as simply as possible, to do only what is absolutely necessary at any given moment, since the very simplicity of the code, together with our knowledge of refactoring [Fowler 99] and, above all, the testing and continuous integration, all mean that changes can be carried out as often as necessary.

4 Practices

But let's get down to brass tacks. What does XP really entail? What exactly are these practices we have been referring to, which are able to bring about this change of mentality when it comes to developing software? Trusting that you, the reader, will excuse the enforced brevity of our explanation we will now give a brief description of these practices.

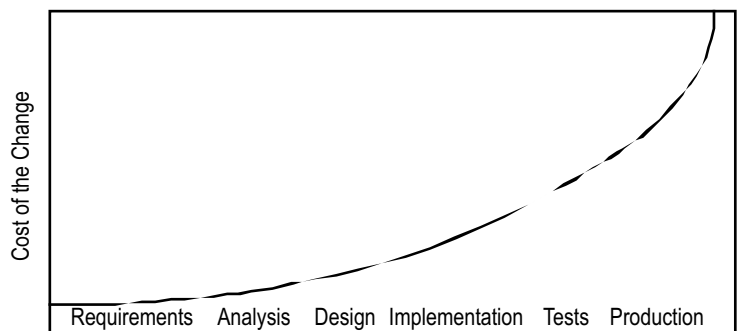


Figure 1: Cost of change in "traditional" software engineering

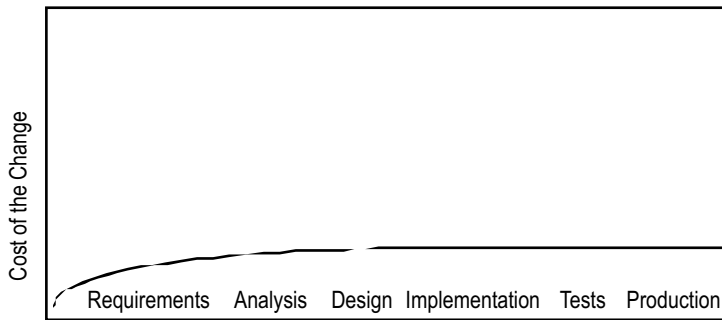


Figure 2: Cost of change in XP

Planning

XP sees planning as a permanent dialogue between the business and technical parties involved in the project, in which the former will decide the scope – what is really essential for the project –, the priority – what should be done first –, the composition of the releases – what should be included in each one – and the deadline for these releases.

The technical people, for their part, are responsible for estimating the time needed to implement the functionalities which the customer requires, for reporting on the consequences of decisions taken, for organising the work culture and, finally, for carrying out a detailed planning of each release.

Small releases

The system first gets into production just a few months at most before it is completely finished. Successive releases will be more frequent –at intervals of between a day and a month–. The customer and the development team will benefit from the feedback produced by a working system and this will be reflected in successive releases.

Simple design

Instead of being hell bent on producing a design which requires the gift of clairvoyance to develop, what XP advocates is, at any given moment, that we should design for the needs of the present.

Testing

Any feature of a programme for which there is not an automated test simply does not exist. This is undoubtedly the cornerstone on which XP is built. Other principles are liable to be adapted to the characteristics of the project, the organisation, the development team... But on this one point there is no argument: if we aren't doing tests, we aren't doing XP. We should be using some automated testing framework to do this, such as JUnit [JUnit] or any of its versions for different languages.

Not only that, but we will write the tests even before we write the class which is to be tested. This is an aid to following the principle of *programming by intention*, that is, writing code as if the most expensive methods had already been written, and so we only had to send the corresponding message, in such a way that the code will be a true reflection of its intention and will document itself. On JUnit's web site, mentioned in the previous

paragraph, you can find interesting articles which explain how these tests should be written.

Refactoring

This responds to the principle of simplicity and basically consists of leaving the existing code in the simplest possible state, so that no functionality is lost – or gained – and all tests continue to be carried out correctly. This will make us feel more comfortable with the code already written and therefore less reluctant to modify it when some feature has to be added or changed. In the case of legacy systems, or projects taken over after they have already been started, we would be bound to need to devote several weeks just to refactoring the code – which tends to be a source of tension with the project managers involved when they are told that the project is going to be held up for several days “just” to modify existing code, which works, without adding any new functionality to it.

Pair programming

All code will be developed in pairs – two people sharing a single monitor and keyboard. The person writing the code should be thinking about the best way to implement a particular method, while his colleague will do the same, but from a more strategic viewpoint:

- Are we going about this in the right way?
- What could go wrong here? What should we be checking in the tests?
- Is there any way to simplify the system?

Of course, the roles are interchangeable, so that at any moment the person observing could take over the keyboard to demonstrate some idea or simply to relieve his colleague. Similarly the composition of the pairs could change whenever one of them were required by some other member of the team to lend a hand with their code.

Collective property of the code

Anyone can modify any part of the code, at any time. In fact, any one who spots an opportunity to simplify, by refactoring, any class or any method, regardless of whether they have written them or not, should not hesitate to do so. This is not a problem in XP, thanks to the use of coding standards and the assurance that testing gives us that everything is going to carry on working well after a modification.

Continuous integration

Every few hours – or at the very least at the end of a day's programming – the complete system is integrated. For this purpose there is what is known as an integration machine, which a pair of programmers will go to whenever they have a class which has passed a unit test. If after adding the new class together with its unit tests the complete system continues to function correctly – i.e. passes all the tests –, the programmers will consider this task as completed. Otherwise they will be responsible for returning the system to a state in which all tests function at 100%. If after a certain time they are unable to

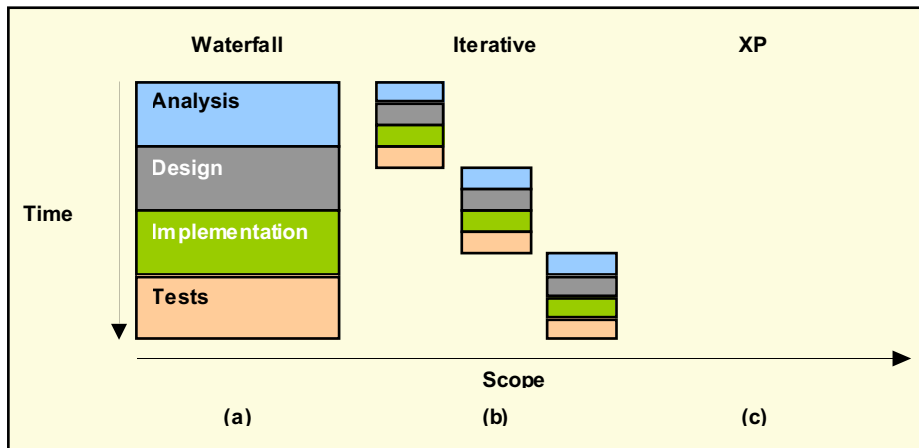


Figure 3: Comparison of long development cycles: (a) the waterfall model (b) shorter iterative cycles, for example, the spiral model and (c) the mix of all these activities which XP employs throughout the whole software development process

discover what it is wrong, they will bin the code and start over again.

40 hours a weeks

If we really want to offer quality, and not merely a system that works – which we all know, in IT, is trivial² – we will want each member of our team to get up each morning rested and to go home at 6 in the evening tired but with the satisfaction of a job well done, and that when Friday comes around he or she can look forward to two days’ rest to devote to things which have nothing whatsoever to do with work. Naturally it doesn’t have to be 40 hours – it could be anything between 35 and 45, but one thing is certain: nobody is capable of producing quality work 60 hours a week.

Customer on site

Another controversial XP rule: at least one real customer should be permanently available to the development team to answer any question the programmers may have for them, to establish priorities... If the customer argues that their time is too valuable, we should realise that the project we have been given is so trivial that they do not consider it worthy of their attention, and that they don’t mind if it is based on suppositions made by programmers who know little or nothing of the customer’s real business.

Coding standards

These are essential to the success of collective property of the code. This would be unthinkable without a coding based on standards which allow everyone to feel comfortable with code written by any other member of the team.

2. Ricardo Devis Botella. *Curso de Experto Universitario en Integración de Aplicaciones Internet mediante Java y XML*. (University Expert Course in the Integration of Internet Applications via Java and XML) University of Oviedo, 2000.

5 Planning

While XP is a code centred method it is not just that. It is also above all a software project management method, in spite of the criticism levelled by many people, perhaps after a too hasty reading of an article such as this one. But for anyone who has taken the trouble to read any of the books explaining the process, it will be clear that planning makes up a fundamental part of XP. The thing is that, given that software development, like almost everything in this life, is a chaotic process, XP does not attempt to find a non-existent determinism but rather provides the means necessary to cope with that complexity, and accepts it, without trying to force it into constraints of heavy-weight or bureaucratic methods.

We wholeheartedly recommended you to read Antonio Escotado’s gentle introduction to chaos theory [Escotado 99], which we believe has a lot to do with the idea behind XP. In short, lightweight methods – and XP numbers among them – are adaptive rather than predictive [Fowler].

The life cycle

If, as has been demonstrated, long development cycles of traditional methods are unable to cope with change, perhaps what we should do is make development cycles shorter. This is another of XP’s central ideas. Let’s take a look at a chart which compares the waterfall model, the spiral model and XP:

6 Conclusions

As we said at the beginning article, XP, just one year after the publication of the first book on the subject, has caused a great furore among the software engineering community. The results of the survey below, commissioned by IBM, evidences the fact that opinions on the subject are divided [IBM 00]:

Pair programming comes in for some especially strong criticism – above all from project managers, though it is an opinion which is doubtless shared by many programmers with an over-developed sense of ownership regarding code (“I did this, and what’s more I am so good at programming and I have such a command of the language’s idioms that only I can understand it”), but a lot is also said about the myth of the 40 hour week, that “all this business about tests is all very well if you have plenty of time, but they are an unaffordable luxury under current market conditions”... and many other vigorous criticisms in a similar vein.

There are also people who say, (and this criticism is perhaps more founded than the previous ones) that XP only works with good people, that is, people like Kent Beck, who are able to make a design which is good, simple and, at the same time – and maybe precisely for that reason – easily extendable, right from the outset.³

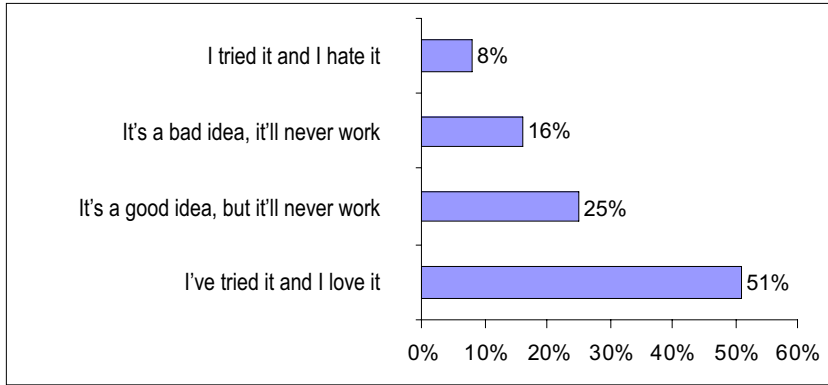


Figure 4: IBM survey (October 2000): What do you think about EXtreme Programming?

One of the things we are trying to say is that XP should not be misinterpreted due to the inevitable superficiality of articles such as the one you are reading. At the end of the day the creator of this method is not some upstart, but one of the pioneers in the use of software templates, creator of CRC files, author of the *HotDraw* drawing editor framework, and the *xUnit* testing framework. Were it for no other reason, it would be worth at least taking a look at this new and exciting software development method.

However, none of the practices advocated by XP are an invention of the method; all of them existed before, and what XP has done is to put them all together and prove that they work.

In any event, Beck's first book is a breath of fresh air which should be compulsory reading for any *software engineer* or *software architect*, to use the term preferred by our friend Ricardo Devis, whatever conclusions you may finally draw about XP. At the very least, it's great fun to read.

3. Raúl Izquierdo Castanedo. *Comunicación privada*. (Private communication)

References

[Beck 99]
 Kent Beck. Embracing Change with eXtreme Programming. Computer (magazine of the IEEE Computer Society). Vol. 32, No. 10. October 1999, pp. 70–77

[Beck 00]
 Kent Beck. eXtreme Programming Explained: Embrace Change. Addison Wesley Longman, 2000. ISBN 201-61641-6

[Beck/Fowler]
 Kent Beck, Martin Fowler. Planning eXtreme Programming. Addison-Wesley. ISBN 0201710919

[Escohotado 99]
 Antonio Escohotado. Caos y orden (Chaos and order). Espasa Calpe, 1999. ISBN 84-239-9751-0. An interesting introduction to chaos theory, which in our view describes the attitude you need to approach software development from an XP point of view.

[Fowler]
 Martín Fowler. The New Methodology. <http://www.martinfowler.com/articles/newMethodology.html>

[Fowler 99]
 Martin Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999. ISBN 0201485672

[Jeffries et al. 00]
 Ronald E. Jeffries et al. EXtreme Programming Installed. Addison-Wesley, 2000. ISBN 0201708426

[IBM 00]
<http://www-106.ibm.com/developerworks/java/library/java-poll-results/xp.html>. Java poll results: What are your thoughts on EXtreme Programming? IBM survey, October 2000

[IEEE CS]
<http://www.computer.org/seweb/Dynabook/Index.htm>. eXtreme Programming. Pros and Cons. What questions remain? The first “dynabook” from the IEEE Computer Society was devoted to XP, with a series of related articles.

[JUnit]
<http://www.junit.org>. JUnit, an automated testing framework for Java, adapted from the framework of the same name for Smalltalk, and available in many other languages. These other versions, under the generic name xUnit, are available at the Ron Jeffries' web site [Jeffries], in the software section.

[Jeffries]
<http://www.xprogramming.com>. One of the most complete XP portals, by Ron Jeffries.