Extreme Participation – Moving Extreme Programming Towards Participatory Design

Markus Rittenbruch, Gregor McEwan, Nigel Ward, Tim Mansfield, Dominik Bartenstein
CRC for Enterprise Distributed Systems Technology
University of Queensland, Monash University and University of Technology Sydney
Level 7, GP South
The University of Queensland, Qld 4072, Australia
+61 (0)7 3365 4310
{markusr, mcewan, nigel, timbomb, dominik}@dstc.edu.au

ABSTRACT
Extreme Programming (XP) is a lightweight software development methodology that has risen to prominence in the last few years. XP and Participatory Design are related in motivation and approach but complimentary in many ways. The authors believe that integrating some Participatory Design approaches into XP substantially improves XP and may even bring some advantages to Participatory Design. This paper summarises XP, compares the two approaches, outlines our experience with XP, draws out some problems with classic XP and suggests some modifications based on Participatory Design.

Keywords
Extreme Programming, User stories, Participatory Design

INTRODUCTION
Over the last few years a software engineering methodology, which has been breaking with several traditional paradigms, has emerged. *Extreme Programming (XP)* [2],[24] is based on four main values: *simplicity, communication, feedback* and *courage* and expresses the necessity to overcome rigid conventions that have accumulated within the area of software engineering over the last decades. It aims to make software development more flexible and focuses on highly flexible environments with quickly changing requirements.

From a participatory design point of view Extreme Programming is interesting for two reasons. First, Extreme Programming implements a highly user-centred approach. Users play a key role during the design process, specifying and designing the system in cooperation with system developers in a strongly iterative, prototype-based process.

In *PDC 02 Proceedings of the Participatory Design Conference,* T.Binder, J.Gregory, I.Wagner (Eds.) Malmö, Sweden, 23-25 June 2002. CPSR, P.O. Box 717, Palo Alto, CA 94302 cpsr@cpsr.org
ISBN 0-9667818-2-1.

Second, several principles of Extreme Programming assist software developers in producing software in a manner that is suitable for a rapid iterative approach. Some principles for instance help developers to overcome "release fear" and to avoid descending into details without consulting the user.

Over the last year, our research project, Information Ecology, has performed a prototypical software development process using an Extreme Programming approach. Our aim was to perform a participatory design approach while focussing on joint code production and the management of a distributed developer team at the same time.

Throughout the process we identified several shortcomings of XP with regard to user participation. Based on these problems and general considerations on the similarities and differences of XP and other participatory design approaches we extended our XP approach in order to represent user contributions in a more complete manner.

Within this article we are going to address three main issues, specifically we will:

1. identify the similarities and differences between Extreme Programming and participatory design approaches. Although Extreme Programming is rooted in another research tradition there are several interesting resemblances. The comparison is a prerequisite for the modification of XP towards a more complete user participation.
2. consider which potential benefits the application of XP could have in the context of a participatory design process. The impact of code production on the whole design process has rarely been addressed within the field of participatory design. We will point out several aspects of XP that will help to perform an effective iterative prototyping approach.

3. describe the modifications to our XP approach showing a possible way to integrate XP and selected participatory design methods.

Overview

The XP process gives an overview of the XP methodology, its history and the challenge it poses to traditional software engineering.

Participatory Design and XP compares XP with several participatory design methodologies. It points out conceptual weaknesses of XP in the context of participatory design and identifies possible contributions of XP to a participatory design process deploying iterative prototyping.

Our System – The Social Portal describes the system we built and describes the original design process as a way of introducing our experience of XP and our motivations for modifying the process.

Problems And Extensions Of The XP Process focuses on the problems that we encountered during our design and suggests extensions to the XP process to overcome these problems and other conceptual weaknesses of XP.

A New Design Phase describes the new design phase we performed based on the modifications on the XP process.

The final chapter sums up the paper and describes our future research.

THE XP PROCESS

This section provides a high level description of the Extreme Programming (XP) methodology for developing software. Our description of XP is based primarily on [2] and [24]

Our intention is not to provide a complete description, but to provide a context for this paper by discussing some of the broader points. We begin by describing some of the philosophy and motivation for Extreme Programming. The next section provides a sketch of the XP development process. Finally, we contrast XP with more traditional software engineering approaches.

Goals and scope of XP (development for dynamic environments)

Extreme Values

One of the key slogans of Extreme Programming is to "embrace change". The four fundamental values of XP, simplicity, communication, feedback and courage, are principles to enable the team to be constantly in touch with and responsive to a changing environment. The source of the change is the constant contact with the user as their concept of the system requirements evolves. XP is designed so that the software can evolve to match the requirements.

Simplicity in XP has two aspects. First there is simplicity in the process itself. Having a simple process means that it is less work for the development team to maintain the practices of XP. Secondly there is simplicity in implementation. In more traditional software engineering emphasis is placed on writing code that is easily extensible to future requirements. Promoting simplicity first means that the resulting software is easy to understand and reduces the time spent on extensions that may never be needed.

XP emphasises strong principles of *communication* within the development team and also between the team and the user. Communication within the team is achieved through frequent planning and design meetings, through sharing around development tasks, and through short daily updates. In this way the knowledge of the system and an awareness of the development process is shared throughout the team. Communication with the users is represented by the role of a customer in the team. A delegate of the user community, which XP calls the *customer* has an active part in the requirements and design of the project. She works in a close relationship with the development team, and takes a major role in planning development.

Feedback at all levels is an integral part of XP. Extensive test suites provide feedback on the software code. Speed of development and the accuracy of project estimates are constantly reflected on and revised. The on-site user representative provides feedback on how well the requirements are fulfilled.

The fourth fundamental value of XP, *courage*, is a call to trust the process. Designing and implementing only the immediate requirements and not thinking about future "possibilities" requires courage from the development team. It also refers to the courage needed to revise existing work extensively in the face of new requirements.

A major feature of XP is the constant feedback and evaluation of the software by users. Having user representatives (customers) actively involved in generating the requirements of the system as well as being a part of the development and planning process results in a very dynamic environment. The customer's concept of the system is constantly revised as they have constant feedback from the implementation.

XP methodology consists of only a few rules and practices that involve little effort to developers. In contrast to many other methodologies, XP makes very minimal use of design documentation (for reasons that will become clear). This lightweight process means that the process can respond quickly to the evolving requirements.

Extreme History

The roots of Extreme Programming lie in the Smalltalk community. Kent Beck and Ward Cunningham originated research on more flexible and agile development methodologies in the late 80s. They refined their informal practise on many projects in the early 90s. A coherent XP process was first performed in 1996 when Kent Beck applied a combination of informal practices to a project at Daimler-Chrysler. Since then, many aspects of XP have been refined and the methodology is continuously evolving.

XP building blocks

This section gives an overview of how XP works. We approach this by sketching the planning and design process and then by explaining the roles of various actors in those processes.

XP is usually explained by describing the practices and rules that should be followed. These may be applied to a development process piecemeal to make it "more extreme". A team must implement most of the practices (there is active discussion about exactly which practices are in that set) before it can claim to be "doing XP".

We have avoided describing most of the practises in detail to keep the description reasonably concise.

Planning and Design

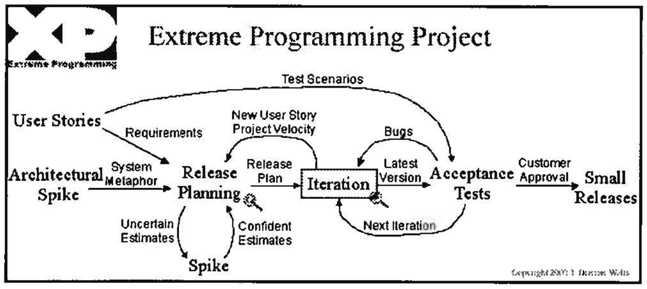

Figure 1 An overview of the XP planning and design process (from www.extremeprogramming.org)

At the heart of the XP planning process are *User Stories*. User Stories are two to three sentence informal descriptions of feature requests or desired working situations written by the customer. These short descriptions form the basis for planning development.

The development of a project is broken up into a series of small *releases*, further divided into *iterations*. An XP release cycle takes two to four months. Iterations take about one to four weeks each.

At the start of each release, the team produces a *release plan*. The release plan consists of the most important remaining User Stories. Selecting the most important User Stories for the release is the job of the customer. The developers provide time estimates for the Stories. For each release, the resources available and the quality of the product are fixed. The team, including the customer, decide on a fixed value for either the schedule or the scope of the project. They decide how long they will take or how much to do, but not both.

This collaborative planning process, with roles for both customer and developer and rules to follow is called *The Planning Game*. Customers and developers "play the game" to decide on what can be included in both releases and iterations.

During the planning game, users and designers select and prioritise user stories by different criteria. Users sort the stories by *value* into three piles:

1) vital for the system to function,
2) less essential, but of good value and
3) nice to have.

Designers sort stories by *risk* into three piles:

1) stories for which precise time estimates that can be provided,
2) stories that can be estimated reasonably well and
3) stories that can not be estimated at all [2].

The game enables the creation of the release plan that contains a candidate set of User Stories, which are both important and achievable in the time available.

At the beginning of each iteration, the customer chooses a smaller subset of those stories that could be achieved in the timescale of an iteration. This forms the *iteration plan*.

The high-level feature descriptions in the User Stories are broken down into specific engineering tasks. This is the point where most of the system design takes place. The existing design is modified to incorporate the extensions required by the iteration User Stories. Iterations are deliberately kept small so that the customer has frequent opportunities to evaluate and provide feedback.

After the completion of each iteration, the system is presented to the customer for evaluation. The customer checks to see that the User Stories selected for that iteration have been implemented satisfactorily. Any Stories that do not pass this testing are fed back into the process for selection in the next iteration.

At the conclusion of a release, as the name implies, a version of the system is released to the client organisation for feedback.

The Team Roles

XP makes five roles explicit within the team. Each team member may have more than one role and each role may have more than one person. The five roles are *programmer*, *customer*, *coach*, *tester* and *tracker*.

The *programmer's* primary role is, naturally, to program. To encourage quality and communication, XP defines some extensions to common programming practices. Firstly, *unit test suites* must be written before the code. This helps to assure quality and also communicates to other programmers the intention of the code. Secondly, programmers engage in *Pair Programming*; working in pairs, with one person looking over the other's shoulder. In this way the code undergoes a peer review as it is written. Possibly the most important benefit of pair programming is that, as pairs are swapped around, team members acquire knowledge of different parts of the system.

All phases of the XP process involve the *customer*. Initially, the customer writes the User Stories that are used in planning. Customers select the User Stories, which they wish to have implemented for the next system release. Next the customer collaborates in writing *acceptance tests*, which define the correct implementation of each User Story.

After the specified Stories have been implemented, customers take part in acceptance testing. Acceptance testing establishes whether the requirements have been accommodated by the system. In the event of customer dissatisfaction, the Story is placed back into the pool of User Stories available for selection in the next release so that the customer can prioritise its completion.

The customer's role is not limited to the planning and testing phases. User Stories are not intended as standalone descriptions of requirements, so the customer is in continuous communication with the developers during design and implementation to provide clarification of the stories.

The customer writes acceptance tests in collaboration with the *tester*. The tester is responsible for acceptance testing at the end of each iteration, running the unit test suites (usually daily) and communicating the results to the rest of the team

The *coach* is responsible for monitoring the team's use of XP. The coach needs to be aware of the process and be able to alter the process or the team if something is not working. Monitoring the process also means making sure that the fundamental values and the practices are being followed.

The *tracker* tracks the progress of the development, communicating the actual speed of development in relation to the estimated times on the User Stories. This information is fed back into the estimation process in future planning sessions. The tracker also produces forecasts of the release schedule during development.

Comparison with other software engineering methods

XP takes several practices from more traditional software engineering methodologies, and makes them "more extreme". More extreme methodologies are less formal and more tightly integrated into the implementation process. They are also performed only as they are needed and only as much as is needed for the immediate task.

Listed below are a number of traditional software engineering practices. We will discuss each of the practices in comparison to the corresponding practice in XP in order to give an overview of the XP philosophy in the light of proven techniques.

Requirements Analysis, Specification and Design

In contrast to the formal documentation and "contractual" communication between customers, designers and programmers in traditional software engineering, XP uses a continuous and informal "conversational" style of communication. The customer provides the designers with User Stories, which are deliberately vague. Designers, customers and programmers are in constant, continuous communication to resolve the details of what the customer requires.

The conversation works both ways. The system is designed and implemented in very small portions and then shown to the customer. The customer is able to modify and extend the requirements as their understanding of the system evolves. To allow quick response to the changing requirements, only a few User Stories are being implemented at one time and the design is as minimal as possible to just cover the current User Stories. The system design is then extended and refined as necessary to implement new requirements.

Code review.

In a traditional software engineering process, Code Reviews are conducted by one or more members of the development team. The reviewers read through the program code that has been written and refine it within the context of the project. The code is reviewed for structure, clarity and documentation as well as its adherence to project style guidelines and architecture. In addition to improving the quality of the code, code reviews also assist with training new team members into the team's coding style. Where the whole team performs the code review or the responsibility for review is moved around the team, it also serves to communicate knowledge of the project throughout the team.

The XP adaptation of code review is pair programming. The pair programming practice specifies that two of the developers work on the same coding task at the same time. One member of the pair types the code while the other sits behind them. Working in a pair provides the benefits of code review as the code is being written. The developer that is not typing is free to consider the context of the programming task. While appearing to be wasteful of programmers, the time is compensated for by the improved code quality at the time of writing. Pair programming has been experimentally validated as speeding up development time [7].

Testing

Traditionally, testing is a phase of development that is carried out after the main coding effort. Often the testing is carried out by a specialised tester who is not one of the programmers. Test cases are designed to cover as much of the logical functionality of the code as possible. Test cases are implemented to call the relevant sections of the code and check the output. Testing checks for logical errors in the code.

Testing in XP fills the same role as in other software engineering processes. The change that XP makes to testing is to require the programmers to write the tests before the code. Every time new code is written on the XP project, a corresponding test case must be written and implemented first. In this way the tests are written in an iterative fashion in parallel with the code by the person that is writing the code. The advantages of the XP methodology are that the practice is less onerous on the programmers, the tests are developed while the context is still fresh in the programmers mind, and there is constant feedback on the state of the code as tests can be run at any stage of development.

Integration Testing

A large system will usually be broken up into several sections for implementation. At some point these pieces have to be integrated to construct the whole system. Integration testing helps to check that all the pieces fit together as intended. The tests ensure that the system as a whole fits the specification. Often integration testing is the most time consuming stage of development.

XP practice is to continuously integrate. As a programmer makes a change to the code, it is integrated with the system daily. Integration and integration testing are incorporated into the development. This means that the testing is performed more often and because it's part of the development of a new feature, happens in the context of the addition. The integration is also performed in smaller chunks of effort.

In summary, XP is a lightweight process. In heavyweight methodologies, development is split into separate stages; requirements analysis, specification, design, coding, testing and integration testing. XP practice rolls all these into one, continuous stage. Everything is done only when it is needed.

The comparisons above do not show all of the differences between XP and more heavyweight software development processes. However, the major areas of software development are discussed to give an overview of the practical differences in Extreme Programming.

PARTICIPATORY DESIGN AND XP

On the first view, comparing XP with participatory design approaches seems like comparing apples with pears. XP is a software engineering methodology that is concerned with classical software-technological issues like code quality, interaction of developers, release planning, etc.

Participatory design is a research field, which has brought forth a huge variety of methods and approaches to realise and facilitate the integration of users into different levels of (software-) design processes. Most participatory design approaches are concerned with the process of design itself. They describe among other things, how design teams are set up, the different roles users play within the design process and the methods (e.g. scenarios [5],[1], mock-ups [9], games [10], ethnographic approaches [8], prototyping [4]) that are used to establish communication between different actors within the design process (e.g. users, designers, facilitators).

The field of software engineering on the whole, has been overall more reluctant to acknowledge the important role of the user within the design process. Even though the emergence of evolutionary and cyclic development methodologies like Floyd's STEPS model [14] and Boehm's spiral model [6] emphasized the strong necessity of user-participation, software engineering as a discipline has embraced user participation rather hesitantly. In many software engineering approaches the understanding of user participation is still often limited to requirement engineering, leaving users only a marginal role within the actual design process.

XP as a new software engineering approach has challenged several traditional paradigms of software design. One of these major changes is the strong focus on user participation. If we take a closer look, XP shares a number of similarities with participatory design approaches in general. It implements an iterative, prototype-based approach, integrating users on different levels of the design process. User representatives (customers) describe their requirements in a non-formal manner (user stories), decide about the implementation of components of the system (release and iteration planning) and judge whether certain aspects of the system have been implemented satisfactorily (acceptance test). The whole process is performed in a strongly iterative manner implementing rapid prototyping and continuous user involvement.

We see the emergence of XP as a possibility. Though incomplete with regard to user participation, XP offers insights about the way software-developers cooperate, that could help to further integrate the work of designers, users and software developers. The following chapter takes a closer look at the relationship between XP and participatory design approaches and will discuss their mutual influence. We are particularly interested in two issues:

1. What are the limitations of XP with regard to user participation? Can XP be enhanced to reflect a broader understanding of user participation based on participatory design approaches?
2. How does the process of software coding as it is performed within XP influence the design process as a whole? Which aspects of XP can be beneficial within a participatory design process?

In the following we will relate XP to some participatory design methodologies on a conceptual and methodological level, we will discuss shortcomings of XP in comparison to participatory design methodologies and discuss the question of what XP can offer within a participatory design process.

Conceptual and methodological similarities

We will consider four aspects of XP in the context of participatory design, user involvement, user stories, release and iteration planning and finally the XP prototyping approach.

In order to compare XP with participatory design approaches several notational differences need to be considered. XP originally represents users only in a representative manner within the customer role. We will henceforth use the term "user" if we talk about all potential users of a system in general and the term "customer" if we refer to particular aspects of the XP customer role. While participatory design approaches normally refer to the people driving the process as designers, XP talk about developers or programmers, due to the software engineering roots of this approach. Although representing different responsibilities the roles of the designer and the developer within the XP methodology intersect. In order to facilitate the comparison we will use the term "designer" from now on, unless we are referring to pure programming in which case we will use the term "developer".

User involvement

Within a participatory design process there are several levels of user participation and user selection. Depending on the size of the organisation, and the limitations of time and resources, the number of users participating actively or passively within the project can vary greatly. During workshops users representing different departments, or different organisational roles might be chosen. In workplace studies and ethnographic approaches the observer will choose work situations that are suitable to represent a wide range of the working context.

XP is rather unspecific about the selection process for the customer role. In general the customer is a user that is supposed to work in the area for which the software is developed. XP is not specific about the number of customers that are appropriate for a particular design process. One customer seems to be common in many projects, but any number of customers is possible. A certain shortcoming of XP is that it doesn't rule out the possibility that organisations nominate a person who might have a general view, but not a detailed view on particular working contexts, as the sole customer. In general, an appropriate user selection is necessary for both participatory design and XP approaches.

User stories

User stories are one of the main building blocks of the XP methodology. To revise their role, users (customers) describe in their own words "what they want the system to do" [24], with unstructured stories of about 3 sentences. User stories are often written down and represented on cards. They are used within the XP process in a threefold manner. First, the totality of user stories represents the informal, continually evolving user requirements for the system. Second, user stories are important elements used during the planning game. Developers estimate the amount of programming time for each release based on the user stories. And third, user stories are used within acceptance tests where users specify conditions to decide whether a user story has been implemented to their satisfaction.

User stories have similarities with several methods found in participatory design. Erickson uses "stories" as a means of communication between designers and users [12],[11]. Stories can be designer stories that reflect the designers experience from prior projects or people's stories that hold information about what people do and think about their work. In comparison to user stories as used in XP, people's stories in Erickson's sense are gathered and interpreted by designers.

Scenarios[1] are commonly used in participatory design [5]. They are similar to user stories in that they describe both situations of system use and system functionality in a non-formal manner. The main difference between scenarios and user stories is, that scenarios often describe a whole working situation covering several work practices and the use of a system within a work context. User stories in comparison are more fragmented and focus on singular use situations.

The process of creating and using user stories also differs. Scenarios are normally created in cooperation between designers and users and are often used in the initial stages of the design process. User stories are mainly utterances of users, created without direct contribution of designers. Consecutive versions of user stories are used during the whole lifecycle of the design process.

[1] we are not referring to scenarios as they are found in object-oriented design, e.g. [16]

Furthermore, a number of similarities between user stories and techniques used in object-oriented modelling can be identified. Methods like use cases, use case diagrams [16] and CRC cards [3], can represent information similar to user stories, though in a more structured manner. CRC cards may be used within an XP process to represent certain programming tasks among the system developers [24].

The use of user stories as a method of communication between users and designers has several potential benefits:

- User stories are small and easy to write. There are not many prerequisites for users to write user stories. User stories can cover several aspects of the development process, ranging from support for a particular working situation, to new necessary features or the improvement of existing functionality.
- User stories are a communication channel between particular users and designers. Designers can ask users for clarification if they do not understand the situation described in the user stories. Users and designers furthermore cooperate in order to define the acceptance test for a particular user story by which users define conditions that help designers to implement close to user needs.
- User stories are integrated within the design process. They are used during several levels (specification, release planning, iteration planning and acceptance tests) of the design process and fulfil multiple purposes.

XP does not articulate exactly how the designer and user interact in making design choices for the implementation of User Stories. Informal discussions before the customer prepares User Stories may be used to raise novel possibilities and discussion of the User Story to clarify its implementation during an iteration are both places in which designers may provide design options.

Although experienced designers may use the process to contribute options and facilitate innovative design, the lack of any practise to support this makes it an *ad hoc* modification. This may mean that less experienced or assertive designers may feel that the customer should carry out the task of specifying the system unaided. This in turn is likely to cause common problems like the "digitalisation of the status quo"

Planning game

As we discussed above, the planning game embodies the tension between what the customer wants and what the developers can deliver by allowing each of them to order the User Stories by value and risk respectively.

This sorting used in the planning game is similar to the use of CRC cards [3] and other card sorting games with the difference that users and designers use card sorting to express their different priorities for the development process.

Although the XP planning game utilizes several roles, it is rather different from games and role-playing found in participatory design (eg. [10], [18], [13]). Games, like the one described in [15] represent the environment in a more complex and playful way, letting the users explore working environments interactively. By contrast, the XP planning game serves only to resolve planning tensions.

Prototyping

The iterative prototyping approach of XP serves two main goals: first, to continuously involve users into the design and evaluation of the system, and second, to overcome release fear by releasing prototypes as early and as frequently as possible. This approach in general resembles many iterative design approaches found within participatory design. Users are participating on different levels of the design process.

Prototyping in XP can be classified as evolutionary prototyping (the system is evolving based on several iterations of a prototype) rather than throwaway or low-fidelity prototyping (a system that is only used for a limited time to demonstrate a particular state in the design process) [20], [22]. Although commitment to code quality enables developers to discard chunks of code during the design process this rarely relates to the prototype as a whole.

The role of the user-customer as part of the design team and the strong cooperation between designers and the user-customers meets Bødkers requirement of "cooperative prototyping" [4].

Summary

It is difficult to directly compare XP with participatory design methodologies in general, due to their different scopes. Still, there are similarities that are mainly grouped around the areas of prototyping and representation of user requirements. The XP prototyping approach is highly iterative and strongly influenced and driven by user decisions (based on User Stories and the planning game). User requirements are represented in a manner can be understood and shared by users in User Stories.

Although User Stories are the main means of communication between users and developers XP does not rule out the use of additional methods such as mockups or scenarios to further clarify requirements. Such methods are always meant to facilitate the communication, but never replace User Stories.

A potential danger of the XP process is the strong focus on selected user representation in comparison to a broader involvement of end-users. The process of selecting user representatives itself is not well specified within XP and is primarily based on the selection made by the customer-organisation

The following section will go into further detail regarding the shortcomings of XP in comparison to participatory design approaches.

Shortcomings of XP in the context of participatory design

Considering XP in the context of participatory design the following aspects of XP could potentially cause problems:

- XP does not support design in context. Users are represented by the customer role. The actual level of intersections between the user needs and the requirements the customer(s) formulate is not validated.
- Workplace studies are not a part of an XP process. XP does not provide the means to integrate results of workplace studies into the process.
- Customers are constantly exposed to the development process. It is likely that they start to identify with development-related problems, potentially losing their focus on user-related issues. This aspect in combination with the former points increases the danger of "tunnel vision" or "coming up with perfect technological solutions to the wrong set of work problems" [8, p. 93]
- Possibilities for the designer to influence the design process are only vaguely defined. On the one hand designers are not meant to interfere with the production of user stories. On the other hand designers lack appropriate practises to integrate design aspects into the process and to facilitate the customer role by providing different design options.

The use of XP within a participatory design process

Many participatory design approaches comprehensively revise the design process. Surprisingly though, the process of programming itself within the software design process is rarely looked at in this context. We are interested in the question, how the act of programming influences the design process and which aspects of coding have to be considered in general in order to support a participatory design process. Since XP is supposed to be suited to flexible, often changing environments, it needed to find ways the change code quickly and efficiently. XP implements 4 main rules to ensure this flexibility:

1. **Common code ownership** Code belongs not to a single developer but to all developers in the team. This ensures that code is produced in a comprehensible manner since all developers of a team have to potentially understand it. Comprehensibility makes it possible to change the code quickly even if the originator of the code is not available.
2. **Pair programming**: Pair programming is another step to ensure that the code is mutually understood by several developers and helps to ensure code quality.
3. **Commitment to code quality**: XP requires a high level of code quality. Sound solutions that solve a problem in an aesthetic manner (in the sense of elegant logic rather than an attractive user interface) are preferred to "quick hacks". This rule does not conflict with the following one, since good solutions are rarely big or complex.
4. **Do the simplest thing possible**: Developers are meant to start with small and simple solutions that solve the problems that the current iteration raises. This rule contrasts with approaches that start with huge conceptual models and architectures and often struggle to deal with the related complexity. In XP the complexity of an implementation can increase over several releases, but is always rooted in simpler approaches that have proven to work.

The above rules document a certain perspective on code production that can influence the whole design process positively. We see possible benefits of XP in three areas: *speed, strong iteration* and *code quality*.

Speed: Code that is easy to change enables developers to implement new requests quickly. In addition, features that did not pass the acceptance test can be discarded efficiently.

Strong Iteration: Rule No. 4 in particular in combination with the planning process helps developers to overcome "release fear". A prototype is presented to the users even though it has minor or major flaws. XP developers can produce systems in a strongly iterative manner with short cycles between releases. Consequently, users can access succeeding versions of prototypes quicker. A quick succession of prototypes ensures that the development process stays dynamic and helps to prevent developments into the wrong direction.

Code Quality: Potential user dissatisfaction is not only caused by the mismatch between user's needs and the systems functionality, but potentially also by faulty code that leads to errors. Pair programming and commitment to code quality lead to software that is less prone to errors and generally increases the utility of the software.

OUR SYSTEM - THE SOCIAL PORTAL

This section describes our initial experiences with XP which motivated our interest in modifying the process as stated in the literature [2]. We describe the system we were developing and our approach to designing it, the way we involved users using XP and some of our difficulties with the classic process.

The Idea Of A Social Portal

One of the goals of the Information Ecology project at DSTC is to enable software to better exploit the broad context (both within the computing environment and beyond it) of the execution of a user command. Although several kinds of context are initially appealing, we initially set out to study just one kind: the patterns in people's social interaction with each other.

Our initial approach was to gather that information by providing an application which supports communication with a list of contacts and use that as a way to capture information. To be a useful research tool, we needed an application which people will use for a significant proportion of their communication, therefore it had to:

- serve a known need
- be more effective than existing solutions
- be primarily web-based to minimise the barriers to adoption.

These criteria led us to the idea of a "social portal".

Portal sites such as My Yahoo! or Lycos or My Netscape attempt to pull all information of interest to the user together in one place. This information is usually organised as channels of information on some topic. Users can typically personalise the channels they see in a portal. For example, users can choose to see channels from newswire services such as Reuters alongside stock portfolio channels, TV listing channels, horoscopes, weather, and so on.

The Social Portal allows portal-style presentation of information from social networks. Rather than solely relying on general channels that may meet the user's information needs, we built a system that also uses social context to recommend information.

Individuals can use the system to send messages to social contacts such as colleagues, friends and family. The receivers of this information can personalise the portal to see the contributions of their friends alongside traditional portal channels.

Initial implementation

The initial system was based on the common portal metaphor of an online newspaper. Rather than receiving news items from a wire service like Reuters in this newspaper, users would receive recommendations of web pages from other users. Like many online newspapers, the Social Portal organises items into channels, one channel for each topic contributed by a given user.

The reverse-side of this design is that each user can make up channels about topics they would typically share and send recommendations to friends or colleagues using these channels.

So, the initial version was based on these two basic notions: recommendations and channels.

Recommendations had a title, a URL (for the thing being recommended), a description and a sender and date. Users soon worked out that they could omit the URL and just use recommendations to send a text message as a news item.

Channels are a conduit from their sender to a group of receivers. Each channel associates a set of recommendations on a common topic (in the opinion of the sender) with a set of receivers.

Each receiver had a page consisting of all the channels they had been sent. They had the means to rearrange the order of the channels in a two-column layout.

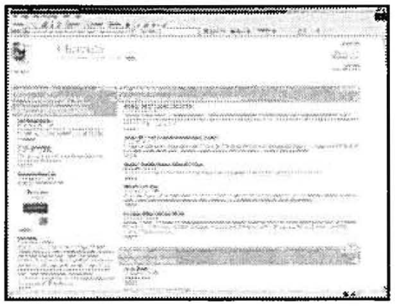

Figure 2 The main page of the social portal

Original User Involvement - Simple Adaptation of XP

We choose XP as a development methodology for several reasons:

- Our programming resources were limited. As a research project we needed a methodology that supported the effective creation of subsequent prototypes at low cost.
- Our developer-base was distributed. Having developers in three different sites distributed over Australia increased the necessity to rely on a methodology that supported cooperation and a a shared process among developers.
- The user-base was distributed. We initially planned to deploy the system at different sites of our organisation and in the long term to make it available to a wider user community on the web. Some of the traditional styles of user involvement were not suited for such a setting. XP seemed to be flexible enough to be adapted for this task.

The goal of our project was to build an application that we could use to capture information about computer-mediated social interaction. We needed an application that a lot of people would use regularly, in place of their existing methods of sharing information through their social networks.

This goal is too broad and exploratory to be translated directly into the User Stories necessary to start the Extreme Programming process. Additionally, we wanted to collaborate with our user community to evolve the vision for a useful social portal system. However, the people who would use the software defined that user community. It did not exist without the software leaving us with a "bootstrap problem"

We overcame these problems in a number ways:

- **ad-hoc customers:** we convinced some members of our initial target deployment community to play the customer role in our design team
- **bootstrap version:** we synthesised an initial version of the system to stimulate User Stories from our customers

The initial target deployment group for the application was our own organisation. One customer was from our organisation's business development section, and one from another research project. Our *ad hoc* customers were asked to use our initial system and write User Stories about how they would like the system adapted.

PROBLEMS AND EXTENSIONS OF THE XP PROCESS

We encountered a number of problems during our initial design phase. Although the customers were satisfied with the system, other users found it difficult to use. It became apparent that the original XP process as we had performed it so far, was too focused on customers as user representatives. In order to overcome this problem and to adapt XP to a broader understanding of user participation we extended the XP process in several aspects. All of the modifications to the process are reflected in additional roles, which add different responsibilities to the design process. In the following we will describe the problems we encountered and the proposed solutions.

Lack of design in context

XP lacks an overall sense of *design in context.* The main reason for this is that the main communication channel for user requirements are user stories. Users might choose to describe working situations and their work context, but they might also be quite focused on pure system functionality. The methodology has no means to ensure that the working context is taken into account. In the sense of [18] it's located on the scale "users directly participate in design activity" but lacks the aspect of "designers participate in users world".

It is obvious that there is an abundance of methods within participatory design, concerned with the understanding of the context the user works in (ethnographic approaches, workplace studies, role-playing games, etc.) The question is how information that can be gathered by using one of these methods can be integrated into an XP process. How does information about the use context influence the XP based design process as a whole?

In order to answer this question we have to focus on the planning game, which is the central hub for how requirements are rolled into the XP process. As we have pointed out before users (customers) have a major influence during the planning game, deciding which aspects of the iterative prototype are supposed to be implemented. In order to introduce findings that have been gained by user-studies we had to introduce another role that represented these aspects during the planning game. Integrating results from user evaluation should also help to overcome the problem that arises when customers have been part of the design team for too long and become "professional customers". There is a potential danger that customers identify themselves with the design process so much that they increasingly loose track of problems and requirements that might be relevant for other users who are not involved in it.

We encountered this problem when we realized that, although the customers seemed to be content with the prototype at that stage, the system seemed to be increasingly difficult to use for new users. New users when confronted with the system reported an overall lack of guidance and help throughout the system. It became obvious that the original customers involved in the design process had become increasingly unaware of problems that new users might encounter. As a consequence these problems had a lower priority and were not addressed sufficiently within the design process.

In order to integrate information about working situations and users that were not represented within the process we did two things:

First, we opened up the process of writing user stories to the whole user community rather than the customers representing a small fraction of users. We provided an **electronic feedback** form that was part of our prototype and enabled users to write user stories whenever the encountered a problem or had a specific requirement. The gathered user stories were integrated into the design process and became part of the planning game. Electronic methods for gathering user feedback become increasingly important in environments where organisational structures become more flexible and works happens in an increasingly distributed manner (see e.g. design processes in networked or virtual organisations [23], [21])

Second we introduced a role called the **user-evaluation customer** to the planning game. The user-evaluation customer has the same rights and obligations as other customers during the planning game. He represents user requirements that have been gained by studying users in their work environment[2]. The results are broken down into

[2] In our design process the user-evaluation customer represented the result that had been gained by constructive-interaction sessions and new-user evaluations

user stories, that are merged with the pool of existing user stories. The user-evaluation customer is meant to represent the user-community based on the user-studies within the planning game. The negotiation between the user-evaluation customer and the user customers (or customers in traditional XP) ensures that different user needs are represented within the process.

Intelligibility of user stories

Another problem we encountered was the abundance of user stories. Since all users could contribute user stories electronically, the pool of user stories was growing rapidly after the first few iterations. While having a large variety of user requirements is desirable in general, we realised that customers became more and more overwhelmed with the amount of new user stories. Especially when new customers were integrated into the design process, they found it hard to gain an overview of the existing stories. To this point user stories had been loosely classified into categories and identical stories were merged.

In order to use user stories more effectively we introduced a **gardener role** into the XP process. The gardener's task was to maintain the user stories in several aspects. First, she was meant to keep the stories current. Since the prototype continuously evolved, several of the user stories expired or their focus changed. Second, she was supposed to clarify user stories with the user who has written the respective stories if the stories were difficult to understand for other customers. Third, user stories were merged or split under participation of the respective customers, if they were dealing with a very similar aspect or covering several aspects respectively. And last, the gardener could add additional material (e.g. paper-based mockup) to make user stories more intelligible to other customers. This procedure was performed in close relationship with the originator of the stories as well.

The overall aim for the gardener was to reduce the amount of user stories, to keep them well structured and current and to enhance their intelligibility. The gardener did not have an active role within the planning game. As an expert for the existing user stories she was present during the planning game, acting as a facilitator in order to clarify questions regarding user stories.

Design vision

The last role we introduced relates to the insufficient role of designers within XP processes. The strong role of customers during the planning game and their independence in writing user stories leads to a lack of possibilities for designers to present their suggestions and take an active part in the process. In order to overcome this problem we introduced a **design customer role**, which enabled designers to take an active role during the planning game. The design customer has equal rights and obligations to the other customers during the planning game. Design suggestions are introduced by designer stories which are written by designers. Similar to user-evaluation stories they are merged with the pool of existing user stories. Design customers negotiate during the planning game with customers and user-evaluation customers about which aspects of the iterative prototype are supposed to be implemented in the next iteration/ release.

Summary

The enhanced XP methodology as it is proposed here comprises three new roles, the user-evaluation customer, the design customer and the gardener. Users (via electronic feedback), user customers, user-evaluation customers and design customers all write User Stories. All roles acting during the planning game can choose the relevant stories freely from the resulting story pool.

A NEW DESIGN PHASE

In the following we describe how the modifications to our XP approach were imp lemented in our design process.

User evaluations

In order to integrate a wider user base into the design process we performed several user evaluations. The results were used by the *user-evaluation customer* within the design game to represent the needs of users who where not represented within the game. We performed two sets of user evaluations, evaluations of new users and evaluations of a broad user base using constructive interaction methods.

New-user evaluations

As we have pointed out before, new users encountered an increasing amount of problems during system use. In order to understand the related problems with this particular user group we performed user evaluations on 10 users who hadn't used the system before. We used thinking aloud [19] and semi-structured interview methods for the evaluations. Each user sat through a half hour session performing several tasks that became increasingly complex. The tasks reflected the functionality of the system. The users were asked to utter their thoughts during system use. Each session was followed by a semi-structured interview covering the usefulness of the system and particular problems that had been encountered during the preceding session. The sessions were videotaped, transcribed and analysed. The result covered a wide area of the system from lacking help functionalities to non-intuitive page design. The result were broken down and represented as separate user-stories.

Cooperative user evaluations

The second set of user evaluations was focussing on the

(cp. chapter User evaluations)

fact that the system was mainly used for cooperative purposes. The test setting reflected this by using "constructive interaction" as evaluation method. We employed Kahler's variety of constructive interaction CITeCS [17] since it strongly focuses on collaborative tasks. The user base was a cross-section of all users within our organisation including people from different departments (accounts, administration, research, training, etc.) and different use-experience. We performed five constructive interaction sessions followed by a semi-structured interview. All sessions were videotaped and transcribed. The results were treated the same way as the results from the new-user-evaluations and represented in user stories.

User-evaluation stories

The user-evaluation customer used the combined user stories from the new user evaluations and the constructive interaction sessions to represent particular user needs. The user-evaluation customer rated the user stories by relevance (how often did the problem/wish occur ?) and urgency (how pressing was the problem/wish ?).

The new planning game

Within our new planning game we had five different main roles:

- Two user customers as user representatives, who were the original customers from the prior design phase;
- One user-evaluation customer;
- One design customer;
- The gardener was present in order to help with questions regarding user stories, but had no influence on the decision process;
- The developer team.

The four customers negotiated which stories to focus on during the next iteration or release. The different types of stories (user stories, designer stories, user-evaluation stories) built a common pool for all customers to choose from. Customers selected user stories that were not necessarily their own stories and tried to build a consensus by identifying related problems and needs.

CONCLUSIONS AND FUTURE WORK

XP is an emerging new methodology that is likely to be used increasingly in software development projects over the next few years. We compared XP with participatory design approaches and pointed out, that XP has a limited conception of user participation. Although users play an important role during the design process, XP lacks means to integrate a wide range of users into the design process and to perform "design in context".

Based on our experiences with XP we enhanced the XP process with the intention to firstly open it for the application of participatory design methods and secondly prepare XP as a possible software development method that could be used within participatory design processes. The methodology was extended by several roles, which reflect problems that were motivated by conceptual comparison with participatory design approaches as well as by the results of our empirical studies.

The consideration of XP in the context of participatory design poses interesting questions regarding the relationship between participatory design and software engineering in general. We addressed a small range of questions such as :

- How far does the culture and attitude of programmers influence the whole design process?
- Which programming practices are beneficial for user-centred design process?

We identified aspects that speed up the design process allowing for more frequent prototyping (common code ownership, pair programming, etc.) as well as exposing developers to an ongoing communication with users. These practices seem to be steps in the right direction, although further research needs to be done in this field.

Our research project on social portals is ongoing. We are currently preparing a new release of our system intended to support applications in several organisations external to DSTC. The additional measures and roles by which we enhanced our process have proven to be beneficial so far. User-customers appreciated the increased intelligibility of user stories treated by the gardener and preliminary user-studies have shown that the system has become more usable for new users. Further research is necessary to explore the applicability and limits of an extended XP approach.

ACKNOWLEDGMENTSThe work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

REFERENCES

1. Bardram, J., Scenario-Based Design of Cooperative Systems. in *COOP ' 98*, (Frankreich, 1998), Aathus University, Denmark, 57-66.
2. Beck, K. *Extreme Programming Explained: Embrace Change.* Addison Wesely Pub. Co., 1999.
3. Beck, K. and Cunningham, W., A laboratory for object oriented thinking. in *OOPSLA*, (New Orleans, Louisiana, US, 1989), ACM Press, 1-6.
4. Bødker, S. Cooperative Prototyping - Users and designers in mutual activity. *International Journal of Man-Machine Studies*, *34* (3).
5. Bødker, S., Scenarios in User-Centred Design

setting the stage for reflection and action. in *32. HICSS'99*, (Hawai, US, 1999), CD-ROM.

6. Boehm, B. A spiral model for software development and enhancement. *IEEE Computer, 21* (5). 61-72.
7. Cockburn, A. and Williams, L. The Costs and Benefits of Pair Programming, Humans and Technology, Place Published, 2000, Available at http ://collaboration.csc.ncsu.edu/laurie/ Papers/XPSardinia.PDF.
8. Crabtree, A., Ethnography in Participatory Design. in *Participatory Design Conference*, (Seattle, Washington, USA, 1998), Computer Professionals Social Responsability, 93-105.
9. Ehn, P. and Kyng, M. Cardboard Computers. in Kyng, M. ed. *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.
10. Ehn, P. and Sjörgen, D. From System Description to Scripts of Action. in Kyng, M. ed. *Design at work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, 241-269.
11. Erickson, T. Design as Storytelling. *Interactions, 3* (4). 31-35.
12. Erickson, T. Notes on design pratice: stories and prototypes as catalysts for communication. in Carrol, J., M. ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, New York, 1995, 37-58.
13. Floyd, C., Mehl, W.-M., Reisin, F.-M., Schmidt, G. and Wolf, G. Out of Scandinavia: Alternative approaches to software design and system development. *Human Computer Interaction, 4* (4). 253-350.
14. Floyd, C., Reisin, F.-M. and Schmidt, G., STEPS to software development with users. in *ESEC '89: 2nd European Software Engineering Conference*, (1989), Springer.
15. Iacucci, G., Kuutti, K. and Ranta, M., On the Move with a Magic Thing: Role Playing in Concept Design of Mobile Services and Devices. in *DIS 2000*, (Brooklyn, New York, 2000), ACM press.
16. Jacobson, I. The Use-Case Construct in Object-Oriented Software Engineering. in Carrol, J., M. ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, New York, 1995.
17. Kahler, H. Constructive Interaction and Collaborative Work: Introducing a Method for Testing Collaborative Systems. *interactions* (may + june). 27-34.
18. Muller, M. and Kuhn, S. Participatory design. *Communications of the ACM, 36* (4). 25-28.
19. Nielsen, J. *Usability Engineering*. AP Professional, Boston, u.a, 1993.
20. Pressman, R.S. *Software Engineering - A Practitioner's approach*. Mc Graw Hill, London, 2000.
21. Rittenbruch, M. and Kahler, H., Supporting Cooperation in a Virtual Organization. in *Ninteehth annual conference on information systems (ICIS)*, (Helsinki, Finland, 1998).
22. Rudd, J., Stern, K. and Isensee, S. Low vs. high fidelity prototyping debate. *interactions, 3* (1). 76-85.
23. Törpel, B., Pipek, V. and Rittenbruch, M. Evolving Use of Groupware in a Service Network. *Journal of Computer Supported Cooperative Work, to appear (2002)*.
24. Wells, J.D. Extreme Programming: A gentle introduction., Place Published, 2002, Available at http://extremeprogramming.org/.