Distributed Extreme Programming: Extending the Frontier of the Extreme Programming Software Engineering Process

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science
in the
University of Canterbury
by
Malcolm Maxwell Williams

University of Canterbury
2003

For my wife, Kemie; my children: Shauneé, Shomari and Malik Amani
who made the greatest sacrifice of affording me time
so that I could explore my dreams.

Abstract

Extreme Programming (XP) is inherently collaborative, which makes it amenable to Computer Supported Cooperative Work (CSCW) support. The collaboration enabling tools and techniques used in XP, such as whiteboards, index cards and co-location, are adequate for their immediate purposes. However, they do not allow sufficient information to survive beyond the end of projects. Long term consequences of their use include the risk of inadequate software maintenance, and limitation of the process' scalability. In this thesis, we explore the opportunity to address these risks by means of a desktop-based client/server experimental groupware application. We exploit existing metaphors and characteristics of XP, such as the 'information radiator' and the natural hierarchical arrangement of the primary project concerns, in order to reduce the gap between user's mental model of normal XP and CSCW enabled XP. We use XML, relaxed-WYSIWIS and a message passing communications system to allow users to interact with the same or different aspects of a project's information space—the *Project Document*—while they collaborate on essential project planning and coordination activities of the Planning Game. We conclude that our choice of deployment architecture and selection of aspects of XP which are augmented offer relevant and more sophisticated support for XP teams than do web-based approaches reported in literature.

Table of Contents

Acknowledgments

First I would like to thank my supervisors Dr. Neville Churcher and Dr. Richard Pascoe for their patience, guidance and keen interest in keeping me on track to complete this thesis.

I also thank the following persons and organisations:

- My office mates for their interest, kind humour and encouragement. Yen-Rong Sun for the pleasant personality and kindness. Behshid Ghorbani for her kind and encouraging words. Also Hao Ding, Sung Bae and Fong Long Chong for their encouragement.

- Carla Molloy and Dr. Surujhdeo Seunarine for their friendship and kindness. Lynette Ling, Dr. Annette Jones, Dorita Knight and the rest of the West Indian community for making me feel at home.

- Micheal JasonSmith, for always being frank with me. His tendency to not be politically correct, gave me a safe and familiar place to get feedback about my project. Joshua Savage for being 'almost West Indian' and a great and patient listener.

- A special thank you to the Government of New Zealand, the New Zealand Vice-Chancellors Committee, Commonwealth Scholarship and Fellowship Plan and the Government of the Cooperative Republic of Guyana for affording me the opportunity of this scholarship. A special note of thank to Ms. Kiri Manuera.

- All of the members of the Software Visualisation Group: Dr. Neville Churcher, Warwick Irwin, Carl Cook and Tony Dale; for their earnest interest in my research, for listening and sharing of their knowledge.

- The Head of Department and staff of the Department of Computer Science and Software Engineering, University of Canterbury, for creating the environment in which I have obtained a deeper appreciation for Computer Science.

Praises and glory be to the Infinite Intelligence whose grace saved me, mended my broken spirit and watched over my reconcilliation from the greatest loss I have ever known, and kept me through this program.

Chapter I

Introduction

Our research explores computer-based support for the Extreme Programming (XP) software development process. Our goal is to support XP teams in their activities in a lightweight unobtrusive manner in order to alleviate risks, such as inadequate software maintenance and lack of scalability, that are associated with XP. The risks arise from the routine use of simple manual tools and techniques, such as index cards and whiteboards and co-location, on XP projects. We exploit the opportunities provided by traditional and relaxed Computer-Supported Cooperative Work (CSCW) techniques in the development of a proof of concept groupware application, called PAM, to address these risks. We developed PAM using an iterative incremental prototyping approach, and each subsequent version benefited from feedback obtained from informal user evaluations. Our research concludes that PAM adequately addresses the risks of normal XP practices.

In the remainder of this chapter, we discuss the evolution of software development processes with the aim of showing how and why XP has emerged as a legitimate process. We also provide an overview of the problem which motivates our research and how we have addressed it. Then we discuss the scope of our research and how the thesis is structured.

1.1 The Evolution of Software Engineering Processes: A Gentle Introduction

Software Engineering deals with the development of large computer-based systems (Pressman 2001, Sommerville 2001). It is generally deemed to be essential to the development of high quality software and the success of development projects. To achieve these goals, software engineering has the concept of a *pro-*

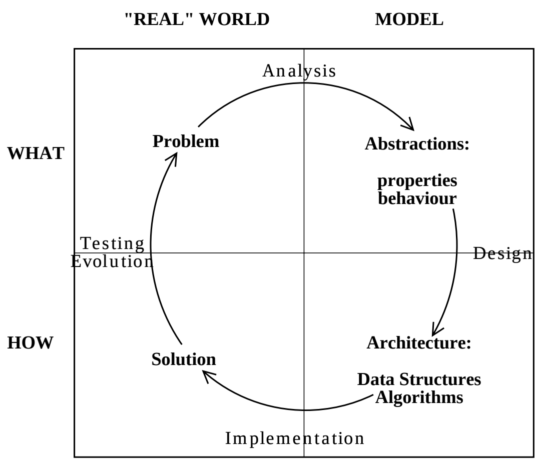


Figure 1.1: The fundamental activities of software engineering processes.

cess at its core. A *process model* serves as a conceptual simplification of an actual process, such as the Waterfall or Rational Unified Process (RUP) model. Four fundamental activities are integral to all processes. These are analysis, design, implementation and testing, illustrated in Figure 1.1 (Zelkowitz, Shaw & Gannon 1979).

Processes have evolved from the time the Waterfall model was first proposed. Several factors, such as the need for rapid delivery of solutions imposed by the 'software crisis' and the dynamics of the modern economy, have influenced this evolution. Nevertheless, the primary goal remains the development of high quality software. We list some of the pervasive and modern influences below.

- More software is needed than the software community can adequately supply on time, within budget and free of errors.

- Advances in hardware technology are faster than in software. Old systems will need to be updated to take advantage of new technology to maintain the competitive edge in businesses.

- Increased time-to-market pressure. Because the new economy is in constant flux, software required to meet specific requirements is often required immediately. Delay in providing solutions often results in a system becoming obsolete before it is delivered.

- System requirements are becoming increasingly more complex as software and hardware advances generate more complex needs, such as virtual worlds, military and medical software.

The evolution of processes is a history of the formulation, refinement and branching of various approaches in response to emerging demands. Modern processes have evolved along two conflicting approaches (Boehm & Turner 2003). These can be classified into two broad categories: heavyweight and lightweight. This classification indicates the nature of the management style, what activities are emphasised, as well as the amount and nature of the intermediate artefacts produced.

Heavyweight Processes

Heavyweight processes are characterised by their emphasis on rigour, reporting, planning, the management function and adherence to strict rules concerning the production of artefacts during development. These models are the result of the emphasis on predictability, accountability and risk management. A SEP is considered heavyweight or high ceremony when there is a high overhead involved in employing the process in terms of artefacts to be produced and especially the management required to ensure that development progresses properly.

Heavyweight processes are suited for large software projects involving hundreds of programmers and long time scales. Toward this end, the emphasis on the role of management, predictability and documentation have resulted in rigid processes that are often criticised for being bureaucratic. A popular heavyweight SEP is Rational Unified Process (RUP), which was developed collaboratively at Rational Software by Ivar Jacobson, James Rambaugh and Grady Booch (Wampler 2002).

Lightweight Processes

Lightweight Processes are characterised by their adaptive nature (Fowler 2003). Lightweight processes work best with small teams. They are also called Agile

Software Development processes and are deliberately fashioned to address the rapidly changing dynamics of the modern economy. In this manner, they are optimised to accommodate to changing requirements. The primary aim of lightweight processes is to get usable software to the customer quickly in small incremental releases.

Lightweight processes include Scrum (Schwaber & Beedle 2001), Crystal (Cockburn 2002), Feature-Driven Development (FDD) (Coad, LeFebvre & De Luca 1999), Dynamic Systems Development Method (DSDM) (Highsmith III & Orr 2000) and the most popular, Extreme Programming (Beck 1999*b*). Lightweight processes embrace a simple philosophy and harness the benefits of modern development paradigms. Traditional concerns such as comprehensive documentation, formal reviews and predictability are supplanted by customer involvement and adaptability (Agile Alliance 2001).

The major criticism of Agile Methodologies is that the process is not suitable for large development projects with hundreds of developers.

Extreme Programming

Extreme Programming (XP), which we dicuss in Chapter 2, came about as part of process evolution. It is the most popular Agile Methodology (Agile Alliance 2001, Miller 2002, Van Der Vyver & Lane 2003). XP is a disciplined yet radical process, that uses commonsense proven techniques in its approach to software development (Paulk 2001, Miller 2002, McBreen 2003). Nevertheless, software is developed using the same concepts highlighted in Figure 1.1, even though they are often named differently. This is further discussed in Chapters 2 and 4.

The work of many software practitioners and experts from other fields—Boehm (1988), Jacobson (1994), Alexander (1979), Lakoff & Johnson (1998) and others—has influenced XP's approach (Beck 1999*a*). For example, with respect to the radical aspects of XP, the assertion that high-impact decisions must be made by occupiers of a structure, (Alexander 1979), influenced the involvement of the customer and their decision making power. Further, the use of a system metaphor (Chapter 2.3, page 20) to aid collective understanding of the software being developed was influenced by Lakoff & Johnson (1998).

XP's radical approach optimises it for collaborative development. It is amenable to the current OO paradigm and several tools and frameworks have been developed to support this process, such as JUnit (ObjectMentor 2001, Beck & Gamma 1998) for the important intensive testing practice. XP is grounded in four values and twelve practices which address many of the problems faced by conventional processes, such as responsiveness to changing requirements and need for rapid delivery. Nevertheless, XP has shortcomings which have long term consequences for the process.

1.2 The Problem

The simple tools used in XP, such as index cards and the common whiteboard, provide inadequate support for the collaboration inherent in the process (Section 3.1, page 28). Though they are effective with small co-located teams, their use has long term consequences. Our research explores computer-based support for XP as a means of addressing some of these consequences. The following are the specific issues we attempt to address in our research.

1. Provision for persistence of XP project concerns

2. Augmentation of the inherent collaboration of XP activities

3. Relaxation of the co-location constraint

4. Support for distributed knowledge sharing about the state of the project.

Provision of computer-based support, to complement or replace existing systems, impacts on the deployment environment. It is desirable that the impacts be positive. However, at times the impact is negative. For example, the new tool introduced may impose restrictions on the normal work flow or otherwise change users mental models of existing processes and protocols. In this regard, the major objective of our research is to fit our prototype system into the context of XP projects so that we do not break the XP process.

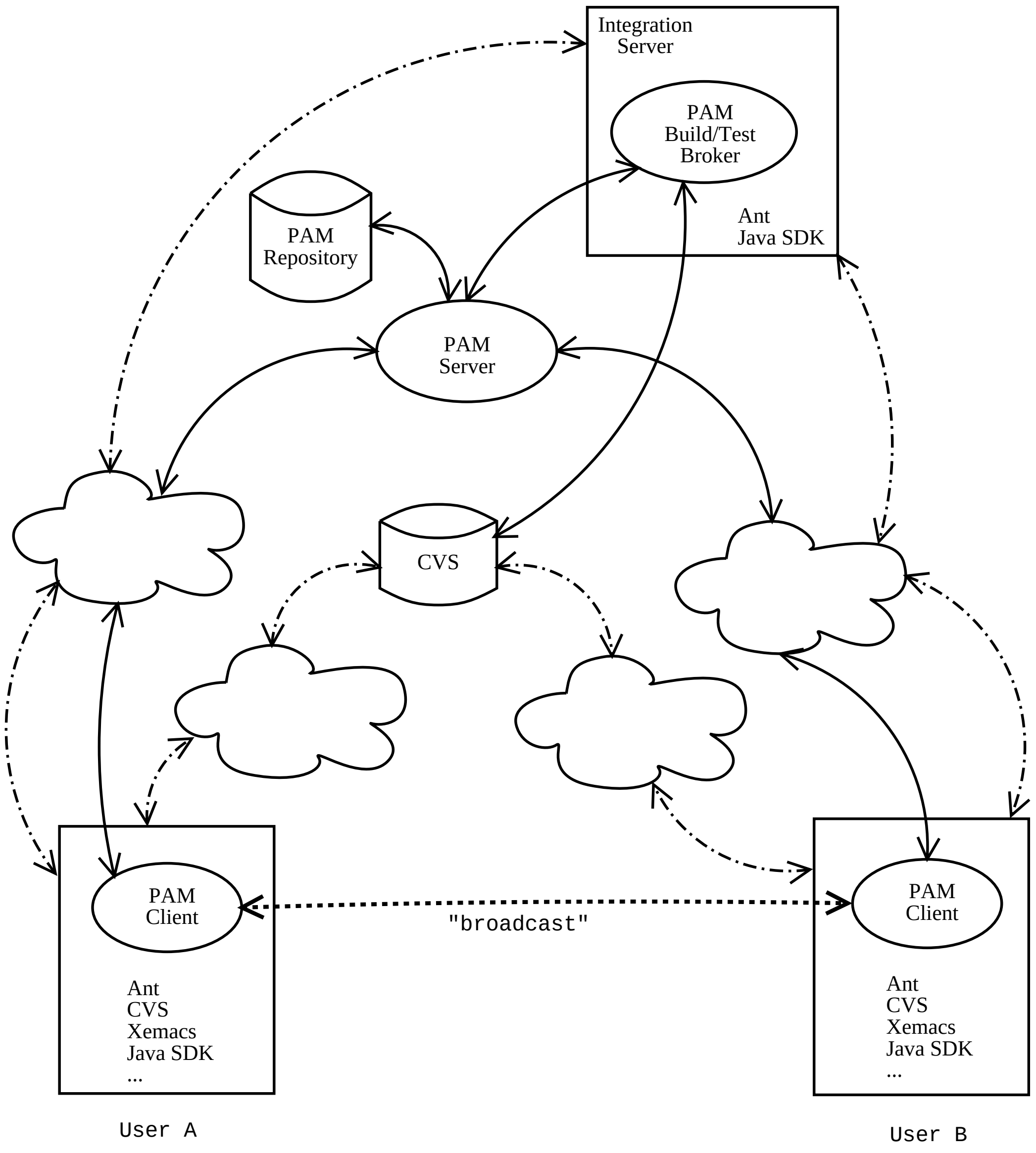


Figure 1.2: PAM Deployment Architecture. The dotted lines indicate existing XP channels of communication, while the bold lines indicate the channels we have added with our prototype system.

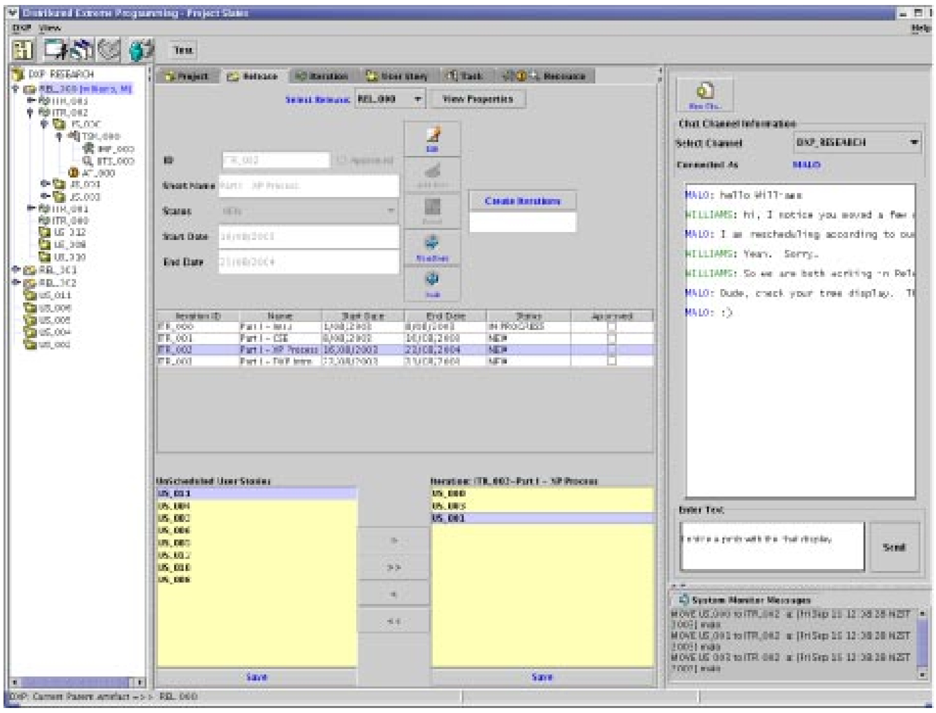
1.3 The Solution

Theoretically XP and CSCW complement each other. XP by virtue of its simple process model and inherent collaboration practices—values people and interactions over processes and tools—is amenable to rapid application development and distributed software development. CSCW is a research area concerned with the study and development of computer-based tools to support group work. Techniques, developed as a part of CSCW researches, have been used successfully in groupware applications to support individuals collaborating on group decision making activities, recreation and so on. Tools such as Microsoft NetMeeting, ICQ and Napster are examples of groupware. They are essentially used for teleconferencing, communication and recreation respectively. These tools indicate the wide application and novelty of groupware.

We presupposed that CSCW support for XP is beneficial for both co-located and dispersed XP teams. Our solution is implemented as a computer-based distributed system. It is a lightweight, specific instance of support for collaborative software development, using XP as the process model. CSCW techniques are used to facilitate the human-computer interaction requirements. The underlying groupware communication is by means of a third party messaging framework called CAISE (Cook & Churcher 2003*a*). The system is deployed as a desktop-based client/server groupware application. Figure 1.2 illustrates the deployment of our groupware. The figure also shows how it fits into the context of existing XP environments.

Our prototype system provides support for XP project planning and coordination (the Planning Game). XP activities can be broadly categorised into project management and program development. Program development in the dispersed context is relatively well supported by tools such as CVS, WikiWiki and so on. Project management tools also exist—for example, Microsoft Project and Mr. Project. However, these tools' support for distributed, real time project management is limited. In achieving its goal, our system emphasises project coordination through production, communication and coordination mechanisms. Figures 1.3 and 1.4 are snapshots of the user interface.

The design of the interface is based on the Clover Model (Sire, Chatty, Gaspard-Boulinc & Colin 1999). This model breaks down the functional requirements


Chat Channel Information

Select Channel: DXP_RESEARCH

Connected As: MALO

MALO: hello Williams

WILLIAMS: hi, I notice you moved a few

MALO: I am rescheduling according to ou

WILLIAMS: Yeah. Sorry.

WILLIAMS: So we are both working on Rele

MALO: Dude, check your tree display. Th

MALO: :)

Iteration ID	Name	Start Date	End Date	Status	Approved
ITR_000	Part I - [illegible]	[illegible]	[illegible]	IN PROGRESS	☐
ITR_001	Part I - CSE	[illegible]	[illegible]	NEW	☐
ITR_002	Part I - XP Process	16/08/2003	22/08/2004	NEW	☐
ITR_003	Part I - DXP Interface	[illegible]	[illegible]	NEW	☐

Figure 1.3: Snapshot Release Management GUI

of groupware into three activities: production, communication and coordination. Production encompasses the creation and maintenance of critical XP data such as User Stories, Release and Iteration schedules. In addition, there is support for client and server side builds and testing. A communication utility supports pair, group and inter/intra-project chat. The coordination mechanism is facilitated through lightweight awareness techniques. This includes awareness of the location of users in the same session as well as real time updates of changes to the project and session. To describe the underlying philosophy of the awareness mechanism we introduce the concept of 'What You Know Is What I Know' (WYKIWIK)—that is, each application, by means of real time updates, obtains and maintain knowledge of the current state of the project. These areas of emphasis, in combination with the metaphors in, and of the user interface—a CSCW enabled information radiator—offer relevant and appropriate support for project planning and coordination activities of XP teams.

Informal evaluation of our experimental groupware suggests that it offers rele-

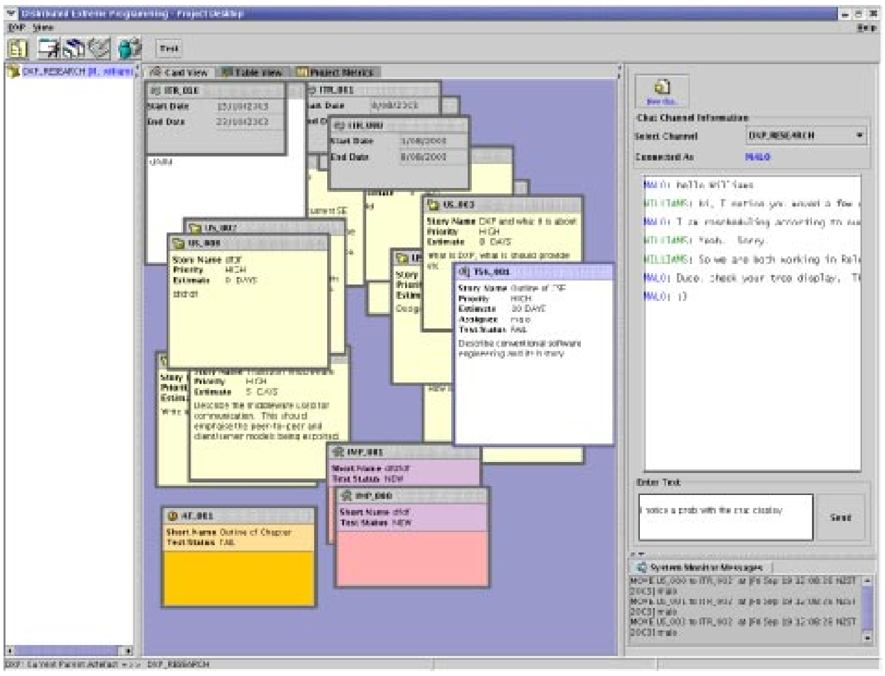


Figure 1.4: Snapshot implementation of Project Desktop metaphor

vant and appropriate support for co-located and dispersed XP teams. Our research also provides a framework for addressing the XP scalability issue. Further, it provides the basis for future comparative evaluation of the distributed XP environment contrasted with standard XP and traditional, non-XP environments. We anticipate that these evaluations will provide empirical evidence to support the promotion of XP as a strategy to enhance the teaching of Object Oriented Technology in academia, and provide the basis for the adoption of XP in industry.

1.4 Thesis Structure

This thesis provides a phenomenological description of our research and implementation of computer-based support for the Extreme Programming process. In Chapter 2 and 3 we discuss the Extreme Programming process and Computer Supported Cooperative Work (CSCW). We show how CSCW complements the inherent collaboration in XP and what challenges are faced when considering computer-based support for the process.

In Chapter 4 we present an analysis of the XP process. We developed several models and extract from these the activities that are essential to XP teams deriving benefits from computer based support. In the next chapter, Chapter 5 we present our groupware from the perspective of the functionality it offers users.

In chapters 6, 7 and 8, discussed the architecture of the system from three perspectives: (1) Configuration and Component Architecure, (2) the Connnectivity Middleware and (3) the User Interface Architecuture.

We then wrap up our discussions with reflection of our research and recommendations of how it can be exended by future work.

Chapter II

Extreme Programming

"I don't think that the most promising ideas are on the horizon. They are already here and have been here for years but are not being used properly."
— David L. Parnas (Eickelmann 1999).

Extreme Programming[1] (XP) was developed by Kent Beck and others in 1996 (published in 1999) and first used on the Chrysler Comprehensive Compensation (C3) project to rewrite the Daimler-Chrysler payroll package (The C3 Team 1998, Williams & Kessler 2000). XP is a radical approach to software development, and this is embodied in its practices (Section 2.3). The approach demands a high degree of discipline from developers and the continuous involvement of the customer (Section 2.2) for the duration of the project. XP derives its name from the extreme levels to which it takes established industry best practices. For example, development is test-driven (test-first programming), up-front design is replaced by incremental 'just-in-time' design (any design not immediately needed is deferred) and formal reviews are supplanted by pair programming (page 19). Consequently, design and review span the entire project rather than being done periodically or at dedicated stages of the process.

XP is structured around a philosophy, four values and a set of twelve practices (discussed in Sections 2.1 and 2.3 respectively). The twelve practices are considered to be enabling practices (Fowler 2001). XP is adaptive, and over the duration of a project teams are encouraged to adapt the process until it fits the development

[1] There has been a recent increase in the documentation about XP. The facts presented in this chapter about XP were obtained from a number of sources: books, articles, XP advocates and the Internet. Some of the primary sources are: (Beck 1999*b*, Beck 2002, Marchesi, Succi, Wells & Williams 2002, Beck & Fowler 2000, Wake 2001, Jeffries, Anderson & Hendrickson 2001, Auer & Miller 2001, Newkirk & Martin 2001, McBreen 2002, Wells 2001, Fowler 2001)

situation, circumstance and purpose of the project. The process is well suited for small teams and vague or rapidly changing requirements (Beck 1999*b*).

XP addresses many practices at Levels 1 and 2 of the Capacity Maturity Model (CMM), which it is considered to complement (Paulk 2001). (CMM was formulated by the Software Engineering Institute (SEI), as a model to measure, help and encourage organisations to improve their software development process (Paulk, Curtis, Chrissis & Weber 1993, Software Engineering Institute 1995).) Paulk suggests that at Level 2, Requirements Management is addressed by User Stories, on-site customer and continuous integration. At Level 3, Organisation Process Focus is addressed at the team rather than the organisation level. Whereas CMM emphasises *what* has to be done, XP focuses on *how*.

In the following sections we discuss the XP process. This discussion highlights the issues addressed and characteristics exploited in our prototype system.

2.1 XP Philosophy and Values

The values central to XP are (1) simplicity, (2) communication, (3) feedback and (4) courage. These values are inter-related and complement each other. For example, communication and feedback supports common understanding of the vision and state of the project. The values are important to XP's philosophy—that is, that for software projects to experience improvement the values must be observed. The four values also contribute to a development culture and attitude that is conducive to the goal of predictable, sustained, and sustainable delivery of software. This combination of culture and attitude—a particular mindset in essence—forms the basis for the discipline necessary for practicing XP. The philosophy and enabling practices (Section 2.3) optimise XP for eventual success of projects through predictable, sustained, and sustainable delivery of software (McBreen 2002).

Simplicity

Simplicity is rigorously advocated by XP and is most obvious in the process model itself (Section 2.5), as well as in the design and documentation artefacts (Section 2.4). For example, refactoring (page 21) reinforces simplicity by frequently and steadily exploring opportunities for improving the design of existing source code and keeping it clean. The process of achieving and maintaining simplicity is plagued with contradicting issues long-term and short-term issues such as producing maintainable code and ensuring predictability in the process respectively. XP addresses these by focusing immediate effort on the current development concern only.

We exploit XP's simplicity in PAM, our proof of concept, computer-based support for XP (See Chapter 5). The independence, small granular size and discreteness of artefacts, such as User Stories, are exploited in order to relax some CSCW protocols (Chapter 3). In addition, the natural hierarchical arrangement of schedules and artefacts forms the basis of the User Interface (UI) model (Chapter 8).

Communication

Communication is an integral part of XP projects. The various skills and expertise of the customer, developers and management are combined into a greater creative effort. The sustainability of this effort relies on open and abundant communication, one immediate benefit of which is quality of feedback.

PAM supports both direct communication (such as synchronous chat) and indirect communication (such as awareness of the actions of other users and asynchronous messages by means persistent notes). These facilities play significant roles in supporting co-located and dispersed XP teams.

Feedback

Feedback spans the entire process from the Planning Game (Section 2.3) through to customer acceptance and delivery of the software (or cancellation of the project). It is important for collective understanding and transparency of the development effort. Adequate and frequent feedback ensures that a high level of control is maintained in the process. This is essential to steering the project, that is, understanding the impact, interference and influence of each aspect of the project as it progresses.

Feedback comes from customer evaluation of the intermediate artefacts and released increments of the software, test results and so on. With feedback everyone has the opportunity to be aware of the state of the product and process. Feedback thus supports informed decision making.

In the PAM system, all project data is treated as constituting a single *Project Document* (Chapter 8, page 120). We use persistence, real time updates and awareness techniques to advise customers and developers of changes to the *Project Document*.

Courage

Courage is essential for progress in the face of uncertainty and doubt. It nullifies the greatest detractors of creativity; that is, lack of confidence and doubt at both the individual and team level. Simplicity, communication and especially feedback play important roles in encouraging and sustaining the courage value.

Courage is a characteristic of emotional intelligence. Emotional intelligence is that human characteristic/skill that enables people to maintain harmony in interpersonal relationships, play and work; in collaborative activities (Goleman 1995). We believe it is a critical work place attribute. Change is typical of projects. Estimates will be inaccurate, requirements will change, systems will crash and so on. In such situations, we believe, the collective emotional intelligence of a workforce makes the difference between project breakdown or temporary schedule creep.

2.2 *XP Stakeholders*: *The XP Team*

A whole team approach is used on XP projects. They all share the risks of the project and are, therefore, considered stakeholders. Everyone involved is equal, but has various roles and responsibilities. This represents a collaboration of business and technical skills in the developing software. None of these roles is necessarily the exclusive property of one individual. There is no designated specialist, only contributors with special skills and knowledge. PAM supports XP team collaboration. For example, PAM supports co-located and dispersed users working together on activities such as when User Stories are being assigned to Releases/Iterations (Section 5.2.5, page 80). The following list outlines the primary roles and primary responsibilities involved in XP development.

- **Customer:** provides the requirements, sets priorities and steers the project.

- **Developers:** transform the requirements into working software. This role can be divided into specialized roles. A single person can take on more than one role depending on the project circumstances. Specialized developers are:

  1. Programmers who estimate feature costs, write source code and execute tests
  2. Testers who help the Customer define the customer acceptance tests, and execute tests
  3. Analysts who helps the Customer to define the requirements.

- **Manager:** This role is divided into three specialised areas. One person may assume all of the roles. The specialised roles are:

  1. Manager – provides resources, handles external communication and coordinating activities.
  2. Coach – helps the team understand the XP process, keeps the team on track: acts as mentor, conflict facilitator, and keeps the system vision and metaphor real
  3. Tracker – watches over the development process in order to offer guidance, where necessary, of how to adapt the process to fit existing circumstances.

2.3 XP Practices

The XP philosophy, values, culture and attitude—the mindset—are supported by a set of enabling practices, which complement one another to define the XP process. These practices can be separated into two broad categories based on their role in the process: (1) Project Management and (2) Application Development. In the following subsections we discuss these practices as a precursor to how they are addressed in PAM. We categorise the twelve practices in Table 2.1.

PAM provides support for the Project Management aspects of XP, with emphasis on the Planning Game and project coordination (Chapter 3).

Project Management	Application Development	
Planning Game	Small Releases	Continuous Integration
On-site Customer	Intensive Testing	Refactoring
Use Coding Standard	Pair Programming	Simple Design
40-hour Work Week	System Metaphor	Collective Code Ownership

Table 2.1: The twelve practices of eXtreme Programming.

2.3.1 Project Management

Planning Game

Planning Game is the term used in XP terminology to describe project planning. In XP, project planning is used to steer the project as it progresses. The goal in this regard is adaptability. This contrasts with the goal of predictability in traditional process planning, which maps out a course of action for the project, from start to finish. There are two key aspects of steering an XP project: (1) Release Planning and (2) Iteration Planning. PAM is designed to support these aspects.

During Release Planning, the Customer lays out the plan for the project from the business' perspective. The desired features of the system are first presented to the programmers, who estimate their costs. Each feature is written down on a 5"$\times$ 3" index card (see Figure 2.1(a) on page 22). The description of the feature is called a User Story (page 22). The cost is a measure of difficulty in terms of how many 'ideal programming days' are necessary to encode the User Story. An 'ideal programming day' is a day in which there are no interruptions or delays. The customer uses the estimates and their knowledge of which features have the greatest potential return on investment, and prioritises them accordingly.

The project duration is divided into a number of Releases of about two to three months each. The highest priority User Stories with a sum cost nearest equal to, but not exceeding the number of 'ideal programming days' in a Release are assigned to the first and subsequent Releases in order. The initial Release Plan is imprecise and is refined regularly as the project progresses. The first step in this regard is Iteration Planning.

At the start of the Iteration Planning activity the User Stories assigned to the current Release are allocated to short term schedules of about two to three weeks, called Iterations. A detailed plan for transforming the User Stories into working deliverable software is mapped out for the first Iteration. Iteration Planning takes place at the start of each subsequent Iteration. As part of the latter Iterations, the Customer presents the User Stories desired for the immediate Iteration as well as those from failed Acceptance Tests (page 23).

Each User Story in the current Iteration is broken down into Tasks (page 23), a number of which define what has to be done to implement the feature in a specific User Story. Developers then take responsibility, or sign up, for Tasks. The Developers responsibility is to ensure that the Task gets implemented. Each Task is estimated by its assignee, independent of the cost of the User Story from which it was derived. This improves the accuracy of the cost estimates per User Story. The Task costs are added and compared with the Iteration velocity, which is the amount of work accomplished in the previous Iteration in ideal programming days (for the first Iteration the velocity is estimated). In order to balance the total Tasks cost with the Iteration velocity the Iteration is adjusted by (a) reassigning User Stories to the next Iteration or (b) bringing forward User Stories from the next Iteration. When the Iteration plan is complete programming commences for that Iteration.

We will refer to this discussion of the Planning Game in Chapters 4 and 5 where we abstract the aspects of XP that we support in PAM, and discuss how the aspects are implemented as system features respectively.

On-site Customer

A customer[2] representative—the "Customer"—is required to be co-located with the developers. The Customer steers the project from the perspective of the busi-

[2] Kent Beck corrects the perception that the 'Customer' refers always to a single person; in his forward in *Questioning Extreme Programming* written by McBreen (2002).

ness' priorities, in addition to being available for enquiries, clarifications and confirmation of any ad hoc issues. Co-location allows the Developers get feedback about issues they find vague or confusing. This prevents the Developers from making assumptions.

Coding Standard

XP attaches extreme importance to source code. Developers comply to a coding standard. This practice supports the pair programming and collective ownership practices by enforcing consistency in the source code. All the code in the system appears as if written by the same programmer. XP does not advocate any specific standard, provided that a consistent standard is used.

40-Hours Work Week

Developers are encouraged to work 40-hours a week. Overtime is allowed when it is beneficial to the project. It is not used as a strategy to correct schedule creep. (Schedule creep is addressed immediately by reexamining the iteration plan and making the necessary adjustments for the next Iteration.) This practice guards against programmer burn out. It ensures that development progresses at a sustainable pace.

2.3.2 Application Development

Small Releases

At the end of each Iteration one or more User Stories, or version, of the software is completed. The working, tested version is handed over to the Customer for evaluation and/or to be released to the end users. Business value is obtained by the Customer every Iteration and project visibility is guaranteed.

This practice empowers the Customer to make informed decisions about the project, such as whether to cancel the project if progress is not satisfactory, and/or what is the next best step to bring more business value. The small and frequent release practice is supported and kept reliable by means of the intensive testing practice, as described in Intensive Testing below.

Collective Code Ownership

The XP team owns the code—that is, ownership is collective as opposed to individual. In this way, everyone takes responsibility for the code and feels free to refactor or make changes, as needed. This practice aims to improve code quality and reduce defects. It makes code modification/refactoring easy and helps with knowledge transfer within the Team. Further, it supports refactoring. Any piece of code can be refactored by any pair of programmers to improve the design of the source code.

Intensive Testing

XP is based on test-driven development. Developers write repeatable Unit Tests to automate testing of the integrity and functions of source code features before the actual source code is written. All tests are required to pass once the source code is written. Whenever code is released to the project repository, the tests collected for the project are executed. This provides immediate feedback if new code breaks the system. Apart from unit testing, periodic acceptance tests are conducted to help the customer determine whether or not the system being developed meets their satisfaction (see Subsection 2.4, page 23).

In PAM we support both local and server side testing (page 85).

Pair Programming

All code is developed by pairs of programmers who sit side by side and share the same computer system. Two roles are defined in this collaboration: (1) driver and (2) observer. Developers are allowed to switch between roles as needed. The driver has control of the input devices, while the observer has the role of a 'real time' reviewer. The observer keeps track of what the driver is doing and assists with corrections where necessary, in addition to thinking about the source code design.

Pair programming also serves to communicate knowledge throughout the team. Pairs switch between tasks and, therefore, specialised knowledge is shared. As the developers learn, their skills improve and they become more productive and valuable to the team and to the company. Though pair programming may seem a desirable aspect of XP to support with computer-based tools, we do not. We explain our reasons in Subsection 4.1.1.

System Metaphor

The system metaphor is the combination of a common vision of how the intended system will work and a collection of names and naming convention. The system metaphor is used to promote efficient communication within the team. The system metaphor is a description that is tailored to educe how the system being developed works. For example, "this program is a dynamic single-page information radiator" may be a metaphor. This is the metaphor used to guide the development of PAM. We use 'dynamic' to mean that anyone can make changes to the *Project Document*, and that those change(s) will be available to other users of PAM. "Information radiator" is the metaphor used by Cockburn (2002) to describe, collectively, the whiteboard, index cards and flipcharts used to keep track of XP projects. XP team member refer to the information radiator in order to be updated about the state of the project.

The system metaphor is refined as the project progresses and the software evolves to reflect the system requirements. Metaphors are useful because they allow the team to conceptualize abstract problem domains from mappings to familiar concrete domains/concepts.

Continuous Integration

The software being developed—the system—is always kept integrated. As each User Story is completed, the source code is integrated into the latest release (after passing all tests). Integration takes place at least twice daily for each pair of programmers. Continuous integration is essential to keep the software reliable and support small releases. It also provides early feedback about what effects new code has on the system.

PAM supports continuous integration by making available up to date knowledge of the current state of the project. For example, developers may refer to PAM to update or determine which tests have failed.

Refactoring

Refactoring, (Fowler 1999) or continuous source code design improvement, complements the Simple Design practice. As the project progresses, refactoring is used to improve the software design. Refactoring aims to remove duplication, while increasing cohesion and lowering the coupling of the code. Intensive testing is very important to refactoring. The Unit Tests verify that refactoring does not break the system.

PAM supports refactoring indirectly through the feedback provided by the testing mechanism.

Simple Design

All software is built to the simplest possible design. Simple is used to mean that the design maps to the functionality of the current version. Design is not done for future anticipated needs, and XP teams are encouraged to maintain this practice based on courage and the belief that “You Are Not Going to Need IT” (YAGNI). Simple Design in XP, like the project planning aspects, is neither a one-off nor up-front activity. Design is carried out for the duration of the project.

Design is incorporated in Release and Iteration Planning, in the decomposition of User Stories into Tasks, and in the programming and refactoring of the source code. No extra work is done with respect to anticipated functionalities. This practice supports the pair programming and the on-site customer practices. It focuses the pair and customer on the current important issue only. The incremental, iterative nature of XP development also lends itself to a deeper understanding of the system. This is necessary for good design, which in turn is essential to sustaining an XP project.

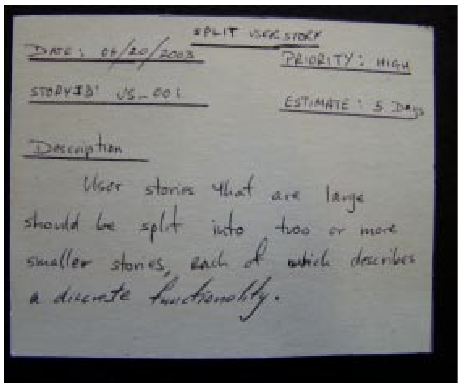
2.4 *XP Tools, Activities* & *Artefacts*

XP Activities (Chapter 4) are actions that create or transform project artefacts, such as the coding task transforms a Task into source code artefact(s). Artefacts are the tangible things produced as part of the project, such as Release and Iteration Plans, User stories, Tasks, Source Code and so on. Tools are the devices used by XP teams to facilitate the activities of the process.

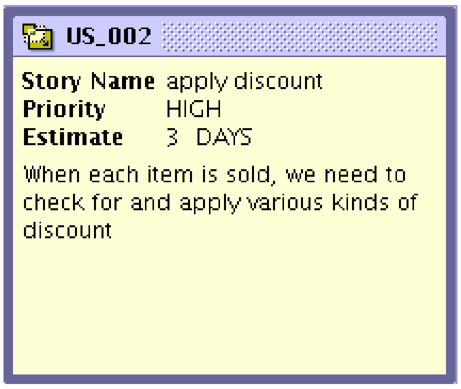
In this section we describe the key XP artefacts and, the activities and tools which are used to create/transform them. We refer to these later (in Chapters 5 through 8) when describing the implementation of PAM.

- **User Stories:** A User Story is a concise expression of a discrete feature of the system that will be discussed with the aim of transforming it into software. User Stories are the primary input into the XP process. Each User Story is written down on a 5”× 3” index card. Figure 2.1 depicts sample User Stories created as part of the development of PAM (Figure 2.1(a)) and during an informal evaluation session (Figure 2.1(b)).

  At the inception of a project, a number of User Stories are prepared by the Customer as part of the Planning Game. As the project progresses new User Stories are included to reflect changing requirements, better understanding of specific features and/or when a large User Story is split (divided into at least two smaller User Stories).


SPLIT USER STORY
DATE : 04/20/2003
PRIORITY : HIGH
STORYID : US_001
ESTIMATE : 5 Days
Description
User stories that are large should be split into two or more smaller stories, each of which describes a discrete functionality.

(a) A User Story card typical of normal XP


US_002
Story Name apply discount
Priority HIGH
Estimate 3 DAYS
When each item is sold, we need to check for and apply various kinds of discount

(b) A User Story card from the PAM System

Figure 2.1: Sample User Story cards.

  Programmers and the Customer collaborate to prepare User Stories. The Customer is responsible for writing and prioritising the User Stories, while the programmers estimate their costs. This collaboration usually takes place with the cards spread out on a flat surface, such as a desktop. We use the desktop metaphor and emulate a *Virtual Desktop* in the client UI of PAM (Section 8.4, page 8.4).

- **Tasks:** A Task (sometimes called an *Engineering Task* (Jeffries et al. 2001)) is an explicit description of a programming task that has to be done to implement a part of a specific User Story. Tasks, like User Stories, are written down on index cards. Tasks are obtained by decomposing User Stories as part of Iteration Planning. This is done as a collaborative brainstorming activity between Developers and the Customer. Programmers ask the Customer to elaborate on aspects of User Stories so as to clear up any ambiguities.

- **Unit Tests:** A Unit Test is a test case or suite written to test the functionality embodied in a source code module, such as a Java class. Unit Tests are created by the Developers, using frameworks such as JUnit (ObjectMentor 2001, Beck & Gamma 1998). They are written prior to the actual source code that they test. This is called test-first development. The Unit Tests source files are stored in the source code repository along with the system's source code.

  In PAM we maintain collections of path names of the Unit Tests, relative to the root of the project tree in the repository (CVS in our implementation). We refer to these collections as *Resources*. We employ the same strategy to handle Acceptance Tests and Source Code—Implementation resource (page 111). Our implementation of PAM is based on the assumption that the project's root directory in the repository is named after the project. We use these resources as input into a Test Execution mechanism which executes local and server side Unit Tests on behalf of a user.

- **Acceptance Tests:** The Customer writes Acceptance Tests for each User Story. The Acceptance Tests describe what the system is expected to do, with respect to User Stories implemented, in order to gain the Customers acceptance. The tests outline what inputs will be used and how the system is expected to respond to the input.

  Developers automate the Acceptance Test by writing appropriate programs. The actual and expected results are automatically compared. In this way, Acceptance Tests can be executed on the server or locally in the same way as unit tests. Acceptance Tests may be executed using the Test Execution mechanism utility with prepared build files.

- **Source Code:** A Source Code file is a complete description, in a particular programming language, of the algorithm(s) and data structures necessary to achieve the specific goal(s) of Tasks, and in turn User Stories. All code is written by pairs of programmers.

  Programming is an application development activity which is outside the scope of our research. PAM keeps track of source code files. This information is used in the Test Execution utility.

- **Test and Defects Metrics:** Tests and defects metrics are records obtained from executing Unit and Acceptance Tests, and also customer feedback from the use of released software.

- **Release Plan:** The Release Plan outlines what User Stories will be implemented over what period; usually of two to three months. It is a general high-level outline of the project schedule.

  The entire team collaborate to produce the Release Plan.

- **Iteration Plan:** The Iteration Plan is more detailed than the Release plan. It outlines what User Stories will be implemented on a two to three week basis. Information available from the Iteration Plan may include the Tasks from User Story decomposition, who specific Tasks are assigned to, new Task estimates, what adjustments were done to make the Iteration realistic and the details of the Iteration velocity.

- **Spike:** A Spike is an exploratory experiment to obtain information to use in decision making when faced with uncertainty. A Spike may be considered as an activity or as a result. Spikes are categorised as architectural, estimate and solution spikes. Examples of spikes in these categories are: the appropriate architecture for the system, the difficulty of implementing a User Story or a Task, and alternative algorithms respectively.

In normal XP, the by-products of spikes are thrown away after the spike is conducted. The information obtained from the spike is used as appropriate—to guide the project (if a metaphor is obtained) or to choose between algorithms (if one alternative proved to be more appropriate for the specific circumstance).

- **Application development tools:** Predominantly, the tools available for XP are related to application development. XP's project management approach is not amenable to current project management tools. In addition, this is a reflection of the extreme importance XP attaches to application development activities.

  Tools used on XP projects are often simple and manual such as index cards, whiteboards and desktops. These tools are used to support collaboration by bridging the gap between developers and customer during planning activities and to store the state of the project. Even though they are effective for their immediate purposes, they are susceptible to hazards (see Section 3.1).

  Computer-based tools are used in XP for application development activities. These tools include compilers, editors, versions systems such as CVS, testing frameworks such as JUnit and debuggers. The lightweight requirement of our solution requires that developers must be able to use the application development tools of their choice, rather than restrict them to proprietary tools.

  Some of the tools used in XP support collaboration—for example, CVS, the index cards and the whiteboards allow developers and the customer to share information. However, the collaborative support offered is inadequate for dispersed teams. PAM adds automated support for the manual tools used in XP. Its architecuture (see Figure 6.1, page 88) complements the existing computer-based tools and benefits both co-located and dispersed teams.

2.5 XP Process Model

Figure 2.2 shows the high-level abstraction of the XP process model (Wells 2001). Release Planning, Iteration, Acceptance Testing and Small Releases are depicted

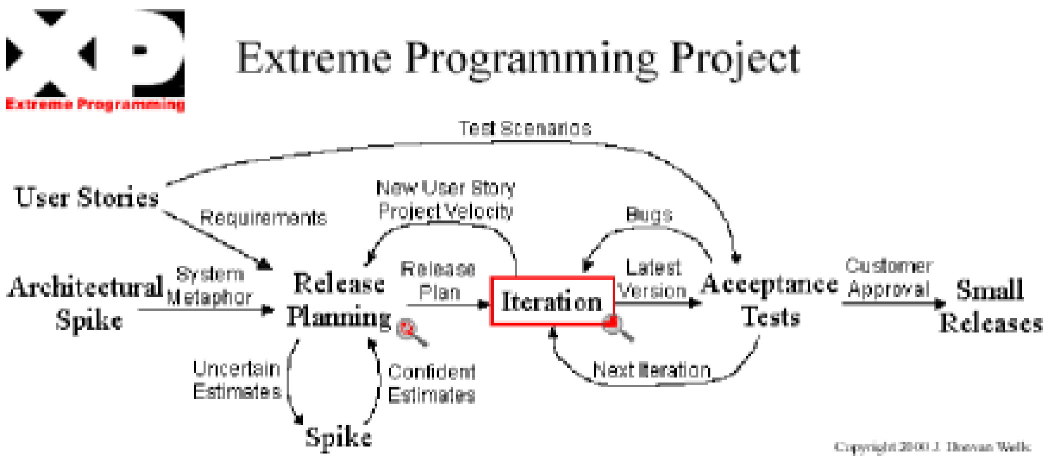


Figure 2.2: XP Project Model

along the critical path. As described in the preceding sections and Chapter 4, these activities are made up of other specific discrete activities. Composite activities, such as Iteration Planning, are usually illustrated using separate diagrams to reveal their details (available at `www.extremeprogramming.org`). The feedback loops in Figure 2.2 are indicative of the iterative nature and the continuous validation inherent in XP. In Chapter 4 we combine these various perspectives and illustrate the XP process using a Finite State Diagram, which we used as part of our activity analysis.

2.6 Summary

Extreme Programming is a legitimate disciplined software engineering process. In this chapter, we discussed the details of the primary concepts involved in the XP process. We indicated what concepts are addressed by our prototype experimental tool, PAM, to augment the collaboration inherent in XP. These concepts will be referenced through this thesis as we describe the the various aspects of our research and the implementation of PAM.

The discussion in this chapter provides the basis for Chapters 3 and 4. We highlighted in this chapter that in order for our prototype system to be lightweight, we must allow developers freedom to use application development tools of their choice. In Chapter 3 we discuss the issues involved in providing computer-based support for XP. We show how XP's inherent collaboration is complemented by existing group-oriented computer technology. We also discuss why we do not explicitly support application development activities. In Chapter 4 we describe our activity analysis of XP, during which we abstract the activities (collaborative and critical to the process), which offer the greatest potential to benefit XP teams through support from computer-based solutions.

Chapter III

Investigating Computer-based Support for Extreme Programming

"The 'human system' and the 'tool system' are equally important in computer-supported cooperative work"— (Engelbart & Lehtman 1988)

In the preceding chapters we discussed how XP emerged as a development process. We highlighted that XP is people/group-oriented and inherently collaborative. In this chapter, we discuss the reasons why these characteristics and group-oriented computer technology have been exploited in the development of PAM. We discuss: (1) why we believe computer-based support for XP will generate value for XP teams whether or not they are co-located; (2) alternative implementation of computer-based support; (3) CSCW and existing technologies which support our research; (4) related research; and finally, (5) our desktop-based client/server approach and motivation for research.

This chapter highlights the two forms of communication required to meet the needs of PAM: (1) point-to-point and (2) broadcast communication. Point-to-point communication is required for critical data management operations which change the state of the project. On the other hand, broadcast is needed for interpersonal and transient communication such as with chat.

This discussion in this chapter highlights several strategic decisions, which will be referenced in later chapters.

3.1 Why Computer-based Support for XP?

The collaborative nature of XP imposes a need for specialised tools. Computer-based support for the Planning Game, which is highly collaborative and loosely structured, has potential to benefit XP teams. We believe that co-location, whether real or simulated, is important during project planning activities. Many important decisions are made during the Planning Game activity. However, the normal XP practice is to write down the results on index cards (for example, the system features in the form of User Stories). These are then pinned to a common whiteboard, called the information radiator.

Index cards and the whiteboard are simple tools. Simple tools, nonetheless, are sometimes not acceptable to many people. Developers may want more flexibility, power of expression, automatically generated code, information to support maintenance and so on. The use of simple tools is testimony to the emphasis on simplicity in XP. Though they serve their immediate purpose—for example, User Stories bridging the gap between the business oriented and technical team members—their use in the Planning Game has serious long term consequences. In this regard, several researches have criticised XP for having shortcomings (Nawrocki, Jasinski, Walter & Wojciechowski 2002, Keefer 2002).

Some of the long term consequences of the normal XP Planning Game practice are:

1. **Inefficient traceability:** 'Paper offices' are subject to hazards such as fading, misfiling, illegibility and fire. In addition, normal XP relies on face-to-face oral communication. Therefore, tracing relationships between paper-based artefacts from mental recall is time consuming and high risk because a team member may forget important issues and decisions.

   We do not advocate replacing the team whiteboard, but, rather aim to augment it (see Chapter 5). Project management tools (for example, MicroSoft Project) can be employed to alleviate the problem of inefficient traceability. However, current project management tools do not readily fit into XP environments due to their focus on predictability, which is contrary to XP's philosophy.

2. **Little provision for software maintenance and evolution:** Insufficient information is carried over from an active project to the maintenance stage once development is complete. In the worst case, User Stories are destroyed once the project is complete. At other times they are given back to the

customer (Jeffries et al. 2001). The state of a past project, at a specific time during its development, would be difficult to reconstruct on the whiteboard, especially if the whiteboard is in use by another project.

3. **Informal 'grapevine' decisions are lost to the project:** Many important decisions are made informally and with a throwaway approach on XP projects. These include the results, intermediate decisions and artefacts obtained from executing spikes. Artefacts produced during spikes are reproduced, if necessary, in the real project. Where the results of spikes are intangible, such as with spikes used to determine the suitability of alternative techniques, they are not recorded.

4. **Memory overhead:** Besides the whiteboard, information pertaining to an XP project mostly resides in the memory of the XP team. XP schedules (Releases and Iterations) are intangible collective concepts, and there are a number of points during a project where critical decisions are made (see Subsection 4.1.2, page 55). This makes recalling knowledge about previous schedules and decisions pertaining to the project tedious and high risk. An example of this is to recall the reason(s) which lead to a User Story being split.

Tools, such as Concurrent Version Systems (CVS), Computer Aided Software Engineering (CASE) applications, Computer Aided Design (CAD) and project management applications, have been developed to support software engineers. However, these tools usually address generic tasks and information exchange, rather than offer collaborative support for dispersed teams (Churcher & Cerecke 1996). Typically, these tools do not offer support for specific processes. Further, they may inhibit the spontaneity that XP tools educe. For example, the simplicity of User Stories cards encourages discussion of their details during requirements elicitation.

Our research investigates computer-based support for XP. This form of support is valuable for co-located as well as dispersed teams because in addition to support for the Planning Game, computer-based support offers benefits for project coordination and persistence of project artefacts. Emergent benefits to be derived are: (1) timely feedback of the state and dynamics of the project, (2) freedom of deployment of stakeholders, and (3) opportunity to enhance XP's scalability. The success of computer-based support depends on the selection of appropriate technology and end-user satisfaction. We discuss enabling technologies in Section 3.2.

Tools that have been developed to relax the co-location requirement of XP are highly likely to be, and so far have been, in the form of computer-based solutions (Kircher, Jain, Corsaro & Levine 2001, Maurer & Martel 2002, Schümmer & Schümmer 2001). Nevertheless, we note that manual methods such as employing a runner/courier service between desks/offices/locations or setting up a dedicated position or department to coordinate, schedule and integrate project work and artefacts are alternatives to computer-based support. It should be noted that these latter manual approaches are highly inefficient.

3.2 Enabling Technologies for Computer-based Support of XP

An immediate question faced by our research is whether or not computer-based support will break XP. This question increases in significance as the separation of the team grows wider in space and in time, as the team move from the 'same room' (co-located) setting to being increasingly geographically dispersed. We believe that the process is not broken so long as the philosophy and values of XP are maintained. The remainder of this chapter relies on this assumption.

Computer-based support raises other issues. The most pressing issues are: (1) How will artefact evolution be coordinated and maintained? (2) How will project scheduling be coordinated? (3) Is persistence required, and if yes, then what form of persistence? (4) What degree of security is appropriate, without compromising XP's shared ownership practice? and (5) How will collaboration within the Team be supported? These questions highlight the need for adequate technological support. On other hand, issues concerning Human Computer Interaction (HCI), ergonomics and the diverse make up and subjectivities of XP teams raise the need for consideration of the social impact. The criteria we use for technological suitability are: (1) the technology must be adaptable in order to fit XP's values and philosophy, and (2) the technology must not put remote teams (indirect communication) at any major disadvantage to co-located teams (direct communication).

The technologies which offer the greatest potential are:

1. Internet: e-mail, web pages, Wiki (Leuf & Cunningham 2001) and so on.

2. Custom groupware application based on Computer-Supported Collaborative Work (CSCW) support.

An examination of the relative strengths and weaknesses of these technologies leads us to eliminate Option 1. The primary reason for this is that these technologies do not adequately support highly interactive synchronous collaboration. In addition, they are subject to the penalties of asynchronous communication. For example, even though Wiki provides shared access to data, the underlying technology makes little provision for resolution of conflicting updates. Further, communication cycles in asynchronous communication—for which they are best suited—is non-deterministic. Thus users may experience irregular time-delayed updates.

The dynamics of XP projects require more responsive technology. Individually, web-based technologies offer inadequate tool support for XP. Furthermore, although these technologies complement each other when combined, research by Kircher et al. (2001) has shown this to have shortcomings (Section 3.4).

Option 2 offers the best potential to provide computer-based support for XP. Therefore, we designed PAM around desktop-based CSCW support (Section 3.3). Computer-based solutions however, impact the environment in which they are deployed. Our analysis of the impact of computer-based support of XP suggests that it presents threats, as well as opportunities to enhance the XP process. The threats to XP predominantly arise from the risks involved in simulating/replacing (in the context of our research we use the term ‘augment’ instead) direct human-to-human communication and collaboration. The conflict between the social protocols used in normal team interactions and the rule-based collaboration enforced by artificial protocols imposes these threats. For example, the shared ownership practice in normal XP will conflict with strict locking protocols that are used in relational database systems.

Ironically, opportunities may be obtained from the artificial protocols that impose rules of engagement when using computer-based support. Where these rules may be found to be restrictive, they can also be harnessed to coordinate and streamline progress. For example, locking protocols can be used to handle concurrent access to shared data in a DBMS. While locking goes against the nature of XP, it may be used to coordinate change and updates in the distributed setting.

When computer-based support encourages wider separation, XP practices that are highly dependent on co-location—that is, pair programming, on-site Customer and shared ownership—will be increasingly affected. Other XP practices susceptible to the impacts of computer-based support are: use of a coding standard and extensive testing. Further, if the computer-based support makes collaboration difficult or otherwise unnatural for XP teams, then the susceptible practices risk being compromised. This may happen if the team decides to use workarounds or drop them all together.

3.3 Computer-Supported Co-operative Work plus XP

CSCW deals with the study of group work and the development of computer-based tools to support group work (Bannon & Schmidt 1989). CSCW research has resulted in the development of several tools to support interactive group work. These tools include: (1) communications tools such as e-mail, newsgroups, bulletin boards and teleconferencing, and (2) group decision support systems such as shared editors and computer-mediated conference tools. CSCW is concerned with issues such as group awareness, communication and coordination of group activities, concurrency control and shared information space for production, involved in supporting group work with computers. The multi-user and interactive computer-based tools based on CSCW study are called groupware. Ellis, Gibbs & Rein (1993) define groupware as “computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment”.

CSCW researches have identified two major influences which impact on the success of CSCW and groupware: group member separation and people factors. Separation has two facets which are described by Nunamaker, Dennis, Valacich, Vogel & George and Rodden (1991) to be: (1) the form of interaction—the time dimension—and (2) the geographical dispersion of the group members—the spatial dimension. This is illustrated in the matrix in Figure 3.1. People factors

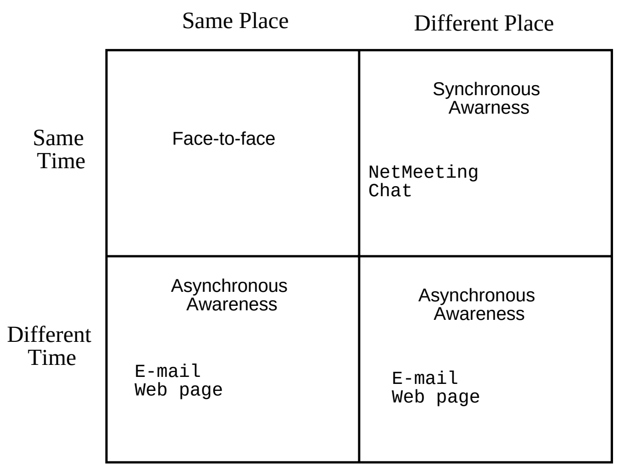


Figure 3.1: CSCW Time-Location Matrix

involves group politics, individual's emotion, personality and how people work in groups.

CSCW is used in a variety of time/location settings. For example, CSCW is used in 'masked' face-to-face meetings where the participants are co-located but anonymity of contribution to the discussion is necessary, as well as in wider distributed contexts where the major purpose is usually getting together irrespective of geographical divide. E-mail and newsgroups are examples of CSCW in use where synchronous communication is not highly important. On the other hand, video-conferencing and chat are examples of CSCW use in situations where asynchronous communications is inappropriate.

CSCW is not the only way to support groups working with shared data and goals. Multi-user systems, such as Computer-Aided Design (CAD) and Database Management Systems (DBMS), such as Oracle, also offer this support. CSCW and the latter differ in the protocols they employed to access the shared data and to perform tasks (Session and Floor Control, Section 3.3.1). CSCW groupware provides awareness information as well as access to shared data. DBMSs, on the other hand, are mainly concerned with secure access to the shared data. Strict locking protocols are used in DBMSs to handle concurrency. Concurrency mechanisms are often more flexible in CSCW and may be supported by social protocols.

To achieve successful CSCW support across the various time/location settings, specific techniques are required. These techniques impose challenges that need to be addressed. We discuss some important challenges in Subsection 3.3.1.

3.3.1 Challenges facing CSCW support for XP

In the preceding section we established that CSCW offers the most potential for computer-based support for XP. Nevertheless, CSCW groupware applications face certain challenges. The significant challenges are:

- Group Member Accessibility: Accessibility is determined by the separation of members by location and time. This influences what can be done, when and how. Communication may be synchronous or asynchronous. For example, the Planning Game requires adequate synchronous communication mechanisms. Annotation of User Stories requires a persistence mechanism so that notes can be available asynchronously.

- Acceptability of Groupware: This is a general problem faced by groupware. Groupware tools should not get in the way of work or put remote users at a disadvantage to co-located users. Groupware that does not augment the group often fails.

  We attempt to address these issue by designing PAM so that it is unobtrusive, lightweight, and fits into the context of XP projects.

- Interoperability: Groupware systems typically augment group work. They must fit into the context of the domain in which they are deployed. Consequently, groupware must facilitate interoperability with existing and anticipated future systems.

  The architecture of PAM supports easy reuse/sharing of data (Chapter 6). We use an XML (Bray, Paoli, Sperberg-McQueen & Maler 2000) data store to achieve this goal.

- Awareness: Groupware applications are required to provide mechanisms for explicit and implicit communication. Part of the information content of the communication is used to facilitate awareness. This comes in many forms, such as telecursors that indicate user's position in a shared desktop. The challenge lies in selecting and developing suitable awareness mechanisms appropriate for specific groupware.

  We provide awareness information in PAM such as the location of users in the *Project Hierarchy* (see Figure 5.2, page 64).

- Session and Floor Control: Groupware systems may provide mechanisms to prevent unauthorized access and avoid conflicting changes by enforcing protocols for floor and session control. Session control deals with access rights to the shared data. On the other hand, floor control deals with permissions to perform certain tasks. For example, editing of collaborative objects may only be allowed by one person at a time.

  In order to comply with XP's team approach and shared ownership practices, we have relaxed floor control protocols.

- Privacy: There is usually conflict between what data and how much of it should remain private and shared in a groupware system. Issues of security, anonymity and accountability influence privacy mechanisms.

  In PAM, all project data is shared through replication. To maintain XP's shared ownership practice, we do not offer private access to the project data. PAM is an information radiator for the project. Therefore, users who wish to experiment with personal ideas may do so outside of PAM.

- Development and Training Costs: Applications cost money. In addition, training is necessary for users. We do not explicitly address this issue in our research.

3.3.2 Why CSCW support for XP?

Theoretically, CSCW complements XP. The collaboration protocols already in use in XP only need to be augmented with CSCW techniques. CSCW techniques, when appropriately adapted, support XP in such a way that neither its philosophy nor values are compromised.

In addition to the benefits discussed in Section 3.1, there are other reasons for supporting XP with CSCW. CSCW shares many characteristics central to XP. For example, CSCW facilitates learning and group work, both of which are central to XP. CSCW is primarily concerned with collaborative work. XP is inherently collaborative and its practices are optimised to support collaboration (Chapters 2 and 4).

XP relies on a team approach and the collaboration within the Team is well defined. Each team member (Section 2.2, page 14) has specific roles and responsibilities. During activities team members adopt the role of 'driver' or 'observer' and use social protocols to coordinate their collaboration. Several drivers can work in parallel without much difficulty. The incremental nature of XP and the small granular sizes of the artefacts ensures that revert operations and/or corrections are not expensive.

3.3.3 CSCW Support for Dispersed XP Teams

In the preceding sections we established that CSCW is an appropriate technology for computer-based support of co-located XP teams. But there are situations when it is necessary that the team be separated. The team may be separated when: (1) teams are necessarily and arbitrarily dispersed, (2) team members are highly mobile, (3) the on-site Customer is too expensive for the client (overseas clients especially), and (4) customer or a developer with specialised skill is shared between projects. These circumstances demand relaxation of the co-location requirement. However, consideration must be taken of the XP practices which are susceptible to team separation.

The relaxation of the co-location requirement of XP has been termed "Distributed Extreme Programming" (DXP) by Kircher et al. (2001). The de facto definition states that DXP is XP with certain relaxation on the close proximity requirement of stakeholders.

CSCW offers appropriate technology to support DXP. Several researches (Section 3.4) investigated CSCW support for dispersed XP teams (Pinna, Lorrai, Marchesi & Serra 2003, Rees 2002, Maurer & Martel 2002, Maurer 2002, Kircher et al. 2001, SourceForge.net 2003, Ellis 2000). Their general trend has been to develop web-based tools and combine them with the utility of existing tools such as

NetMeeting, e-mail, web pages, Wiki technology and so on. Combining the utility of disparate tools can be used to achieve common goals. However, this approach suffers from consistency and interoperability issues. We believe that dispersed XP teams can be supported by simpler means.

Compared with support for co-located teams, computer-based support for dispersed XP teams places extra demands on the supporting technology. Issues of awareness, session and floor control, member accessibility and so forth (Section 3.3.1), require keener attention and support. In this regard, we provide a summary of some significant researches in the following section.

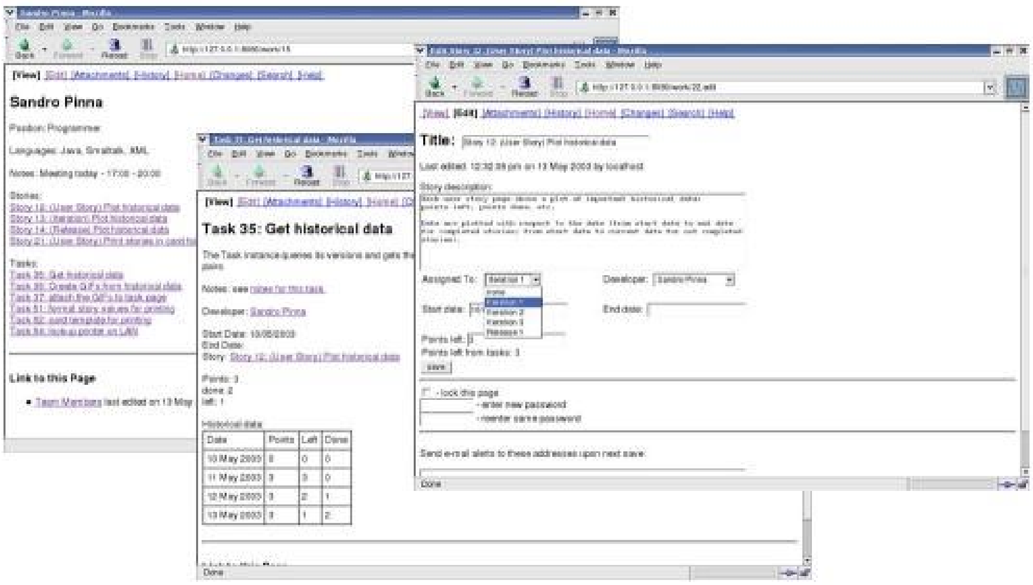
3.4 Related Work on CSCW support for XP

Various facets of XP have been prioritised in previous researches that offer support for dispersed XP teams. Some (Subsection 3.4.2, for example) support just User Story card management, while others (Subsection 3.4.1) have attempted more holistic approaches. We provide a critical analysis of these solutions in this section. This analysis also explains where our approach differs from and/or enhances previous work.

3.4.1 Developing a Tool Supporting XP Process

Description: XPSwiki is a web-based tool developed to support the Planning Game (Pinna et al. 2003). This tool is based on Wiki technology. Figure 3.2 displays a screenshot of some of its XPWiki pages. XPSwiki keeps track of project changes by means of a versioning system. It also manages data pertaining to releases, iterations, User Stories, Tasks and Acceptance Tests. Navigation of XPSwiki pages follows the natural tree-structure of XP project schedules and artefacts.

Critique: XPSwiki is a holistic approach to computer-based support for XP. However, it is limited in some ways: (1) it lacks adequate provision for synchronous interactive collaboration, which is central to XP; (2) it lacks an awareness mechanism; (3) it makes no provision for integration and unit testing; and (4) the Wiki technology used renders it susceptible to problems of timely resolution of conflicting updates.



Sandro Pinna

Position: Programmer

Languages: Java, Smalltalk, XML

Notes: Meeting today - 17:00 - 20:00

Stories:
Story 12: (User Story) Plot historical data
Story 13: (Iteration) Plot historical data
Story 14: (Release) Plot historical data
Story 21: (User Story) Print stories in card format

Tasks:
Task 35: Get historical data
Task 36: Create GIF's from historical data
Task 37: attach the GIF's to task page
Task 51: format story values for printing
Task 52: card template for printing
Task 54: look up profile on LAN

Link to this Page

- Team Members last edited on 13 May

Task 35: Get historical data

The Task instance queries its versions and gets the pairs

Notes: see notes for this task.

Developer: Sandro Pinna

Start Date: 10/05/2003
End Date:
Story: Story 12: (User Story) Plot historical data

Points: 3
done: 2
left: 1

Historical data

Date	Points	Left	Done
10 May 2003	0	0	0
11 May 2003	3	3	0
12 May 2003	3	2	1
13 May 2003	3	1	2

Title: Story 12: (User Story) Plot historical data

Last edited: 12:32:08 pm on 13 May 2003 by localhost

Story description:
[illegible]

Assigned To: Iteration 1 — Developer: Sandro Pinna

Start date: [illegible] — End date:

Points left: 3
Points left from tasks: 3

save

- lock this page
- enter new password
- reenter same password

Send e-mail alerts to these addresses upon next save:

Figure 3.2: Screenshot of XPSwiki web pages.

Significant feedback was received from users of XPSwiki. This feedback suggests that computer-based support for XP has the potential to promote the adoption of XP in industry, as well as helping developers learn the Planning Game.

3.4.2 A Feasible User Story Tool for Agile Software Development

Description: DotStories is a web-based tool developed to replace cardboard User Story index cards (Rees 2002). DotStories supports multiple projects. Projects are hosted as a collection of several web pages which define a User Story group. User Stories are presented as "User Story Teglets" (Rees 2002), a small rectangular area on a web page that facilitates direct manipulation, visualization of User Story cards and formatting features such as indenting, bold and so on. Common User Story manipulation tasks such as create, edit, split and delete are permitted. Teglets can also be grouped. DotStories support projects with several hundred stories.

Critique: DotStories partially addresses distributed XP. It is essentially a single-user tool with access to a collection of User Stories data. It does not support collaboration. Teglets afford direct manipulation, but this is only available in the client browser. Therefore, for group discussions involving more than two persons, teglet print-outs or a projector is required.

DotStories is an alternative to the normal User Story cards. However, even though Rees agrees that User Stories form the central work product of XP to steer the process, we view the one-dimensional approach of DotStories as limited in the context of the real needs of dispersed teams.

3.4.3 Process Support for Distributed Extreme Programming Teams

Description: Maurer & Martel (2002) introduced MILOS [1], a process-support environment for distributed XP teams. MILOS is web-based and offers support for project coordination, information routing, team communication and pair programming (Figure 3.3). It was originally developed to support process execution and organisational learning for virtual teams. MILOS includes features such as a WorkFlow Engine that handles task management issues; NetMeeting to support the Planning Game and pair programming; a Personal To-Do List, Help and user authentication. Common tasks such as create, delete and editing are allowed on User Stories and XP Tasks.

Critique: MILOS also partially addresses distributed XP, but offers more utility than Rees DotStories. MILOS also has limitations. First, the use of NetMeeting restricts it to the Microsoft Windows platform. It also limits conference participation to one-to-one sessions. Like DotStories, MILOS supports direct manipulation of User Stories, in addition to Tasks. However, the manipulation is not collaborative.

A significant finding from the use of this tool is that though video conferencing is desired, it is often unnecessary once the developers are already acquainted.

[1] Minimally Invasive Long-term Organizational Support

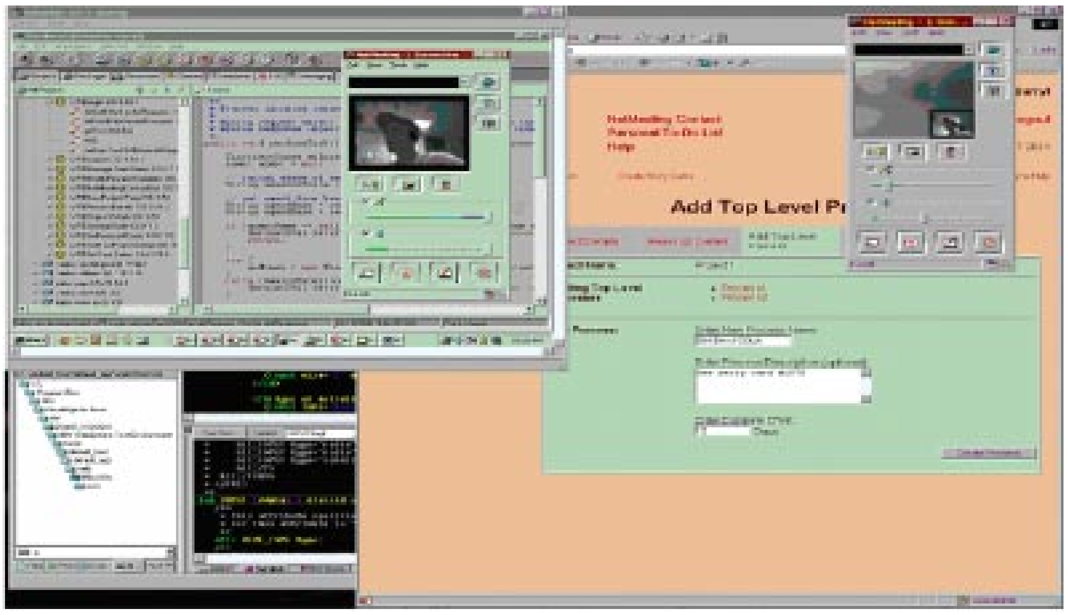


Figure 3.3: Snapshot of MILOS depicting an active pair programming session.

3.4.4 Support for Distributed Teams in eXtreme Programming

Description: Schümmer & Schümmer (2001) developed a tool called TUKAN, which supports distributed programming and the Planning Game. It is built on top of the COAST[2] (Schuckmann, Kirchner, Schümmer & Haake 1996) groupware framework. TUKAN offers virtual meetings, synchronous collaboration, communication channels and awareness information, such as where a programmer is in a source file and the 'distance' of one programmer from another with respect to potential conflicts. These, in addition to WYSIWIS and several visualisation techniques, are employed to support distributed pair programming (multiple persons in reality) and integration.

TUKAN virtual meetings use an 'activity explorer' in the Planning Game. Common User Story tasks and annotations are permitted. Communication is via audio and text chat.

[2] COAST supports highly interactive, synchronous applications and offers transaction-controlled access to replicated shared objects and optimistic concurrency mechanism.

Critique: TUKAN offers a holistic approach to support for distributed XP. Results from this research suggests that distributed pair programming may lead to a breakdown in strategic discussion. In addition, Schümmer & Schümmer (2001) present an in-depth analysis of the requirement for distributed, collaborative computer-based support for XP. This analysis highlighted communication, collaboration and coordination as essential requirements.

TUKAN does not offer direct manipulation of User Story cards. Through the use of COAST, TUKAN offers better communication and collaboration than tools that use NetMeeting. This is due to the use of replication, as opposed to the application approach used in NetMeeting. Information is not available about TUKAN treatment of User Stories transformations. Nevertheless, TUKAN demonstrates the feasibility of a holistic approach.

3.4.5 Distributed Extreme Programming

Description: Kircher et al. (2001) were the first to report an attempt to provide computer-based support for distributed XP teams. Their support came in the form of a tool called Web-Desktop. They claim that only four of the XP practices are affected in a distributed environment: the planning game, pair programming, continuous integration and the on-site customer. Kircher et al. (2001) suggest that effective communication in teams does not necessitate co-location. Communication can be supported through the use of technologies such as video-conferencing, e-mail and chat. The objective of this original research was to provide a working distributed environment that would keep distributed XP teams in the realms of XP.

Web-Desktop uses the client/server architecture model. Downloadable applications from a dedicated server handle most developmental and management processes. XP activities such as the Planning Game, pair programming, continuous integration and the on-site Customer are supported through the use of existing tools: video conferencing, inter-operable NetMeeting, e-mail and application sharing software with regular editors. CVS serves as the configuration management tool and integration is carried out directly from client systems.

A number of challenges were encountered in this research. However these provided useful insight into the challenges of CSCW support for dispersed XP teams. The major challenges faced during this research were: (1) one-to-one limitation of NetMeeting conferencing; (2) the inconvenience of using simple text file for capturing user stories; (3) operating system interoperability issues; (4) narrow bandwidth for conferencing; (5) lack of uniform access to the source code repository; and (5) internationalization issues such as keyboard layout and character mapping in different countries.

Critique: The limitations of this approach were mainly due to interoperability issues. Separate applications were used together for a common purpose. Since no modifications were made to adapt these tools to fit the demands of their new environment, it is not surprising that interoperability and cumbersome nature of tools and artefacts became an issue.

This research demonstrates that existing tools can be used to support distributed XP teams. However, special consideration must be made in terms of interoperability, applicability and flexibility of the selected tools.

The results appear to suggests that a flexible, custom tool would offer the best potential to support distributed XP.

3.4.6 Section Summary: An Analysis of Existing CSCW support for XP

All of the solutions presented in researches have focused primarily on User Story management and/or pair programming. This, we believe, stems from the fact that User Stories drive XP development projects, and that pair programming is highlighted by Beck (1999*b*) as a vital part of XP. Further, these solutions (apart from Web-Desktop) focused on support for specific tasks rather than the augmentation of XP team collaboration.

The major drawbacks of these approaches include: (1) one-to-one collaboration limitation imposed by NetMeeting, (2) overhead involved in setting up and maintaining disparate tools, (3) irregular time-delayed updates of web-based solutions using technologies such as wiki and (4) inadequate suport for synchronous collaborative activities such as User Story elicitation during the Planning Game.

These researches, however, provided useful insight into CSCW support for XP. Significant requirements, design issues, pitfalls and difficulties have been reported. We use these to guide our design and implementation of PAM. Our analysis of computer-based support for XP indicates that computer-based support is required to be lightweight, consistent, fully collaborative, unobtrusive and must support users' choice of development tools.

3.5 *Our Approach*: *Custom Desktop-based client/server CSCW Groupware*

We implemented PAM as a desktop-based client/server groupware tool. Our implementation decision is influenced by the findings of previous work, our analysis of the activities of XP teams and close examination of XP's project structure (Chapter 4). A major objective of our approach is to support teams by enabling them to sustain the XP philosophy and values irrespective of whether they are co-located or not. We believe that the philosophy is more important than the practices. Nevertheless, we are cognizant that the practices are essential support for the philosophy.

3.5.1 Motivation of our research

We question the overall benefit of the web-based approach used in the researches discussed above. The support of composite activities and tools, such as pair programming and User Story elicitation, appear to be the major thrust of these researches. This task-oriented approach suggests that the interdependence of activities, the various roles and responsibilities within the Team, and the purpose of existing XP tools—User Story cards and the whiteboard, for example—were not considered. Nevertheless, the drawback of these web-based tools may be attributed to the limitations imposed by the immaturity of the technology.

We believe that the way XP activities are structured (or unstructured) is deliberate. For example, User Stories and the collaborative Planning Game bridge the gap between the non-technical business-oriented Customer and the technical, non-business oriented Developers, by providing a mutually understandable interface to support requirement elicitation. It is imperative that new tools developed to augment XP activities complement and integrate with existing tools and practices. Otherwise, their introduction will cause contradictions, such as interference with the spontaneity of the normal XP User Story elicitation activity.

In previous researches, new metaphors and activity patterns were often introduced to achieve certain ends. Rees's Teglets, though they maintain the card metaphor, sacrificed collaborative direct manipulation of the cards. The high degree of individuality introduced by this tool, may result in a breakdown in the normal XP communication. The opportunities for knowledge sharing, which the Planning Game supports, may then be lost.

With PAM, our aim is to reduce the cognitive distance between the mental models of the computer supported distributed XP environment and the real world collaboration and coordination in normal XP. We expect that PAM will fit into the natural flow of the normal XP environment.

3.5.2 Why use a desktop-based client/server approach?

Two dilemmas concerning the choice of appropriate deployment of PAM were encountered early in our research. These were whether or not to choose (1) Desktop-based or Web-based platform and (2) Client/Server or Peer-to-Peer architecture. Previous researches reveal that communication, coordination and collaboration are essential to the success of computer-based support for XP. These issues have caused much difficulty for web-based tools. Our decision to use a desktop-based client/server approach is based on the following considerations.

Desktop-based versus Web-based

Web-based and desktop-based distributed systems have relative advantages and disadvantages. There is also a significant difference in the capabilities of the technologies currently available to support these platforms. Web-based technology has more reach (as in browsers and the Internet), but desktop-based technologies offer more sophisticated support for HCI demands and novelty of groupware applications.

We use the desktop-based approach because it offered mature technology. The primary reasons are:

1. To support highly interactive, collaborative activities such as Developers'

negotiation for taking Tasks responsibility (page 83) and planning meetings when User Stories are negotiated, the spontaneity of the normal XP practices must be maintained.

2. Computer-based support is equally important for co-located and dispersed teams.

3. Desktop-based applications, in combination with network technologies, provide a convenient way to prototype applications before optimising and porting to the web.

4. We presupposed that the use of a desktop-based approach will provide insight to guide the development of future versions of PAM for web-based deployment.

Client/Server versus Peer-to-Peer

The other deployment dilemma faced was how to choose between client/server and peer-to-peer architecture. In a peer-to-peer configuration, disadvantages may be manifested as risks. Risks may include (1) asymmetric maintenance across the project and (2) interleaved updates, which can lead to issues of data integrity and availability. An advantage is the freedom of self-administration and keeping data private. On the other hand, client/server deployment has the advantage of central administration, but the concomitant disadvantage of having a single point of failure.

We use the client/server architecture because it supports higher degrees of data availability and integrity. Further, the client/server architecture is appropriate for the replication information sharing technique (see Section 8.2). The client/server architecture requires point-to-point communication for collaborating with client applications. This is used for critical data management operations. On the other hand, broadcast communication is required for communication of transient information such as chat messages and awareness.

3.6 Summary

XP projects are vulnerable to the hazards of the 'paper office'. This presents many real and potential problems, such as, loss of information and inefficient traceability. These consequences are mainly the result of inadequate techniques used to keep track of XP project (in particular, the Planning Game). Several researches have been carried out to alleviate these issues, in addition to relaxing the co-location requirement.

Previous researches have been predominantly web-based. The web offers wider dispersal of XP teams. However, the technologies available for web-based applications are not sufficiently mature to support the highly collaborative group work typical of XP projects. This has resulted in various degrees of success in previous approaches.

Our approach differs from previous researches. We employ CSCW techniques with a desktop-based client/server as opposed to a web-based approach. We highlighted that two forms of communication, point-to-point and broadcast, are required to satisfy the multifaceted communication requirements and operations in the PAM groupware. In the following chapter we discuss the activities, people, interactions and tasks supported by our PAM.

Chapter IV

Computer-based Support for XP Activities

In Chapter 3 we identified the benefits of CSCW support for both co-located and dispersed XP teams. We discussed the reasons why support for the Planning Game and project coordination take priority over support for development activities such as pair programming and debugging. We developed several models of the XP process based on information available from a variety of sources (see Chapter 2) in order to understand the work flow and relationships that exist in normal XP. We build on the previous chapters by using our model of the User Story life-cycle to show the XP activities for which we provide CSCW support. We also discuss the various roles and responsibilities of the XP Team, and finally, how they collaborate to accomplish project goals.

The discussion of this chapter sets up (Chapter 6) in which we discuss implementation details with specific emphasis on exploiting CSCW techniques. We conclude that computer-based support for XP activities with the most collaboration has the greatest potential to benefit XP teams.

4.1 XP Concerns, Relationships and Activities

We present various perspectives of the XP process in Figures 4.1 through 4.4 and Table 4.2 through 4.4. We call these tables the Activity Tables. Each figure and

Schedules	**Artefacts**		
	Analysis	Design	Implementation
Project Release Iteration	User Story	Task Acceptance Test Spike	Source Code Test Case

Table 4.1: Classification of primary XP project concerns.

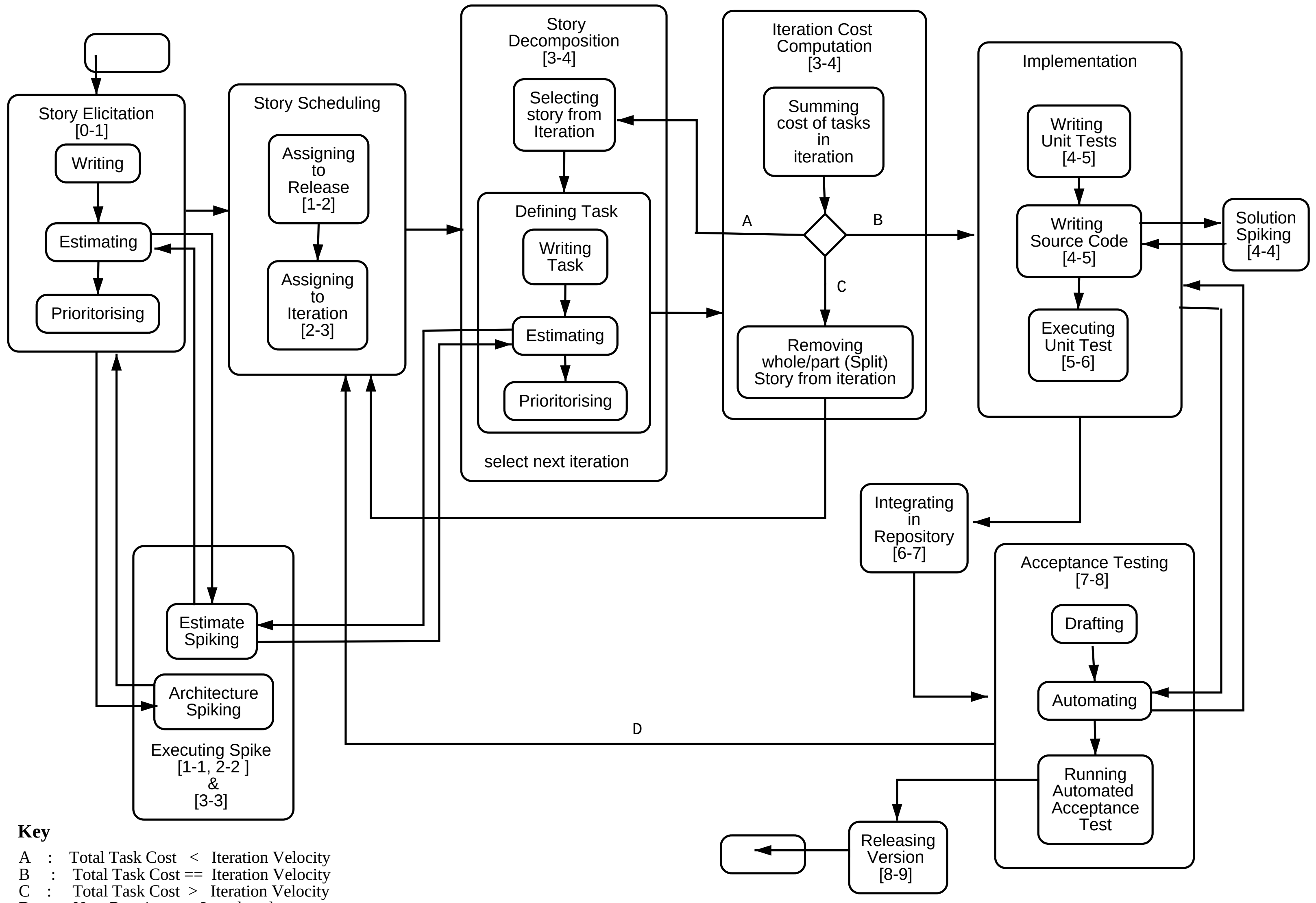
table highlights specific aspects of XP, such as the hierarchical arrangement of its schedules and artefacts (Figures 4.2 and 4.3). The various activities, roles, interactions, project concerns and constraints involved in an XP project are interdependent. This is due, in part, to the process' incremental, collaborative development and intensive testing characteristics. We use the various perspectives to complement one another in order to describe the relationship, project concerns and activities involved in XP.

We have classified the primary project concerns in Table 4.1. In this table and in Figure 4.3, the project concern *Task* is a programming task derived from decomposing User Stories during Iteration planning (Section 2.4, page 23). *Test Case* describes code written to automate Unit and Acceptence Tests. We separate this concept to highlight the relationship that exists between tests code and the source code of the software under development. We introduce these project concerns here because we reference them in the rest of this chapter and in Figure 4.3, which describes other relationships that exist between them.

4.1.1 Aspects of Computer-based Support for XP

Our analysis of the Activity Tables shows that XP revolves around the transformation of User Stories. From these tables and Figure 4.4 we derived the life-cycle of a User Story. We illustrate the life cycle in Figure 4.1. This figure shows how features of the problem being solved are captured (that is, the process of story elicitation) and processed until an incremental version of the system being developed is released. Further, when Figure 4.1 is cross-referenced with the Activity Tables, it can be seen that most activities affecting the User Story life-cycle involve collaboration between the Customer and Developers. The exceptions are programming and integration activities.

We observed that the Planning Game activities (Table 4.2) always involve collaboration between the Customer and Developers. In Section 3.1 (page 28) we discussed that the Planning Game practices put XP at risk, because of their long term consequences. Further, the Planning Game is a Project Management activity, while programming and integration are aspects of Application Development (Table 2.1, page 16). We believe that Application Development is adequately supported with tools such as Concurrent Versions System (CVS). The separation


Story Elicitation [0-1]
Writing
Estimating
Prioritising
Story Scheduling
Assigning to Release [1-2]
Assigning to Iteration [2-3]
Story Decomposition [3-4]
Selecting story from Iteration
Defining Task
Writing Task
Estimating
Prioritising
select next iteration
Iteration Cost Computation [3-4]
Summing cost of tasks in iteration
A
B
C
Removing whole/part (Split) Story from iteration
Implementation
Writing Unit Tests [4-5]
Writing Source Code [4-5]
Executing Unit Test [5-6]
Solution Spiking [4-4]
Integrating in Repository [6-7]
Acceptance Testing [7-8]
Drafting
Automating
Running Automated Acceptance Test
Releasing Version [8-9]
D
Estimate Spiking
Architecture Spiking
Executing Spike [1-1, 2-2] & [3-3]

Key

A	:	Total Task Cost < Iteration Velocity
B	:	Total Task Cost == Iteration Velocity
C	:	Total Task Cost > Iteration Velocity
D	:	New Requirement Introduced

Figure 4.1: State Transition Diagram illustrating the life cycle of a User Story. The numbering of the composite states correspond, broadly, to the transitions in Figure 4.4.

of activities, in combination with our analysis of the User Story life cycle provides an appropriate means for us to define the boundary of PAM. We conclude that computer-based support for XP activities with the most collaboration has the greatest potential to benefit XP teams.

Based on the preceding, we choose to provide computer-based support for some aspects of the User Story life cycle. These are: (1) Story Elicitation, (2) Story Scheduling, (3) Story Decomposition and (4) Unit and Acceptance Testing. This selection is reflected in the composite states in Figure 4.1. We also keep track of source code files produced as part of programming activities (Chapter 8) and provide support for communication. These aspects of XP are supported in our current version of PAM (Chapters 5 through 8). The features are:

1. **Virtual Meetings:** All of the aforementioned aspects of XP rely on communication and feedback. Therefore, we provide communication support in the form of: (1) a Chat utility, (2) a Virtual Desktop utility and (3) an interactive Sketcher utility. Collectively these utilities support Virtual Meetings.

   Virtual meetings have potential to alleviate the problems experienced by large and/or dispersed teams in normal XP, such as sharing a common desktop or whiteboard during planning sessions/discussions. (Session is used to mean a period of collaboration involving informatoin sharing and working towards a common goal). For example, the limited space available at a desk or whiteboard may cause some members of the team to be unable to see the artefacts. This may affect their ability to make meaningful contribution.

2. **Collaborative XP Tools:** In addition to the Virtual Desktop, we provide collaborative tools to manage XP project concerns and keep track of the project's progress. They also support collaborative activities such as User Story scheduling. Further, we facilitate all of the primitive activities, such as writing User Stories, of the specific aspects—the composite activities—selected for support. These tasks are highlighted in Figure 4.1 as the discrete states of the User Story life cycle.

3. **Persistence and Session & Project Monitoring:** In order to alleviate the long term consequences of the Planning Game practices, we also support

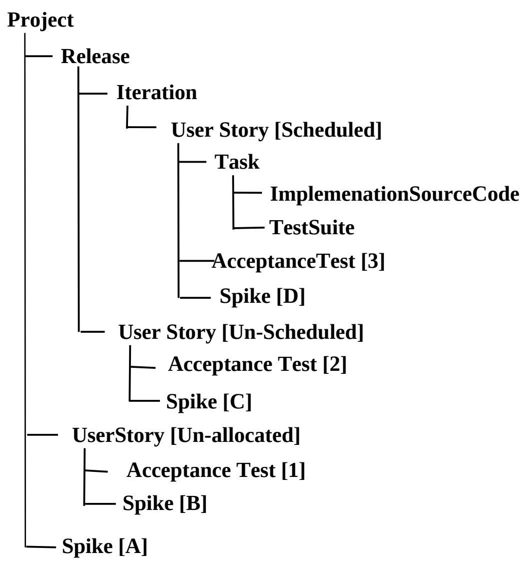


Figure 4.2: Tree representation of XP project hierarchy. The labels in brackets distinguish instances of the same type of concern.

persistence of project data. Persistence is responsible for ensuring that the project data survives over the duration of a session and from one session to another. The persistence mechanism resides on a server.

We also provide a monitoring system to keep track of activities that change the state of the project. The monitoring system tracks changes such as when and by whom project concerns are created and/or modified, and the session in which the user(s) were participating.

4. **Security System:** This supports authentication of users dispersed over a network. Ensuring that the shared ownership practice is not compromised and at the same time making provision for reusability and sharing of data involves a tradeoff in security. In our current implementation of PAM we support both aims. We do not impose strict security session control protocols. These considerations, however, are outside the scope of our research.

4.1.2 Relationships between XP Project Concerns Explained

In this and the following sections we describe the relationships that exist between XP project concerns and activities. We discuss these here because we exploit them in the user interface of PAM (Chapter 8, page 116).

Figure 4.2 shows that XP concerns exhibit a natural hierarchy. This is derived from the Class Diagram in Figure 4.3, which illustrates another perspective. These figures show that a Project is made up of Releases which in turn are made up of Iterations. Each of these may be associated with mutually exclusive User Stories, which are themselves related to Tasks and Acceptance Tests. A User Story must be allocated to an Iteration before it can be associated with (decomposed into) Tasks. Tasks and Acceptance Tests are implemented through programming operations to produce source file Resources.

Figure 4.3 also highlights specialised concepts and relationships involved in XP. For example, Acceptance Tests depend on Tasks to validate User Stories. The region of the figure bordered by the rectangle, highlights operations. These operations are related to the Application Development aspect of XP. As stated above, this aspect of the XP process is outside the boundary of our research. We include the classes depicted in the shaded region to show the relationship between Resources and the project concerns—Tasks, Acceptance Test and Spikes—to which they are indirectly related in our implementation of PAM (page 111).

The classes Project, Release, Iteration, User Story, Task, Acceptance Test, Spike, UnitTest Case and Source Code in Figure 4.3 correspond to the classes classified in Table 4.1 and described in Chapter 2. The class named Operation is an abstract class that encapsulates the events and activities that are required to implement and/or carry out the specific objectives that are specified in the Task, Acceptance Test and Spike classes.

Figure 4.3 does not explicitly show the sequence of activities involved in a project. We describe these activities in the following subsection, with the aid of Figure 4.4 and the Activity Tables.

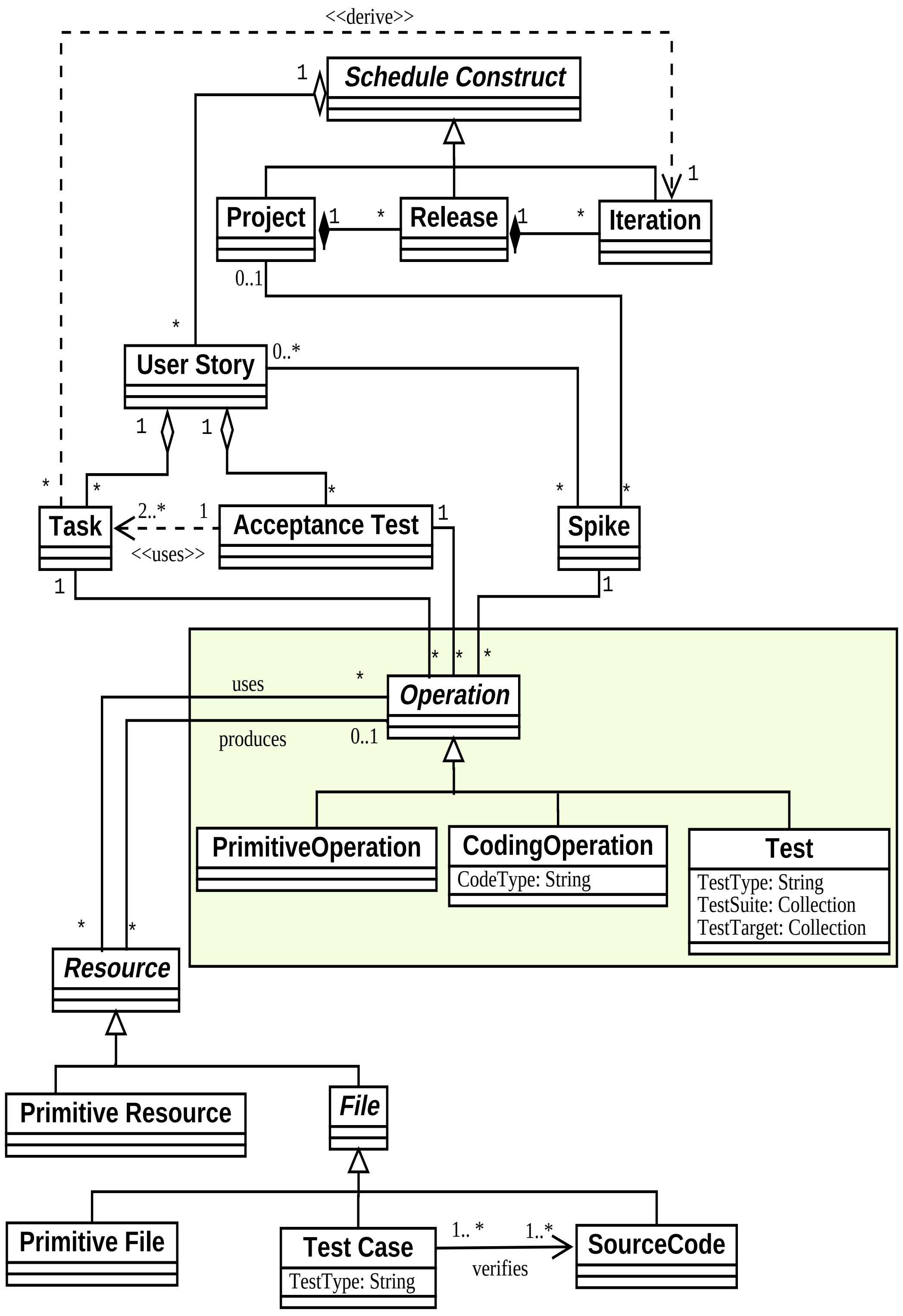


Figure 4.3: UML Class diagram illustrating the relationships that exist among XP project concerns.

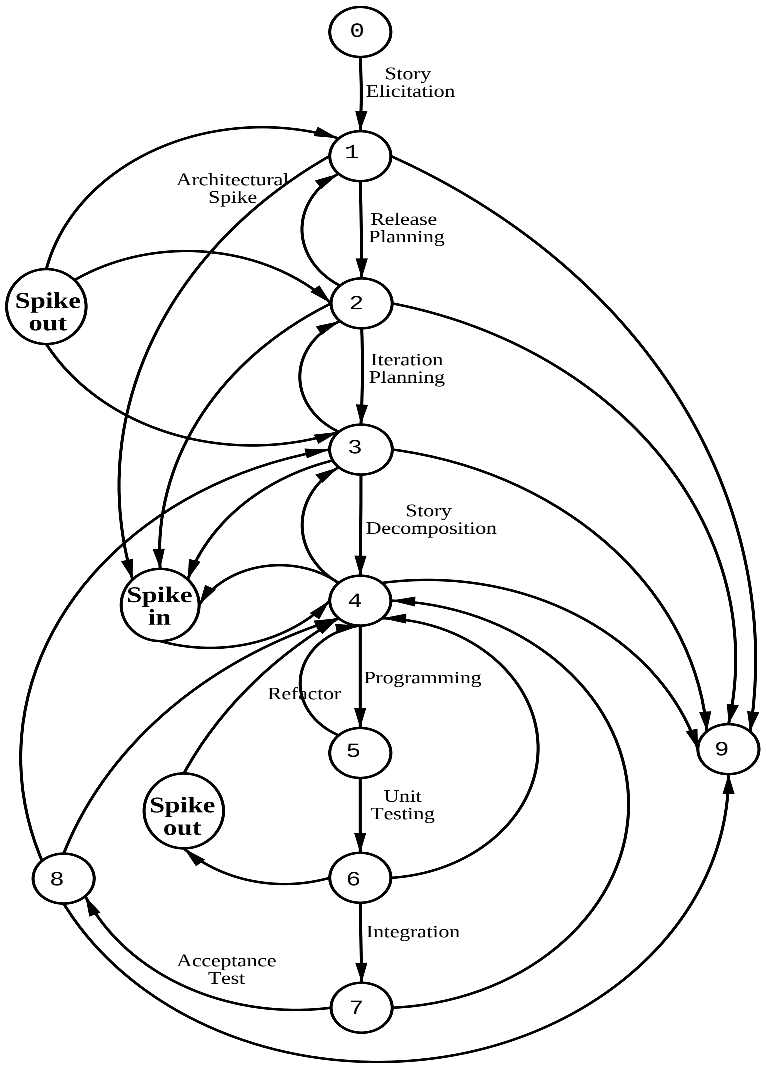


Figure 4.4: Finite State Diagram illustrating the forward and feedback transitions of the XP process. The state labels are for identification only.

4.1.3 The Ebb and Flow of XP Projects

XP is a highly iterative process in which progress is made in small steps. This has two major effects: it makes problem localisation and correction relatively easy and it increases the number of places/times at which critical decisions concerning the project can be made. For example, the Customer is empowered to make decisions like whether or not to cancel the project if progress is not to their satisfaction.

We model XP as a Finite State Diagram (FSD) in Figure 4.4 to illustrate the primary XP activities that influence project state changes. Usually, XP is described using separate diagrams, each of which usually addresses a specific aspect of the process. Our FSD highlights the various states at which critical decisions are made, such as at the beginning of Release and Iteration Planning (state 1 and 2 respectively), User Story decomposition (state 3) and after Acceptance Tests (state 8) are executed. The FSD also highlights that a decision to cancel the project may be made at several states. This is a result of the quality of feedback obtained on XP projects. However, the FSD does not show the low level task associated with the transition activities. We use the Activity Tables to complement the FSD to describe these primitive tasks.

The list of activities in the Activity Tables follows the main thread through an XP Project. The tables show (1) the activities involved in each transition (for example, transition ‘1 to 2’, “Release Planning”), (2) the pre and post-conditions that cause and result from the transitions, and (3) the team members’ interactions and the roles involved. It also shows what conditions may cause the project to return to a previous state. This does not necessarily involve corrections. For example, when an Iteration is complete, the project takes the transition ‘from 3 to 2’ to commence the next Release. This is in fact indicative of progress in the process.

Spikes are shown with ‘Spike In’ and ‘Spike out’ states, which serve as connection points to keep the diagram from being clustered. In the Activity Tables we indicate Spike transitions with a single asterisk. The double asterisk indicates that Acceptance Tests may be written for User Stories at any time prior to and during the Iteration in which the User Story is implemented.

The Activity Tables show that XP is inherently collaborative, since all of the activities involve some form of collaboration. The Release and Iteration Planning activities (Table 4.2), which correspond to the Planning Game, involves the collaboration of customer and developers, and at times the entire team. In normal XP, User Story cards support this collaboration. We posit that support for these activities, because of the diversity of the collaborating participants, will benefit the entire team. With adequate support everyone has the opportunity to know the state of the project provided they can establish a connection to the PAM server.

Table 4.3 shows that apart from executing Acceptance Tests and writing new User Stories, the programming, testing and integration activities are primarily carried out by collaborating developers. As we stated in Chapter 2, page 25, application development activities are adequately supported by existing tools such as CVS, compiler and debuggers. In addition, there is ongoing research to investigate support for distributed pair programming (Baheti, Gehringer & Stotts 2002, Hanks 2003).

State		**Actors**		**Sequence of Activity**	**Condition**		**Collaboration**	
fr	to	Driver	Observer		Pre-	Post-	Manual	DXP
0	1	C	Team	Write User Stories	Project Initiated	Collection of User Stories	√	√
1	4*	D	C	Do Architectural Spike	System Architecture unclear	alternative architectures	√	
6	1	D	C	Adopt a spike alternative	spike executed	Project architecture, metaphor	√	
1	2	D	C	Estimate User Story cost	user story exist		√	√
		C	Team	Prioritise User Stories	estimates available	All stories well defined	√	√
		C	Team	Define Release(s)	sufficient stories to start project		√	√
		C	Team	Assign Stories to Release	release duration set	Draft release plan	√	√
2	1	Team		Revise release plan	current plan impractical	improved release plan	√	√
2	4*	D	C	Do Estimate Spike	Uncertainty wrt story cost	spike solutions implemented	√	√
6	2	D	C	Update story estimate	spike executed	Improved estimates		
2	3	D	C	Define Iteration(s)	Release plan drafted	Iterations per Release set	√	√
		C	D	Assign Stories to Iteration	previous iteration complete/full	Draft iteration plan	√	√
3	2	Team		Revise Iteration plan	schedule creep	up-to-date iteration plan	√	√
3	2	Team		Commence next Release	previous release complete	Iteration plan initiated	√	√
3	4*	D	C	Do Estimate Spike	Uncertainty wrt story cost	spike solutions implemented	√	√
6	3	D	C	Update story estimate	spike executed	Improved estimates		
3	4	D	C	Decompose Story into Tasks	iteration plan set	collection of tasks	√	√
		D	C	estimate tasks	tasks defined	initial task estimates	√	√
		D	C	Iteration cost evaluated	stories in iteration decomposed	iteration cost	√	
						Iteration plan confirmed	√	√
		C		**Create Acceptance Test	User Story complete	Acceptance Test Defined	√	√
4	3	C	D	Revise Iteration plan	Sum(task) $\neq$ Sum(story) estimate	well defined iteration	√	√
4	3	Team		Revise Iteration plan	schedule creep	adjusted iteration plan		
4	3	Team		Commence next Iteration	previous iteration complete	Story decomposition initiated	√	√

Table 4.2: Typical XP Features of the Planning Game (Release and Iteration Planning).

State		**Actors**		**Sequence of Activity**	**Condition**		**Collaboration**	
fr	to	Driver	Observer		Pre-	Post-	Manual	DXP
4	5	D	D	Write unit test	Task defined	test suite/case(s)	√	
4	5	D	D	Do SpikeSolution	Uncertainty wrt task algorithm	alternative algorithms	√	
4	5	D	D	Write source code	Task defined		√	
					Unit test written	source code module(s)	√	
4	5	D	D	Write Acceptance Test	Acceptance Test defined	automated acceptance test(s)	√	
4	5	D	D	Refactor existing code	obsolete/complex code	improved code/system design	√	
4	5	D	D	Fix code errors	Error/Bugs in code	patched code	√	
5	4	D	D	schedule refactoring	code design needs updating	improved/extended module	√	
5	6	D	D	Execute Unit Test	source code module written	test results obtained	√	
6	4	D	D	Schedule Bug/Error fix	Unit Test failed	task revisited	√	
6	4	D	D	Schedule next task	Unit Test pass	task to encode selected	√	√
6	7	D	D	Integrate source code	unit test pass	new version of system	√	
7	4	D	D	Schedule Bug Fix/refactoring	Integration breaks current version	obsolete/bad module localised	√	
7	4	D	D	Schedule next task	Integration successful	next task to encode selected	√	
7	8	C	D	Run Acceptance test(s)	New version ready	Results of acceptance test	√	
8	3	C	D	Write new User Story	Changed requirement needed	new user story added	√	
8	4	D	D	Schedule Bug/Error fix	Acceptance test reveal bug	new task added	√	
8	4	D	D	Schedule next task to encode	Acceptance Test passed	new version	√	
8	9	Team		End project	application inappropriate	customer cancels project	√	
8	9	Team		End project	application complete	working software	√	

Table 4.3: Typical XP Features of application development (Programming, Testing and Integration).

State		**Actors**		**Sequence of Activity**	**Condition**		**Collaboration**	
fr	to	Driver	Observer		Pre-	Post-	Manual	DXP
1	9	C	Team	End project	application inappropriate	customer cancels project	√	
2	9	C	Team	End project	application inappropriate	customer cancels project	√	
3	9	C	Team	End project	application inappropriate	customer cancels project	√	
4	9	C	Team	End project	application inappropriate	customer cancels project	√	
		C	D	Common Operations				
		C	D	–change	concern detail inaccurate	updated concern	√	
		C	D	–remove	concern no longer needed	refined project requirement/definition	√	
		C	D	–Split user story	story too large	refined user stories	√	

Table 4.4: Common Operations of the XP process; including project termination criteria.

4.2 Summary

In this chapter we showed that the roles, activities, project concerns and relationships involved in XP are interdependent. We presented various perspectives of the process to illustrate and describe the various aspects of the interdependence. For example, the Activities Tables, lists the set of activities that are carried out as part of the XP process. Based on our analysis of these perspectives and consideration of the long term consequences of the Planning Game practices, we decided to provide support for specific aspects of a normal XP process, such as User Story elicitation and User Story Scheduling. The support is implemented in our experimental prototype system, PAM.

In the following chapter we describe PAM from a users perspective. This description sets up the remaining chapters in which we discuss the design and implementation issues and decisions.

Chapter V

The CSCW Enabled Information Radiator

In this chapter we introduce our prototype system, which we have named PAM. PAM is a distributed system with a client/server architecture (Section 3.5, page 44). We discuss the details of its deployment in Chapter 6. All XP team members interact exclusively with the PAM client. A designated person (perhaps the Coach) will be responsible for administrative tasks such as setting up projects and user administration. We describe PAM as a CSCW enabled information radiator. This description captures the metaphor used to guide the development of PAM and emphasises PAM's primary purpose.

In the preceding chapter we established the aspects of XP which are supported by PAM. We described our analysis and explained why specific aspects were selected. In this chapter, we describe how these aspects are supported by features in the current version of PAM. First we discuss the fundamental user interface components and metaphors, as well as related feedback from informal evaluations of PAM. We believe that this will aid in making the ensuing discussion about the use and utility of PAM clearer. We advise that PAM is not an Integrated Development Environment (IDE). Further, PAM in its current version is an experimental groupware application only, and is not intended for commercial use.

5.1 The Big Picture: A Tool called PAM

The user interface of the PAM client consists of three major components: (1) a Project Hierarchy display, (2) a work area and (3) a communication utility (left, center and right of Figures 5.1 and 5.3 respectively). These components correspond to the three activities, coordination, production and communication, of the Clover Design Model(Sire et al. 1999) for groupware applications (see Section 8.1, page 112).

The components are part of a common window, thus forming a composite

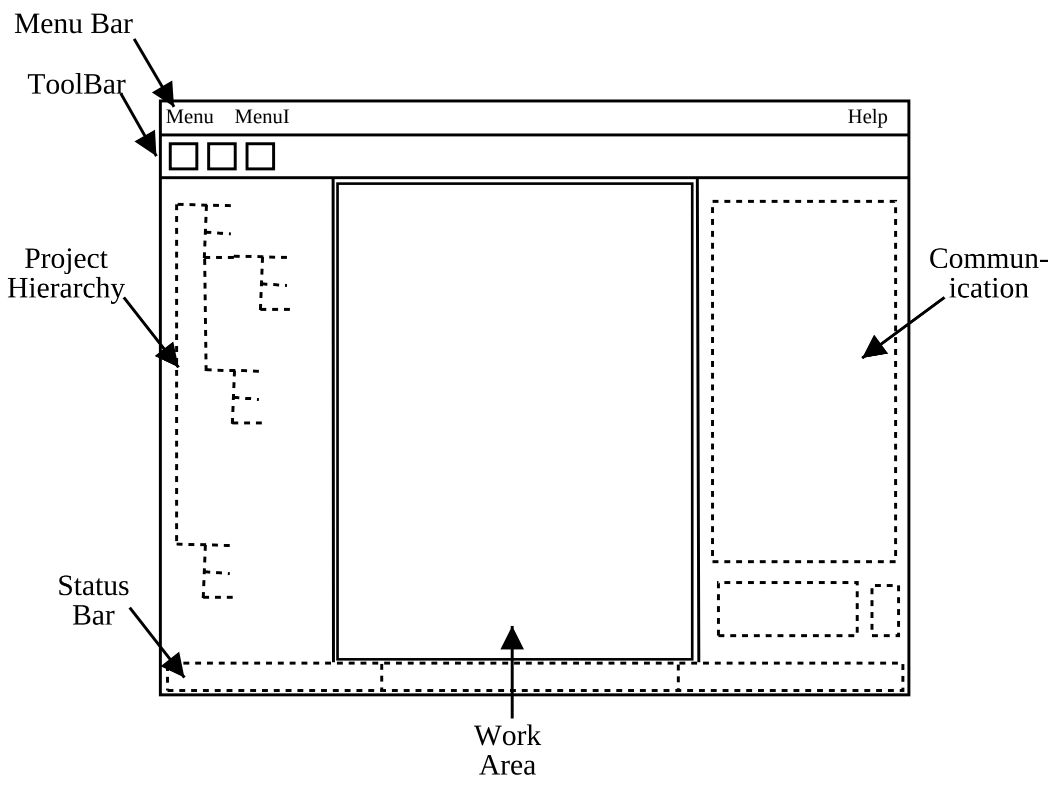


Figure 5.1: Illustration of the prototype system User Interface model

interface in which they are deliberately kept together. This configuration reduces the need for users to switch between windows in order to work and collaborate. Anecdotal evidence reveals that switching between screens and windows (for example, an instance of NetMeeting and a separate collaborative text editor) is distracting and counter-productive. This observation is supported by informal tests (Laurillau & Nigay 2002). Nevertheless, independent single-window tools can be supported.

The **Project Hierarchy** display reflects the natural hierarchical arrangement of XP projects concerns. This is illustrated in Figure 5.2. The Project Hierarchy is used in the UI for navigation and as part of the awareness mechanism, which provides users with information on the location and activity of others. Together, these functions assist in coordinating interaction between users and in the interface. A user's location is indicated in the Project Hierarchy by a list in brackets to

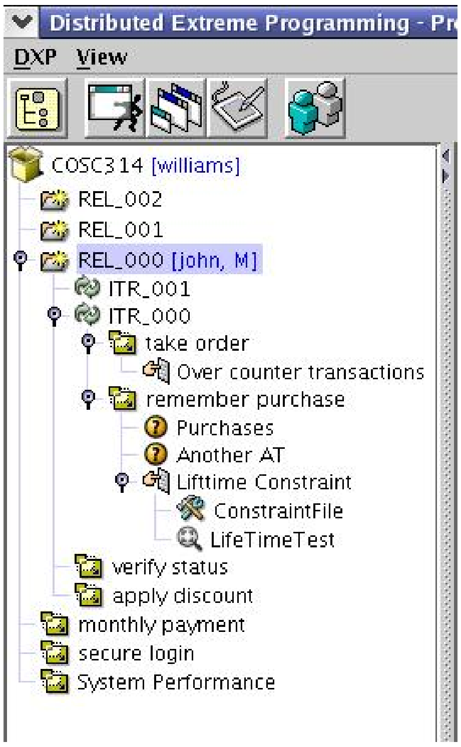


Figure 5.2: Snapshot of Project Hierarchy illustrating the breakdown of an active project and the location of users.

the right of the element name. For example, Figure 5.2 shows that users *williams* and *john* are working at the level of Project (COSC314) and Release REL_000 respectively. The user whose screen is being shown in the example, has logged in as Malo (indicated by the first letter only, *M*) and is at Release REL_000, along with *john*. This scenario may be an indication that *john* and *malo* are collaborating on some aspect of Release REL_000. In this situation *john's* and *malo's* work areas will the same, while *williams'* will be different. Alternatively, *john* and *malo* may be working in different views. This example indicate the flexibility of the floor control protocol in PAM. This form of flexibility in groupware where users are able to collaborate but are free to view different aspects of the shared information space is referred to as *relaxed-WYSIWIS (What You See Is What I See)* (Stefik, Bobrow, Foster, Lanning & Tatar 1987). Tools which employ this technique are termed *collaboration-aware* (Lauwers & Lantz 1990). The awareness information

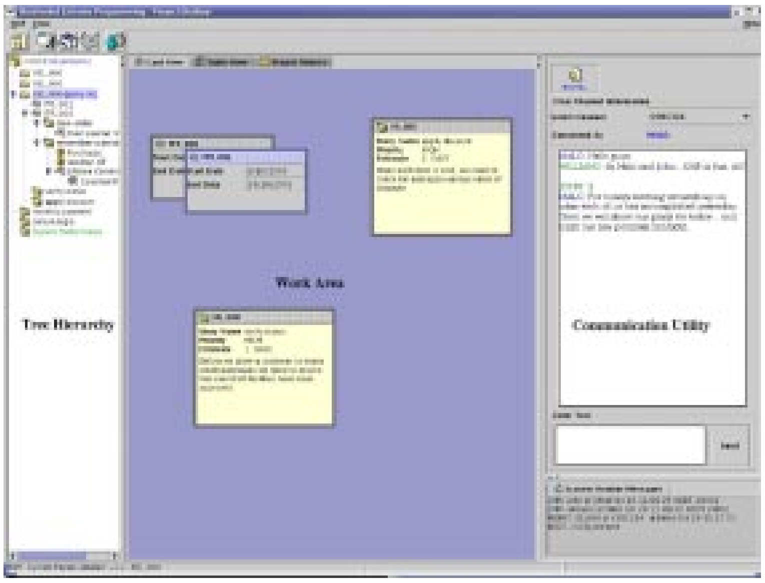


Figure 5.3: PAM client User Interface showing the Tree Hierarchy, Work Area (currently used by the Virtual Desktop) and the Communication Utility components.

helps to avoid conflicts.

In terms of navigation and coordination within the interface, when a user selects a specific element in the Project Hierarchy, the context of that project concern is brought into focus. This is evident in Figure A.2, which illustrates the Virtual Desktop containing representations of the project concerns that are immediately associated with Release REL_000: Iterations ITR_000 and ITR_001 and User Stories US_002 and US_000. A user may follow another, to synchronise their screens, by selecting the same element in their Project Hierarchy. Since there are three possible interfaces which may occupy the work area, users may use the Chat utility to agree on a common view. This overhead may be alleviated by providing awareness of which view each user is currently using (See Chapter 9, "Future Work"). Such HCI elements are not crucial for our proof of concept system.

The communication utility is made up of two components: (1) a **Chat** utility and (2) a system activity notification panel, called **System Monitor Messages**. These are illustrated on the right of Figure A.2. The current version of the Chat utility supports text chat (See Section 9, Future Work). The activity notification panel to the bottom of the Chat interface keeps a log of activities affecting the project and session state. Users are made aware of project state changes as they occur and notification of when users join and leave the current session. In addition, users may refer to System Monitor Messages to view the history of activities for their session. This is another facet of the awareness mechanism.

The work-area is the main component of the PAM client UI. It serves as a place holder for the three utilities primarily responsible for production. These are:

1. **Virtual Desktop:** We use the metaphor of a 'Virtual Desktop' (Reference Section 8.4, page 116) to describe our emulation of the common desktop used on XP projects during the Planning Game. In the Virtual Desktop, collaborative objects (the shared information about a single project concern, such as a User Story) are represented as cards (See Figure 5.3). The cards support direct manipulation with remote feedback. They may be moved around and grouped during meetings. For example, Figure A.8 shows users *john* and *williams* negotiating for Tasks by moving them to a place of their convenience in the interface. When a card is moved by a user, that user's name appears in the titlebar.

   The Virtual Desktop supports filtering (See Figure A.5, Appendix A), Drill Down (navigating from a parent concern to show its associated children in the Virtual Desktop) and Common Operations (see Subsection 5.2.3).

2. **Stacked Slates:** We use the metaphor 'stacked slates' to describe the layout of forms in the tabbed pane (Reference Chapter 8.4, page 116 and Figure A.3, Appendix A). The forms in the Stacked Slates are primarily used to update the details of current data and, in the case of Release and Iterations, to provide collaborative support for scheduling User Stories (Subsection 5.2.4).

   The Stacked Slates also support navigation. This happens when a user switchs between tabs, such as moving from the project level (Figure A.6) to a specific Release, REL_000 for example, (Figure A.7), by clicking on the "GoTo..." button. Navigation is also supported between levels—users can move, for example, from Figure A.6 to Figure A.3 by clicking on the

"User Story" tab. Figure A.3 displays Tasks and Acceptance Tests which are associated with User Story US_001.

3. **Sketcher:** The current version of the Sketcher utility (Figure A.4, Appendix A) supports collaborative drawing of simple diagrams. Sketcher is included as an example of the sort of tools that can be implemented in future versions of PAM.

All of the preceding components and utilities complement each other to fulfill the various aspects of PAM outlined in Section 4.1.1 (page 49). For example, The Sketcher and the Virtual Desktop complement the communication utility to fulfill Virtual Meeting aspect.

Before we discuss user interactions with these utilities, we will discuss some UI issues.

5.1.1 Of Relaxed CSCW and Informal Feedback...

In our implementation of PAM we use relaxed-WYSIWIS to support users working with the same or various aspects of the shared *Project Document* (page 121) from different perspectives. Strict WYSIWIS is used for direct manipulation of the cards in the Virtual Desktop and for User Story scheduling. XP projects typically have between 80 to 120 User Stories. If these User Stories are distributed across ten Releases then there will be at approximately ten User Story cards being handled at a time. In addition, the small granular size and discreteness of the collaborative objects, we believe, do not warrant the overhead involved in providing strict WYSIWIS capability for Common Operations. For example, a User Story is only usually a sentence or two long and, thus, does not take long to create or update. Consequently, character level strict-WYSIWIS is not needed. To illustrate our point, we suggest the contrast of our User Story with a shared document, such as a source code file, which is significantly larger. PAM provides lightweight awareness of imminent Common Operations in order to alleviate the lack of strict-WYSIWIS. Notification is provided in the System Monitor Message panel (bottom-right of Figure A.4, Appendix A).

Feedback from informal evaluation of PAM reveals that the relationship between selected nodes in the Project Hierarchy and what appears in the Virtual

Desktop and the Stacked Slates is counter-intuitive (Reference Figure A.3, Appendix A). This observation is, however, skewed. While users of IDEs such as Together ControlCenter (Borland Software Corporation 2003) express confusion, infrequent IDE users do not. Frequent users of IDEs expect that the details of the selected collaborative object will be displayed. However, in our implementation, the relationship between the two components is 'selected parent'–'displayed children' (or 'displayed descendants' by menu option; see Figures 5.6(a) and A.5).

Here again, the small granular size of the artefacts and their discreteness is the reason behind the interface design. The artefacts by themselves convey little information. Compare, for example, the User Story in Figure 2.1(a) (page 22) with Figure A.3 (page A.3). It is obvious that the contextual information in Figure A.3 provides much more useful information. Further, collaborative objects are not frequently edited once created; especially after forming children relationships.

Formal user evaluation might appear to be the logical way to test PAM. However, this must be done in the context of long term studies. These studies must first seek to determine whether the concept of distributed extreme programming is valid. Provided that DXP is found to be valid, the next best step would be to carry out an evaluation to compare client/server and web-based approaches to DXP. These general evaluations will provide the basis for evaluation of PAM.

Evaluations of PAM could be conducted to determine the suitability of the client/server architecture to support dispersed and co-located teams. Provided, here again, that these evaluations attest to the suitability of the architecture, further focused evaluations may then be considered. Focused evaluations will investigate specific aspects of the PAM system. HCI usability evaluations may be carried out on individual clients. These may subsequently be extended to involve collaborative usability. Nevertheless, proof of concept systems, such as PAM, do not depend on these focused evaluations to assist validation.

In the current implementation of PAM, long term evaluation is supported by means of the System Monitor logs (see Subsection 6.2.3, page 92). The logs are stored in XML format and keep track of transactions as they changed the state of the project. These logs may be used outside of PAM to analyse the evolution of projects and to determine how PAM is being used.

For meaningful information to be obtained from evaluation of PAM, the evaluations must be carried out in the context of a reasonable size project of approximately six months duration with a team of between ten to twenty members. Tests to determine whether PAM supports user queries, such as finding User Stories given a search value, and the effectiveness of the awareness mechanism can be carried out periodically. These evaluations will require substantial time and a stable human resources environment from which to draw participants.

The difficulties involved in the evaluation of PAM as a CSCW system arise from the many issues that must be addressed. These are: (1) HCI issues of usability and ergonomics; (2) distributed system issues such as concurrency, synchronisation and efficiency of communication; (3) people factors, group dynamics issues, individual needs and psychology, politics and so on; and (4) impact on XP projects with regards to pair programming, shared ownership and so on (Reference Section 3.2, page 33). In addition, the size and duration of a test project appropriate for a formal evaluation must be realistic. Insufficient resources would have made it necessary for one or more of the preceding facets be left out, thus compromising the rigour of the evaluation.

Instead, we conducted a number of informal evaluations involving a research group of five members. Feedback about the usability the user interface were used to guide subsequent versions of PAM. During these informal tests we observed that users were sometimes inclined to work individually. This may be attributed to the excitement of being the first one to break the system. However, we used this as encouragement for our use of relaxed CSCW techniques. Feedback from these evaluations also confirms reports from previous groupware evaluation researches. What we learned fits with other expectations; that is, that consideration of the socio-technical realities of a CSCW systems is concomitantly important to its success and makes it difficult to evaluate (Engelbart & Lehtman 1988, Baker, Greenberg & Gutwin 2001, Ross, Ramage & Rogers 1995, Grudin 1988).

Users examined the utility offered by PAM during informal evaluations. The participants were from two separate groups. The first group consisted of graduate Computer Science students who were involved in ad hoc testing of utilities such as User Story elicitation (page 77) and signing up for Tasks (page 83). Feedback from these tests were used to enhance the design of the user interface. The second group was comprised of members of a graduate research group. Informal testing with the second group was more structured.

In the latter evaluations, the research group collaborated in the Planning Game activities for an Inventory Management System. (The collaborative objects shown in the running example of the discussion about PAM features and support that follows were created as part of these evaluations.) In each evaluation session, one participant adopted the role of customer and the others were developers. First the participants engaged in User Story elicitation (see page 77). Several User Stories were created by the customer. Participants used the Chat utility extensively to communicate with each other as the User Stories were negotiated. During Release and Iteration planning (page 5.2.5) participants allocated User Stories to Releases and Iteration respectively. User Stories in the first Iteration of the first Release were then decomposed into Tasks (see Figure A.8). Participants used social protocols while using the Chat utility in order to coordinate their collaboration. All of the utilities offered in PAM were used during the course of these evaluations both as a matter of convenience and exploratory experiments. Feedback from these evaluations suggest that PAM offers relevant and worthwhile support for XP teams. In addition, feedback also suggested that some aspect of the user interface stand to benefit from attention to HCI usability expectations.

In the preceding discussion we presented the fundamental UI components of PAM and advised of some user feedback. We also discussed our reasons for using informal instead of formal evaluation to test the utility of PAM. In the following sections we describe how PAM supports group work on XP projects.

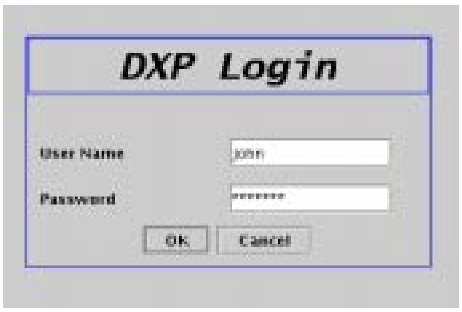
5.2 A Day in the Life of PAM

The PAM client is instantiated by running a shortcut script (the script contains the Uniform Resource Locator (URL) address of the PAM server) or from the command line. At startup, the user is presented with the dialog in Figure 5.4(a). The user supplies authentication data in the logon screen and clicks ‘OK’ to request authentication and permission to access projects maintained on the PAM server. Upon successful authentication by the PAM server, the projects to which the user has access rights are returned to the client application and displayed in the Virtual Desktop. There they are displayed in the form of cards (Figure 5.4(b)). The user then selects and loads a project.

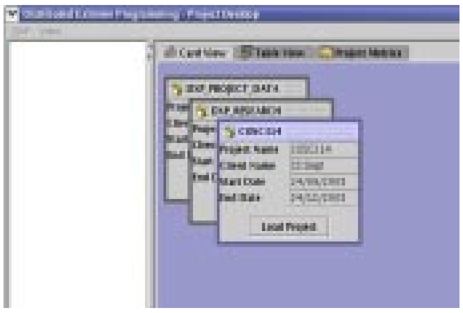
Type of collaborative object	Amount
User Stories	100
Releases	8
Iterations (10 per Release)	80
Tasks (10 per User Story)	1000
Unit Test per Task	1000
Acceptance Test per User Story	100
Total: 2288 @ 1kb = 2.288 MB	

Table 5.1: Breakdown of XP project in terms of project concerns to compute Project Document size.

When a project is loaded, the current session and project state on the server are replicated on the client system. The session state consists of awareness information such as who is in the current session, their location in the Project Hierarchy, temporal information such as which User Stories are marked for reassignment (See Figures A.6 and A.7, Appendix A) and so on. We discuss the awareness mechanism in Chapter 8. The project state consists of the *Project Document*, which we anticipate will typically have an approximate maximum size of 2.5 MB (We discuss the concept of the *Project Document* in Section 8.5.2, page 121). Table 5.1 shows the breakdown of project concerns for an example reasonable sized XP project. We chose the data replication sharing approach for strategic purposes. It differs from shared desktop and shared application approaches. We discuss the reasons in Section 8.2, page 112.



(a) Login Screen.



(b) Virtual Desktop showing Project cards.

Figure 5.4: Snapshots depicting login form and available projects.

While the preceding happens in the background, the user is presented with the Project Hierarchy, Virtual Desktop and communication composite interface. The exact display depends on the current state of the loaded project. For new projects, one node—the loaded Project—appears in the Project Hierarchy and no cards are displayed in the Virtual Desktop. For projects in progress, all of the project concerns are listed as nodes in the Project Hierarchy and cards for each concern immediately related to the Project, such as Releases and unallocated User Stories (see Figures 4.2 and 5.2), are displayed in the Virtual Desktop.

When the project is fully loaded, the user becomes an active participant in the session. The user can then interact with the rest of the system—that is, other participants and PAM. Goals are achieved through users performing operations. We discuss these in the following section.

5.2.1 User Interaction with PAM

PAM supports XP by:

- Alleviating the long term consequences of the normal Planning Game by providing persistence of project data. This alleviates issues that may arise from practices such as the use of a temporal whiteboard (see Section 3.1, page 29).

- Relaxing the co-location requirement by providing support for dispersed teams.

- making the project whiteboard—the information radiator—conveniently accessible. This supports collective understanding of the current state of the project, tracking and historical cross-referencing.

- Augmenting the stand-up meetings by means of Chat, Virtual Desktop utilities and shared data.

- Allowing customers to monitor the project whether they are on-site or not.

- Supporting server-side unit testing with real code, as well as client-side testing (mock objects most often used for client-side tests).

To obtain these values users are required to carry out operations in the PAM environment. Though other clients may be implemented differently, operations such as create, edit, delete, annotate and viewing the properties of project concerns are common to all collaborative objects. We refer to these operations as *Common Operations*. In the following subsections, we describe how a user navigates the PAM client user interface. We then describe how Common operations are performed. Finally we describe how other artefact and project state specific operations—for example, assigning a User Story to a Release and the splitting of User Stories—are performed.

5.2.2 Navigation in the PAM client User Interface

The fundamental components of the PAM client user interface provides various views of the *Project Document*. For example, the Virtual Desktop depicts the project as a collection of cards. The Project Hierarchy, on the other hand, as a hierarchical arrangements of nodes, while the Stacked Slates depict the project as a collection of forms. The work area can be occupied by only one of these production views at a time. This reflects a compromise between the reducing the overhead of users switching between screens and having access to multiple perspectives of the document simultaneously.

In Section 5.1 we describe how a user may navigate the Project Hierarchy and Virtual Desktop. We also described the parent-children relationship between the Project Hierarchy and Virtual Desktop on one hand, and the Project Hierarchy and Stacked Slates on the other. These views are synchronised. For example, if the Current Parent Artefact is Release REL_000 then the collaborative objects associated with REL_000 will be displayed in the Virtual Desktop or the Stacked Slates, whichever is visible. If the user navigates the interface and brings another project concern into focus while using the Stacked Slate view, the Virtual Desktop will be populated to show the immediately associated project concerns.

Synchronisation of the views is internal to a PAM client. Views are not explicitly synchronised between clients. This is a deliberate departure from the use of strict-WYSIWIS (Section 5.1, page 5.1). Users manually synchronise their views when they need to share the same view. We discuss this design decision in Chapter 8. Suffice to say, it supports flexible use of the interface. Nevertheless, the client application may be extended to provide a strict “follow me” mode such that collaborating users may be directed through the interface by a designated leader. The equivalent is available in PAM through the use of social protocols and Chat.

Navigation and operations are also supported in the client through the following components.

Toolbars: There are two types of toolbars in PAM: (1) the application toolbar (see Figures 5.1 and 5.3) and (2) the toolbars on the Stacked Slates and individual forms (see Figure A.7). The application toolbar allows a user to switch between the production views, as well as hide/show the communication view and/or the Project Hierarchy (either or both result in the work area being allocated more screen real estate). The form toolbars offer support for the Common Operations, such as save, revert, add note and so on. They also support visiting the next or previous tab in the Stacked Slates.

Menus: PAM has has an application menubar and popup menus that are available in the Virtual Desktop and Project Hierarchy. Actions performed in the application toolbar are also available in the menubar. The use of the popup menus is discussed in the next section, “Common Operations”.

Icons: Icons are used in the menus and toolbars. However, the important icons are those used in the Project Desktop, cards and tabs of the Stacked Slates. These icons are depicted in Figure 5.5, which is a cutout from the Stacked Slates. The icons represent the names listed, in addition to the Implementation, Acceptance Tests and Unit Tests resources respectively. These icons are used, similarly, in the Project Hierarchy.

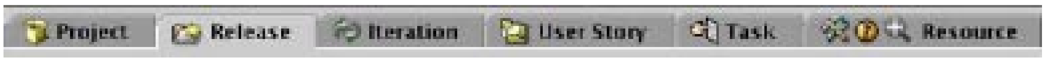
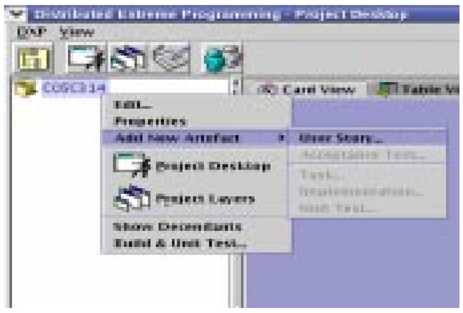
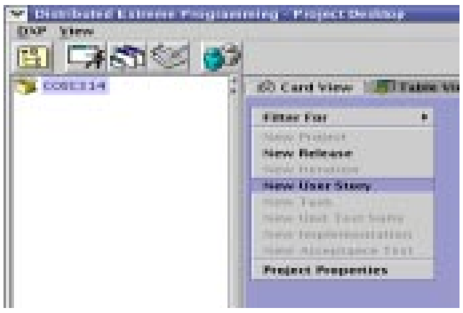


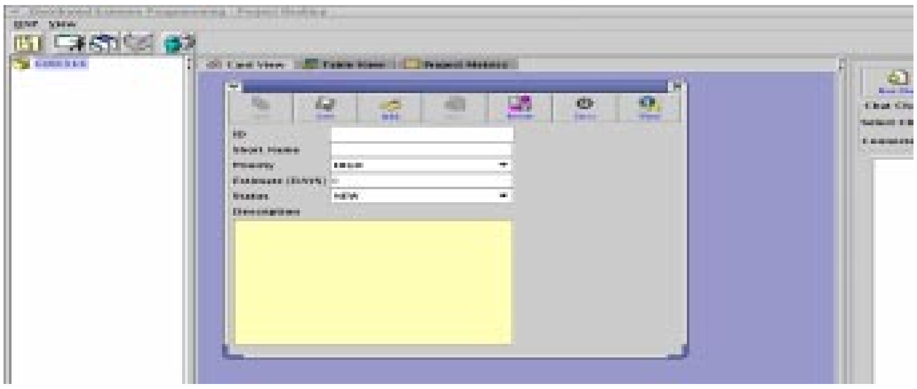
Figure 5.5: Icons in the client user interface



(a) Popup menu of the Project Hierarchy



(b) Popup menu of the Virtual Desktop



(c) User Story data input form

Figure 5.6: Snapshots showing menus and and example of a form used in Common Operations.

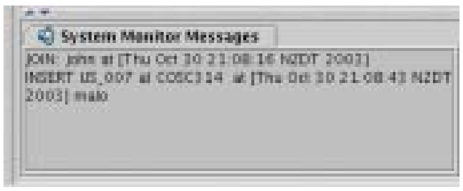
5.2.3 Common Operations

Common Operations are permitted from three places in the PAM client: the Virtual Desktop, the Project Hierarchy and the forms in the Stacked Slates. We use a User Story to illustrate how these operations are performed.

***Create*:** To create a User Story, the user defines the User Story in the client interface and then commits it to the system. This process is initiated with a right mouse click on either (1) the Project, Release or Iteration node in the Project Hierarchy, or (2) the Virtual Desktop. This action displays the menu in Figures 5.6(a) and 5.6(b) respectively. The user selects the appropriate menu option—“New User Story”, for example—to display the form in Figure 5.6(c). The User Story is defined by filling out the form with appropriate details. The user uses the menu bar to commit (“Save”) or alternatively, “Cancel” the operation.

The User Story becomes a collaborative object and part of the system when it is saved. The definition is sent (Section 7.3, page 104) to the PAM server where it is added to the User Story repository (see Subsection 6.2.2, page 91). The server then broadcasts (page 104) the new User Story to all connected PAM clients, including the sender. In so doing, the state of the project is shared in real time.

Remote stakeholders are alerted to the creation of the new User Story—the changed project state—by means of the awareness mechanism. Awareness is provided in these ways: (1) an entry is recorded in the System Monitor Messages panel (See Figure 5.7), (2) a User Story card is added to the Virtual Desktop, (3) a coloured node is added to the Project Hierarchy (we use the colours green, red and black to indicate new, recently updated and regular nodes respectively), and (4) if the new User Story has a child association with an Iteration, a row is added to the table in the Iteration tab of the Stacked Slates to reflect this relationship.



```
JOIN: john at [Thu Oct 30 21:08:16 NZDT 2003]
INSERT US_007 at COSC314 at [Thu Oct 30 21:08:43 NZDT
2003] malo
```

Figure 5.7: Snapshot of System Message Monitor

All project concerns may be created similarly, with the exception of the Project concern. The Project is created from the Virtual Desktop only, and by the Administration (most likely the Coach). The menus in Figure 5.6 display enabled/disabled options depending on what type of collaborative object can be added as a child of the Current Parent Artefact, that is, the selected element in the Project Hierarchy (see bottom left of Figure 5.3, page 65). For example, only User Stories can be added as immediate children of the project object (see Figure 4.2, page 52).

Alternatively, once a project is in progress and Iterations have been defined, User Stories can be added directly to an Iteration through the Iteration tab of the Stacked Slates form (see Figure A.7). The User Story definition is first created, then committed. The other collaborative objects may be created in the same conditional manner from the relevant tab.

***Edit*:** Edit operation on a User Story is initiated by a right mouse click on the User Story node in the Project Hierarchy or the User Story card in the Virtual Desktop or by using the menu bar in the "Iterations" tab of the Stacked Slates. Committing changes is similar to committing new User Story definitions.

***Delete*:** Delete operation is permitted only from the Project Hierarchy. The project concern being deleted must not have children associations.

***Annotate*:** A Note Editor is available to add notes to collaborative objects. Each note has sender, receiver, date and description data. Notes are added by right clicking on the card in the Virtual Desktop or selecting 'Add Note' from the toolbars available in forms. Adding a note is a form of editing, and, therefore, treated similarly.

***View Properties*:** While forms in the Stacked Slates display details of collaborative objects, the Tree Hierarchy shows only the name and/or ID and the Virtual Desktop shows only a subset of the details. To view the details of a collaborative object displayed in the Virtual Desktop or the Project Hierarchy the user right clicks on the collaborative object representation and selects 'View Properties' from the popup menu.

5.2.4 User Story Elicitation

User Stories form the first input into an XP project. They are usually accompanied by an architectural spike, which experiments with alternative architectures and develops a metaphor for the system (See Figure 2.2, page 26). In the normal XP Planning Game, the Customer is guided by the Developers in developing the

User Stories for the project. The Customer and Developers use social protocols in face-to-face meetings to flesh out the details of User Stories. Once consensus is obtained on the detail, the Customer writes a User Story on an index card and adds it to the project.

PAM supports collaborative story elicitation. It encourages the use of social protocols and facilitates meetings through the Chat and Sketcher utilities. Story Elicitation through PAM involves the Common Operations. The Customer may collaborate with Developers using the Chat utility to discuss User Story descriptions. Alternatively, the Customer may create some User Stories independently, and then meet with the Developers to discuss them. During elicitation, User Stories are created, edited, deleted and/or annotated as described in the preceding subsection. In addition, each User Story cost is estimated and then prioritised.

Estimating is done by the Developers and prioritising is done by the Customer. Both of these activities constitute Edit operations. During the process of estimating a User Story, the Developers may execute a Spike to improve the accuracy of the estimate.

When the Team is satisfied with the elicited User Stories, they are scheduled. We discuss the Story Scheduling process next.

5.2.5 User Story Scheduling

Story scheduling is an activity of Release and Iteration Planning (Section 2.3.1, page 16). The Customer assigns User Stories to Releases by order of their priority. The resulting distribution of User Stories and Releases makes up the Release Plan. Iteration Planning, unlike Release Planning, is a three part process during which the Team: (1) set up Iteration and assign User Stories, (2) decompose User Stories into Tasks, and (3) review and adjust Iteration based on the Iteration velocity. We describe part (1) below and show how PAM supports (2) is supported in Subsections 5.2.6. Item (3) is not automated in PAM but is supported.

***Setting up the schedules*:** PAM supports collaborative User Story scheduling. In order to schedule User Stories Releases are first created. This allows Iterations to be created within Releases (Figure 4.3, page 4.3 illustrates the composition association between the schedule constructs). Single Release and Iteration

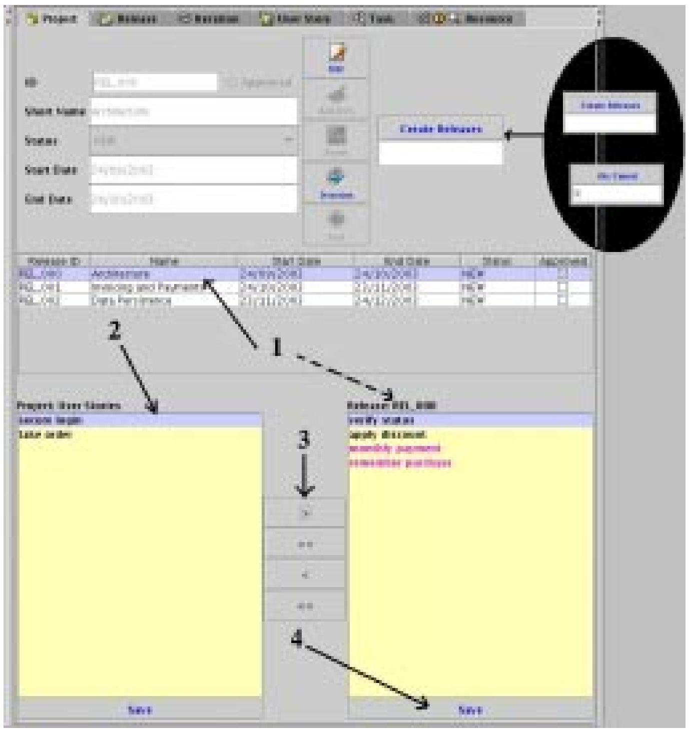


Figure 5.8: Release Form Tab illustrating the process by which User Stories are assigned to a Release. It also shows the two step process of creating Releases (the black circle with form cutouts).

collaborative objects can be created from the popup menu in the Virtual Desktop or Project Hierarchy. In this case, the Project or a Release node must first be selected in the Project Hierarchy. Alternatively, one or more Releases can be created using the Project tab in the Stacked Slates. The Stacked Slates imposes constraints on the order in which collaborative objects may be added. That is, the appropriate parent object must first exist.

As Figure 5.8 shows, the process for creating multiple Releases and Iterations is similar (see also Figures A.6 and A.7). The user clicks the "Create Releases"/"Create Iteration" button. The button label changes to show "Save/Cancel". The associated text box prompts the user to enter an amount. The user then commits the Releases/Iterations to the system.

Awareness of new Releases and Iterations takes place in a similar manner to a ‘create’ Common Operation. In this case Releases and Iterations are added to the Project and Release tabs respectively.

***Assigning User Stories to Schedules*:** When the schedules are set up, User Stories can be assigned to them. The user does this using the Stacked Slates. It should be noted that although it is possible to provide *Drag-and-Drop* operation to assign User Stories in the Project Hierarchy and Virtual Desktop, we do not. The reason is twofold. In PAM, the chances of missing the target with a Drag-and-Drop operation is higher than in a single-user application because other users will be changing the structure of the Project Hierarchy and the layout of cards in the Virtual Desktop. Further, undo operations are difficult in distributed applications.

Secondly, in normal XP, User Stories are moved back and forth between schedules before a final decision is made. We support this activity by allowing a User Story to be in transit. This is illustrated in Figure 5.8. The in transit User Stories, coloured in magenta, were moved from the Project level to Release REL_000. Users are free to move between Releases in the table to view the User Stories that they have been allocated. The user may also move User Stories between the Project and other Releases/Iterations.

To move User Stories between schedules, the user must perform four tasks. The user will: (1) select the target schedule in the table (User Stories currently assigned to, or in transit in the selected target schedule appear in the list on the right), (2) select a User Story in the left hand list and (3) click on the appropriate direction button located between the lists to move the selected User Story (or all) to the list on the right, where it/they are coloured magenta. Finally, the user will (4) commit the in transit User Stories to the target schedule. We illustrate this process in Figure 5.8 (the numbers indicate the steps).

We use WYSIWIS to support the activities involved in scheduling User Stories. When a User Story is moved between schedules, all PAM clients in the session are updated in real time. Users who have their views synchronised see the User Stories as they are moved from one list to the other. Further, any user in the session has the right to commit the assign operation. PAM clients joining the session while User Stories are in transit will be updated with appropriate session state information from the server. Consequently, the project and session state is consistent irrespective of when a client joins the session.

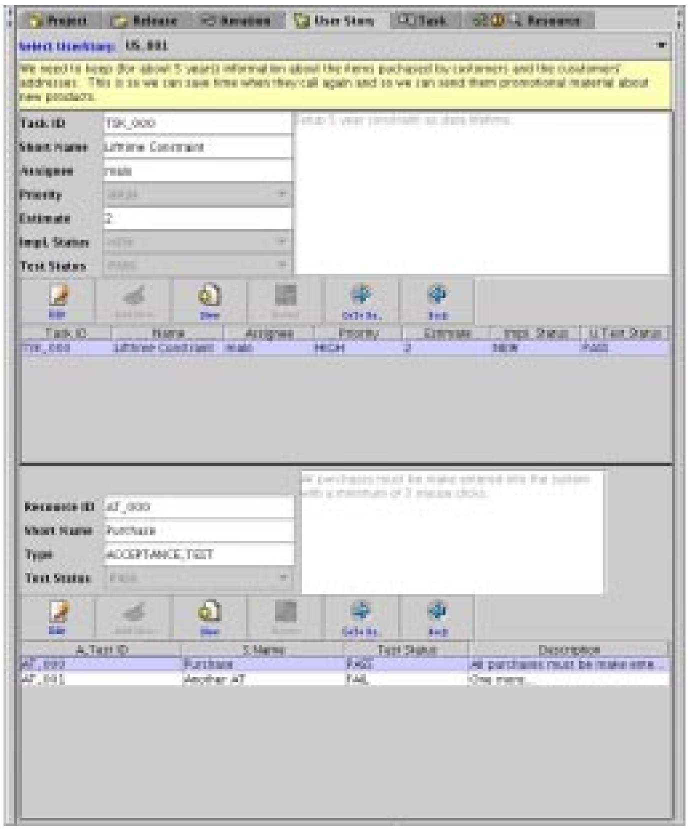
Creating Iterations and assigning User Stories to Iterations is performed in the same way. In this case, however, the unassigned User Stories from both the Project and the current Release (See Figure A.6) are displayed in the left hand list of the Release Tab. When there are in transit User Stories in the current Release, they appear in the left hand list, coloured in magenta. The awareness of in transit User Stories is transparent across both Project and Iteration tabs. Figures A.6 and A.7 shows User Story with short name *System Performance*, appearing in both the Project and Release Tabs because it has been moved to, and is in transit, in Release REL_000.

The use of relaxed-WYSIWIS allows users to collaborate while assigning User Stories to schedules or to join the activity at any time and still be updated appropriately. In the current version of PAM the awareness of in transit User Stories is limited to the Project and Release Tabs. This mechanism may be enhanced in future versions (see Chapter 9).

5.2.6 Story Decomposition

Story Decomposition is the second major aspect of Iteration Planning. In normal XP, User Stories are decomposed one at a time, but in PAM several User Stories may be decomposed simultaneously (see page 5.2.6). The Customer and Developers further analyse and discuss each User Story, in order to define the Tasks needed to implement it. When the Tasks are defined, they are estimated by the Developers.

Next, the Iteration cost is calculated (see Section 2.3.1, page 17). The total cost of all the Tasks in the current Iteration is compared with the current Iteration velocity, or with the number of ‘ideal programming days’ in the Iteration if it is the first Iteration. If the total Task cost is greater than the Iteration cost, then one or more stories are reassigned to the next Iteration or a User Story is split; whichever balances the Iteration. On the other hand, if the total Task cost is less than the Iteration velocity, then one or more User Stories are brought forward from the next



Select UserStory: US_001

We need to keep (for about 5 years) information about the items puchased by customers and the customers' addresses. This is so we can save time when they call again and so we can send them promotional material about new products.

Task ID	TSK_000
Short Name	Lifetime Constraint
Assignee	maik
Priority	HIGH
Estimate	2
Impl. Status	NEW
Test Status	PASS

Setup 5 year constraint as data lifetime.

Task ID	Name	Assignee	Priority	Estimate	Impl. Status	U.Test Status
TSK_000	Lifetime Constraint	maik	HIGH	2	NEW	PASS

Resource ID	AT_000
Short Name	Purchase
Type	ACCEPTANCE_TEST
Test Status	PASS

All purchases must be make entered into the system with a minimum of 3 mouse clicks.

A.Test ID	S.Name	Test Status	Description
AT_000	Purchase	PASS	All purchases must be make ente...
AT_001	Another AT	FAIL	One more...

Figure 5.9: Snapshot of User Story Decomposition Form

Iteration (see state "Iteration Cost Computation" in Figure 4.1, page 50). When the Iteration Plan is set, the Developers 'sign up'—that is, take responsibility—for a number of tasks (see Subsection 2.3.1, page 17). This constitutes the Iteration Plan.

Defining Tasks: PAM allows several User Stories to be decomposed simultaneously. However, the Team will need to use social protocols to structure the discussion and activity. We do not impose floor control, though this could be added to clients if desired.

User Stories may be decomposed in three ways: (1) in the Virtual Desktop, (2) from the Project Hierarchy and (3) in the User Story Tab of the Stacked Slates. In order to collaborate, the users follow the driver by synchronising their screen and location based on the location of the driver in the Project Hierarchy, with the aid of the Chat utility. We consider decomposition in the Stacked Slates. The user—in this case the Customer—selects the User Story from the combobox labeled ‘Select User Story’ (see top of Figure 5.9). The observers synchronise their location. The user—in this case the Developer—then performs a create Common Operation with the user of the toolbar. Notes may also be added to Tasks. Alternatively, Tasks are created from the Project Hierarchy or in the Virtual Desktop as described in Subsection 5.2.3 Common Operations.

Signing up for Tasks: Signing up for Tasks can be done in the Stacked Slates or in the Virtual Desktop. The Virtual Desktop is more convenient, because it supports users—in this case the Developers—negotiating for Tasks. During the signing up activity, all interested Developers synchronise their Virtual Desktop by selecting, in the Project Hierarchy, the relevant ancestor of the Tasks to be negotiated. If the ancestor is not also the parent of the Tasks, then the user will use the “Show Descendants” menu option (see Figure 5.6(a)) in the Project Hierarchy to show descendants of the selected project concern. Each User then filters their display for Tasks only (see Figures A.5 and A.8), after which they group Tasks of interest by dragging them to a location within the Virtual Desktop. When a card is dragged in the Virtual Desktop, the name of the user dragging the card is displayed in the title bar of the card. Figure A.8 show users *john* and *williams* negotiating for Tasks TSK_001 and TSK_000 respectively (this is from the perspective of user Malo). A Developer takes responsibility for a Task by means of an Edit Common Operation (future versions of PAM will support multiple select and drag, as well as a “Take ownership” operation. See Section 9).

5.2.7 Handling Spikes

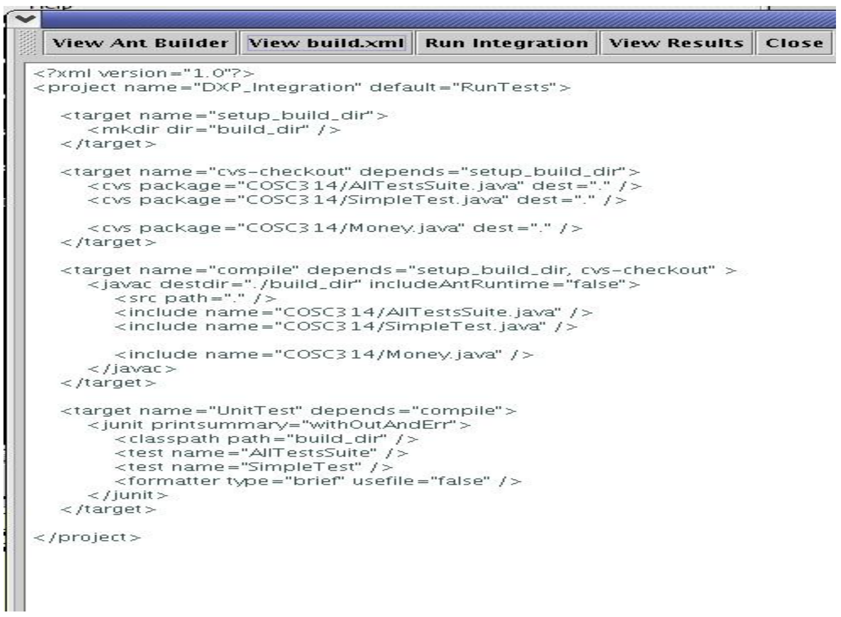
In PAM, spikes are treated as a special type of Task. AT the project level, architectural spikes are manifested as a special purpose User Story with associated Tasks. That is, a User Story is created to describe the objective of the spike. One or more Tasks are then created to define what has to be done to obtain the results of the

Figure 5.10: Snapshot of User Story Decomposition Form

spike and so on. Results from the spike are recorded as notes of the User Story. The process involved in performing Common Operations on Tasks is the same for spikes.

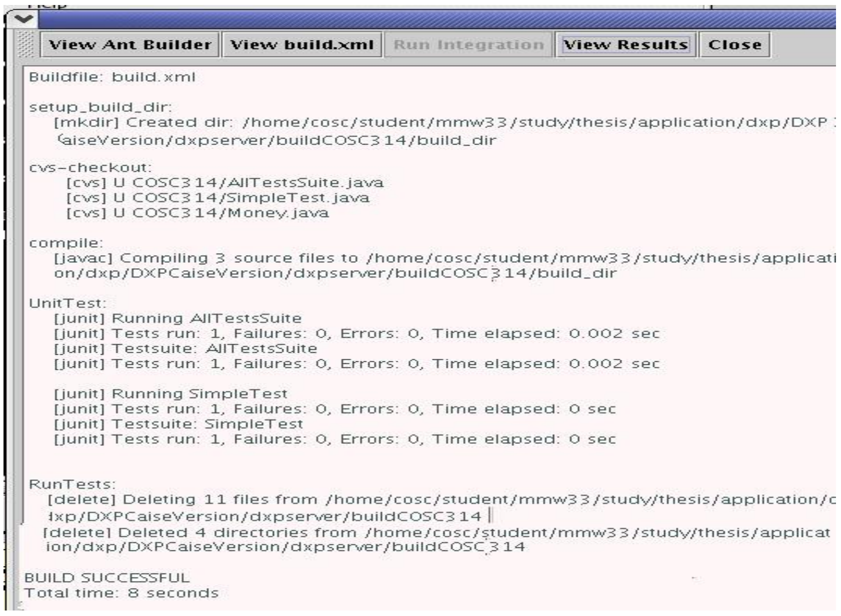
Why do we provide support for Spikes?

Information obtained from spikes (Section 2.4, page 24) is useful for historical referencing, project planning and coordination. Executing spikes involves activities of Application Development and thus are external to PAM. However, they may be documented in the normal way as Tasks. An example of the usefulness of recording spike results, regardless of the throwaway practice, follows. Architectural spikes conducted in the initial stage of the development of PAM compared several middleware alternatives such as CORBA, Java RMI and JGK (later renamed CAISE). Though the source code and design artefacts from these experiments have been discarded or survive only in the form of documentation, Chapter 7 serves as a record of the spike results.



```
<?xml version="1.0"?>
<project name="DXP_Integration" default="RunTests">

   <target name="setup_build_dir">
      <mkdir dir="build_dir" />
   </target>

   <target name="cvs-checkout" depends="setup_build_dir">
      <cvs package="COSC314/AllTestsSuite.java" dest="." />
      <cvs package="COSC314/SimpleTest.java" dest="." />

      <cvs package="COSC314/Money.java" dest="." />
   </target>

   <target name="compile" depends="setup_build_dir, cvs-checkout" >
      <javac destdir="./build_dir" includeAntRuntime="false">
         <src path="." />
         <include name="COSC314/AllTestsSuite.java" />
         <include name="COSC314/SimpleTest.java" />

         <include name="COSC314/Money.java" />
      </javac>
   </target>

   <target name="UnitTest" depends="compile">
      <junit printsummary="withOutAndErr">
         <classpath path="build_dir" />
         <test name="AllTestsSuite" />
         <test name="SimpleTest" />
         <formatter type="brief" usefile="false" />
      </junit>
   </target>

</project>
```

(a) Illustration of *build.xml* file generated by PAM.



```
Buildfile: build.xml

setup_build_dir:
   [mkdir] Created dir: /home/cosc/student/mmw33/study/thesis/application/dxp/DXP
   CaiseVersion/dxpserver/buildCOSC314/build_dir

cvs-checkout:
     [cvs] U COSC314/AllTestsSuite.java
     [cvs] U COSC314/SimpleTest.java
     [cvs] U COSC314/Money.java

compile:
   [javac] Compiling 3 source files to /home/cosc/student/mmw33/study/thesis/applicati
   on/dxp/DXPCaiseVersion/dxpserver/buildCOSC314/build_dir

UnitTest:
   [junit] Running AllTestsSuite
   [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.002 sec
   [junit] Testsuite: AllTestsSuite
   [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.002 sec

   [junit] Running SimpleTest
   [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0 sec
   [junit] Testsuite: SimpleTest
   [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0 sec


RunTests:
  [delete] Deleting 11 files from /home/cosc/student/mmw33/study/thesis/application/c
  dxp/DXPCaiseVersion/dxpserver/buildCOSC314
  [delete] Deleted 4 directories from /home/cosc/student/mmw33/study/thesis/applicat
  ion/dxp/DXPCaiseVersion/dxpserver/buildCOSC314

BUILD SUCCESSFUL
Total time: 8 seconds
```

(b) Illustration of the feedback from server-side execution tests defined in the above Figure.

Figure 5.11: Snapshot of Ant file and results screens.

5.2.8 Handling Tests

PAM supports the execution of local and server-side Unit Testing. Figure 5.10 depicts a snapshot of the Test Execution utility. The Test Execution utility uses the information— relative path names to files stored in the CVS repository on the server—maintained by Implementation and UnitTest collaborative objects. For example, Figure 5.10 shows that the *Money.java* file is an Implementation file, while the *AllTestsSuite.java* and *SampleTest.java* are examples of JUnit test suite and test case respectively (see also Figure A.5, card ID IMP_000 which is coloured peach). The Test Execution utility uses Apache Ant build system.

Users may select to use a prepared Ant build file or to generate one in the Test Execution interface by selecting Implementation classes and the Unit Test cases (suites) to test them and requesting that that a build file be created. The generated build file is displayed in the utility (see Figure 5.11(a)). The user may then run the test locally or on the server. In the case of a server side build and test, feedback is returned by the server and presented to the user in text format in the utility interface (see Figure 5.11(b)).

Acceptance Testing may be carried out similarly. However, in the current implementation, Acceptance tests can only work with prepared Ant build files.

5.3 Summary

In this chapter we presented our proof of concept desktop-based groupware application. We use the XP 'information radiator' metaphor and describe our groupware system to be a CSCW enabled information radiator, which we call PAM.

We discussed informal evaluations which were carried out to guide the development of PAM. These evaluations examined the utility offered by PAM and found that PAM offers relevant support for XP teams.

We also discussed the requirements of formal evaluations of PAM. This discussion highlighted that for formal evaluations of PAM to be meaningful, evaluation of the concept—DXP—on which PAM is base has to first be evaluated. Other aspects of PAM, which require long term evaluations, are required to be carried out before addressing the focused HCI usability issues. Though these evaluations are important, our proof of concept application does not depend on them.

PAM satisfies the aspects of XP which were identified in Chapter 4 to be beneficial to computer-bases support. In the latter part of this chapter we presented these utilities from the perspective of users. In the following chapter, we address the implementation details of PAM.

Chapter VI

PAM: Configuration and Component Architecture

So far, we have discussed the potential benefits that computer-based support offers XP teams (Chapter 3). We also showed that CSCW is an appropriate technology to provide this support. In the preceding chapter, we presented PAM—the CSCW Enabled Information Radiator—a proof of concept experimental groupware application. We described the utilities provided and showed how users interact with PAM to augment and extend the reach of the collaboration inherent in XP. In the design and implementation of PAM, we employed a mix of traditional and relaxed CSCW protocols to achieve specific ends. For example, while we use relaxed-WYSIWIS for Common Operations, we use more traditional WYSIWIS to assign User Stories to schedules and direct manipulation of cards in the Virtual Desktop (Section 5.2, pages 75 and 78). In Chapter 4 we discussed our analysis of the XP process and showed how we determined the activities that offered the greatest potential to return value from computer based support. In this in and subsequent chapters, we discuss the design and system architecture of PAM.

We discuss the architecture of PAM from the perspective of its configuration and components, connectors and interface in this chapter, and Chapters 7 and 8 respectively.

6.1 Deployment Architecture

PAM is implemented as a desktop-based client/server distributed system (Section 3.5). As discussed, this architecture offers the flexibility required to support the multi-faceted needs of computer-based support for XP. These include lowering risks of data loss and integrity, and promoting efficient collaboration by means of awareness and adequate support for communication. In addition, it supports achievement of specific goals (for example, User Story elicitation) by linking

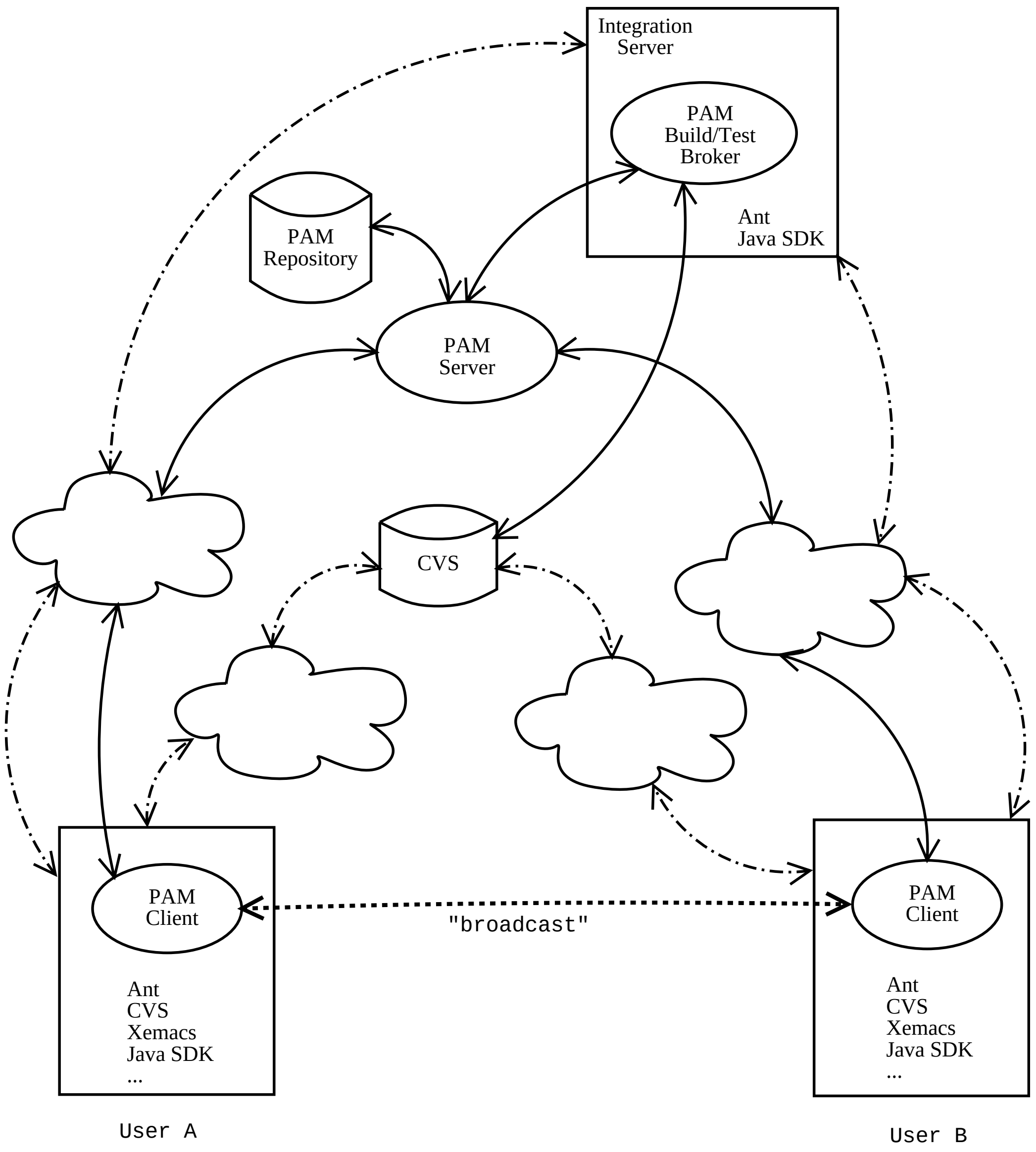


Figure 6.1: PAM Deployment Architecture

individuals, groups and collaborative objects involved in project state changing activities.

Figure 6.1 illustrates the deployment of the PAM. This figure shows how PAM complements the existing tools and communication channels used on XP projects. The dot–dash lines indicate the channels of communication used for operations such as retrieving a copy of source code from the repository. The integration server and the versions system—CVS—are tools typically used on traditional XP projects. The integration server is used to integrate new code into the latest good release of the system. The versions system allows users to work on the same project code, and assists them to identify and resolve conflicts when they occur. Despite shortcomings, such as the asynchronous feedback about conflicts and the overhead involved in resolving them, versions systems satisfy their purpose.

As depicted in Figure 6.1 we have added to the XP environment: (1) the PAM server and repository, (2) PAM clients, (3) the Build/Test Broker. These sub-systems are discussed in the remainder of this chapter. The solid lines indicate the communication channels that support critical data management operations involving PAM client interaction with the PAM server and repository. The dotted line, "broadcast", highlights the transient communication between clients such as chat and lightweight awareness such as the in transit state of User Stories (page 78). Broadcast communication in the PAM environment complements the face-to-face communication in XP, in addition to supporting real time awareness of project state changes. The two forms of communication highlighted here, meet the communication needs of PAM, and are discussed in Chapter 7.

6.2 PAM Server Sub-system

The primary components of the server sub-system are: (1) a Server Driver, (2) runtime Repositories, (3) a Persistence Manager, (4) a Build/Test Broker and (4) a Data Store (See Figure 6.2). The main purpose of the server is to provide a central point for maintaining the state of projects and brokering server-side operations for clients such as updating the Data Store, maintaining session state information, and user authentication.

The PAM server supports multiple projects simultaneously. Though these projects are independent, there is a communication channel between them. Users may use this channel to communicate across projects. The data and awareness information is project specific. PAM's support for several projects allows a single large project to be divided up into smaller 'XP-sized' projects. PAM allows each user to have access rights to several projects, but allows them to working with one at a time on a given server. We posit that this may be a way to address XP's

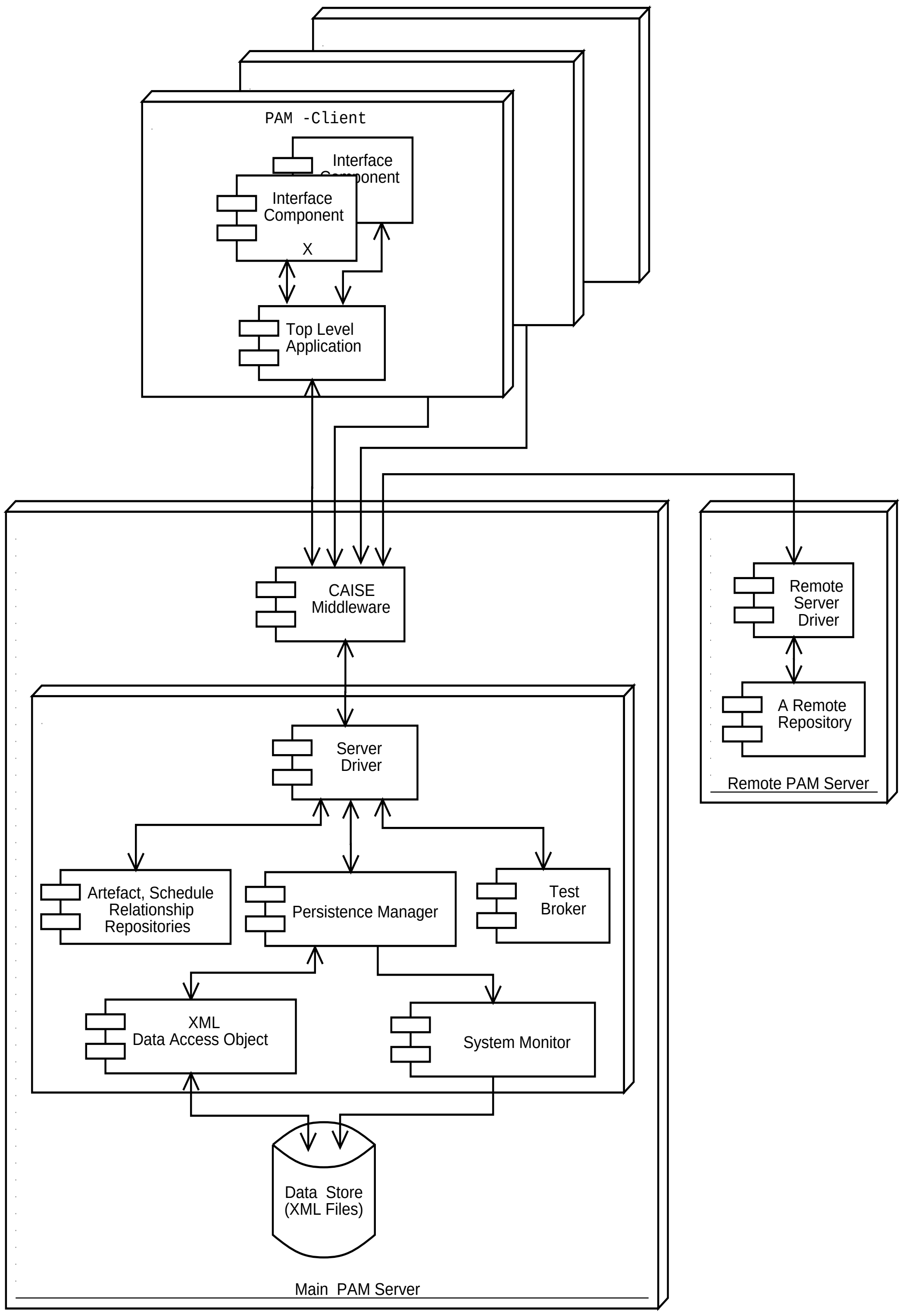


Figure 6.2: PAM System Architecture

scalability issue (see Chapter 9).

6.2.1 The Server Driver

The Server Driver is the interface between the PAM clients and the components of the PAM server. Messages received by the Server Driver are forwarded to the relevant server component for processing (see Subsection 7.3.2). Figure 6.2 illustrates the Server Driver and its associations in the PAM environment.

6.2.2 Runtime Repositories

There are three broad categories of repositories in the server.

1. Schedule Repositories: these maintain collections for Projects, Releases and Iterations

2. Artefact Repositories: these maintain collections for User Stories, Tasks, Acceptance Tests, Unit Tests and Implementation artefacts

3. Relationships: this repository maintains collections of parent-child associations between XP concerns for each project.

A repository is a collection of sets of a specific type of project concern, such as User Stories and Tasks, for a number of projects. In each repository the sets of project concerns are mapped each to the project to which it belongs. Repositories are responsible for storing the results of Common Operations (page 75) performed on clients, and broadcasting the resulting project state change(s) to other clients. They also use the Persistence Manager to handle data storage on their behalf.

Relationships between XP concerns are maintained as collections of Relationship objects. Relationship objects encapsulate specific details about the parent and child in each parent/child association of the Project Hierarchy. For example, a Relationship object has been created for the association between Project COSC314 and Release REL_001 in our examples in Chapter 5. The collection of relationship objects for a project serves as the data model for the Project Hierarchy component in the PAM client.

The 'Project Repository' serves a dual purpose. First, it provides the services of a regular repository for Projects handled by the server. In addition, it handles user authentication, a need which arises from the association between users and the projects to which they have rights.

6.2.3 Persistence Manager

The Persistence Manager provides an interface to the Data Store. It uses two components to carry out the actual storage and retrieval of data. These are: (1) the System Monitor and (2) an XML Data Access Object (DAO). The System Monitor manages data pertaining to the configuration management, instrumentation and the incremental changes to the project state, while the DAO manages data pertaining to the *Project Document*. The Persistence Manager insulates the rest of the server system from the raw representation of the data.

System Monitor

PAM's server-side System Monitor component is responsible for maintaining records of activities on the server. These records are stored as log files: (1) Transaction Log, (2) Debugging and Performance Log, and (3) Usage Log. The Transaction Log maintains records of activities which affect the state of the project, while the Usage Log contains information about when each user joins or leaves a PAM session and so on. These logs are maintained as separate files on the server.

The log files are stored in XML format. Data contained in the Transaction Log is useful for decision making regarding resource allocation, planning and so on. For example, Transaction Log entries show what User Stories are created, split, edited and so on. In this manner, the Transaction Log supports configuration management. They provide audit trails of incremental changes to the project state. These logs are valuable for long term studies of PAM and distributed XP.

The Debugging & Performance Log file stores instrumentation data. Instrumentation deals with performance issues in computerised systems. The Usage Log is relatively simple and is primarily intended to provide human resource management information. On the other hand, it provides a means of analysing the way XP is used in practice. During development, these log files provided information that was used to locate problems and guide the incremental prototyping of PAM.

For example, during informal evaluation we used these logs to localise a recurring runtime error. One participant's activities severely stressed a particular user interface component. This caused the system to crash, thus interrupting the evaluation session. Prior to implementing the System Monitor, localisation of this problem proved to be elusive. However, when the logs became available, an analysis of the Performance and Usage logs revealed the problem.

System Monitoring offers many important benefits to users of PAM. It protects users from the details of background processing, which would otherwise be done manually. For example, it records how User Stories change over time. It also helps the developer to keep track of how the system operates in the deployment environment. This is needed since some scenarios in the deployment environment, such as failure in the network and hardware or unpredictable network behaviour, are not easily modelled in the development environment. This may result in some aspects of the system escaping adequate testing prior to deployment.

Implementation of the System Monitor

We considered two alternative means of implementing the system monitoring functionality: (1) the use of external utilities provided by the Operating System, such as syslog on Unix OS, and (2) the implemention of the logging functionality as part of the server sub-system. Option (1) imposes deployment constraints, which has potential to affect PAM's platform independence. Therefore, we implement the monitoring functionality as part of the PAM server.

PAM is implemented in Java version 1.4.1. The *java.util.logging* package provides an API that supports flexible logging of events in an application. We use this framework to implement the System Monitor. In Figure 6.3 we illustrate the Java Logging Framework.

In the Java Logging Framework a *Message* originates in an application's component, for example a PAM repository, and is passed to the *Logger* object. The *Logger* creates a *LogRecord*, fills in the details of the Message and passes it to the *Handler*. The *Handler* uses a formatter to format the LogRecord—in PAM, as an XML file. The formatted LogRecord is then passed to an output stream such as a socket or bytestream. The *Filter* filters LogRecords according to their level of severity.

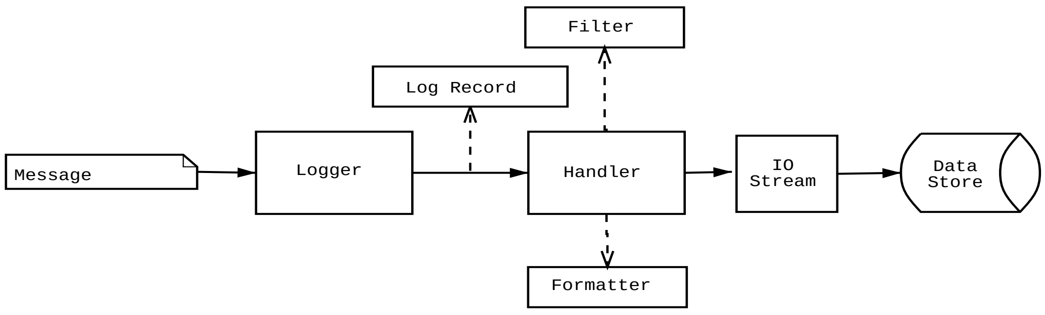


Figure 6.3: Java Logging Framework used in PAM

The log files are stored in XML format. The entries in the log file are stored as *Record* elements. Each *Record* has the elements level, sequence, date, logger and message contained in a *Log* . We include below an example of an log Record. The self-documenting, tag based format of XML makes the logs' data amenable for use by external programs and also for direct examination by humans.

```
<record>
<level>INFO</level>
<sequence>39</sequence>
<date>Wed Nov 26 09:23:01 NZDT 2003</date>
<logger>transactions_server.datastore_monitor.SystemMonitor</logger>
<message>
  <project>COSC314</project>
  <parentArtefact>COSC314</parentArtefact>
  <instruction>UPDATE_MODE</instruction>
  <artefactAffected>US_004</artefactAffected>
  <sender>cosc359.cosc.canterbury.ac.nz_1069791725654</sender>
  <receiver>SERVICE_ID_USR</receiver>
  <userName>john</userName>
<message>
</record>
```

We reuse the Java Logging Framework terminology in the tags of the log files. *Level* describes the severity of the message being logged. Severity is denoted by a description such as INFO, SEVERE or WARNING. The *Sequence* denotes the order in which the log entry is processed, beginning with a zero count. The *Date* element captures the date and time of the logged event. The *Logger* element describes the system component in which the log entry originated. The *Message* element—the data that is to be stored—is a description of the event/activity that is being logged.

XML Data Access Object

PAM is independent of the specific features of the Data Store. This is achieved with the use of the DAO design pattern (Sun Microsystems, Inc. 2003). The DAO adapts the Data Stores API—in our case, a Document Object Model (DOM)—to suit the way in which data is handled in PAM (Section 6.2.2). It also allows the Data Store to be changed without having to rewrite data handling code within PAM. For example, if an XP project team uses RDBMS, then they can replace/extend the current DAO to handle the specific RDBMS API. This enhances the overall flexibility of PAM.

In the current version of PAM we use an XML DAO. The XML DAO handles storage and retrieval of the project's data stored in XML files on the server. The XML DAO uses a validated DOM. The DAO retrieves the project data from the XML files and stores it in the DOM. The DOM by virtue of its hierarchical structure, maintains a tree of projects. Each project is stored as a hierarchical arrangement of the projects concerns which is reflected in the *Project Document*. Requests to store or retrieve data are made via the Persistence Manager.

6.2.4 The Data Store

The project data is stored as regular text files in XML format. We chose XML files because: (1) XML offers an amenable format to reflect the natural hierarchy of XP project concerns, (2) plain text files lend themselves easily to the lightweight nature of PAM, and (3) XML can be easily read and interpreted by both humans and external programs. These characteristics make it easy for PAM to support reuse and sharing of the System Monitor's log and project data with other XP tools and applications. The XML files are stored within the PAM directory hierarchy.

6.2.5 Test Broker

In addition to local unit testing (Subsection 5.2.8, page 85), PAM also supports unit testing on the server. We use the Apache Ant (The Apache Software Foundation 2003) build system for both local and server side execution of builds and tests, which is common on “real” XP projects. Nevertheless, other build systems may be used in the place of Ant such as Make. The *Build/Test Broker* (see Figure 6.2) component of the server is the interface between the PAM system and Ant. The Build/Test Broker sends feedback about the build and tests to the client that initiated the request.

Test requests are sent to the server as messages. The message data contains details of the source code to be tested and the unit test case or suites to use. (This information is available in the client interface; see Subsection 5.2.8, page 85). The Build/Test Broker uses this data to create an Ant build file—*build.xml*—and places it a build directory on the server. It then loads Ant, which reads the *build.xml* file and executes the instructions.

As the build and tests are executed, the feedback is sent to the client. Upon completion of the build and test, the build directory is cleaned up and the *Test Broker* reset to accept another test request. In the event that more than one use attempts to use the Build/Test Broker, the second user is advised that a build is currently being executed.

6.3 PAM Client Sub-system

The client sub-system consists of three main components: (1) the Top-Level Application, (2) views for handling XP project concerns and (3) a communication utility (Section 5.1, page 62). The use of the client interface and features is discussed in the preceding chapter and the principles used in their implementation are discussed in Chapter 8.

The Top-Level Application component serves a dual purpose. First, it is the interface between the CAISE middleware (Chapter 7) and the client interface components. In this way, it separates a client from the rest of the PAM system. Secondly it provides a graphical user interface container for the interface components.

Messages received from the CAISE middleware are stored by the Top-Level

Application in a queue. We discuss the message processing mechanism in Section 7.3.2. The Top-Level Application also has a System Monitor component whose primary purpose is to log Debugging and Performance activities. The method of storage is the same as the Server System Monitor. The use of these logs are mainly for development and maintenance purposes.

6.4 Summary

In this chapter we discussed the details of PAM's server and client sub-systems. We described the functions and relationships between the major components of each sub-system. The sub-systems collaborate to achieve the goals of the groupware, that is, to alleviate the long term consequences of the Planning Game, in addition to supporting dispersed XP teams. We discussed the sub-systems independently because they are decoupled. The transport mechanism, called CAISE, supports communication between them adn is discussed in the following chapter.

Distributed systems help to solve some problems, such as in our particular case, relaxing the co-location requirement of XP. However, developing distributed systems introduces problems. These problems are often related to the messaging/transport system that holds the distributed components together. An example of such a problem may occur when a client sends a request to the server to reserve some resource, and for any number of reasons the client is dropped from the system without receiving a response. Messages sent between client and server can be inexplicably lost. Components within the system can hang. These are some of the things that can happen to disrupt a distributed system.

In the following chapter, we discuss the connectivity middleware which supports PAM. The middleware is based on message passing. We show how this mechanism supports the communication needs of the PAM environment. We also discuss why we use a message passing transport, as opposed to remote objects.

Chapter VII

The Connectivity Middleware

In this chapter we discuss the connectivity middleware that provides the communications service between PAM sub-systems. Its role is to allow PAM's sub-systems to exchange data while hiding from them the complexity of the underlying distributed environment—that is, the operating system and network technologies.

We have identified two fundamental types of communication requirement for the PAM system. These are broadcast and point-to-point communication. In the first case, the middleware must support broadcasting of non-critical transient data such as that generated by the Chat utility. In the second case, it is required to support communication between the clients and the server for critical data management operations such as when a new collaboration object is being added to the system and must be stored in the repository (see Section 5.2.3, page 75). To meet these needs we use a message passing middleware called CAISE (Cook 2003). In the following sections we show how CAISE supports the communication needs of PAM.

7.1 Requirements of the Middleware

We executed spikes to assess which available middleware offers the best potential for supporting the communications needs of PAM (see Section 7.2). The criteria listed below were used in the assessment.

1. **It must provide a robust and reliable transport system:** PAM supports several projects, each with multiple users participating in collaborative sessions. Certain operations in the PAM client interface, such as moving User Story cards, generate many messages. The communication system is required to handle these situations while maintaining adequate performance

and reliability. We anticipate that each project supported by PAM will have an average team size of eighteen members.

2. **It must maintain knowledge of all online PAM sub-systems:** Direct point-to-point and broadcast communication are required to support the multi-faceted needs of PAM. An example of point-to-point communication is the transfer of user authentication data from the PAM client to the server. Broadcast communication takes place when the sender does not need to know to which client and/or server a message is being sent. This includes chat messages and notification of updates from the server to all clients.

3. **It must decouple PAM clients from PAM server(s):** The server and client sub-systems in PAM are logically and physically separate. PAM clients and servers, excluding the main server, do not need to know about one another. All they need to become a part of the PAM environment is the URL of the main server. This is necessary for flexibility and scalability. It must be possible for additional server services to be added to the PAM system or for the secondary PAM servers to be moved to different physical systems with little or no effect on the clients. For example, the Build/Test Broker component (Section 6.2.5, page 96) may be separated from the main PAM server, implemented as an additional server and deployed on a physically separate system without PAM clients being affected (see Figure 6.2).

4. **It must be lightweight:** A major aim of PAM is for the system to be lightweight. That is, it must fit into the context of a normal XP environment with minimum negative effects. Consequently, it is essential that third-party systems supporting PAM must facilitate this aim. The communication system must, therefore, not impose a need for special configuration or require proprietary component support in the depolyment environment.

In Table 7.1 we provide a summary rating of the criteria used to determine the suitability of some available middleware. From the alternatives we choose CAISE because it satisfies the main criteria—that is, lightweight, simplicity and the flexibility of the communication mechanism. Though that table shows CAISE

Criteria	CORBA	Java RMI	RPC	CAISE
Maturity	***	**	***	*
Robustness & Reliability	***	**	**	**
Simple API	**	**	*	***
Lightweight	*	*	**	***
Decouples client from server	**	**	*	***
Flexibility of communication	**	**	*	***

Table 7.1: Comparison of some available middleware.

has low rating for robustness and reliability we expect that these will be enhanced as the middleware matures.

The greatest demand (from the perspective of PAM's needs) on the connectivity middleware is based on the need for a robust and reliable transport system. This is especially important to PAM. PAM implements metaphors—for example, the Virtual Desktop—that are used to emulate real world practices and activities of normal XP (see Chapter 5). Consequently, the middleware is required to support tasks such as moving a widget (for example, a User Story card) in one client application space and have the movement reflected in other client application spaces in real time. Such tasks generate streams of messages, which are dispatched to all clients. In order to avoid "hot spots" and jerky processing (especially for GUI rendering) dissemination and processing of messages must be quick. Less demanding on the middleware's robustness and reliability is the need to support broadcast text chat messages, as well as awareness of the transit state of User Stories (see Section 5.2.4, page 80).

The middleware is also required to handle the transfer of both large and small data collections. For example, a large data collection transfer takes place when a project's state is being replicated on a client from the server. On the other hand, a small transfer occurs when a chat message is being broadcasted.

7.2 Why use a message-passing middleware?

Different types of middleware are available. They differ in terms of their Application Programming Interface (API), their functionality and the way in which they work. The main types of middleware are Remote Procedure Call (RPC), Object Request Brokers (ORB) and Message-Oriented Middleware (MOM). RPC works by having a client invoke procedures of the server application. The client is tightly coupled to the server because the code to call server procedures are complied in the client. With ORB, on the other hand, clients invoke methods on server objects. These popular middleware are parts of standards, such as CORBA, and are meant to guide the development of services and interoperable software. These middleware are popular. We conducted spikes to examine the suitability of some of them—CORBA, RPC and Java RMI—for the communication needs of PAM, and found they do not adequately satisfy all criteria listed in Section 7.1. For example, CORBA and RPC are not lightweight, and do not decouple the server and clients respectively.

On the other hand, MOMs decouple clients from the server. They simply acts as an intermediate and passes encapsulated messages to and on behalf of clients and servers. For example, when a User Story is created, it is encapsulated in a message along with an `insert` instruction code and sent to the server. Messages include data such as sender, recipient, payload and type or instruction code (see Figure 7.2 for description of PAM message format). The recipient interprets the message and performs the appropriate action(s). A messaging system also aids in scalability; adding clients and servers does not incur changes to the messaging system. It also aids in making the overall architecture robust; failure in one component does not affect unrelated components.

Message-passing is the technique used in MOMs. It is simpler than the techniques used in other middleware. The API for a message passing systems is often easy to use and less complex than their remote procedure and objects counterparts.

7.3 The CAISE message-passing middleware

We use the CAISE (Cook & Churcher 2003*b*, Cook & Churcher 2003*a*) message-passing middleware for the communication system in PAM. Figure 7.1 illustrates

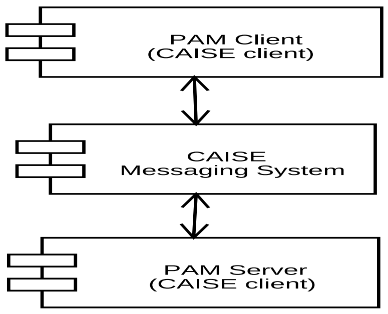


Figure 7.1: CAISE-DXP Relationship

the relationship between the CAISE messaging middleware and PAM applications—that is, the PAM client and server in the figure. CAISE also supports scalability; adding clients and servers to the PAM environment does not necessitate changes to the messaging system. Finally, it makes the overall architecture of PAM robust. Failure of a client or additional server service does not affect unrelated consumers of the specific service. CAISE also provides a simple API, which is easy to use.

Communication in the PAM environment is divided into point-to-point (client/server) and broadcast communication. Point-to-point communication occurs, for example, when a new project concern, such as a Task, is created and added to the Task repository on the server. In such a communication, the client prepares a message and sends it to the middleware with instructions to save it in the corresponding repository. The middleware forwards the message to the server, which makes arrangement for it to be saved. Broadcast communication, on the other hand, occurs when, for example, a User Story is placed in transit in a Release or Iteration, or when the server sends notification of a project state changing operation, such as an insert, to all remote client applications. In the first case, the original client application makes a request for the middleware to broadcast the new User Story state, along with data about the schedule to which it is being allocated.

With respect to operation of PAM, the important CAISE functionalities are:

1. **Registration of clients.** CAISE maintains a `Registry` of listeners. It treats each application, called a "CAISE client", as a listener (see Figure 7.1), regardless of the role the specific application plays in the distributed system. Each CAISE client has a Registry object which it uses to perform opera-

tions offered by the CAISE middleware API such as joining `Channels` (see Item 2, “Creation of distinct communication groups”) and so on.

In the PAM environment, the server and client sub-systems are CAISE clients. The sub-systems use the CAISE API (see Item 4, “Flexible message passing”) to pass messages by broadcast and point-to-point.

2. **Creation of distinct communication groups.** CAISE has two communication groups: (1) `Meetings` and (2) `Channels` (see Figure 7.3 and Subsection 7.3.1). A `Meeting` is a container concept. Channels, on the other hand, are the actual communication pathways via which messages are passed. A Meeting may contain any number of Channels. CAISE supports several Meetings. CAISE clients may join and leave Meetings and Channels as needed, and have their associations updated in the Registry accordingly.

   PAM handles many projects. The first PAM client that requests to work on a specific project creates a `Meeting` named after the project. Otherwise, it would simply join the `Meeting`. Within the project `Meeting`, two `Channels` are initially created: one for use by Chat utility, named `CONFERENCE`, and the other for communication with the server and other clients. The latter is named after the project (COSC314, from our running example). Broadcast and point-to-point communication are differentiated by means of the API method called (see Item 4, “Flexible message passing”). Several `Channels` may be created to support the use of the Chat utility by separate groups. Messages arriving on the `CONFERENCE` channel are displayed in the Chat utility irrespective of the channel to which the Chat utility is currently connected.

3. **Encapsulation of message objects.** Messages are serialised and enclosed in `Envelopes`. The sending sub-system passes the `Envelope` to the CAISE middleware, which forwards it point-to-point to the PAM main server or as a broadcast message to clients on a specific channel (see Item 4, “Flexible message passing”).

   Each `Envelope` contains information about the `Sender`, `Receiver`, `Channel`, and the data—a PAM defined message. The data in the Envelope may be any serialisable object. In this regard, we use an `XP_Data` object to encapsulate PAM messages (see Figure 7.2 for class representation). This message encapsulates the collaborative objects, instructions, project, user and other essential data elements that are used in communication between PAM sub-systems. The `XP_Data` is serialised and placed in an `Envelope` before being passed to the CAISE middleware.

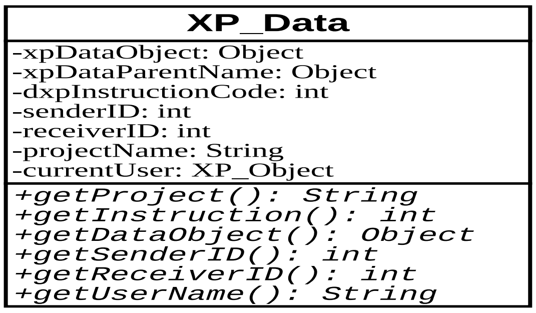


Figure 7.2: XP_Data Class

The `Envelope` is collected by the Top-Level Application or Server Driver (Subsections 6.2.1 and 6.3), where applicable, and extracted from the MessageQueue by the `MessageProcessor` (see Section 7.3.2). The recipient information is examined and the Envelope is then passed to the appropriate component for processing.

4. **Flexible message passing.** CAISE defines three methods in its API to effect its message passing. These are: (1) `send`, (2) `sendToAll` and (3) `sendToOthers`. The `send` method requires the registered name of the recipient CAISE client. This method is used to send an Envelope point-to-point. The main PAM server's name is known by the client sub-system. When a PAM client sends an Envelope to the PAM server, the name of the client is retrieved from the CAISE envelope. The PAM server uses this name to communicate with the specific client. Otherwise it uses the CAISE API, `sendToAll` and `sentToOthers`, to broadcast a single message to all clients (for example, Chat messages) including the sender, and all other clients except the sender respectively.

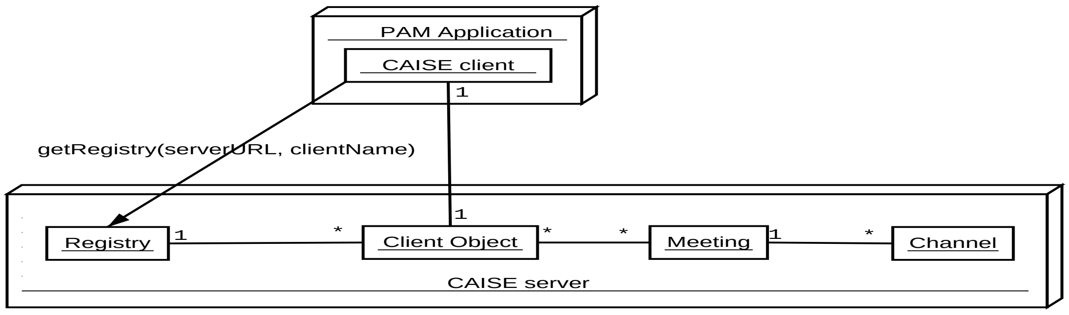


Figure 7.3: CAISE Architecture

7.3.1 Registering as a CAISE Client

Figure 7.3 illustrates the relationship between the internal CAISE concepts and a CAISE client application. The server consists of the CAISE middleware and the components of the PAM server sub-system. At startup the CAISE middleware is loaded by the Server Driver. The Server Driver registers itself with CAISE and proceeds to load the rest of the server components. It then waits for PAM clients to join and/or for messages to broker.

PAM clients—and the PAM main server when it starts up—connect to CAISE by first requesting a reference to the `Registry` with a `getRegistry()` call. This call passes as parameters, the URL of the system on which CAISE is running, and a unique name by which each PAM client will be identified in the `Registry`. The PAM server uses the name `DXP_SERVER`, while each PAM client creates a unique name from the local host URL and system time.

On the server, CAISE creates a `Client` object for each CAISE client and maps it to the unique name supplied. Each CAISE client requests and maintains a one-to-one association to their respective `Client` object. CAISE clients create and/or joins `Meetings` and `Channels` in the CAISE middleware with appropriate calls to the `Registry` object. Each CAISE client is responsible for maintaining a queue for messages—Envelopes—sent to it by the CAISE middleware. We discuss in the following section how these messages are processed.

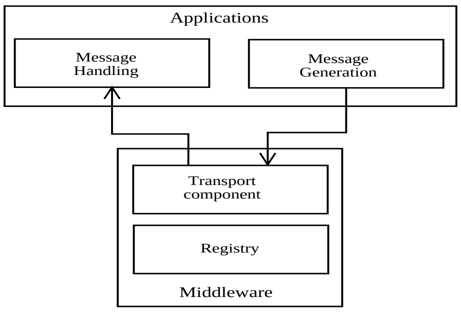
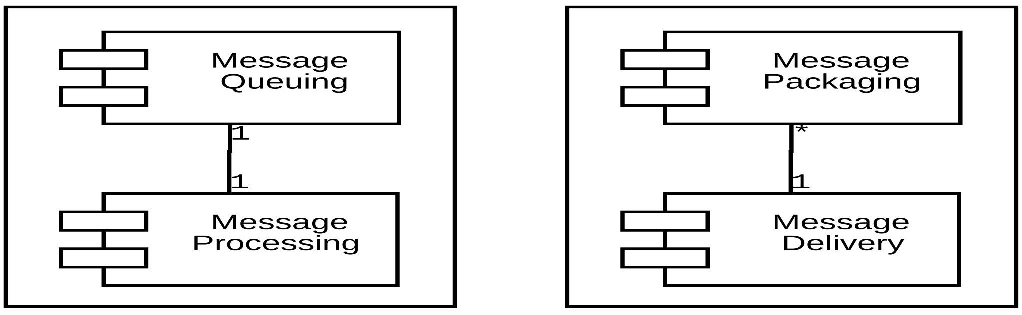


Figure 7.4: Illustration of PAM message handling architecture.



(a) Message Handling Component (b) Message Generation System

Figure 7.5: PAM Application Communication Components.

7.3.2 PAM Message Processing System

To complete the communications system, CAISE client applications implement a message handling system to handle the pooling and processing of messages receive from the middleware. This is illustrated in Figure 7.4.

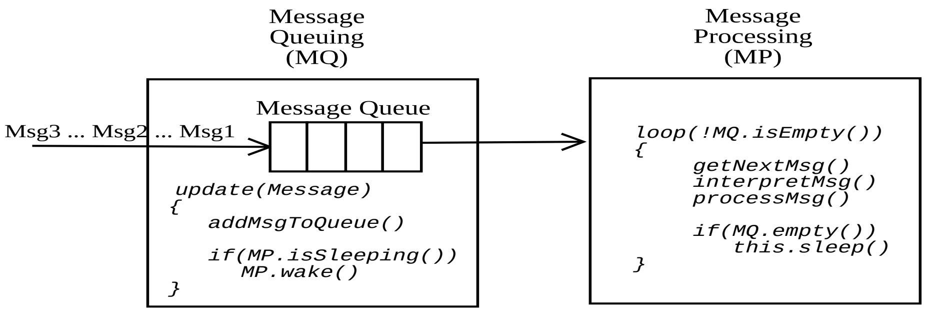
The PAM message handling system (for clients and server) consists of two primary components: (1) a message queuing component and (2) a message processing component (see Figure 7.5(a)). The message queue is updated with messages sent from the middleware. The message processing component extracts a message from the message queue, interprets the message and passes it to the relevant components to carry out the operation(s)/service(s) requested.

We consider three main threads executing in the sub-systems. These threads are: (1) GUIThread—the default thread for the Java GUI, (2) MessageQueuing—the main application thread that observes the middleware, and (3) MessageProcessing—the thread that retrieves messages from the queue maintained by the Message-

Queuing thread. When a PAM sub-system—server or client—is loaded into memory, threads MessageQueuing and MessageProcessing are created. MessageQueuing waits on events from the middleware and MessageProcessing monitors the MessageQueuing for arriving message(s). On receipt of a message, MessageQueuing inserts the message into the back of the message queue. The MessageProcessing thread extracts the message from the front of the queue, identifies which component it is meant for and passes it along accordingly. The MessageProcessing thread pools the queue intermittently. If there is a message in the queue it is retrieved and processed; otherwise the thread sleeps. Figure 7.6 illustrates the the pseudo-algorithm of the implementation of the messaging system.

It is possible for the `MessageProcessing` component to use a message prioritising scheme. For example, whereas messages that affect the project state can be handled sequentially, transient messages, such as messages advising of the new position of a User Story card, can be given high priority and processed quickly, or even redirected to another thread for immediate processing.

PAM clients request services and respond to service requests by sending messages to the CAISE middleware. This is a relatively simple process. Figure 7.5(b) illustrates the two primary components involved. The component which has to send a message encapsulates the data in an XP_Data object, adds it to an Envelope and uses the `Client` object to pass it to the middleware. The CAISE API method used determines if the message is sent point-to-point or by broadcast. For example, the Chat utility encapsulates `String` objects in XP_Data objects and uses the `send` to have CAISE broadcast the message. Outgoing messages may originate from different parts of the PAM client system and occur arbitrarily in time.



```
update(Message)
{
    addMsgToQueue()

    if(MP.isSleeping())
        MP.wake()
}
```

```
loop(!MQ.isEmpty())
{
    getNextMsg()
    interpretMsg()
    processMsg()

    if(MQ.empty())
        this.sleep()
}
```

Figure 7.6: Pseudo Message Processing Routine

Therefore, we separate the sending of messages from the processing of incoming messages.

7.4 Summary

In this chapter, we noted that there are two primary forms of communication required to support collaboration between the PAM sub-systems. In order to fulfill these requirements we used a message passing middleware. Our choice of CAISE was based on the results of a spike which was carried out in order to assess the suitability of available transport technology to satisfy specific criteria based on the objectives of the PAM environment.

In the next chapter we discuss the PAM client user interface. We examine the principles used in the model and show what metaphors and characteristics of XP were used to guide the design.

Chapter VIII

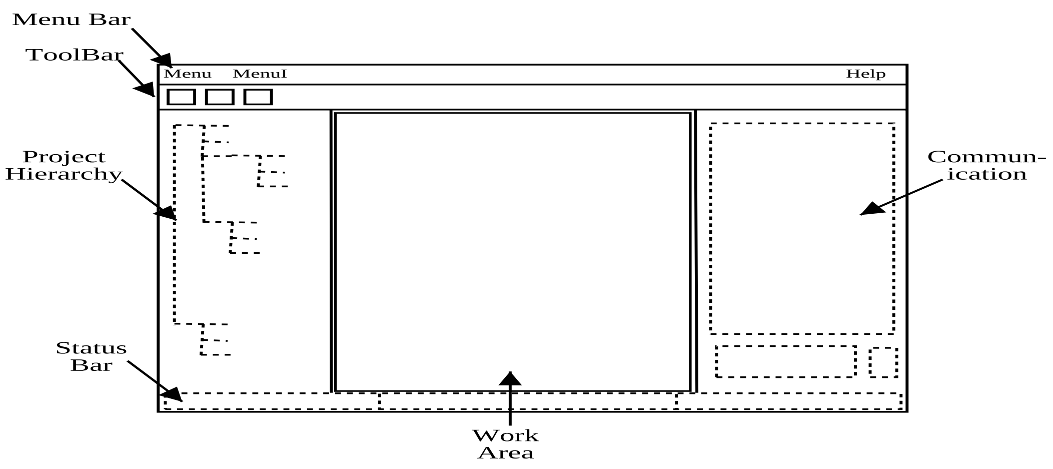
PAM Client User Interface

In this chapter we discuss the underlying principles of the User Interface (UI) of the PAM client sub-system. As stated in Chapter 4 The UI supports manipulation of project concerns, as well as collaboration and coordination of XP planning activities—the Planning Game.

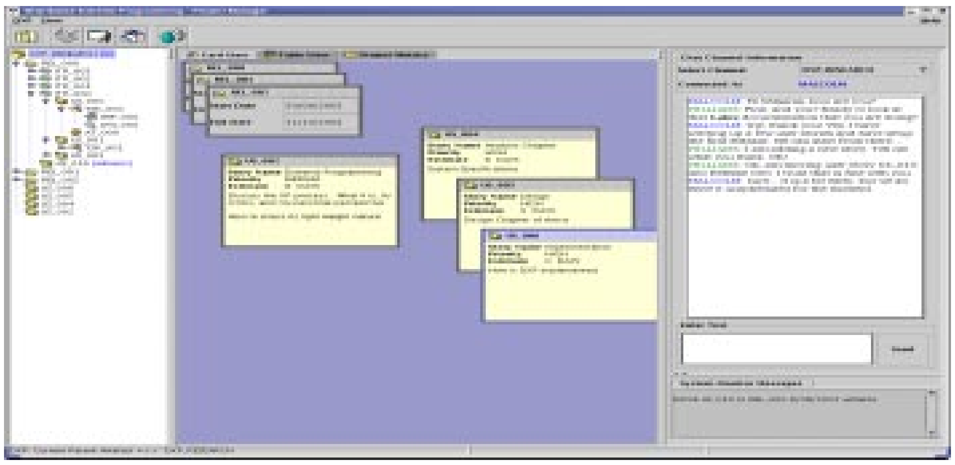
User interfaces are an important part of modern computer systems. They play a significant role in the usability of these systems (Telles 1990, Nielsen 1993). Usability is concerned with the issues of learnability, ease of use, memorability, errors and user satisfaction, with respect to computer systems. In essence, usability deals with the efficiency of supporting users in performing tasks and accomplishing goals. It is important because it encompasses user interaction with the system, as well as the impact the system can have on users. Our design of the client interface is influenced by these needs. In the long term user trials will guide the evolution of the client user interface. At this moment, providing appropriate functionality as proof of concept is our major goal.

The actions of any user of PAM impacts on other users. An action can potentially change the state of the project or the session. Consequently, an awareness mechanism (Subsection 5.2.3 and Section 8.6) forms an essential and important part of PAM. Awareness plays a significant role in strengthening the overall usability of PAM. In this regard, it plays a significant role in collaborative usability since it helps users to coordinate their work while maintaining awareness of what other participants are doing in the system.

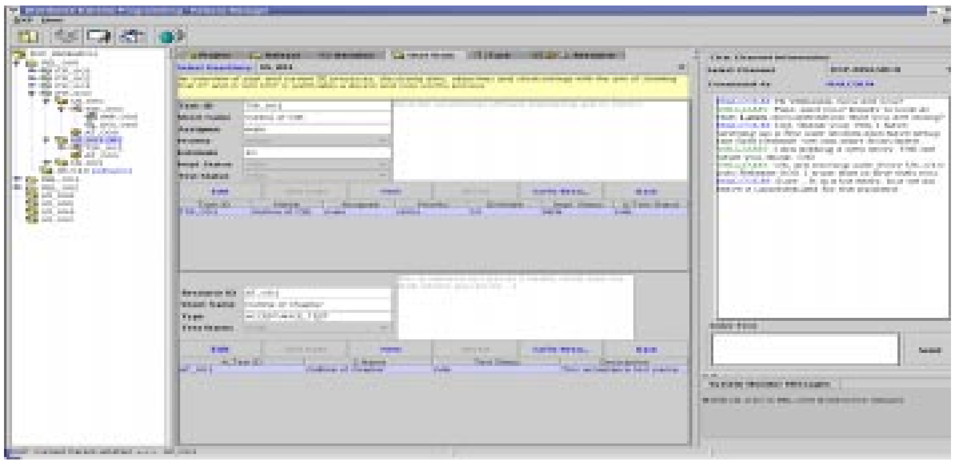
The architecture of the PAM client UI is based on a philosophical approach to the analysis of XP activities (Chapter 4), the Clover Design Model (Section 8.1) and the Model-View-Controller (MVC) design pattern (Gamma, Helm, Johnson & Vlissides 1994). The interface architecture supports multiple orthogonal perspec-



(a) DXP User Interface Model



(b) Implementation snapshot showing 'Desktop' Metaphor



(c) Implementation snapshot showing 'Stacked Slates' Metaphor

Figure 8.1: Illustrations of the UI Model and Screen shots after implementation.

tives and means of performing tasks and interacting with the *Project Document*. Figures 8.3 and 8.6 (pages 114 and 117) are the fundamental models of the perspectives which were obtained from our analysis of XP. We exploit the natural hierarchical arrangement of the XP project concerns, and the simple, incremental and iterative nature of the process, in the design of the UI.

We also take advantage of the similarity of activities involved in transforming project concerns. In Chapter 5, we described these as Common Operations and highlighted the activities that are common to both schedules and artefacts. The common operations are exploited in the UI in order to maintain consistency in performing tasks.

We use Java Swing widgets to implement views and interaction metaphors, such as Virtual Desktop and Card metaphors, that are consistent with metaphors used in normal XP. In combination with the connectivity middleware (Chapter 7) and the PAM server, these views support the groupware activities of communication, production and coordination recommended by Calvary, Coutaz & Nigay (1997). (We discuss the Clover Model in Section 8.1.) User interface support for these activities are illustrated and discussed in Chapter 5. In Section 8.5, we discuss the principles that work together to emphasise the groupware activities in the GUI. Figure 8.1(a) (repeated here for convenience) illustrates the UI framework, which supports communication, a work area for production and a tree display to facilitate some aspects of the overall coordination.

Notwithstanding the emphasis of people over the process, XP projects revolve around the management: (a) of schedules and (b) of artefacts (see Chapter 4). Schedule management deals with the handling of issues pertinent to the XP schedule concepts: projects, releases and iterations. Artefact management deals with the handling of tangible deliverables of the XP process such as User Stories and Tasks, and resources produced and used by Unit Tests, Acceptance Tests and Implementation operations. We use the concept of a *Resource* to serve as named collections of pointers to the external resources that are stored in a file system. For example, a Unit Test Suite is a collection of relative path names of Unit Test source code stored in the project repository. Figure 4.2, page 52, illustrates the *Implementation*, *Unit Test Suite* and *Acceptance Test* resources as leaves in the hierarchy.

The evolution of the project from inception to delivery of part or whole of the functional software—or project failure—is the history of the coordination of schedules, artefacts and resources. In the following sections, we build on Chapter 4 to describe how collaboration between users and the evolution of schedules and artefacts are supported in PAM. First we discuss the Clover Design Model and the approach used in developing the UI model.

8.1 The Clover Design Model

Collaborative activities have been classified, based on the Clover Design Model, into three categories: communication, production and coordination (Calvary et al. 1997). Different components of a groupware system, such as a multiuser toolbar, may support all of these activities (Sire et al. 1999). In this model, communication handles information exchange. Coordination handles resource allocation and maintenance of dependencies between tasks. Production is concerned with all the actions that go into creating and maintaining a collaborative object. These activities map directly into features that PAM needs to support: communication, project coordination, and creation and manipulation of shared XP concerns (Chapter 4).

The aspects of XP which we need to address, such as User Story elicitation (Chapter 4), are best supported with a model that emphasises all of the preceding groupware activities. We note that a major goal of PAM is to support project and collaborative activities. We therefore, attempted to strengthen the coordination component of PAM by means of the awareness mechanism.

8.2 Information Sharing Strategy

We use data replication to support information sharing in the PAM groupware application (see Figure 8.2 for illustration of architecture). An alternative strategy is to use shared application/desktop data sharing technique used by groupware such as the open source Virtual Network Computing (VCN)(Richardson, Stafford-Fraser, Wood & Hooper 1998) developed by AT&T Laboratories Cambridge. Other popular shared desktop applications are Microsoft NetMeeting. The application sharing technique uses a low replication architecture and usually imposes strict floor and session control protocols in order to support collaboration.

This implies strict-WYSIWIS in the groupware, which makes application sharing inappropriate for PAM (see Section 8.5.2).

Information sharing is needed in PAM (as with other distributed collaborative applications) for: (1) shared project and session state, (2) consistency of data in the PAM system and (3) client-side updates. The data replication technique used in PAM satisfies these criteria. Further, it supports the extension of PAM to provide offline processing and synchronisation (see Chapter 9).

8.3 Support for Relationships and XP Activities in the User Interface

We use the metaphor of a *Project Hierarchy* to describe the hierarchical arrangement of project concerns because it captures the essential relationships between them. While schedules exhibit a natural hierarchy only (Figure 8.3(a)), artefacts also exhibit network relationships (Figure 8.3(b)). In our implementation of PAM client interface, we exploit the hierarchical relationships. To achieve this, it is necessary at times to introduce redundancy.

When we model conventional XP the redundancy only occurs at the point where the project concerns are associated with actual source files and other real resources (see Figures 4.3). For example, Tasks are associated with source code files through unit testing and implementation operations. The effects of the redundancy are mitigated because the real resources and operations are outside the boundary of PAM. We separate the real resources from PAM by means of Resource abstractions. These abstractions are collections of pointers to the external resources (see Figure 8.4). Figure 4.2 illustrate the hierarchy as a tree arrange-

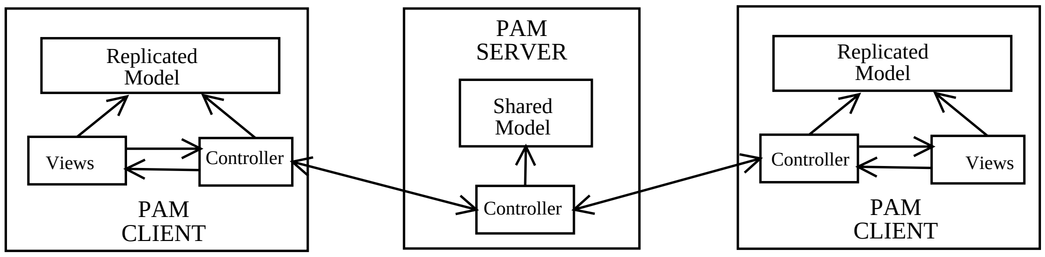
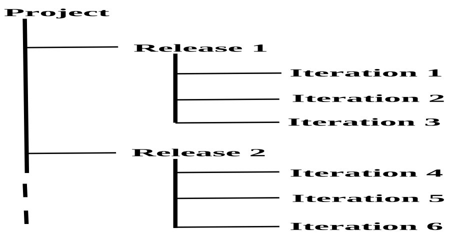
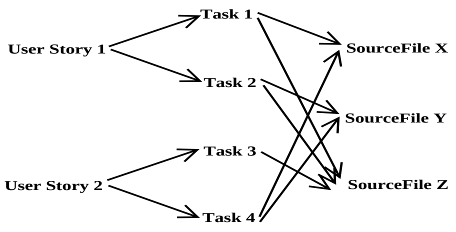


Figure 8.2: Illustration of PAM replication model highlighting MVC architecture.



(a) Hierarchical Relationship between Schedules



(b) Network Relationship between Artefacts

Figure 8.3: The mixed relationship models between schedules and artefacts

ment. The hierarchical arrangement of XP project concerns, lends itself easily to being adequately visualised with a JTree Swing widget.

Where do Schedules get their Meaning?

A schedule is an intangible construct. The important properties of schedules include: identification, type, start date and end date, apart from its children. Of course, extra data can be collected about this concept. These include, but are not limited to: client, development firm, status, assigned resources, parent schedule and children concerns. In PAM, management of the various schedules involves similar activities. These activities are described in Chapter 2 and the Activity Tables in Chapter 4.

The semantic of a schedule is dependent on its context. The context is defined by its relationships with other schedules and the artefacts allocated to it. For example, Figure A.7, depicts a 'Stacked Slate' view showing the context of Release REL_000. In this view it is evident that release REL_000 has two Iterations, ITR_000 and ITR_001, of which Iteration ITR_000 has two User Stories. In addition, the *System Performance* User Story is in transit in the REL_000, along with four other unallocated User Stories that can be allocated to either of the Iter-

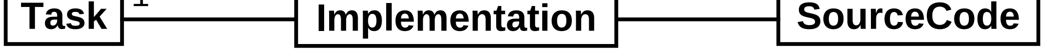
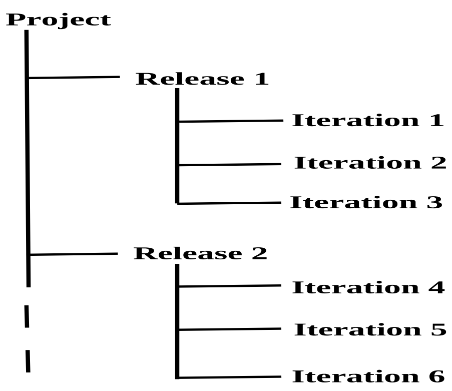
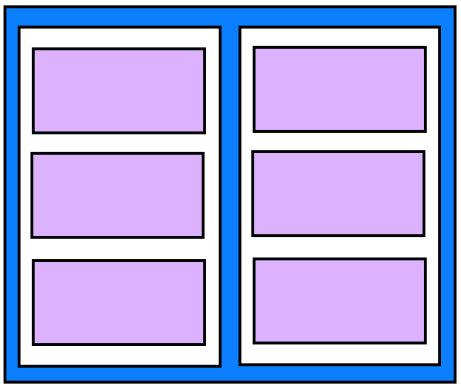


Figure 8.4: Task-Implementation-Source Code Relationship



(a) Graph Representation



(b) Tree Map Representation

Figure 8.5: Alternative views of a project's schedule hierarchy, with two releases and six iterations.

ations ITR_000 or ITR_001. When we compare this with the specific data about Release REL_000 (ID, Short Name, Status, Start and End Dates) shown at the top of Figure A.7, it can be seen that the context view provides much more useful information.

A tree representation provides the context and relationships in the Project Hierarchy. Figures 8.5(a) and 8.5(b) illustrate two possible ways in which the hierarchy can be represented. The figure depicts graph and tree map (exaggerated for effect) representations of a simple schedule hierarchy. We exploit these representations in the design of the user interface. For example, the Project Hierarchy is based on the the representation in Figure 8.5(a).

8.3.1 Artefact related and Common Activities of XP

Our discussion in Chapter 4 revealed that some XP activities are common to all project concerns. The schedules have similar activities, while the artefacts have a significant amount of activities in common, such as create, edit and delete. We described these as Common Operations in Chapter 5. Other activities carried out on projects are running tests, decomposing User Stories into Tasks and so on. These are specific to particular artefacts. The common activities provided us an opportunity to generalise the user interface. In this way we gain consistency in how project concerns are manipulated through the PAM client interface. For example, editing of project concerns is carried out in the same manner, regardless of the artefact under consideration.

In the following sections we look at the principles behind our model of the user interface. We then combine these principles with support for awareness and the activities to be supported to complete the user interface model.

8.4 DXP Client GUI: The Underlying Principles

The basis of the user interface is the extension of the metaphors behind the schemata in Figure 8.5. We extend the tree metaphor (the trunk, branches and leaves) to represent the hierarchy of the project concerns. The visualization of this metaphor (see Figure 5.2) provides a holistic view of the project. Specific details can be brought into focus by addressing a specific component of the tree. We extend the container metaphor of the tree map schema (Figure 8.5(b)) to represent a conceptual multi-layered perspective of the project. Each layer serves as a container for concerns of one or more adjacent layers. For example, Figure A.5 depicts the project layer in the Virtual Desktop containing project concerns from other layers such as Release, Iteration and Tasks.

Two independent perspectives can be distinguished in the tree map metaphor. These are illustrated in Figures 8.6(a) and 8.6(b). These figures expose the layers of the hierarchy—that is, project concerns such as Project, Release, Iteration and User Stories layers—in different ways. These perspectives are the basis for the Virtual Desktop and Stacked Slates metaphors discussed below. In XP, the real world equivalent of the project concern at these layers is discrete—for example, an index card on which a Task is described.

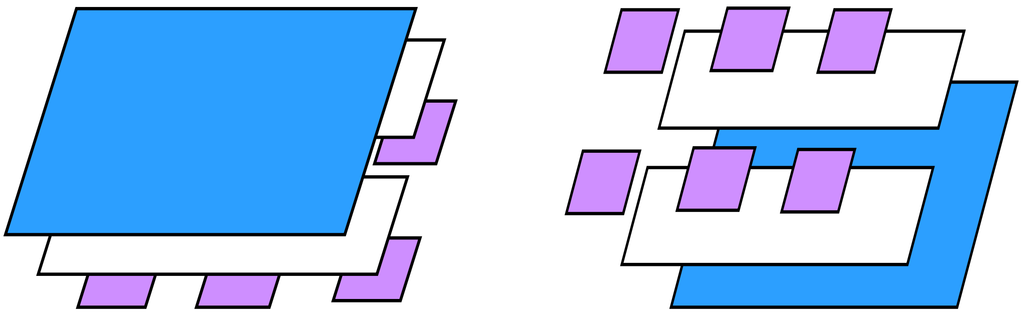
We use the schema in Figure 8.5(b) as the basis of three metaphors for viewing and manipulating aspects of the Project Document. The metaphors are: (a) *Drill Down* (b) *Stacked Slates* and (c) *Virtual Desktop*.

Figures 8.6(a) shows a top-down perspective of the tree map schema. The top layer can be used to represent the project, while each subsequent layer can be used to represent the Releases, Iterations, User Stories and so on. As shown, the number of concerns at each subsequent layer increases as the number of layers increase. For example, there are three levels depicted in Figure 8.6(a), with one, two and six concerns respectively. In addition, each element in the second layer contains three concerns of the third layer. This containership relationship is consistent with the arrangement of XP project concerns.

We note that the layers are naturally flat and two dimensional. We exploit this metaphor by defining forms to display the context of concerns at five layers: Project, Release, Iteration, User Story, Tasks and Resources. We implement these layers as depicted in Figure A.7 using named tabbed panes. We use the metaphor of *Drill Down* to describe the navigation from a specific concern to show a view of its children associations. For example, a *Drill Down* operation is performed when one moves from the project into one of its Releases. Figures A.6 and A.7 shows the before and after of the implemented *Drill Down* metaphor in use (a Drill Down operation in the Stacked Slates view, from the Project level to REL_000).

The *Stacked Slates* and the *Drill Down* metaphor are used for depth-wise navigation and construction of the *Project Document*. The top-down schema provides an appropriate model for this implementation. Each layer in this model is a potential form. The details on these forms may range from the details of a specific concern to details about the layers adjacent to that concern. For example, Figure A.7 shows the Project level form depicting the Releases in the project, which are from the next level. As each layer is visited, that layer is brought into focus and takes up the available work area. The collection of layers in this model represents a stack of forms.

Constructing the hierarchy depth-wise corresponds to the process of planning increasingly smaller schedules, beginning with the largest—the Project—and fin-



(a) Top Down View (b) Bottom Up View

Figure 8.6: Layers in the the project hierarchy; basis for drill-down and desktop metaphors.

ishing with Iterations. After one iteration is complete, the next is commenced. A Release is completed when all of the Iterations in it are complete. The next release will then be traversed depth-wise to its first iteration, and the preceding process repeated. These are examples of breath-wise navigation of the *Project Document*. The Drill Down metaphor thus maps to the real world practices of XP and its iterative nature.

We also use the metaphor of a *Virtual Desktop*. This too is an extension of the tree map representation and is illustrated by the schema in Figure 8.5(b). The underlying container metaphor maps to the real world XP practice of using the open surface of a desk to lay out and discuss project concerns particularly User Stories. The lowest layer depicted in the schema, maps directly to the Project, which then serves as the desktop—container—into which the Releases, Iterations, User Stories and so on are added. This metaphor supports direct manipulation of the cards representing project concerns on the *Virtual Desktop*. In this manner, it emulates the common XP Planning Game desktop. Further, it helps to reduce the gap between user's mental model of normal XP and CSCW supported XP.

The *Virtual Desktop* metaphor as implemented has a few shortcomings. Representing all concerns as *Cards* in the *Virtual Desktop* encourages one to develop the same cognitive model of schedules and artefacts. This is not consistent with reality. De-emphasising the schedules may result in a more accurate model. This can be achieved by using different representations for schedules, artefacts and resources. In our implementation, we use colour to highlight the difference. The *Drill Down* metaphor is used for navigation to expose the context of concerns displayed in the current view. We use properties forms (see Figure 5.6(c)) to facilitate viewing of the details of specific project concerns (see Subsection 5.2.3 for details).

The metaphors discussed here provide orthogonal perspectives of the project's arrangement and details. Each is suited for interaction with the project in specific ways. For example, the *Virtual Desktop* supports collaborative direct manipulation of *Cards* in the *Virtual Desktop*. This maps directly to the way the Planning Game activity is conducted in normal XP. The *Project Hierarchy* metaphor is used for direct—pseudo-random—access to specific concerns. The *Stacked Slates* metaphor is used to follow the work flow of XP, and may be used as a means of

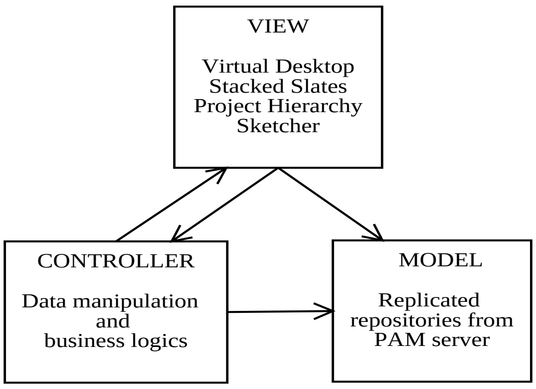


Figure 8.7: Model-View-Controller Design Pattern used in PAM

developing understanding of the process.

The various views discussed thus far are independent. Combinations of them offer tremendous potential to support navigation, viewing, editing (production) and awareness in the user interface. The way in which these combinations work together is the focus of the user interface architecture. This is discussed in the next section.

8.5 PAM Client GUI: The Architecture

The PAM client is based on the MVC design pattern. Figure 8.7 illustrates the separation of the model, view and controller components of the PAM client subsystem. Java Swing widgets make up the *Views*. In our implementation, the views are most often separate from the logic that manages the interactions of the client application and enforces XP rules and constraints in the system. The logic objects constitute the *Controller*. The *Model* is made up of the data that is retrieved from the server at startup and updated in real time for the duration of the session. The MVC design pattern effectively separates presentation from content. It thus enhances the extensibility and flexibility of the PAM client. For example, another, or novel, perspective of the *Project Document* can be added relatively easily to the PAM client using the existing model.

8.5.1 The Information Radiator Metaphor

In the preceding section we discussed the principles of the client application GUI. The spatial arrangement of the architecture is illustrated in Figure 8.1(a) (page 110). The *Project Hierarchy* metaphor is to the left of the screen. The *Work Area* serves as the base container to hold the *Virtual Desktop*, *Stacked Slates* metaphors and the Sketcher Utility. To the right is the communications component. Communication is integral to XP and, therefore, is an essential part of the PAM client interface.

The client GUI is devoted to supporting the groupware activities of communication, production and coordination. The *Project Hierarchy*, along with the metaphors that fit into the work area, support production and some communication. For example, awareness information presented in the *Project Hierarchy* (colour coded node status) and in the *Work Area* (movement of *Cards*) serve as a lightweight communication and coordination mechanism. The communication component—the Chat and System Monitor Messages panel—complements the lightweight awareness (Section 8.6). This is manifested as support for indirect human-to-human communication and the normal XP social protocols, which provide another mechanism for coordination. We discussed how users interact with these metaphors in Chapter 5.

The main tools of normal XP used in the Planning Game are index cards, whiteboards and flipcharts. These are usually set up so that they are visible to everyone, and act as “information radiators” (Cockburn 2002). The metaphors and architecture discussed above, provide equivalent “information radiators” in addition to guaranteed persistence, traceability and support for arbitrarily dispersed users.

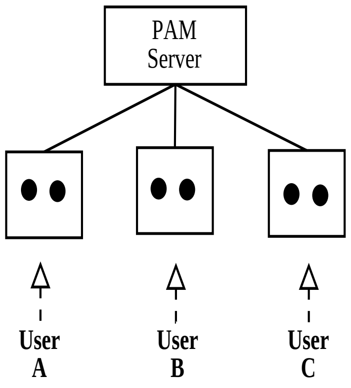
8.5.2 User Interaction with the Project Document

Schedule and artefact manipulation—XP project activities— are not done in isolation. PAM as we described in Section 8.1 is more of a hybrid distributed system than a client/server. All project data, that is, all details pertaining to the current software being developed with the exception of the source code files, are stored on the server. We use the metaphor of a *Project Document* to describe the complete set of project data.

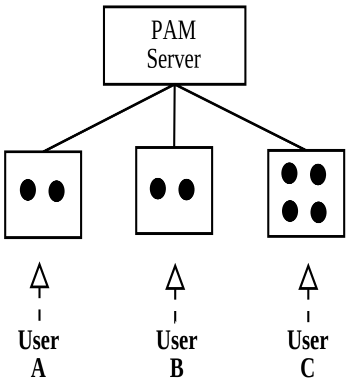
Software development deliverables are usually copious and discrete. This is especially evident with source code files and documentation. Even though XP significantly reduces the amount of documentation deliverables, still deliverables such as User Stories, Tasks and source files remain to be managed. The Project Document is dynamic and grows during the project. While using PAM, various views of the *Project Document* are referenced and changed by different users. These actions sometimes conflict. Consequently, the PAM client interface model includes support for flexibility when users are viewing different aspects of the Project Document. Users are free to change their views without fear of affecting other user's ability to view or interact with other aspects of the project. This flexibility is referred to as relaxed-WYSIWIS. We illustrate this with the following example.

Suppose users **A**, **B** and **C** are collaborating on an Iteration plan. They may share the same view, such as the Release tab of the *Stacked Slated*. If user **C** prefers to use the *Virtual Desktop* or even wants to view a different aspect of the project, he/she is free to switch views without obstructing or otherwise affecting the collaboration of **A** and **B**. **A** and **B**, however, are made aware of the presence and location of **C** in the *Project Hierarchy* (Section 8.6). This allows users **A** and **B** to infer **C's** potential actions. Figure 8.8 illustrates this scenario. In this model, users collaborate by combining the utility of PAM with the social protocols already in use on XP projects. The users will communicate in order to agree on which view to use for collaboration. They may then synchronise their views. The real time updates propagated in PAM take care of synchronisation of the information in the views.

Our collaboration model deviates from traditional CSCW protocols. Typically, participants in a CSCW session collaborate on a single artefact such as a drawing in a whiteboard application. The artefact is usually small and the elements in the shared information space tightly coupled. Different views are thus not usually required. For these applications, the common technique exploited is WYSIWIS (What I See Is What You See). Collaborators typically share a common desktop or application (Section 8.2). The individual small sizes and discreteness of elements in the Project Document, the iterative nature of XP, the separation of projects into



(a) Before: A, B and C using the same view



(b) After: C using different view from A and B

Figure 8.8: Illustration of project document viewing flexibility; while maintaining collaboration.

small development units (Releases and Iterations) and the need for several different views of the *Project Document* to be in use simultaneously and unobtrusively across the PAM environment makes strict WYSIWIS inappropriate for PAM.

We introduce the concept of "What You Know Is What I Know" (WYKIWIK) to describe our implementation of relaxed-WYSIWIS. WIKIWIK works at the PAM application level. Knowledge of the current state of the PAM system state is maintained at this level by means of real time updates. All connected PAM applications thus share the same knowledge of the state of the PAM system—that is, the project and session state—at every instance in time. This knowledge is translated and made available to users through the awareness mechanism. The awareness mechanism supports freedom to be in any view but still enables lightweight peripheral awareness of state changes. Collaborators using the same view are also provided with lightweight awareness. The fine details of the state change are also available. Remote users only need to switch explicitly to the view in which the fine details are available, if they so desire. Lightweight, unobtrusive awareness is facilitated through the use of low impact techniques, such as colour changes and small discernible movement.

8.6 Exposing the Awareness Mechanism

In PAM, none of the activities supported are conducted in isolation (as is the case with single-user applications). The actions of any user impact on the state of the project or the session in which that user is working. Therefore, awareness of what others are (a) about to do, (b) are doing and (c) have done, is essential to the coordination component in PAM. The user interface is the primary means of interacting with PAM and, therefore, the sensible place to facilitate awareness.

There are two types of state changes that affect PAM: (1) session state change and (2) project state change. A session describes one or more PAM client applications connected to the DXP server to perform work. An example of a session state change is a user joining or leaving an existing session. A project state change occurs when user interaction results in a change in the Project Document such as an insert, a deletion or an update. All client applications in the same project session are updated with the state changes.

To facilitate system wide awareness of changes, we implemented mechanisms for sending and receiving state change updates and for handing incoming updates. The architecture and mechanics of sending and receiving state change updates are discussed in Chapter 7. The mechanism for handling incoming updates is responsible for updating client application data with changes received from the server and for presenting appropriate awareness information to the system user. The awareness information can be in the form of audio or visual cues, such as changing of the colours of the nodes in the Project Hierarchy.

Unobtrusive collaboration is a major goal of PAM. PAM provides awareness of the actions of remote participants. This includes awareness of the presence of remote users in the Project Hierarchy and their actions. Lightweight awareness of their intentions and actions, such as initiation of an update and commit operation, is supported in the communications utility by means of the Message Display Panel (see Figure 5.7). This display can be examined for historical information as enabling the viewer to keep abreast with a peripheral concern's manipulation.

User's actions affect specific concerns or their associations: a suitable place to provide feedback about a users presence is therefore relative to the concerns in the Project Hierarchy as a list (see Section 5.1, page 63, for details of how this is manifested in the interface). The presence of the remote user in the Project

Hierarchy also aids inference about the next possible set of actions from which they may select to execute. By looking at the tree, one can easily determine: (1) who is working in the current session, (2) the artefact that each user is working with and (3) the set of potential actions that each user can effect.

Awareness may also be supported in the Stacked Slates and Virtual Desktop. Awareness in the currently implemented Virtual Desktop includes, feedback about which user is moving a card and the movement of the card. This form of awareness is exploited in activities such as Developers taking responsibility for Tasks (see page 5.2.6). In future versions of PAM the visual cues and widgets used in representing concerns and defining the forms may be upgraded to provide more awareness feedback. For example, the use of multiuser scrollbars would allow the size of the Virtual Desktop to be increased.

8.7 Summary

The approach we used in the analysis and design of the PAM client user interface has resulted in the implementation of a UI that reflects the nature of conventional XP and supports the unobtrusive collaboration of dispersed and co-located teams. In the design of the user interface, we exploit existing metaphors and characteristics of XP, as well as established design patterns. These design decisions are based on the aspects of XP which, based on our activity analysis (Chapter 4), offered the most potential to benefit XP teams if augmented with computer-based support. We also provide an awareness mechanism in PAM which supports dispersed teams. The current implementation adequately supports the aspects necessary for supporting XP teams.

The current PAM interface is experimental only. The separation of the PAM client from the server and the means of communication between them allow additional new interface implementation to be added to the PAM environment. For example, interfaces can be developed for use with Personal Digital Assistant (PDA) without obstructing the existing system.

PAM is designed to complement normal XP. It is anticipated that users will often swap between the PAM environment and normal XP practices, such as the common whiteboard and desktop, as a matter of convenience. Therefore, in the current implementation we place specific emphasis on supporting the maintenance of the same mental model of the XP process when XP teams use PAM to complement their normal XP practices. In Chapter 5 we discussed the implementation of these interfaces from the perspective of users of the system. The discussion and the implementation illustrate the utility of the model discussed in this chapter, and show how the mental models are maintained.

In the next chapter we recap our discussion of PAM and provide some feedback about the insights obtained during our research. We also discuss future work for enhancing our support for XP.

Chapter IX

Discussion and Future Work

Our research was motivated by the opportunity to alleviate the co-location requirement of the Extreme Programming process. We posited that relaxing this constraint would benefit XP teams and enhance the scalability of the process. Initial analysis of the activities and needs of XP teams indicated that a high level of support for time and space independent communication offers tremendous potential for the achievement of our goals. In this regard, initial work centered on executing spikes to select an appropriate connectivity middleware and the development and investigation of collaborative tools which can be integrated to provide support for dispersed XP teams (see Section 7.2).

During the course of our activity analysis (Chapter 4), it was evident that even though computer-based support for co-located XP teams is beneficial, there was a lack of such tools. We found that XP teams predominantly used available tools, such as compilers, build systems and versions systems, for application development activities. Thus there existed a need for project management support. However, current project management tools (for example, Microsoft Project) are not well suited to XP.

We subsequently extended the scope of our development to include support for production activities of XP. Several informal evaluations were conducted during the incremental prototype development of the groupware. Feedback obtained helped to guide development of subsequent versions. The result of our research is a desktop-based client/server groupware application, which offers support for XP project planning and coordination—the Planning Game. In addition to satisfying the goals of our research, PAM also shows promise, with appropriate extensions to support application development, as a valid integrated tool for collaborative software development.

9.1 Reflections of our Research: A Gentle Discussion.

In Chapters 2 and 3 we discussed the XP process and CSCW. We described the nature of XP and showed how the simple manual tools and techniques used on XP projects to support collaboration, such as index cards, whiteboards and co-location, put the process at risk. The primary risks identified were the potential for inadequate software maintenance and lack of scalability of the process. These shortcomings give rise to the need for specialised tools. We concluded that CSCW is an appropriate technology with which to investigate computer-based support for XP.

Analysis of the requirement and potential impact of computer-based support for XP raised socio-technical issues. These include consideration of platform support, deployment architecture, usability, utility, capability and the potential impact on the existing protocols typical of XP projects. Utility is important because it is concerned with the return on investment in developing the system—that is, whether or not PAM provides useful, worthwhile functionality. Capability concerns whether it does what it is supposed to do. Usability deals with the efficiency of supporting users of PAM in accomplishing tasks. Informal evaluations suggest that PAM meets utility and capability satisfaction.

PAM is not meant to replace XP, but rather to complement it. Our desktop-based client/server approach deliberately differs from the web-based approaches reported (see Section 3.4) because our examination of their features shows that they do not provide adequate support for XP teams so that they benefit from the intent of XP activities (Chapter 4). For example, none of the web-based tools allow direct manipulation of User Story cards. Direct manipulation of the cards has been established as a critical aspect of the XP Planning Game activity and is meant to educe spontaneity and bridge the gap between the Customer and the developers. PAM fits into the context of XP environments (see Chapter 4 and Section 6.1).

In Chapter 4 we analysed the activities and nature of XP and found that a higher level of support for collaboration is needed than is provided in web-based tools. We categorised XP project activities as application development and project management activities. Both aspects involve XP team members working closely together, which is indicative of the inherent collaboration in XP. Further, our analysis shows that XP projects revolve around the creation and transformation of User Stories.

We identified the highly collaborative aspects of the User Story life-cycle (for example, User Story elicitation) and designed PAM to augment them. We found that support for these aspects depend on broadcast and point-to-point communications. These forms of communication complement each other by supporting transient information exchange, such as awareness of in transit User Stories, and crucial data management operations respectively. We executed spikes to investigate the suitability of some available middleware for the communications requirements of PAM. We assessed the connectivity middleware in Chapter 7 and provide a summary rating of the criteria used to determine their suitability in Table 7.1.

We choose CAISE for the connectivity middleware because it satisfies the main criteria—that is, lightweight, simplicity and the flexibility of the communication mechanism. Though the table shows CAISE has low rating for robustness and reliability we expect that these will be enhanced as the middleware matures. In keeping with our expectations, recent versions of CAISE has showed measurable improvement. The maturity and development of CAISE is a result of a mutually beneficial collaboration between our project and (Cook & Churcher 2003*b*) the developers of CAISE. While CAISE provided the utility needed for our research, our project provided a test environment for it. We reported on issues as the arose. These were always handled within reasonable time.

Formal evaluations of PAM suggest that it offers adequate support with benefits for both co-located and dispersed XP teams. We discussed the features of PAM in Chapter 5 and show that they satisfy the aspects identified for support in Chapter 4. The metaphors used in the client interface (Chapter6) are based on existing XP metaphors and characteristics. For example, we use the metaphor 'information radiator' to describe PAM, since it provides up to date information about the state of the project in a similar manner to the common whiteboard used on XP projects. The natural hierarchical arrangements of XP project schedules and artefacts are utilised in the Project Hierarchy and the Stacked Slates interface components. The Virtual Desktop is an emulation of the common desktop used for the Planning Game. Support for communication and coordination of users collaboration while working on the `Project Document`, in addition to the metaphors used in PAM allows users to maintain the same mental model of the XP process when they use PAM to complement the process.

Though PAM fulfills its primary purpose it is experimental only. In the following section we discuss how PAM can be refined and tailored to suit unique needs.

9.2 User Evaluation

XP is a relatively simple process. Nevertheless, by comparison with CSCW standards it is complex. PAM combines CSCW and XP and therefore, demands non-trivial evaluations for meaningful information to be obtained. Further, PAM is neither a technique nor an application that is independent of a wider context. Rather, PAM is designed to complement and fit into the existing context of an intricate group oriented problem domain.

Experimental design to evaluate PAM must take all of the preceding aspects into consideration. In this regard, we suggest several evaluations each examining a slice of the overall context, as opposed to individual aspects such as usability and short term comparative analysis. We recommend that the goal of evaluations must be to determine whether or not computer-based support for XP puts remote and co-located teams using PAM, to complement XP, at significant disadvantage to co-located teams practicing normal XP. We know that certain disadvantages exists. For example, the inability to match the efficiency of face-to-face communication. Evaluations, however, may be carried out to quantify these disadvantages. In this regard, the best suited context will need to be examined. For example, the comparison of co-located groups working with and without the use of PAM, and also dispersed XP teams using PAM compared with dispersed teams using conventional methods. In view of the disadvantages we posit that a reasonable measure of acceptability would be if the utility of PAM out-weights the shortcomings and users are willing to pay live with it.

In Chapter 5 we provided a discussion of the issues that are essential for meaningful evaluation of PAM. Several researches have been conducted with the aim of supporting distributed XP teams (Section 3.4). Each of these approaches, however, has used different approaches. Opportunities thus exists for collaborative comparisons and evaluations between different groups. Evaluation must be carried out in the context of a reasonable size project of at least six months duration with teams of between ten to twenty members. Collaboration between researchers also provide opportunities for international experiments. These investigates another context in which dispersed teams may be deployed—that is, in separate countries and different time zones. The result of these collaboration has potential to provide deeper insight into the real needs of XP and therefore enable researchers to provide stronger support for distributed XP teams.

Rigorous meaningful evaluation of the usability, the appropriateness of PAM system architecture and the impact of PAM on XP projects require much more time and human resources than are available for our research. We consider usability evaluations of PAM as premature. Further, elaborate usability tests are considered to be a waste of resources, when meaningful tests can be carried out informally with as little as five users(Nielsen 2000). Though necessary, their usefulness in the context of research is dependent on PAM having first been formally evaluated and found to be beneficial to XP teams, and that the usability evaluation is being conducted to determine what has to be done to enhance PAM's utility and capability. Provided that utility and HCI evaluations show that PAM is beneficial to XP teams, evaluations of appropriateness of the system architecture and the impact of computer-based support for XP may follow. Results from these latter evaluations can be used to optimise PAM for commercially use.

9.3 Opportunities for extending and enhancing PAM

PAM is an experimental groupware application developed using 'half-pair' programming (yours truly) and the XP approach as far as technically possible. We used an iterative incremental prototyping approach. Each increment addressed issues identified during informal evaluations and user testing. All of the implementation and testing was carried out by a single developer. Consequently, collaborative user interaction issues were only revealed during informal evaluations and some may still remain. We discuss below, means of addressing these issues that may enhance and/or extend PAM.

The PAM server provides a simple interface to PAM client applications which decouples the clients from the server. This separation, in addition to the relatively small size of the `Project Document` and the message passing connectivity middleware, allows PAM to be extended. It is possible and feasible to develop additional client interfaces for use with the current implementation without negative impact. For example, new interface metaphors or interfaces for other platforms, such as PDAs, may be developed to extend the availability project state information. The PAM server accepts messages of a defined format. Each message contains information such as sender, receiver, the data and instructions type. The PAM server interprets the instruction code performs the appropriate actions. Developers of new PAM client applications are only required to know the message format and the codes used for communication and to obtain the project concern class library. This library includes classes for each concern in addition to an abstract class with the codes used in PAM.

Groupware applications are challenged to emulate the immediacy of co-located groups. PAM faces similar challenges. The PAM client interface can be enhanced by strengthening and extending the awareness mechanism. For example, in addition to showing user's location in the Project Hierarchy, awareness to indicate which view (Virtual Desktop, Sketcher or Stacked Slates) the user is in may also be provided. The available space in the Virtual Desktop may be increased by adding multi-user scrollbars. Session state change information, such as in transit User Stories, are currently only available in certain views. This was done to illustrate the concept; however, the awareness mechanism may allow this information to be available system wide. In addition, support for activities such as developers signing up for Tasks may be enhanced with support for multiple select and drag, as well as a "Take ownership" operation available in the Virtual Desktop.

The resources and time available for our research was not adequate both to design and develop PAM, and to design and conduct formal experiments. However, the System Monitor (page 92) logs meaningful data about how the system is being used. These logs are stored in XML format, which make them amenable to analysis both in and outside the PAM environment. The System Monitor logs can be analysed to determine how PAM is being used and for metrics gathering. For example, it can be used to track how User Stories are being created, edited and so on, as well as to replay the evolution of User Stories (that is, the projects). Though the logs do not support subjective evaluation, we posit that the quantitative analysis which it supports offers equally important benefits.

9.4 Summary

This chapter presented a recapitulation of our research. We discussed our initial motivation and how our research evolved as we gained further insight into the XP process and the nature of support required. The feedback provided by informal user evaluations provided was a significant source of insight. We highlighted the socio-technical intricacies of our experimental prototype system, PAM, and made recommendations for meaningful rigorous evaluations. Further, we discussed means of extending and enhancing the current implementation of PAM.

Chapter X

Conclusion

This thesis investigated the opportunity of combining CSCW with XP as a means of alleviating the shortcomings of the XP process. XP is a modern lightweight process. It was developed to address the need for rapid delivery of working software and adaptability to changing requirements. In order to facilitate these goals, XP place the people above the process. In this regard, the process is optimised for collaboration. The tools and techniques defined for the XP process are intended to support the collaborative nature of XP activities. The tools which support application development are adequate. On the other hand, the manual tools used for project planning and coordination do not allow sufficient information to pass over into the maintenance stage. These impose long term consequences on the process. In addition, the co-location practice affects the scalability of the process.

The shortcomings of XP offers opportunities for computer-based support. CSCW by virtue of its research emphasis and focus on development of computer-based tools as resources to support group work was identified as an appropriate technology to alleviate XP's shortcomings. CSCW is appropriate because it complements XP and offers techniques to augment the existing collaboration.

Analysis of XP activities shows that the most benefits are to be derived from computer-based support for those activities with the highest levels of collaboration. These were identified to be the activities associate with the Planning Game—that is Release and Iteration Planning. In conventional XP these activities are carried out with close cooperation of the customer and developers. Two major forms of communication were deemed necessary for supporting XP teams with computer based support: broadcast and point-to-point. Broadcast communication is mainly used to handle transient non-critical information exchange between collaborators and for awareness. On the other hand, point-to-point is used for critical data management operations.

We designed and developed a desktop-based client/server groupware application as a proof of concept to support co-located as well as dispersed teams. We exploited existing metaphors and characteristics of conventional XP, in order to minimise the difference in user's mental models of XP and CSCW supported XP. Our groupware tool offers relevant and appropriate functionalities and addresses the shortcomings of XP. We provide persistence for project data such as User Stories and Tasks. In this regard, information about XP projects can now survive beyond the end of projects and be used in subsequent analysis of the process itself. Support is provided which makes project state information available synchronously and asynchronously. We describe our groupware as a CSCW enabled information radiator.

Our groupware tools is experimental only. Therefore, there is opportunities for enhancing and extending it. The awareness mechanism in the current version is not fully optimised. Session control protocols may be added to allow collaborating pairs or groups to have a designated driver who can leads the rest of the group through the information space. In addition, the data replication information sharing technique can be enhanced to allow client side caching of project data and offline work. This enhancement will also need to address synchronisation. The emphasis of our research was to provide functionality. Therefore, improvements can be made in terms addressing HCI issues in of the interface. Finally, planned evaluations my be conducted to quantify the utility offered by our groupware.

Several informal evaluations were conducted during the course of the groupware's development. Feedback from its use suggest that computer-based support benefits XP teams. The client server architecture supports extension of our groupware. Additional interfaces, customised for specific needs can easily be added to the environment without impact on the existing system. We conclude that our approach addresses real needs of XP activities and teams, and look forward to extending our work further in the future.

List of Figures

List of Tables

Appendix A

Screen Shots of the Prototype System

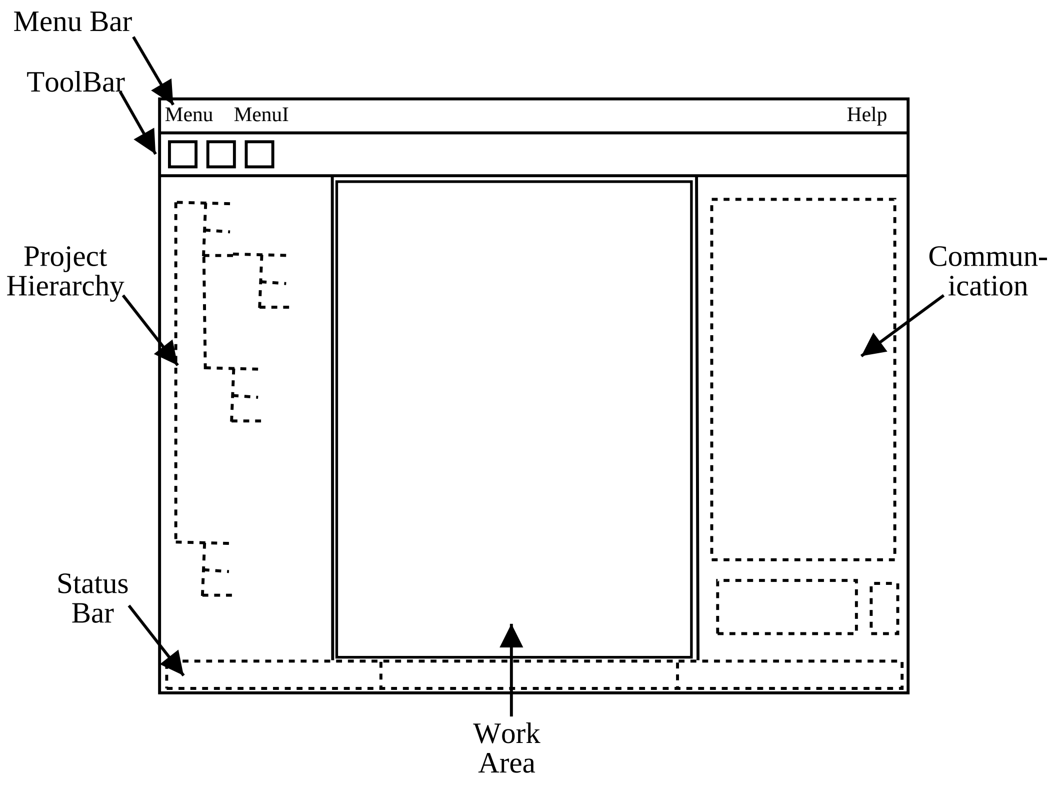
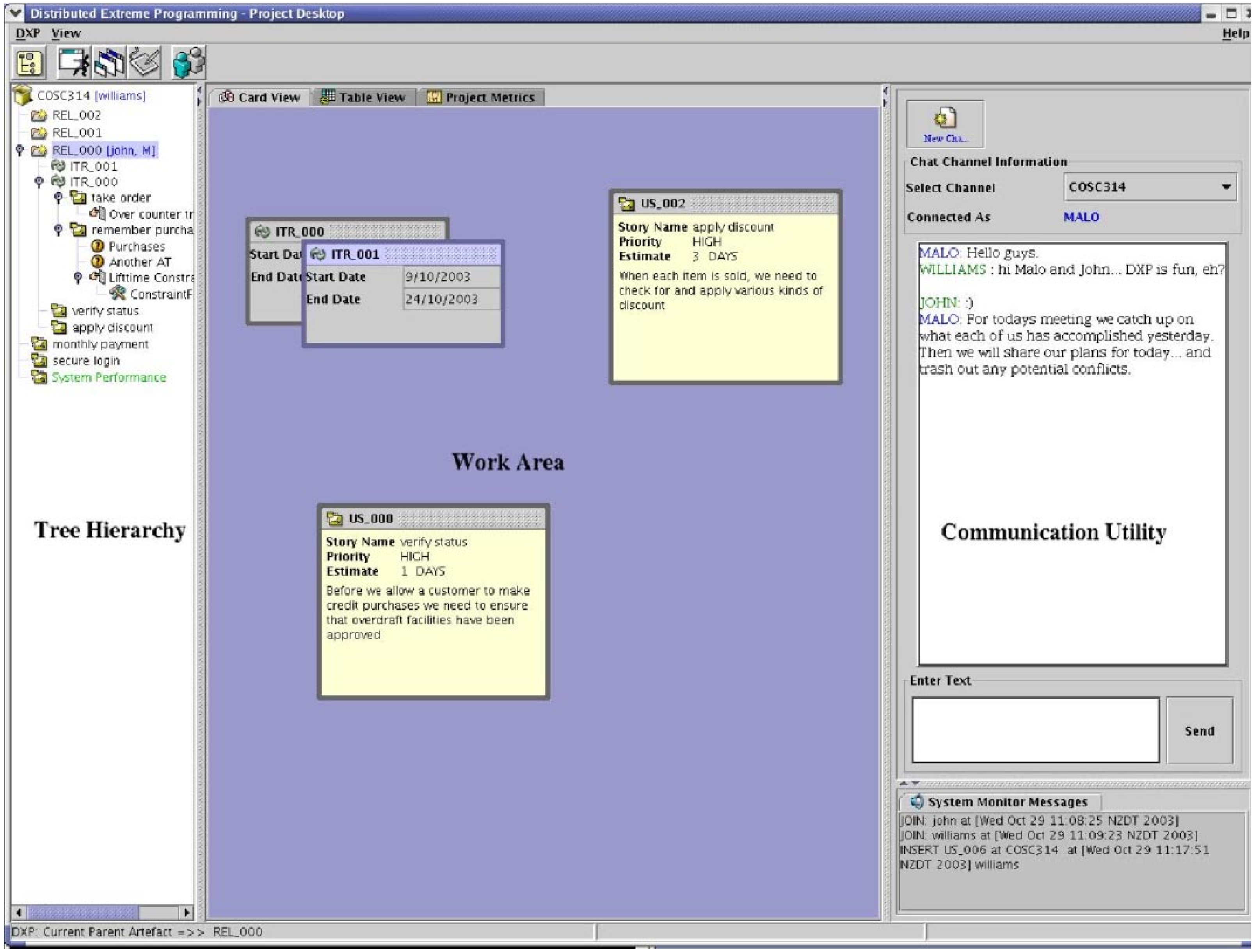


Figure A.1: Illustration of the prototype system User Interface model



COSC314 [williams]
REL_002
REL_001
REL_000 [john, M]
ITR_001
ITR_000
take order
Over counter tr
remember purcha
Purchases
Another AT
Lifttime Constra
ConstraintF
verify status
apply discount
monthly payment
secure login
System Performance

ITR_000
Start Dat
End Date

ITR_001
Start Date 9/10/2003
End Date 24/10/2003

US_002
Story Name apply discount
Priority HIGH
Estimate 3 DAYS
When each item is sold, we need to check for and apply various kinds of discount

US_000
Story Name verify status
Priority HIGH
Estimate 1 DAYS
Before we allow a customer to make credit purchases we need to ensure that overdraft facilities have been approved

Chat Channel Information
Select Channel COSC314
Connected As MALO

MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

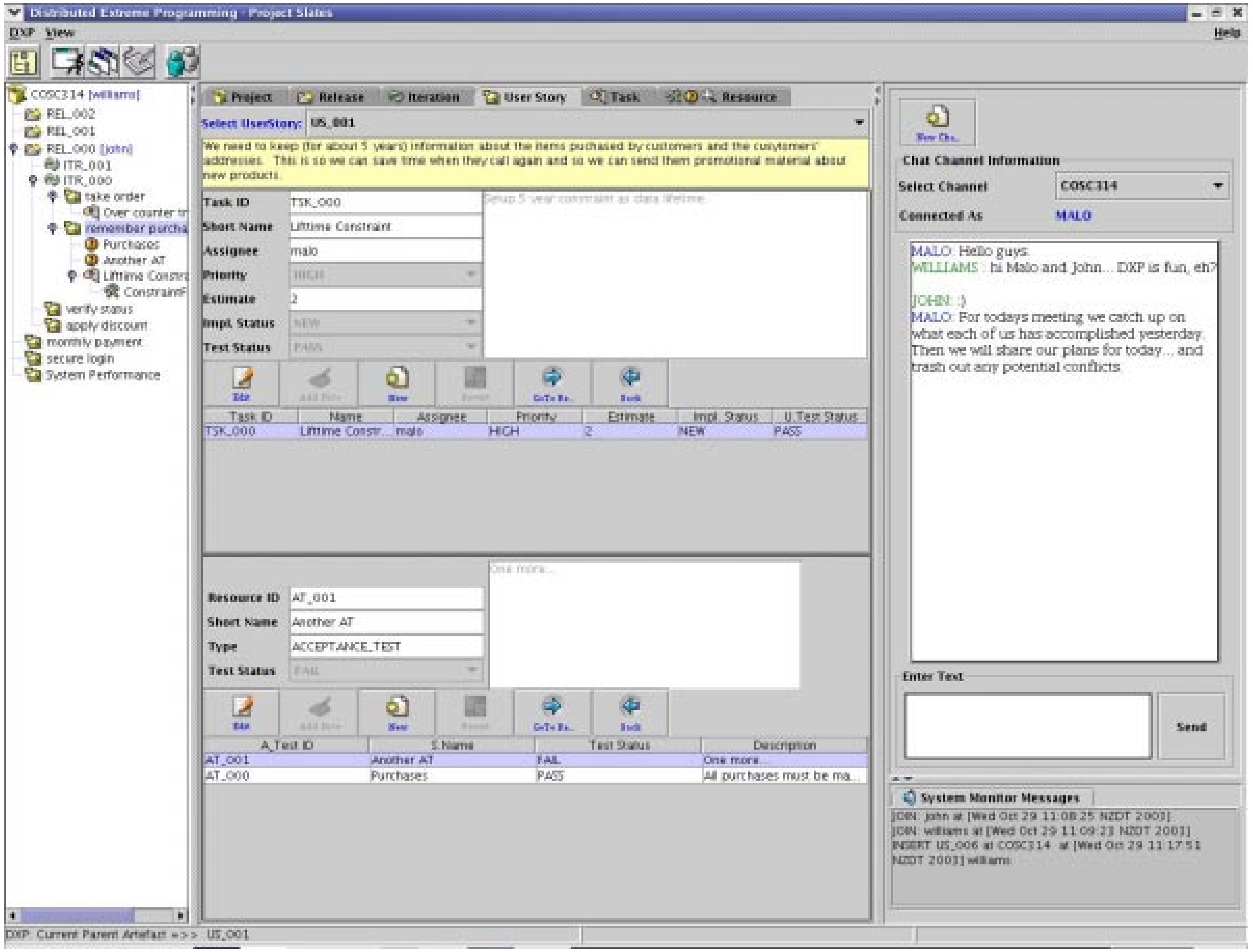
JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

Enter Text

System Monitor Messages
```
JOIN: john at [Wed Oct 29 11:08:25 NZDT 2003]
JOIN: williams at [Wed Oct 29 11:09:23 NZDT 2003]
INSERT US_006 at COSC314  at [Wed Oct 29 11:17:51
NZDT 2003] williams
```

DXP: Current Parent Artefact =>> REL_000

Figure A.2: PAM client User Interface showing the Tree Hierarchy, Work Area (currently used by the Virtual Desktop) and the Communication Utility components, to the left, middle and right respectively.


Distributed Extreme Programming - Project Status

Select UserStory: US_001

We need to keep (for about 5 years) information about the items puchased by customers and the cusytomers' addresses. This is so we can save time when they call again and so we can send them promotional material about new products.

Task ID	TSK_000
Short Name	Liftime Constraint
Assignee	malo
Priority	HIGH
Estimate	2
Impl. Status	NEW
Test Status	PASS

Setup 5 year constraint as data lifetime.

Task ID	Name	Assignee	Priority	Estimate	Impl. Status	U.Test Status
TSK_000	Liftime Constr...	malo	HIGH	2	NEW	PASS

Resource ID	AT_001
Short Name	Another AT
Type	ACCEPTANCE_TEST
Test Status	FAIL

One more...

A_Test ID	S.Name	Test Status	Description
AT_001	Another AT	FAIL	One more...
AT_000	Purchases	PASS	All purchases must be ma...

Chat Channel Information

Select Channel	COSC314
Connected As	MALO

MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

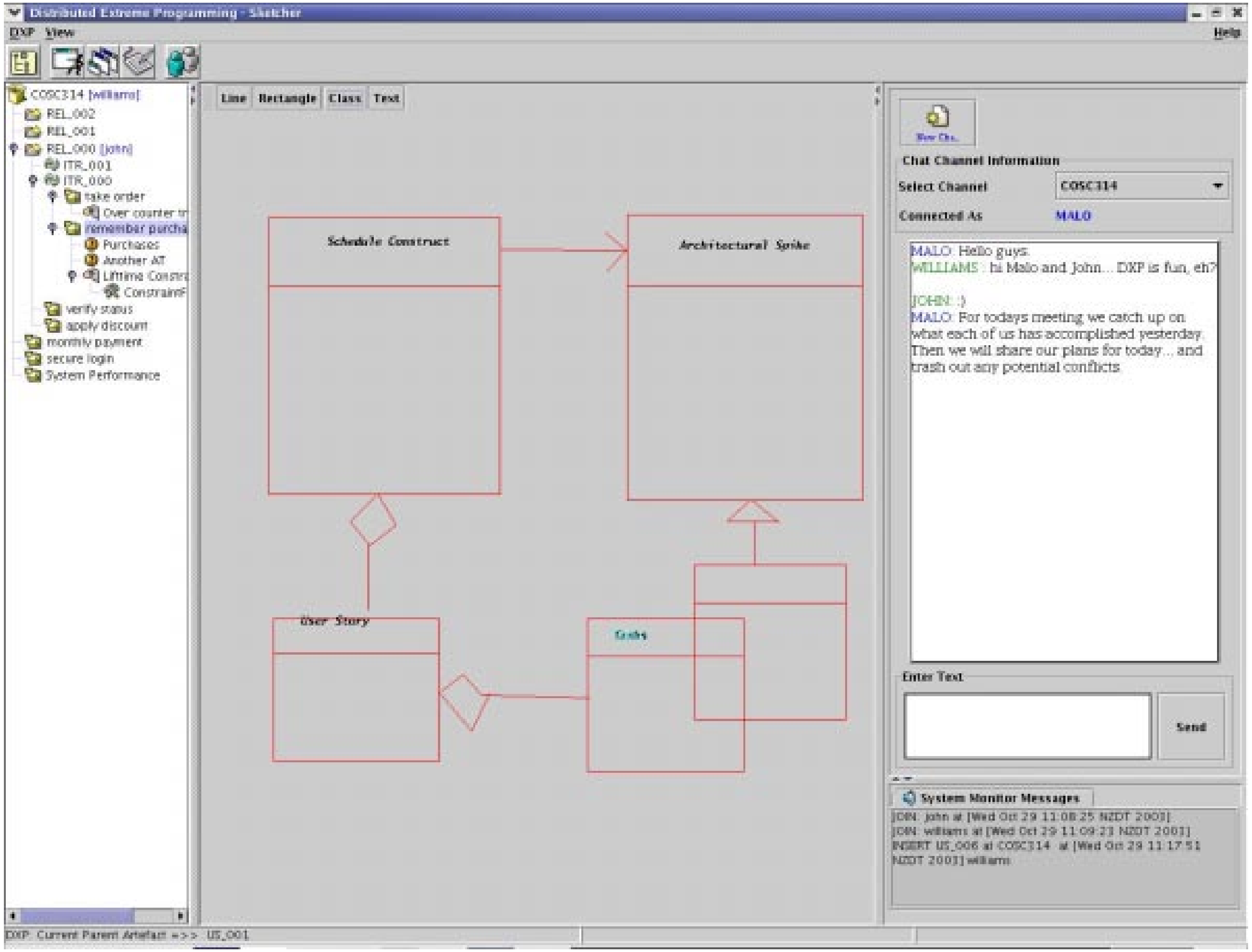
JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

System Monitor Messages

```
JOIN: john at [Wed Oct 29 11:08:25 NZDT 2003]
JOIN: williams at [Wed Oct 29 11:09:23 NZDT 2003]
INSERT US_006 at COSC314  at [Wed Oct 29 11:17:51
NZDT 2003] williams
```

DXP: Current Parent Artefact =>> US_001

Figure A.3: PAM client User Interface showing TabPanes.

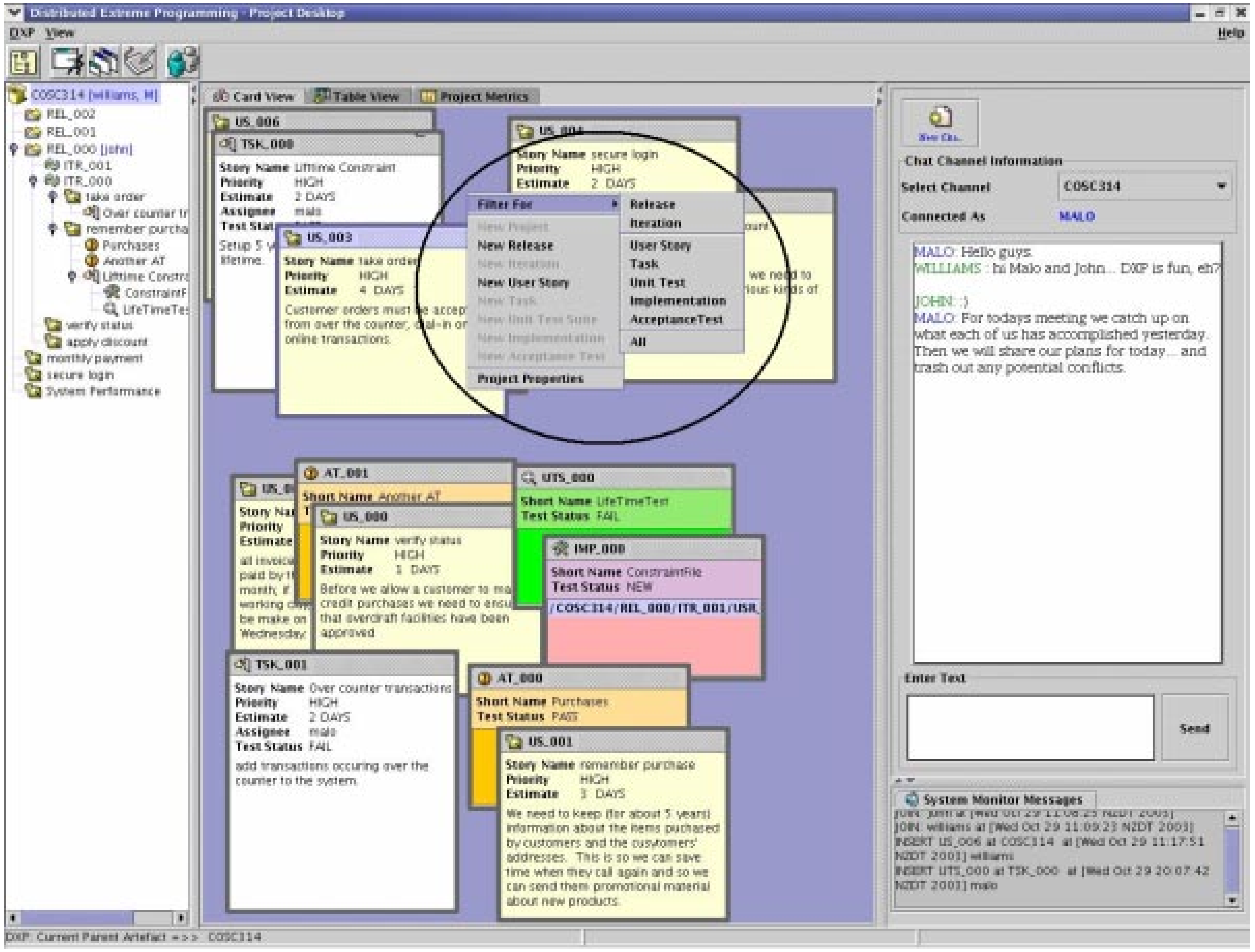


MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

```
JOIN: john at [Wed Oct 29 11:08:25 NZDT 2003]
JOIN: williams at [Wed Oct 29 11:09:23 NZDT 2003]
INSERT US_006 at COSC314  at [Wed Oct 29 11:17:51
NZDT 2003] williams
```

Figure A.4: Sketcher Utility, with minimal features demonstrates simple diagram drawn collabotatively.



COSC314 [williams, M]
REL_002
REL_001
REL_000 [john]
ITR_001
ITR_000
take order
Over counter tr
remember purcha
Purchases
Another AT
Liftime Constra
ConstraintF
LifeTimeTes
verify status
apply discount
monthly payment
secure login
System Performance

US_006

TSK_000
Story Name Liftime Constraint
Priority HIGH
Estimate 2 DAYS
Assignee malo
Test Stat
Setup 5 y
lifetime.

US_004
Story Name secure login
Priority HIGH
Estimate 2 DAYS

US_003
Story Name take order
Priority HIGH
Estimate 4 DAYS
Customer orders must be accep
from over the counter, dial-in or
online transactions.

US_0
Story Na
Priority
Estimate
all invoice
paid by t
month; if
working d
be make on
Wednesday.

AT_001
Short Name Another AT

UTS_000
Short Name LifeTimeTest
Test Status FAIL

US_000
Story Name verify status
Priority HIGH
Estimate 1 DAYS
Before we allow a customer to ma
credit purchases we need to ensu
that overdraft facilities have been
approved

IMP_000
Short Name ConstraintFile
Test Status NEW
/COSC314/REL_000/ITR_001/USR_

TSK_001
Story Name Over counter transactions
Priority HIGH
Estimate 2 DAYS
Assignee malo
Test Status FAIL
add transactions occuring over the
counter to the system.

AT_000
Short Name Purchases
Test Status PASS

US_001
Story Name remember purchase
Priority HIGH
Estimate 3 DAYS
We need to keep (for about 5 years)
information about the items puchased
by customers and the customers'
addresses. This is so we can save
time when they call again and so we
can send them promotional material
about new products.

Select Channel COSC314
Connected As MALO

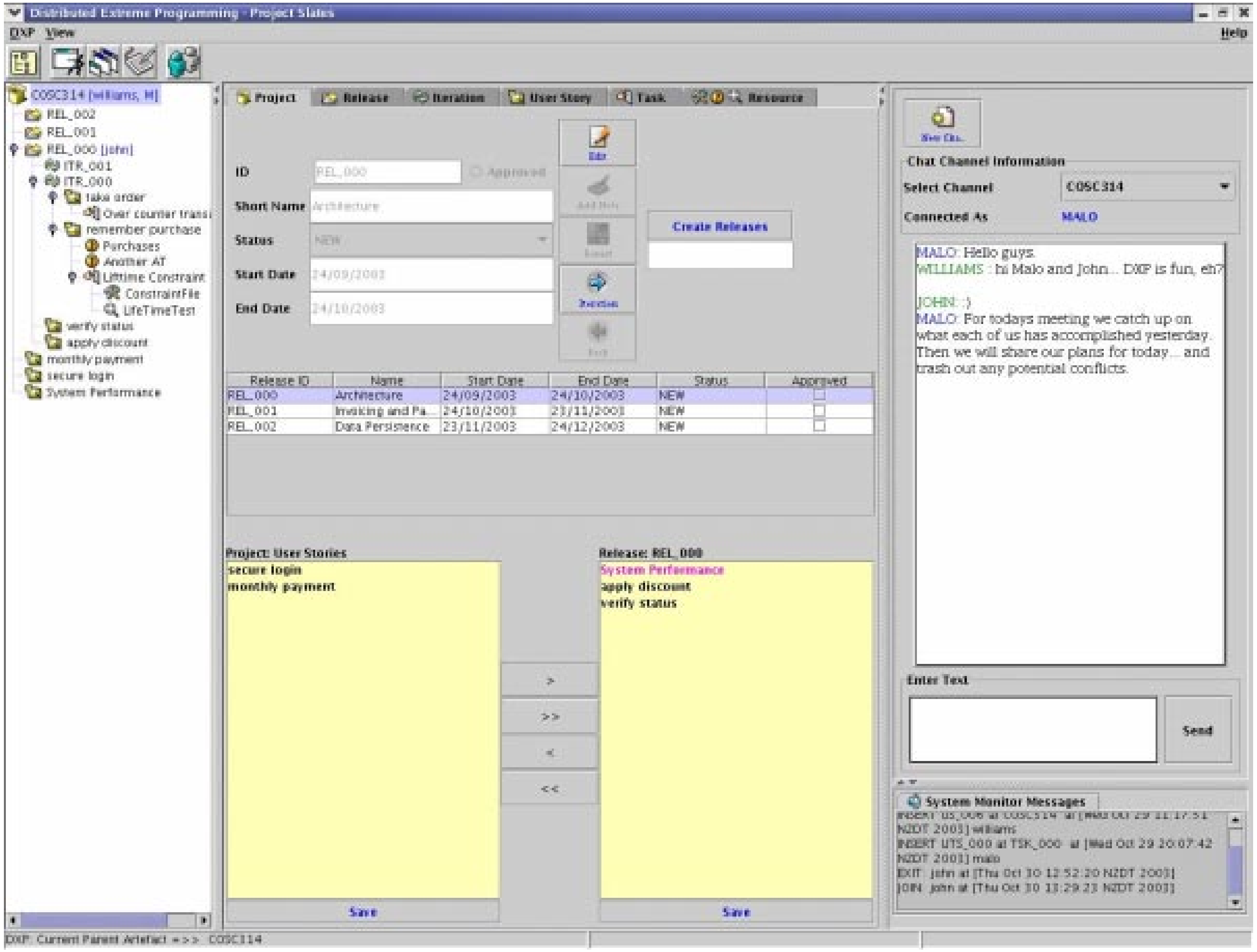
MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

```
JOIN: john at [Wed Oct 29 11:08:25 NZDT 2003]
JOIN: williams at [Wed Oct 29 11:09:23 NZDT 2003]
INSERT US_006 at COSC314  at [Wed Oct 29 11:17:51
NZDT 2003] williams
INSERT UTS_000 at TSK_000  at [Wed Oct 29 20:07:42
NZDT 2003] malo
```

DXP: Current Parent Artefact =>> COSC314

Figure A.5: PAM Desktop displaying all Decendands of the Project.



COSC314 [williams, M]
REL_002
REL_001
REL_000 [john]
ITR_001
ITR_000
take order
Over counter transi
remember purchase
Purchases
Another AT
Liftime Constraint
ConstraintFile
LifeTimeTest
verify status
apply discount
monthly payment
secure login
System Performance

ID	REL_000
Short Name	Architecture
Status	NEW
Start Date	24/09/2003
End Date	24/10/2003

○ Approved

Release ID	Name	Start Date	End Date	Status	Approved
REL_000	Architecture	24/09/2003	24/10/2003	NEW	☐
REL_001	Invoicing and Pa...	24/10/2003	23/11/2003	NEW	☐
REL_002	Data Persistence	23/11/2003	24/12/2003	NEW	☐

Project: User Stories
secure login
monthly payment

Release: REL_000
System Performance
apply discount
verify status

Chat Channel Information
Select Channel COSC314
Connected As MALO

MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

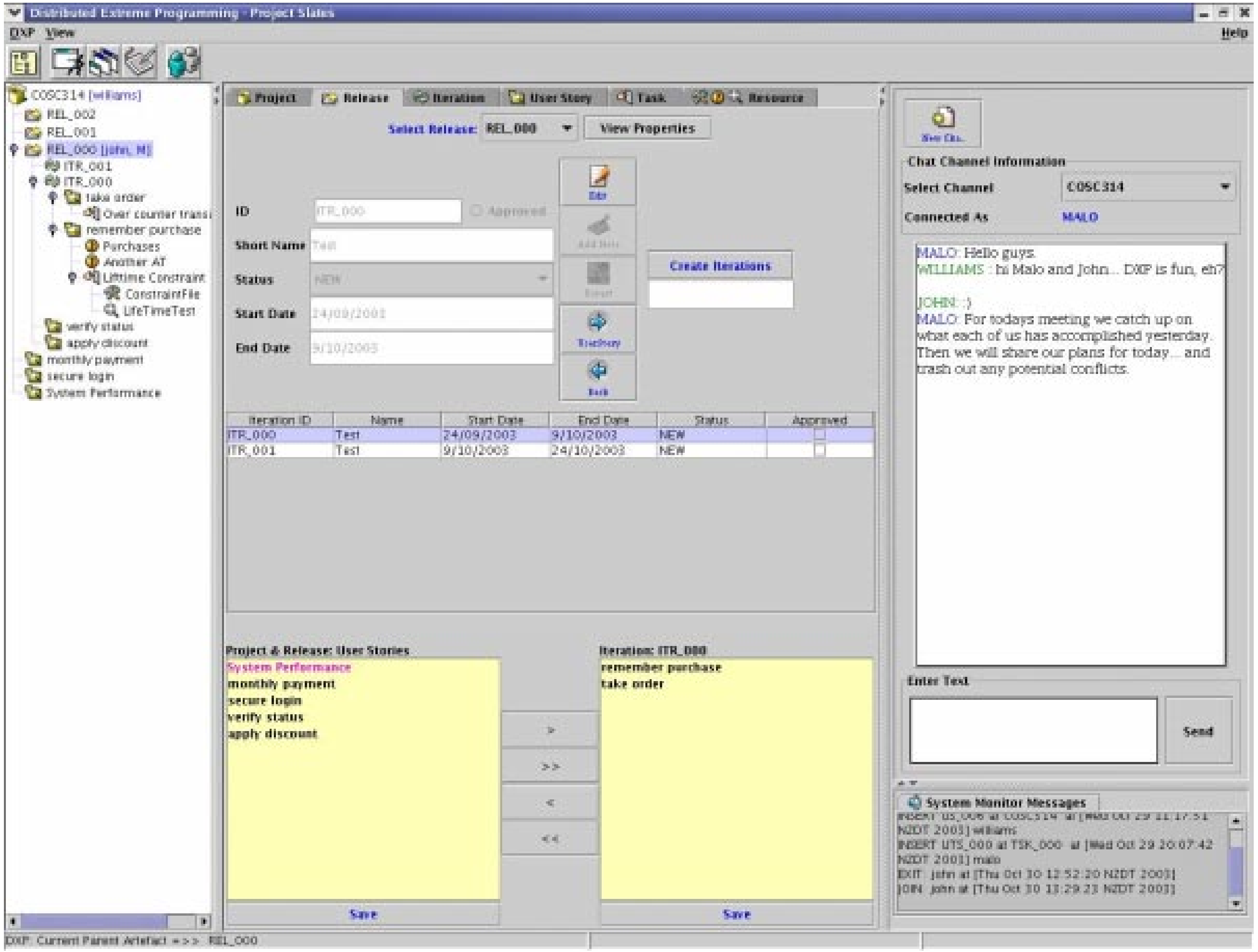
JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

System Monitor Messages

```
INSERT US_006 at COSC314 at [Wed Oct 29 11:17:51
NZDT 2003] williams
INSERT UTS_000 at TSK_000 at [Wed Oct 29 20:07:42
NZDT 2003] malo
EXIT john at [Thu Oct 30 12:52:20 NZDT 2003]
JOIN john at [Thu Oct 30 13:29:23 NZDT 2003]
```

DXP: Current Parent Artefact =>> COSC314

Figure A.6: PAM Stacked Slates displaying Release Slate.



Select Release: REL_000

ID	ITR_000 ☐ Approved
Short Name	Test
Status	NEW
Start Date	24/09/2003
End Date	9/10/2003

Iteration ID	Name	Start Date	End Date	Status	Approved
ITR_000	Test	24/09/2003	9/10/2003	NEW	☐
ITR_001	Test	9/10/2003	24/10/2003	NEW	☐

Project & Release: User Stories

- System Performance
- monthly payment
- secure login
- verify status
- apply discount

Iteration: ITR_000

- remember purchase
- take order

Chat Channel Information

Select Channel	COSC314
Connected As	MALO

MALO: Hello guys.
WILLIAMS : hi Malo and John... DXP is fun, eh?

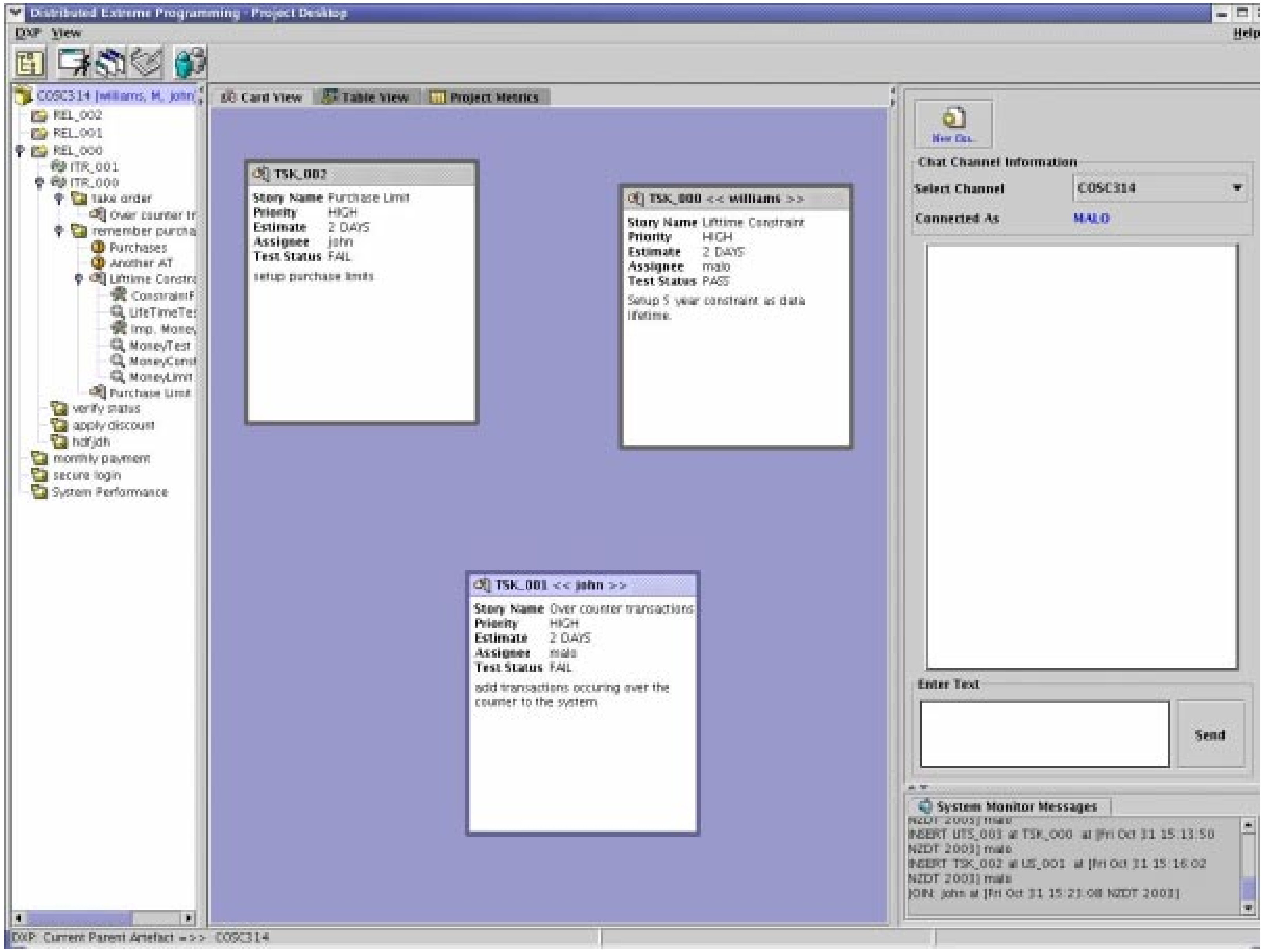
JOHN: :)
MALO: For todays meeting we catch up on what each of us has accomplished yesterday. Then we will share our plans for today... and trash out any potential conflicts.

Enter Text

System Monitor Messages

```
INSERT US_006 at COSC314 at [Wed Oct 29 11:17:51
NZDT 2003] williams
INSERT UTS_000 at TSK_000 at [Wed Oct 29 20:07:42
NZDT 2003] malo
EXIT: john at [Thu Oct 30 12:52:20 NZDT 2003]
JOIN: john at [Thu Oct 30 13:29:23 NZDT 2003]
```

Figure A.7: PAM Stacked SlatesDesktop displaying Iteration Slate.



TSK_002

Story Name	Purchase Limit
Priority	HIGH
Estimate	2 DAYS
Assignee	john
Test Status	FAIL

setup purchase limits

TSK_000 << williams >>

Story Name	Liftime Constraint
Priority	HIGH
Estimate	2 DAYS
Assignee	malo
Test Status	PASS

Setup 5 year constraint as data lifetime.

TSK_001 << john >>

Story Name	Over counter transactions
Priority	HIGH
Estimate	2 DAYS
Assignee	malo
Test Status	FAIL

add transactions occuring over the counter to the system.

Select Channel	COSC314
Connected As	MALO

```
NZDT 2003] malo
INSERT UTS_003 at TSK_000  at [Fri Oct 31 15:13:50
NZDT 2003] malo
INSERT TSK_002 at US_001  at [Fri Oct 31 15:16:02
NZDT 2003] malo
JOIN  john at [Fri Oct 31 15:23:08 NZDT 2003]
```

DXP: Current Parent Artefact = > > COSC314

Figure A.8: Task assignment

References

Agile Alliance (2001), ‘Agile alliance’, Internet URL. ”www.agilealliance.com”.

Alexander, C. (1979), *The Timeless Way of Building*, Oxford University Press.

Auer, K. & Miller, R. (2001), *Extreme Programming Applied: Playing to Win*, Addison-Wesley.

Baheti, P., Gehringer, E. & Stotts, D. (2002), Exploring the efficacy of distributed pair programming, *in* D. Wells & L. Williams, eds, ‘XP/Agile Universe 2002’, pp. 208–220.

Baker, K., Greenberg, S. & Gutwin, C. (2001), Heuristic evaluation of groupware based on the mechanics of collaboration, *in* M. R. Little & L. Nigay, eds, ‘Engineering for Human-Computer Interaction: 8th IFIP International Conference, EHCI 2001’, Vol. 2254 of *Lecture Notes in Computer Science*, Springer.

Bannon, K. J. & Schmidt, K. (1989), Cscw: four characters in search of a context, *in* ‘Proceedings of the First European Conference on Computer Supported Cooperative Work’, pp. 358–372.

Beck, K. (1999*a*), ‘Embracing change with extreme programming’, *Computer* **32**(10), 70–77.

Beck, K. (1999*b*), *Extreme programming explained: embrace change*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Beck, K. (2002), *Test Driven Development: By Example*, Addison-Wesley Pub Co.

Beck, K. & Fowler, M. (2000), *Planning Extreme Programming*, Addison-Wesley.

Beck, K. & Gamma, E. (1998), 'Test-infected: Programmers love writing tests', *Java Report* **3**(7), 51–56.

Boehm, B. (1988), A spiral model of software development and enhancement, *in* 'Computer', pp. 61–72.

Boehm, B. & Turner, R. (2003), *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley.

Borland Software Corporation (2003), 'Together controlcenter', Internet URL. "`www.togethersoft.com`".

Bray, T., Paoli, J., Sperberg-McQueen, C. M. & Maler, E. (2000), 'Extensible markup language (xml) 1.0 (second edition)', Internet URL. "`http://www.w3.org/TR/2000/REC-xml-20001006`".

Calvary, G., Coutaz, J. & Nigay, L. (1997), From single-user architecture design to pac*: a generic software architecture model for cscw, *in* 'Proceeding of the ACM CHI'97', Addison-Wesley, pp. 242–249.

Churcher, N. & Cerecke, C. (1996), Groupcrc: Exploring cscw support for software engineering, *in* J. Grundy & M. Apperley, eds, 'OzCHI'96: Proceedings of Sixth Australian conference on computer-human interaction', IEEE Computer Society Press, pp. 62–68.

Coad, P., LeFebvre, E. & De Luca, J. (1999), *Java Modeling In Color With UML: Enterprise Components and Process*, Prentice Hall.

Cockburn, A. (2002), *Agile Software Development*, Addison-Wesley.

Cook, C. (2003), Computer-Supported Collaborative Software Engineering, PhD thesis, University of Canterbury. Work in progress.

Cook, C. & Churcher, N. (2003*a*), An extensible framework for collaborative software engineering, *in* ‘APSEC 2003: 10th Asia-Pacific Software Engineering Conference’, IEEE press, Chiangmai, Thailand. accepted, to appear.

Cook, C. & Churcher, N. (2003*b*), A pure–java group communication framework, Technical report, University of Canterbury.

Eickelmann, N. (1999), Software engineering notes. http://www.acm.org/sigsoft/SEN/parnas.html.

Ellis, C. A., Gibbs, S. J. & Rein, G. L. (1993), ‘Groupware: Some issues and experiences’, **34**(1).

Ellis, J. (2000), ‘Xpcgi’, Internet URL. ”`http://xpcgi.sourceforge.net/`”.

Engelbart, D. & Lehtman, H. (1988), ‘Working together’, *BYTE* **13**(13), 245–252.

Fowler, M. (1999), *Refactoring: Improving the design of existing code*, Addison-Wesley. Fowler seems to be the authority on refactoring.

Fowler, M. (2001), ‘Is design dead’, Internet URL. `http://www.martinfowler.com/articles/designDead.html`

Fowler, M. (2003), ‘The new methodology’, Internet URL. `www.martinfowler.com/articles/newMethodology.html`

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Goleman, D. (1995), *Emotional Intelligence*, Bantam Books, New York.

Grudin, J. (1988), Why cscw applications fail: problems in the design and evaluation of organization of organizational interfaces, *in* ‘Proceedings of the 1988 ACM conference on Computer-supported cooperative work’, ACM Press, pp. 85–93.

Hanks, B. F. (2003), Tool Support for Distributed Pair Programming, PhD thesis, University of California at Santa Cruz.

Highsmith III, J. A. & Orr, K. (2000), *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, New York: Dorset House.

Jacobson, I. (1994), *Object-Oriented Software Engineering*, Addison-Wesley.

Jeffries, R., Anderson, A. & Hendrickson, C. (2001), *Extreme Programming Installed*, Addison-Wesley.

Keefer, G. (2002), ‘Extreme programming considered harmful for reliable software development’, http://www.avoca-vsm.com/Dateien-Download/ExtremeProgramming.pdf.

Kircher, M., Jain, P., Corsaro, A. & Levine, D. (2001), Distributed extreme programming, *in* ‘XP2001 - eXtreme Programming and Flexible Processes in Software Engineering’, http://www.xp2001.org/xp2001/conference/papers/Chapter16-Kircher+alii.pdf.

Lakoff, G. & Johnson, M. (1998), *Philosophy in the Flesh*, Basic Books.

Laurillau, Y. & Nigay, L. (2002), Clover architecture for groupware, *in* ‘Proceeding of the ACM CSCW’02’.

Lauwers, J. C. & Lantz, K. A. (1990), Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems, *in* ‘Proceedings of the ACM CHI 90 Conference on Human Factors in Computing Systems. ACM’.

Leuf, B. & Cunningham, W. (2001), *The Wiki Way: Collaboration and Sharing on the Internet*, Addison-Wesley Pub Co.

Marchesi, M., Succi, G., Wells, D. & Williams, L. (2002), *Extreme Programming Perspectives*, Addison Wesley.

Maurer, F. (2002), Supporting distributed extreme programming, *in* ‘””’.

Maurer, F. & Martel, S. (2002), Process support for distributed extreme programming teams, *in* ‘””’.

McBreen, P. (2002), *Software Craftsmanship*, Addison-Wesley.

McBreen, P. (2003), *Questioning Extreme Programming*, Pearson Education, Inc.

Miller, R. W. (2002), ‘Demystifying extreme programming’, Internet URL. `www-106.ibm.com/developerworks/java/library/j-xp0813/`

Nawrocki, J., Jasinski, M., Walter, B. & Wojciechowski, A. (2002), Extreme programming modified: Embrace requirements engineering practices, *in* ‘IEEE Joint International Conference on Requirements Engineering’.

Newkirk, J. W. & Martin, R. C. (2001), *Extreme Programming in Practice*, Addison-Wesley Pub Co.

Nielsen, J. (1993), *Usability Engineering*, Morgan Kaufmann.

Nielsen, J. (2000), ‘Why you only need to test with 5 users’, Internet URL. `www.useit.com/alertbox/20000319.html`

Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R. & George, J. F. (1991), ‘Electronic meeting systems to support group work’, **31**(7), 40–61.

ObjectMentor (2001), ‘Junit’, Internet URL. ”`http://www.junit.org/`”.

Paulk, M. C. (2001), Extreme programming from a CMM perspective, *in* ‘XP Universe’.

Paulk, M. C., Curtis, B., Chrissis, M. B. & Weber, C. V. (1993), ‘Capability maturity model’, *IEEE Software* **10**(4), 18–27.

Pinna, S., Lorrai, P., Marchesi, M. & Serra, N. (2003), Developing a tool supporting xp process, *in* F. Maurer & D. Wells, eds, 'XP/Agile Universe 2003', pp. 150–160.

Pressman, R. S. (2001), *Software Engineering: A practitioner's approach*, 5 edn, Mc Graw Hill.

Rees, M. J. (2002), A feasible user story tool for agile software development, *in* 'APSEC'.

Richardson, T., Stafford-Fraser, Q., Wood, K. R. & Hooper, A. (1998), 'Virtual network computing', *IEEE Iternet Computing* **2**(11), 33–38.

Rodden, T. (1991), 'A survey of cscw systems', *Interacting with Computers* **3**(3), 319–353.

Ross, S., Ramage, M. & Rogers, Y. (1995), 'Petra: participatory evaluation through redesign and analysis', *Interacting with Computers* **7**(4), 335–360.

Schuckmann, C., Kirchner, L., Schümmer, J. & Haake, J. M. (1996), Designing object-oriented synchronous groupware with COAST, *in* 'Computer Supported Cooperative Work', pp. 30–38.

Schümmer, T. & Schümmer, J. (2001), Support for distributed teams in extreme programming, *in* G. Succi & M. Marchesi, eds, 'eXtreme Programming Examined', Addison Wesley, pp. 355–377.

Schwaber, K. & Beedle, M. (2001), *Agile Software Development with SCRUM*, 1 edn, Prentice Hall.

Sire, S., Chatty, S., Gaspard-Boulinc, H. & Colin, F.-R. (1999), 'How can groupware preserver our collaboration skills? designing for direct collaboration', *Human-Computer Interaction – INTERACT'99* .

Software Engineering Institute (1995), *The Capability Maturity Model: Guidelines for Improving the Software Process*, 1 edn, Addison-Wesley.

Sommerville, I. (2001), *Software Engineering*, 6 edn, Addison-Wesley.

SourceForge.net (2003), 'Xplanner', Internet URL. "`http://http://www.xplanner.org/`".

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. & Tatar, D. (1987), 'Wysiwis revised: early experiences with multiuser interfaces', *ACM Transactions on Office Information Systems* **5**(2), 147–167.

Sun Microsystems, Inc. (2003), 'Data access object', Internet URL. `java.sun.com/blueprints/patterns/DAO.html`

Telles, M. (1990), Updting and older interface, *in* 'Proceedings ACM CHI'90 Conference'.

The Apache Software Foundation (2003), 'The APACHE ANT project', Internet URL. `ant.apache.org`

The C3 Team (1998), Chrysler goes to "extreme", *in* 'Distributed Object Computing', pp. 24–2. Ann Anderson, Ralph Beattie, Kent Beck, David Bryant, Marie DeArment, Martin Fowler, Margaret Fronczak, Rich Garzaniti, Dennis Gore, Brian Hacker, Chet Hendrickson, Ron Jeffries, Doug Joppie, David Kim, Paul Kowalsky, Debbie Mueller, Tom Murasky, Richard Nutter, Adrian Pantea, and Don Thomas are the C3 Team, Chrysler Corporation.

Van Der Vyver, G. & Lane, M. (2003), Using the new generation of is development techniques in effective group learning: A pilot study of a team-based approach in an it course, *in* 'Informing Science and Information Technology Education Conference'.

Wake, W. C. (2001), *Extreme Programming Explored*, Addison-Wesley.

Wampler, B. E. (2002), *The Essence of Object-Oriented Programming with Java and UML*, Addison-Wesley.

Wells, D. (2001), ‘Extreme programming: A gentle introduction’, Internet URL. `www.extremeprogramming.org`

Williams, L. & Kessler, R. (2000), ‘All i really need to know about pair programming i learned in kindergarten’, *Communications of the ACM* **43**(5), 108–114.

Zelkowitz, M., Shaw, A. & Gannon, J. (1979), *Principles of Software Engineering and Design*, Prentice-Hall.