Till Schümmer

A Pattern Approach for End-User Centered Groupware Development

Published by Josef-Eul-Verlag, Köln, Lohmar, Germany, 2005.
Summary

New network technologies have changed the ways how people interact and collaborate over a distance. The understanding of such interaction is, however, still limited. This thesis discusses how new groupware applications can be built that support such networked groups. The main focus thereby is not on how to support application specific tasks – which is in many aspects comparable to single user support – but on the group specific aspects like group formation and group maintenance.

The approach presented in this thesis sheds light on the problem of groupware development by taking a closer look at theoretic approaches to design. Especially, the situatedness of design as it is propagated by the philosopher Martin Heidegger and the architect Christopher Alexander motivate that groupware development has to pay special attention to the group's situation. Empowering the end-users to express their needs in a specific group situation is therefore crucial to any groupware development.

The Oregon Software Development Process presented in this thesis reflects these ideas. It fosters the communication and interaction between developers and end-users during all phases of software development and ensures that end-user involvement and end-user tailorable is achieved.

Groupware patterns serve as an educational means for empowering end-users to behave like groupware development experts. They describe how to design social interaction in groupware systems as well as technical aspects of groupware systems. A selection of groupware patterns is presented in this thesis.

The approach is validated through analyzing its impact in different case studies: the first case is a two year development project of the collaborative learning system CURE involving six developers and a large user community; the second case describes the application in three smaller student projects.

About the Author

Till Schümmer, born 1973 in Frankfurt/Main, Germany, studied computer science at the Darmstadt University of Technology from 1994 to 1999 finishing with a diploma in computer science. From 1999 to 2002, he participated in the DFG supported PhD program "Infrastructures for Electronic Commerce" at the Darmstadt University of Technology. During his years in Darmstadt, Till collaborated with the CONCERT division of the Fraunhofer IPSI institute where he participated in the development of the COAST groupware framework and several applications for supporting group interaction in work settings and in e-commerce contexts. In 2002, he moved to the FernUniversität in Hagen, where he developed groupware applications that support distributed students in their learning process. His insights of 10 years of groupware development are captured in this thesis for which Till received a Dr. rer. nat. in computer science in July 2005.
Preface

Groupware is becoming increasingly important as a means for supporting collaborative work in distributed organizations. Although we find a plethora of groupware systems in the research domain, only few groupware application are widely used. One reason for this deficit may be the problem of matching the ever-changing end-users’ needs with the characteristics of a groupware system. Researchers propose several approaches to this problem: from customized development to end-user tailoring. However, no adequate approach has yet been demonstrated.

This book presents a solution that allows end-users to actively participate in groupware development. The resulting groupware meets the initial requirements of the users and can be tailored as requirements and practices will inevitably change during later use. The solution is rooted in both, agile software development processes and pattern languages. Based on an in-depth review of existing theories on design and problem solving, an approach consisting of two components is constructed: Firstly, an adapted agile software development process, called the Oregon process, is presented, which combines end-user participation with rapid, iterative software development. Secondly, a pattern language is presented, which enables end-users to understand and participate in the design of the groupware. Together, these components facilitate communication between end-users and developers, and enable end-users to drive the development process.

The approach is validated through a detailed case study. The case shows that the approach was successfully used in a 2 years larger groupware development project at a distance learning university.

The book provides the reader, who is interested in successful groupware development, with a description of the process and the pattern language. The case study and many examples help the reader to appropriate the method to other usage contexts.

Hagen, July 2005
Prof. Joerg M. Haake
Abstract

The development of groupware applications is still a difficult task. One main reason is that both developers and end-users are not aware of possible solutions for supporting group interaction. A second aspect is that group interaction involves many users, which makes the definition of requirements difficult.

Developers and end-users thus need to be supported in the requirements elicitation and the design of tools that help to satisfy these requirements. This thesis discusses groupware development processes from three perspectives: (1) the abstract view on development that is not bound to the specific nature of the developed artifact, (2) the software development perspective that focuses on processes and tools for the development of software, and (3) the groupware development perspective, which brings together software development aspects with social aspects and thus addresses the task of groupware development from a socio-technical view.

We first derive requirements from the abstract view on development that is mainly influenced by the theories of the architect Christopher Alexander. These theories suggest that end-users should be in control over the development process. To reach this level of control, they have to be educated regarding possible good practices – the patterns – for reshaping their environment. The goal is that end-users shape their environment in structure-preserving transformations that are modifications that help to increase the set of satisfied requirements by keeping existing structures intact or whole, as Alexander would call it.

These core requirements are then related to current software and groupware development processes. A special focus is put on evolutionary processes, agile methods, and participatory design approaches. We will show that these processes, methods and approaches solve specific aspects of the requirements, but that there is still the need for a process that supports groupware development in a way so that end-users can construct structure-preserving transformations.

To fill this gap, we propose a new development process, the Oregon Software Development Process. It fosters end-user participation, evolutionary growth, and reuse and exchange of design knowledge. This process supports development activities so that the different steps in the development lead to structure preserving transformations. This process advocates the use of end-user centered patterns as the most important artifacts during the development of socio-technical systems.

To illustrate the application of the process we first present a small set of patterns from a pattern language for group formation based on enhanced group awareness. It addresses a small subset of problems in groupware development (the group for-
mation phase). The application of these and other patterns from a larger groupware pattern language is demonstrated for the development of the collaborative learning environment CURE. We will discuss the different levels of participation and the means for educating end-users in the design of collaborative systems. Experiences from this project show how the iterative application of groupware patterns leads to a coherent design of a collaborative application through a sequence of structure preserving transformations. The educative support of the patterns is further investigated by reporting on student projects in which students that were not familiar with groupware development designed systems for supporting different types of groups.

Finally, the main contributions of this thesis are summarized. They are

– the introduction of the Oregon Software Development Process that fosters end-user participation,

– a new format for design patterns that is especially focussed on end-users,

– a pattern language for group formation based on enhanced group awareness and the presentation of selected patterns in an elaborated format,

– a collection of thumbnails of additional patterns from a pattern language for groupware development,

– a case study of applying the process and the patterns in a participatory design process of the collaborative learning platform CURE, and

– the evaluation of pattern languages regarding the role of structure preserving transformation in the context of the CURE project.
Zusammenfassung


Es bedarf deshalb eines Ansatzes, der Entwickler und Anwender beim Auffinden der Anforderungen und bei der Entwicklung von Werkzeugen, die diese Anforderungen erfüllen, unterstützt. Im Rahmen dieser Dissertation werden als Reaktion auf diese Erkenntnis Entwicklungsprozesse für kooperative Anwendungen aus drei verschiedenen Perspektiven betrachtet:

1. Die abstrakte Sicht auf Entwicklungsprozesse, die nicht an eine bestimmte Art von zu entwickelnden Systemen gebunden ist. Hierunter fallen theoretische Überlegungen zum Verständnis von Entwurfsprozessen, wie sie in den Philosophien von Martin Heidegger (Situiertheit allen Handelns und die Theorie des Bruchs), Jan Christian Smuts (Holismus), Donald Schön (Reflexion) oder Christopher Alexander (Strukturerhaltende Transformationen unter Zuhilfenahme von Entwurfsmustern) zu finden sind.

Die abstrakte Sicht dient als Basis für die Definition von Anforderungen für ein ideales Vorgehensmodell für die Entwicklung kooperativer Systeme. Hierbei liegt ein besonderer Fokus auf den Theorien von Christopher Alexander. Dieser legte nahe, dass der Benutzer stets die Kontrolle über den Entwicklungsprozess behalten solle. Da der Benutzer seinen Kontext und seine Wünsche am besten
kennt, diese jedoch oft nur schwer kommunizieren kann, sollte der Benutzer in
die Lage versetzt werden, seine Umgebung selbst zu gestalten. Für Alexander,
einem Architekten, lag das Augenmerk dabei zunächst auf der Gestaltung von
Gebäuden. Die neuesten Theorien Alexanders schlagen jedoch eine Ausweitung
auch auf andere Gestaltungsfelder vor.

Damit der Benutzer die Kontrolle über seine Umgebung behalten kann, ist es
erforderlich, dass er mit kognitiven Werkzeugen ausgerüstet wird, die es ihm
erlauben, gut durchdachte Entwurfsentscheidungen zu treffen. Entwurfs muster
sind ein solches Werkzeug. Sie bilden den Benutzer, indem sie wiederkehrende
Probleme und deren Lösungen so darstellen, dass sowohl die Lösung als auch
die Begründung für die Lösung einsichtig wird.

Alexander zeigt weiter, dass einige Entwurfs muster Transformationen des Kon-
texts des Benutzers beschreiben, die den Charakter der Umgebung intakt las-
sen. Wichtige Merkmale, die für das bisherige Funktionieren der Umgebung
verantwortlich waren, dürfen nicht verletzt werden. Aspekte der Umgebung,
die bisher nicht perfekt funktionierten, müssen hingegen verbessert werden.
Sofern Entwurfs muster dies gewährleisten, stellen Sie für Alexander struktur-
erhaltende Transformationen dar.

Da solche Transformationen die Ganzheit des Systems stets stärken, sollte
es das Ziel eines Entwicklungsprozesses sein, genau solche Transformationen
hervorzubringen.

2. Die Theorie der Software-Entwicklungsprozesse. Diese Sicht nimmt vor allem
Prozesse und Werkzeuge für die Entwicklung von Computersystemen in den
Blick. Zentrale betrachtete Modelle sind

- sequenzielle Modelle (z.B. das Wasserfall-Modell),
- iterative Modelle (z.B. der Unified Process), in denen die Entwicklung des
  Systems in mehreren Iterationsschritten durchlaufen wird, bei denen jede
  Iteration eine zusätzliche Annäherung an die Ziele der Benutzer bringt,
- agile Methoden (z.B. eXtreme Programming), die auf extrem kurze Ite-
  rationen und eine vermehrte Kommunikation zwischen Benutzern und
  Entwicklern ausgerichtet sind und
- partizipative Vorgehensmodelle, bei denen die Interaktion mit dem Be-
  nutzer und seine Einbindung in den Prozess im Vordergrund stehen.

Ergänzt werden diese drei Sichten durch die Betrachtung von Software-Ent-
wurfsmustern, die bei der Softwareentwicklung eine wichtige didaktische Rolle
spielen.

In der Arbeit zeigen wir, dass die Modelle alle jeweils nur Teile aspekte der An-
forderungen erfüllen. So haben sequenzielle, iterative und agile Modelle einen
Fokus auf der Unterstützung der technischen Aufgaben im Lebenszyklus eines
Entwicklungsprojektes. Partizipative Vorgehensmodelle und Entwurfs muster
legen hingegen besonderen Wert auf die Einbeziehung des Benutzers.
Entwurfsmuster unterstützen den Benutzer bei der Erarbeitung und der Er- schließung von Lösungskonzepten und spielen daher eine wichtige didaktische Rolle.

3. Vorgehensmodelle für die Entwicklung kooperativer Systeme bilden die dritte Sicht. Hierbei liegt der Fokus auf den Anpassungen, die vorgenommen wurden, um der Komplexität der Entwicklung kooperativer Anwendungen im Vorgehensmodell Rechnung zu tragen. Dabei untersuchen wir eine soziotechnische Sicht auf den Entwicklungsprozess, die sowohl die Interaktion zwischen Benutzern als auch die Gestaltung der Technologie im Blickfeld hat.

Auch bei den Modellen für die Entwicklung kooperativer Systeme gibt es keinen Vertreter, der sowohl die Aufgaben im Lebenszyklus des Entwicklungsprozesses als auch die didaktischen Anforderungen hinsichtlich der Kompetenzenerweiterung der Benutzer erfüllt.

Um diese Lücke zu füllen, stellen wir in dieser Arbeit einen neuen Entwicklungsprozess vor: den Oregon Software Development Process (OSDP). Dieser legt Wert auf

– eine möglichst umfassende Einbindung des Benutzers, bei der der Benutzer die Rolle eines gebildeten Designers und Entwicklers spielen kann,
– das evolutionäre Wachstum des Systems unter Berücksichtigung von strukturierenden Transformationen, und
– den Austausch von Wissen und Erfahrungen sowohl zwischen Entwicklern als auch zwischen Benutzern.

Eine wichtige Rolle spielen im OSDP Entwurfsmuster, die in einer sowohl für den Entwickler als auch für den Benutzer verständlichen Sprache verfasst sind.


ZUSAMMENFASSUNG


Abschließend können die Hauptbeiträge dieser Arbeit wie folgt zusammengefasst werden:

– Die Arbeit schlägt den *Oregon Software Development Process* vor, in dem vor allem Benutzerbeteiligung unterstützt wird,

– sie definiert ein *Format für Entwurfsmuster*, das vorrangig für die Nutzung durch Endbenutzer (also Laien auf dem Gebiet der Softwartechnik) ausgelegt ist,

– sie stellt eine Sammlung von *Entwurfsmustern zur Unterstützung der Gruppenbildung mittels verbesserter gegenseitiger Wahrnehmung der Benutzer* vor und beschreibt ausgewählte Entwurfsmuster in ihrer vollen Länge.

– Die vollständigen Entwurfsmuster werden durch eine *Sammlung von Kurzfassungen von Entwurfsmustern für die Entwicklung kooperativer Systeme* ergänzt.

– Eine Fallstudie zeigt den Einsatz der Entwurfsmuster an einem konkreten Beispiel, der kooperativen Lernplattform CURE.

– In der Fallstudie wird schließlich deutlich, wie strukturerhaltende Transformationen bei der Entwicklung eines kooperativen Systems hilfreich sein können.
Acknowledgements

This thesis is the result of many years of research in the context of groupware systems. Many people accompanied me on my path of research. Now it is time for me to express my gratitude for the numerous interactions that helped me to clarify my thoughts presented in this thesis.

Thanks are due to my advisor, Jörg M. Haake. He employed me 1995 as a research student in the CONCERT research division of the Fraunhofer IPSI. Since then he was the most constant companion and followed my research for the last 10 years until now. His constructive criticism was challenging for improving this work.

The employment at the CONCERT division opened the opportunity for me to collaborate in an ambitious research team and contribute to the groupware development framework COAST. As member of the COAST development team together with Christian Schuckmann, my brother Jan Schümer, Hans Scholz, and Holger Kleinsorgen, I could experience the new directions for groupware development and gained insights into many technical issues that had to be considered when building collaborative applications.

In 1999, I first met Alejandro Fernández who moved from Argentina to Germany to work at the CONCERT division. He was the person who introduced me to the early work of Christopher Alexander. This encounter was pathbreaking for my future research. Other colleagues who accompanied these first steps in the direction of a pattern-oriented view on groupware development were Robert Slagter (from the Telematica Institute in the Netherlands) and Torsten Holmer who also worked at the CONCERT division. Robert triggered me to consider the aspects of tailoring and end-user involvement as important parts of groupware development. Torsten contributed his incredible enthusiasm for new ideas.

I appreciate the support from my current colleagues at the distributed systems group at the FernUniversität in Hagen. Stephan Lukosch jumped on the train of groupware patterns research and became the most important discussion partner regarding the development and application of groupware patterns. Thanks are due to Mohamed Bourimi, Anja Haake, and Britta Landgraf (in alphabetical order) who agreed to participate as developers in the CURE development project. Wolfram Schobert contributed to the development of the collaborative pattern editor CoPE. He burned the midnight oil when I had the idea of new functionality and wanted to demonstrate it on the other day (e.g. at the CHI2004 workshop on HCHI patterns).

Special thanks are due to my shepherds at various EuroPLoP conferences (in chronological order): António Rito Silva, Martine Devos, Kristian Eloff Sørensen,
Joseph Bergin, Andreas Rüping, Lise B. Hvatum, and Alan O’Callaghan. All of them provided invaluable feedback to the patterns presented in this thesis. Special thanks are due to Joseph Bergin (Pace University – New York) for accepting the invitation to participate in my PhD committee.

Above all, I have to thank my beloved wife Jutta and my two sons Noah and Liam. Jutta gave her invaluable mental support and helped to create an environment in which this work could be written. My sons had to miss their father for several months while I was writing the final version of this thesis – with the side effect that Noah (2 years) is now able to perfectly pronounce the German word *Doktorarbeit*.\textsuperscript{1}

To express my thanks to all of you and the innumerable researchers who challenged me in discussions, I decided to write this thesis in plural voice. Even though some contribution were only small bricks in the building of this thesis, they all brought me forward in my evolving my ideas of groupware development. Thank you!

\textsuperscript{1}German for PhD thesis.
Contents

Chapter 1  Introduction  .............................................. 1
  1.1  Motivation ....................................................... 1
  1.2  Problem Statement .............................................. 4
  1.3  Approach ......................................................... 5
  1.4  Organization of this thesis .................................... 5

Chapter 2  Theory of Development Activities  ..................... 7
  2.1  Heidegger – Understanding Situatedness ..................... 10
  2.2  Alexander – The Synthesis of Form ............................ 11
  2.3  Rittel and Webber – Wicked Problems ....................... 14
  2.4  Schön – Reflection in Action .................................. 15
  2.5  Alexander – Patterns in the Context of Design ............. 17
    2.5.1  Garfinkel – Ethnomethodology .......................... 19
  2.6  Alexander – The Oregon Experiment ......................... 21
  2.7  The Scandinavian Approach – Participatory Design ........ 24
  2.8  Alexander – The Nature of Order ............................ 25
  2.9  Takeuchi and Nonaka – The Rugby Approach ............... 31
  2.10 Groupware Development Requirements ........................ 33

Chapter 3  State of the Art ............................................ 39
  3.1  Software Development Processes ............................. 39
    3.1.1  Sequential Non-Iterative Development .................. 40
    3.1.2  Iterative Development Processes ........................ 45
    3.1.3  Agile Methods for Software Development ............... 49
    3.1.4  Participatory Design and End-User Development ....... 60
    3.1.5  Patterns in the Context of Software Development ..... 65
    3.1.6  Discussion of Software Development Processes ........ 76
  3.2  Groupware Development Processes ............................ 78
    3.2.1  A critical review of Dewan’s waterfall process model for group-
          ware development ........................................... 78
## Contents

### Chapter 3

3.2.2 Agile Processes for Groupware Development ........................................ 81
3.2.3 Application of Participatory Design Methods ........................................ 82
3.2.4 CSCW Patterns ...................................................................................... 85

3.3 Summary of the first part ........................................................................... 91

### Chapter 4

4.1 The Oregon Software Development Process .............................................. 95
4.1.1 Conceptual Iteration ............................................................................... 95
4.1.2 Development Iteration ........................................................................... 98
4.1.3 Tailoring Iteration .................................................................................. 101
4.1.4 The relation between the different iteration types .................................. 102
4.1.5 Requirements addressed by OSDP ......................................................... 102
4.1.6 Comparison to the State of the Art ......................................................... 105

4.2 The Pattern Representation ....................................................................... 106
4.2.1 Single Pattern Representation ................................................................ 106
4.2.2 Pattern Abstraction Scale ...................................................................... 112

### Chapter 5

5.1 Importance of Group Formation ................................................................. 113
5.2 A Pattern Language .................................................................................... 116
5.2.1 Overview ............................................................................................... 116
5.2.2 FROM SHARED DATA TO SHARED WORK ........................................ 120
5.2.3 LOCAL AWARENESS .......................................................................... 127
5.2.4 PRESENCE INDICATOR ...................................................................... 135
5.2.5 USER LIST .......................................................................................... 141
5.2.6 GAZE OVER THE SHOULDER ......................................................... 145
5.2.7 ELEPHANT’S BRAIN .......................................................................... 150
5.2.8 DAILY REPORT .................................................................................. 158
5.2.9 HELLO HELLO ................................................................................ 163

5.3 Summary ................................................................................................. 169

### Chapter 6

6.1 Observation Method ................................................................................. 171
6.2 Context of the CURE Project .................................................................... 174
6.3 Conceptual Iterations in CURE ................................................................. 175
6.3.1 Results from the Conceptual Iterations ................................................ 178

6.4 Development Iterations in CURE ............................................................. 179
6.4.1 A sequence shaping the collaboration environment ............................... 180
6.4.2 A sequence addressing group awareness ............................................. 185
CONTENTS

6.4.3 A sequence addressing privacy and group formation . . . . . . . . 192
6.4.4 Results from the Development Iterations . . . . . . . . . . . . 202
6.5 Tailoring Iterations in CURE . . . . . . . . . . . . . . . . . . . . . 205
  6.5.1 The Hello Hello pattern . . . . . . . . . . . . . . . . . . . . . 205
  6.5.2 The emergence of CSCL patterns during tailoring iterations . 206
  6.5.3 Results from Tailoring Iterations . . . . . . . . . . . . . . . 209
6.6 Student Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
  6.6.1 Results from the Student Projects . . . . . . . . . . . . . . . 213
6.7 Observations Made by Peer Reviewers and Shepherds . . . . . . . . 214
6.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 216

Chapter 7 Conclusions and Future Work 219
  7.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 219
  7.2 Main Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
  7.3 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222

Appendix A Additional Thumbnails 225

Appendix B The CoPE Environment 247
  B.1 Linking Patterns in a Pattern Language . . . . . . . . . . . . . . 247
  B.2 Example Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . 251

Appendix C Tailoring Mechanisms in CURE 253

Bibliography 259

Index 278
Chapter 1

Introduction

New network technologies have changed the ways how people interact and collaborate over a distance. By staying connected over a network, new ways of collaborative work could evolve. Instead of working face-to-face most of the time and rare communication with remote people, today many people collaborate with remote peers via the Internet. In the work domain, employees in distributed companies collaborate in distributed work groups, workers in distant parts of a virtual organization form dynamic ad-hoc teams for a step in the production process, and people participate in virtual communities to increase their professional capabilities, e.g. in communities of practice (e.g., Wenger (1998)). This development is also visible in private life, where users participate in use communities of interest and communities of purpose to make their life easier or more interesting.

This thesis discusses how new groupware applications can be built that support such networked groups. The main focus thereby is not on how to support application specific tasks – which is in many aspects comparable to single user support – but on the group specific aspects like group formation and group maintenance.

1.1 Motivation

Consider as an introductive example the case of online communities. Preece (2000) defines online communities as a set of “people, who interact socially as they strive to satisfy their own needs or perform special roles, such as leading or moderating.” These people share interests or needs, which make up the purpose of the community. They use group policies (i.e., rules and assumptions) to guide social behavior and computer systems to mediate the interaction. An early example is the legendary WELL (Rheingold, 1998).

Sometimes the members also/only meet in person. In this case, the role of the technology is mainly reduced to bringing people together. Examples are flashmobs (e.g., the Birmingham flashmob in 2003 – cf. (Revell, 2004)) where large groups of people gather at a predefined time in public spaces to have fun and perform objectively meaningless actions.

An increasing number of community providers has realized that virtual com-
munities can become very binding, if they reach the phase of social involvement. Rovai (2002), further defines four characteristics of online communities: spirit, commonality of expectation and goals, trust and interaction. These characteristics are common to most group processes. The community needs a spirit that helps members to distinguish themselves from other people. The spirit is often supported by a shared set of goals. Community members have to establish trust in order to interact freely. Interaction in turn helps to grow trust.

There are different classifications of communities. Most classifications distinguish communities of interest, communities of purpose, and communities of practice (cf. Marathe (1999), Carotenuto et al. (1999)).

Members of a community of interest share the same interests in a topic (and often a common background). Examples are discussion groups on television shows or people interested in planets of the solar system. Some authors (e.g. Carotenuto et al. (1999)) also define communities of passion, which are very close to communities of interest. The difference is that the members are more involved in the community’s topic up to the point where they become passionate advocates. Actually, a community of interest can become a community of passion. An example is a discussion group on a TV show that became a fan club of the show’s host.

Communities of purpose consist of members who share a common (short term) goal. For instance, customers at a virtual bookstore share the goal of finding and buying books. They all have to go through the same process (i.e. selecting the item and checking out) and they can help one another reaching the goal. Thus, the community of purpose has a functional purpose and it may dissolve after the goal is reached. In contrast to communities of interest, the members don’t necessarily share the same interests and therefore, they are not likely to start activities that exceed the community’s purpose (Carotenuto et al., 1999).

If the members of a community share a common profession, the community is called a community of practice. Their members reflect on the way how they perform their tasks and enhance their ways of working in a community learning process. Since the community’s topic is the member’s profession, members are normally highly involved in such communities. Concrete communities of practice are for instance Smalltalk programmers who meet in a Smalltalk users group to shape the process of programming. In some cases, it makes sense that the community is established by the merchants. Given the example of the Smalltalk users group, the main vendor has been heavily involved from the beginning. The well known experts are employees of the company, and their involvement in the online discussions is beneficial for both parties (customers and vendor): the customers can get help from the experts, and the vendor can steer the discussion regarding his business plans (carefully up to a specific degree; too much advertising would put off the customers).

Marathe (1999) adds another type of community: a community of circumstances, which is defined by common circumstances such as current life situations, e.g. children reaching puberty. Interaction in these communities is often personally focused and third parties are not involved in the community. Therefore, these communities may not easily be influenced, but they can lead to strong bindings to a seller, if this
seller is present in important life stages.

While technology that supports the above types of groups in their tasks (e.g., exchange of messages and documents) is widely available, fewer attention has been on the need of social group interaction, e.g. the need of users to find relevant peers or the need of users to establish trust. But this social group interaction is one of the most important issues in the design of successful groupware.

A famous definition of the term groupware defines groupware systems as “intentional group processes plus software to support them” (Johnson-Lenz and Johnson-Lenz, 1981). This definition includes different aspects that we have to consider when designing groupware solutions:

- The core of the definition is the group. A group of users wants to interact using groupware. Naturally, the members of this group should play an important role during the design of groupware. The groupware design has the purpose to create a solution, which satisfies the user’s needs. Thus, end-user requirements have to be the central issue during groupware design.

- The group interacts in an intentional group process. The interaction between humans thus needs to play an important role during the design of any groupware solution. It has to become clear who interacts with whom. It has to be considered how strict the intentional group process is, ranging from unplanned interaction in virtual communities up to formally structured workflows in a distributed workgroup.

- The process is supported by software. The fact that software is mentioned at a third place here emphasizes that the software itself should be a supportive means to ease the interaction between humans. The software should be adapted to the users’ needs to best fulfill its supportive role. At this point the software developer comes into play. As the software supports the group process, the software developer should support the users in adapting or creating a software that fits the process.

Compared to a focus on design that has the goal of supporting the group in the manipulation of content, support for social group interaction needs a larger focus on the relations between users. Tools for manipulation are in most cases used by one user (even in collaborative systems). So, they affect the relation between the user and the shared artifacts. Social interaction, on the other hand, affects the relation between users and needs to address issues like trust and privacy. The focus should thus be on human-computer-human interaction (HCHI), while the design of tools for the manipulation of artifacts mainly affects human-computer interaction. The design of tools therefore focuses on the interaction of the user with the artifact and considers the human-human interaction as a marginal aspect.

To provide customized designs of groupware mechanisms, we have to make use of a design process that is flexible enough to adapt to the group’s needs. Experiences with the design of single user applications have already shown that many software development projects fail because of requirements inadequacies (Dorfman, 1997).
such cases, the customer is typically involved in the early stages of the project as a source of design requirements. This set of requirements is then implemented by the software developers and subsequently the customer assesses the result. However, if the requirements were not specified correctly, the customer receives a product that does not match her needs.

Unfortunately, these socio-technical requirements are often less clear to all stakeholders involved in the development of groupware applications. Two reasons make this part of groupware development difficult:

- While in single user tasks (like word processing or image editing) only one actor interacts with an artifact, groupware needs to support the interaction of many users with each other. The reacting interaction partner is thus not a technical deterministic artifact but a non-deterministic human user.

- The users are not as used to these new opportunities for interaction compared to the use of single user applications.

The theory of socio-technical design treats a community from two perspectives: the social system including group processes, roles, and flow of information and the technical system that includes tools used within the community like IT infrastructure or buildings. In a socio-technical perspective these two systems are highly interrelated. A socio-technical perspective on groupware design has to be aware of three key aspects (Bikson and Eveland, 1996):

- It is difficult to predict the reciprocal effect of changes to either the social or the technical system.

- The process used to create the socio-technical system will affect the acceptance of the system.

- Both social and technical systems change over time.

This leads us to the problem statement of this thesis.

1.2 Problem Statement

The development of groupware applications is still a challenging task for both users and developers. One big problem lies in the appropriateness of requirements and the unfamiliarity of users with tools for group support. Especially, aspects for supporting social interaction between group members are often still technology driven and provide a low appropriation to the group needs.

In order to improve this situation, this thesis aims at the development of a process and tools that help the developers and the end-users in finding and implementing group-specific groupware solutions, which fit the needs of the user groups’ social interaction. While the development process should be considered from a holistic
groupware perspective, it should be applied to problems arising in early phases of collaboration (i.e., group formation and socializing).

1.3 Approach

The approach of this thesis is to empower the end-users and developers to play the role of an informed designers in the development process of the groupware system. This is achieved by two means: a groupware development process that structures the development activities and groupware patterns that help to communicate design knowledge.

The proposed development process – the Oregon Software Development Process – fosters the communication and interaction between developers and end-users during all phases of software development and ensures that end-user involvement and end-user tailorable is achieved. The proposed pattern language consists of high-level patterns for designing social interaction in groupware systems and complements low-level pattern collections that focus more on technical aspects of groupware development. The patterns are used to empower end-users to participate in the development process and make the often implicit requirements and constraints of solutions explicit.

The approach is validated through analyzing its impact in different case studies: the first case is a two year development project of the collaborative learning system CURE involving six developers and a large user community; the second case describes the application in three smaller student projects.

1.4 Organization of this thesis

In chapter 2, we analyze the problem of groupware development from a broader view and derive requirements for an ideal development process that focuses on end-user participation and the education of end-users and developers. A special focus will be on the design theories of the architect Christopher Alexander.

Chapter 3 provides an overview of the state of the art. This covers three different areas: Process models used in general purpose software engineering, process models used for groupware development, and the state of current pattern approaches in the areas of software development, human computer interaction, and groupware design. We will show how current approaches address the requirements identified in chapter 2 and show that a new approach is needed to fill the gaps of each model.

Chapter 4 presents the approach. First, the process model is explained and related to the underlying models found in the state of the art. Since the process model relies on the use of patterns that are readable for end-users and groupware developers, a new pattern format is needed. Thus, we discuss how a pattern has to be structured to serve both means. Since a full pattern catalogue for groupware systems would go beyond of the scope of this thesis, chapter 5 presents only a
small excerpt of a pattern language for groupware development focusing on group formation through group awareness.

In chapter 6, we report on the application of the process and the patterns in the context of two different case studies: The first case is a two-year project performed by a group of software developers and a large user community. It applied both the process and the patterns in order to develop a collaborative learning environment. The other case involves three student projects that mainly applied the patterns in different contexts.

Finally, chapter 7 summarizes the main contributions made by this thesis, briefly compares them to the state of the art, and suggests possible directions for future research.
Chapter 2

Theory of Development Activities

The development of groupware provides a special challenge to requirements analysis: the developed artifact is both, a technical support system and a social process.

In this chapter, we will identify requirements for the development process of technical artifacts as well as social protocols. To understand the requirements it is important to first understand the process of development at a general level.

We define development as the modification of an artifact or a process with the goal to reshape the context so that a specific action can be performed (more easily) in the resulting context. According to Merriam-Webster (2000), we interpret the term artifact as “something created by humans usually for a practical purpose” and the term context as “the interrelated conditions in which something exists or occurs”.

Three questions are important for development.\(^1\)

1. **What** is the direct goal (or requirement) that is reached by the modification.
2. **Why** is this goal relevant (i.e., what is the rationale behind the modification).
3. **How** is this goal reached (i.e., how does the solution modify the context).

![Figure 2.1: Different levels of abstraction in development.](image)

The different questions address the problems of development on different levels of abstraction. Refer to fig. 2.1 for a concrete example where the requirement of the user (the *what*) is the wish to start the engine of a car. This requirement became

\(^1\)Note that Cockburn (2000) uses the same questions and abstraction levels in the context of use-case authoring.
relevant in a larger problem context (namely the need to drive a car) that can be determined by asking, why the user wants to start the engine. The solution or the concrete design that satisfies the requirement is to heat the glow plug (as one step necessary to start the engine). In most cases, the granularity of the described actions becomes smaller, the lower the level of abstraction is (although we will argue together with Alexander that it will not result in a tree structure with decreasing sizes of requirements – cf. section 2.2).

On which level a specific question needs to be asked depends on the focus of the current development. If the user would for instance state the user requirement of driving a car, the answer to the why question would be that the user wants to travel from Frankfurt to Hagen. One part of the solution (the how question) would be to start the engine.

The task of a development activity is to move from higher abstraction levels to lower levels of abstraction and map different activities like requirements elicitation, artifact design, and artifact implementation to these levels of abstraction.

The field of requirements engineering (Nuseibeh and Easterbrook, 2000) addresses these questions both from a user and from a developer perspective. The goal of requirements engineering is to identify people with an interest in the design (the stakeholders) and their personal needs. These needs have to be documented and communicated between all stakeholders (and the developers) and finally a solution has to be designed and implemented that satisfies the needs. The challenge is to understand and discover the needs and to find a solution that satisfies all considered stakeholders.

In other disciplines like architecture or industrial design, these questions are often subsumed under the problem of designing the artifact (e.g., Alexander, 1979). This leads us to a problem in terminology: While in the development of software systems design is often considered as the process of creating software structures that resolve a specific set of requirements (e.g. creating an object-oriented model), the other disciplines have a wider conception of design that includes also the analysis of requirements and the implementation.

We will use the wider definition at this stage of this thesis and speak of design as “a matter of making” Fallman (2003). Designers create or change the structure of the designed artifact in order to reach a design goal. Fallman identified three different accounts of design: the conservative account which is influenced by a mechanistic engineering approach, the pragmatic account that situates itself in the fields of social sciences, and the romantic account that considers design as a matter of art. The different approaches to design imply different processes and result in different qualities of the designed artifact. Table 2.1 compares the different accounts of design.

Traditional groupware development has often addressed the problem from a conservative account leading to a technology driven solution. In times where the technical problems of data distribution, network communication, or consistency management were a open issues, such an approach to groupware development was acceptable. Since the problems could be formulated as technical requirements, the development process could successfully address the problems with an engineering
<table>
<thead>
<tr>
<th>Conservative Account</th>
<th>Pragmatic Account</th>
<th>Romantic Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>An analytic information processor who looks at requirements that are all visible.</td>
<td>A reflective person who looks at a system as participant or external observer.</td>
</tr>
<tr>
<td>Problem</td>
<td>Not well defined or structured. It will be defined during design.</td>
<td>Bound to the situation. Experienced and set by the designer.</td>
</tr>
<tr>
<td>Product</td>
<td>Result of the process.</td>
<td>Outcome of the dialogue between the designer, the user, and the world. Situated in the world.</td>
</tr>
<tr>
<td>Process</td>
<td>A rational search process; fully transparent.</td>
<td>A reflective conversation.</td>
</tr>
<tr>
<td>Knowledge</td>
<td>Guidelines; design methods; scientific laws.</td>
<td>Experience that suggests how each problem should be tackled.</td>
</tr>
<tr>
<td>Role model</td>
<td>Natural sciences; engineering; optimization theory.</td>
<td>Human sciences and sociology.</td>
</tr>
</tbody>
</table>

Table 2.1: Accounts of design (in part based on Fallman, 2003). This thesis focuses on a pragmatic account of design.

But nowadays where these technical problems are solved to a large extent, the socio-technical problems become more important. The development of social processes however requires another perspective on design that takes the non-determinism of human behavior into account.

We will thus put a focus again on the roots of current design methodologies and the underlying conceptions. The main focus lies on models that follow a pragmatic approach (this is the reason why the column of the pragmatic account appeared in bold font in table 2.1. We will show that reflection and situatedness play an important role in identifying requirements and that the process needs to take place in a dialogic way that involves developers as well as end-users who communicate the different requirements. We will also provide arguments for an evolutionary understanding of requirements and a continuous adaptation of the design. This
results in the need of a process in which requirements engineering as well as design and implementation are continuous activities that go hand in hand during the whole development of the desired system.\textsuperscript{2}

One important contribution to the evolution of design processes is the doctoral thesis of Stahl (1993). In his thesis, he related the theories of design of different schools (Alexander, Schönb, and Rittel) and related them to the often cited Heideggerian understanding of situatedness and breakdown. The following sections will partially follow Stahl's analysis by focussing on the same schools of design. His views will be extended by more recent theories of Alexander.

\section{2.1 Heidegger – Understanding Situatedness}

In his main work \textit{Sein und Zeit}, Heidegger (1927) unfolds a theory of understanding being. He proposes the theory of situatedness as a central approach for understanding being in general. \textit{Situatedness} (In-der-Welt-sein) describes a situation where the individual cannot understand his being without taking the current context, the situation, into account. Every being is always situated, and so is every action.

Heidegger makes this clear in his discussion of tool use (ibid., p. 66ff.): instead of focussing on the tool, the user of the tool focuses on its determination in the specific context (ibid., p. 69). In most cases, the perception of the tool is secondary; the knowledge about the tool and its use is \textit{tacit knowledge}. The user approaches the tool with a specific pre-understanding and uses the tool according to this understanding.

The tacit knowledge about the tool – or as Heidegger calls it in a more general view the \textit{ready-to-hand} (\textquotedblleft Zuhandenen\textquotedblright) – becomes explicit when a user’s pre-understanding does no longer match the perceived situation (ibid., p. 75). Heidegger refers to this point in time as the \textit{break} (\textquotedblleft Bruch\textquotedblright). In English, it is often called the \textit{breakdown}. Conditions in which a break occurs are:

\begin{itemize}
  \item the lack of a ready-to-hand in a situation where this ready-to-hand would have been used and its existence was taken for granted before,
\end{itemize}

\textsuperscript{2}Nuseibeh and Easterbrook (2000) argue that requirements engineering should span the whole life cycle of a software system. The need for design and implementation as activities that take place in parallel throughout the process may contradict traditional approaches to software development at a first glance. The reason why we argue for a parallel view is that the three views actually provide access to the problems that need to be solved at different levels of abstraction. Instead of forcing the designer to stay at one level of abstraction and provide a complete solution at this level before he moves to a lower level, we argue that the designer needs to flexibly move between the different levels to fully understand each level. This means that design, for instance, cannot be fully understood without creating a (partial) implementation. However, the implementation informs the design again and thus has implications on the different design decisions (cf. Reeves (1992) who even argued that developers have not designed anything until they have provided an implementation and thus should focus on all different levels of design including coding itself and choose that level that is appropriate for each situation). The same is true for analysis: we will present the theory of wicked problems that argues that each change in the design will probably affect the requirements again (cf. section 2.3).
2.2. ALEXANDER – THE SYNTHESIS OF FORM

- the unsuitable behavior of the ready-to-hand so that it does no longer serve its purpose, or
- the existence of another object that hinders the proper use of the ready-to-hand.

The ready-to-hand becomes obtrusive and existent. It is this context in which one can understand the essence of the ready-to-hand and the situation in which one is. The reflection happening in breaks leads to an understanding of the relations between an artifact, its user and the context.

Although the focus of Heidegger is primarily ontological, the conception of situatedness and breaks provide important insights to the theory of design. As Stahl (1993, p. 147) pointed out, the understanding of requirements requires a situatedness of the actors. In the situation, the user has full access to all dependencies that currently constitute the context. Thus, to make a design decision designers need to be physically or in thought situated and the situation needs to be as close as possible to the situation of the anticipated users.

2.2 Alexander – The Synthesis of Form

The early works of the architect Christopher Alexander, especially his Notes on the Synthesis of Form (Alexander, 1964), were driven by the goal of combining mathematic theory with processes needed for designing buildings (or any general form).

Alexander distinguishes between the form as the part of the environment which the designer is able to shape and the context in which the form is going to be embedded. The problem of design is to “put the context and the form into effortless contact or in frictionless coexistence.” (ibid., p.19) The interaction between the shape and its context is defined by a set of forces. One can distinguish between conflicting forces (different needs of the context and the shape) and enforcing forces (needs that are aligned).

Fitness is achieved if the conflicting forces are minimized and enforcing forces are maximized. The problem that Alexander sees is that we are not able to name all forces. And even if we were able to enumerate all forces, this list would still be endless and the interaction of the forces would still require a field description in which the relations between all forces could be expressed (ibid., p. 24).

To overcome the problem of not being able to state all forces, Alexander proposed to “think of the requirements from a negative point of view, as potential misfits, [because then] there is a simple way of picking a finite set [of forces]. This is because it is through misfit that the problem originally brings itself to our attention.” (ibid., p.26). From this quote, it becomes obvious that Alexander understands the process of understanding needs in an Heideggerian way. Again, the role of a good design is to reduce negative conflicting forces.

But even if only misfit forces are considered, these forces are still too complex to be addressed all at once. The goal of Alexander thus is to
“picture the process of form-making as the action of a series of subsystems, all interlinked, yet sufficiently free of one another to adjust independently in a feasible amount of time. It works, because the cycles of correction and re-correction, which occur during adaptation, are restricted to one subsystem at a time.” (ibid., p. 43)

This implies an iterative process with small iterations that focus on locality (positions in the problem space that have a relatively low connectivity to other parts of the problem space). The process of finding these subsystems is defined by Alexander as a simple algorithm. Assume that all misfit variables $x_1, \ldots, x_m \in M$ are arranged as nodes in a graph. If two nodes of the graph are connected by an edge, the two requirements represented by the nodes are affected if one of them is changed. The process of decomposition starts with finding subsystems that have many connections within the system and only few connections to other subsystems. These subsystems should be roughly of the same size or scope. They are then addressed as a single requirement and the clustering is repeated until the whole set of requirements is organized as a tree.

The division of the set $M$ into relatively independent clusters takes place during analysis, while the combination of subsystems to larger systems is a step of synthesis. In the late 1960s, there were attempts to automate this process by software systems. One example is the CLUSTR system (Milne, 1971) that took a set of requirements and a connectivity between the requirements as an input and calculated a tree structure that represents a valid decomposition of the problem. The main problem why these approaches cannot work is that in complex contexts the requirements will not be complete and are not independent from changes in the design. Alexander already mentioned this as one of the main problems of his approach (that was the reason for focussing only on misfits, but this shifted focus only gradually reduced the problem of uncertainty).

Until today, many development processes ignored two essential parts of Alexander’s approach:

1. In his essay *a city is not a tree*, Alexander (1965) made clear that the structure that evolves while clustering the interacting misfits will not lead to a tree structure. It is instead a semilattice. The difference is that in a semilattice the different nodes of one level can interact with different nodes at a higher level instead with only one parent in a tree structure.

   Alexander (ibid.) provided an example of a city that was composed as a tree (cf. fig. 2.2). As it can be seen in the map, one central axis serves as the central bus. A main artery (as Alexander called it) connects the bus with urban areas. Each area finally is reached by subsidiary arteries. The problem with such a structure is that exchange between different regions is only possible via main axes. Only very few requirements connecting each smaller part with the next larger one were considered and exchange or *life* as Alexander calls it is hardly possible in such structures.

   A more abstract example for this problem is shown in fig. 2.3. Six requirements are partitioned according to their relatedness. Requirements $\{1, 2\}$ are handled
2.2. ALEXANDER – THE SYNTHESIS OF FORM

Figure 2.2: The city of Brazilia (left) is composed as a tree (abstract version of a figure in (Alexander, 1965)).

Figure 2.3: Decomposition using a tree structure (example from (Alexander, 1965)).

together and requirements \{3, 4, 5\} are also addressed in an isolated subsystem. The result of the latter set of requirements is combined with requirement 6 and finally all subsystems are integrated. The advantage of this approach is that the interfaces between each pair of subsystems are very small. This makes composition easy. But this partition of requirements creates a solution that ignores many relations between the different subsystems (e.g. the relation between \{3, 4, 5\} and \{1\}. Exchange between the different subsystems is thus not well supported.

If we compare the tree structure with the semilattice (as shown in fig. 2.4), we can see that the exchange between subsystems occurs to a larger degree. The structures overlap and thus reduce the frictions between the different subsystems.

2. Alexander did not intend that all requirements could be named à priori.

The latter issue has been further discussed by Rittel and Webber, who coined the term of wicked problems, which will be discussed in the next section.
2.3 Rittel and Webber – Wicked Problems

Rittel and Webber (1973) termed the notion of wicked problems in design. Their basic assumption is that in such problems the requirements change whenever a design step is performed. Since the design action itself changes the context of a problem, it is very likely that the context needs to be reanalyzed before applying the next design step. During this analysis step, a new requirement may evolve, which can be stronger than the strongest requirement seen in the previous analysis phase.

Figure 2.5 shows this relation: If we consider an initial design in place, this design interacts (in the notion of Heidegger) with its context. It is always situated and the relations to the context can be analyzed and prioritized (shown as arrows in fig. 2.5). The prioritization demands that a new part of the design is put into place (the black circle in fig. 2.5). But since this new design interacts with its context, it will also change the context of the first design. Thus, the prioritization made with only the first design in place will no longer be valid after adding the second design.

The problem definition in this sense co-evolves with the design. And each design step will bring up new problems that need to be addressed. In theory, this would lead to an infinite set of design iterations and a monotonically increasing set of
relations of design with its context. For practical reasons, the cycles stop when a specific quality threshold is reached.

Another important property of a wicked problem is that each instance of this problem is unique: since there cannot be two identical initial contexts (Heraclitus already coined this insight with his quote παντα ρει – Everything flows, nothing stands still), the initial set of relations is always unique and thus no sequence of design steps will be identical to other sequences (although different sequences may be comparable).

Malotaux (2001) hit the point by identifying two requirement paradoxes:

1. Requirements should not change for reliable results, however the requirements do always change.

2. Developers don’t want requirements to change, however, since the risk of changing requirements is known, developers try to provoke requirements change in early phases of the project.

Fitzpatrick (1998) provided arguments why groupware design is a wicked problem. She states that

The problem of how to understand what cooperative work means and how to design systems to support that work is an inherently complex one that defies simple definitions and solutions (ibid., p. 13).

The author is especially concerned about uncertainty regarding common solutions in the research community of CSCW and the variety of different experiences drawn from workplace studies. It is, according to Fitzpatrick, important to relate knowledge from an abstract level with knowledge from concrete instances.

Fitzpatrick criticizes that groupware design has so far often focussed on supporting one specific interaction (e.g. document co-authoring or decision making). These solutions, however, do not match with the wicked nature of group interaction. Instead, groupware design should focus on situated solutions that take an evolving group context into account. Further on, Fitzpatrick argues that action-based approaches that try to capture human interaction in pre-defined workflows are unsuitable. Again the reason lies in the wicked nature of group interaction: when trying to predefine a workflow, the context in which this workflow is applied (including all stakeholders current needs) may not change at runtime. When it changes, users will be confronted with a break as Heidegger defined it.

2.4 Schön – Reflection in Action

An often used perspective on groupware design is influenced by the work of Schön (1983) who examined how professional users deal with break situations as defined by Heidegger. According to Schön, practitioners act using their implicit knowledge of how to perform an action in response to challenges imposed by a specific context. Activities of this kind were referred to as knowing-in-action. Whenever the results
of the action do not map to the intended results (i.e. the context does not behave as anticipated), the practitioner lives through a breakdown (note that Schön uses a different term than Heidegger who uses the term break). Intuitive actions are now subject to reflection. The practitioner stops the action and to understand or reflect about

(a) what he was trying to do (and thus makes the implicit knowledge about activities explicit), and

(b) what the forces are that prevent his intuitive actions to succeed. It is important that these forces are experienced at the time when the breakdown occurs since this supports a better understanding of the context that cannot be easily anticipated in an unrelated design session.

Reflection opens new directions for action. In the context of tool design, this may mean a new way of using the tool or a way of updating or tailoring the tool. The knowledge about the breakdown, its reasons (obtained through reflection), and its solution is kept as implicit knowledge by the practitioner.

![Figure 2.6: The Plan-Do-Study-Act cycle.](image)

The Plan-Do-Study-Act cycle (PDSA – fig. 2.6) attributed by Deming to Shewhart but fully elaborated by Deming (Deming, 1982, p. 88) has a comparable goal: Users should reflect on break situations and invent a solution. If the solution was successful, it should be remembered as a best-practice procedure to enhance the quality of the design cycle. The steps are the following:

Planing combines three activities:

1. users analyze a current problem,
2. they think about a solution (a design) that solves the problem, and
3. they also think about a measure for assessing the success of the solution.

Doing is the step where the users implement the solution according to the plan.

Studying (or Checking, as Deming called it) means that the users apply the assessment measure. If the result is satisfying, they move on to the Acting phase. If not, they think about a new solution in the planning phase.
2.5. Alexander – Patterns in the Context of Design

Acting helps to capture the solution. If the previous steps led to the desired success, the solution is used to change the standard procedures for action. Users learn that the action was appropriate and add it to the canon of best practices.

Larman and Basili (2003) interpreted the PDSA cycle as the first occurrence of iterative development and design. Since each iteration only addresses exactly one problem, it creates a gradually improved catalogue of practices. Note that the development of software and especially of groupware applications is much closer to the application of PDSA to the manufacturing of an artifact since the created software is only built once and from then on copied without any additional effort.

2.5 Alexander – Patterns in the Context of Design

In *A city is not a tree*, Alexander (1965) clarified that decomposition always needs to investigate the context of the unit of densely connected requirements. In *The timeless way of building*, Alexander (1979) proposed a new way of addressing connected requirements that puts an emphasis on the context of the problem – the pattern. Early forms of patterns were formulated as a mathematical description of problem spaces introduced in Alexander’s *Notes on the Synthesis of Form*. The first reference can be found in (Alexander et al., 1968). In this publication, Alexander defined a pattern as a rule with two parts:

“the PATTERN statement itself, and a PROBLEM statement. The PATTERN statement is itself broken down into two further parts, an IF part, and a THEN part. A pattern reads like this: 

\[
\text{IF: } X \text{ THEN: } Z / \text{PROBLEM: } Y
\]

\(X\) defines a set of conditions.

\(Y\) defines some problem which is always likely to occur under the conditions \(X\).

\(Z\) defines some abstract spatial relation which needs to be present under the conditions \(X\), in order to solve the problem \(Y\).” (ibid., p. 15)

The patterns were organized in a semi-lattice structure (as discussed in section 2.2) so that different levels of scale in the process of analysis could be reached. All patterns were organized in a pattern map shown in figure 2.7.

Later on, Alexander moved away from a strictly mathematical understanding of patterns. In *A Timeless Way of Building*, he defined it as a morphological law that explains how to design an artifact in order to solve a problem in a specific context. Patterns have several properties that inform design:

- They make the problem explicit instead of just stating a solution. The reader of a pattern (ranging from lay persons to experts) is provided with a description of the problem that helps him to compare the implicitly perceived problem with other known problems.
They state the conflicting forces and explain how these forces change when applying the solution.

Alexander described the process of finding patterns in our environment:

"First define some physical feature of the place, which seems worth abstracting. ... Next, we must define the problem, or the field of forces which this pattern brings into balance. ... Finally, we must define the range of contexts where this system of forces exists and where this pattern of physical relationships will indeed actually bring it into balance." (Alexander, 1979, pp. 249, 251, and 252)

The resulting Alexandrian patterns as they were used in A Pattern Language (APL) (Alexander et al., 1977) are written in natural language and include the following elements:

Name: Patterns start with a name that describes the solution. Examples in APL are Quiet Backs, Indoor Sunlight, or Sitting Circle.
Sensitizing photography: Under the name follows a photography of an environment where the pattern is in place. The photography fulfills two purposes: (1) it gives the reader a first idea what the solution may look like, and (2) it helps the reader to better remember the pattern.

Context description: The context of a pattern is describing the space in which the pattern can be embedded. It often includes references to patterns on a higher level of scale that have been considered before (the incoming edges in fig. 2.7).

Problem statement: The problem statement marks the start of the pattern’s body. It is first expressed in a short bold-faced paragraph and continued with the discussion of forces.

Discussion of forces: The discussion of forces helps the reader to better understand the problem and the relation to the context. In the Alexandrian pattern format, the forces are discussed and resolved in the text between the problem statement and the solution statement.

Solution statement: The solution statement summarizes how the forces were resolved in the forces discussion. Again, it is a short bold-faced paragraph. Together with the problem statement, it forms a thumbnail of the pattern which already communicates the most important parts of the pattern. This allows readers to get a quick access to the pattern without reading too much. If the pattern looks relevant, a reader will read the forces section.

Diagram: In the context of architecture, patterns should, according to Alexander, be drawable. If the author is not able to draw a picture, then the described solution is no pattern. This extreme criteria for patterns is, however, not necessary in other problem domains like social interaction patterns where the solutions help to model dynamics.

Related patterns: The final part of the Alexandrian pattern format is the related patterns section. It lists other patterns that are intended to be considered together with the pattern. It often includes references to patterns on a lower level of scale (the outgoing edges in fig. 2.7).

APL includes 253 patterns on different levels of scale. The patterns are highly connected, which reflects the interdependence of the addressed forces. While the patterns in APLMSC could still be shown as an acyclic semi-lattice, the patterns of APL can no longer be arranged in an acyclic graph since they include 241 cycles (this may be the reason why Alexander no longer included overview diagrams).

2.5.1 Garfinkel – Ethnomethodology

Patterns play an important role in another field that is relevant to the field of design and especially the field of requirements analysis: Ethnomethodology (Garfinkel, 1967).
Ethnomethodology provides an approach on understanding the ways how people interpret their perception of the world. In short, the main idea is that all human interaction will be based on a categorization of phenomena to existing understandings of the world. If, e.g., one person communicates with another person, she tries to frame the utterances of her communication partner in the context of the conversation.

One illustrative example of Garfinkel (1967, p. 38) reports on the conversation of a couple and reveals how each utterance is subject to interpretation and how it is related to background knowledge:

\textit{Husband: Dana succeeded in putting a penny in a parking meter today without being picked up.}

To understand this utterance, one has to relate it to background knowledge that is shared between the communication partners:

- The couple has a son called Dana who is 4 years old.
- The husband picked him up from kindergarten.
- After picking him up, they stopped somewhere since the son filled the parking meter.
- Normally the son needs to be picked up, but today he reached the parking meter on his own.
- This means that the son has grown or perfected his motoric capabilities.

\textit{Wife: Did you take him to the record store?}

The wife has made the above interpretations and now wonders, where they stopped.

- She knew that her husband wanted to stop at the record store.
- The question is whether he was at the store before picking up his son or after that.
- If he was not stopping at the record store, the wife implicitly asks where else her husband and her son stopped.

In the context of design, these implicit assumptions have to be communicated between participants in the design process. But since one can never understand any communicative act without relating it to previous knowledge, there will be no complete record of assumptions. Therefore, the understanding of the requirements will stay ambiguous. Two constructs help to reduce the ambiguity: users should be as explicit as possible when reporting on requirements, and they should make implicit patterns of actions explicit.
This is the part where patterns in the Alexandrian view come into play: Each communication partner knows behavioral patterns that are followed by the other communication partner. In the above example, the wife knew that the son was always picked up to reach the parking meter. This was a practice that was common to both wife and husband. However, the story becomes interesting since the pattern is modified for the first time. The son no longer has to be picked up, which means that the wife can also change her repository of patterns.

Another implicit pattern is that the son normally gets picked up from kindergarten. This is not mentioned in the conversation, but it is a daily recurring activity that is common to both wife and husband. Patterns describe habits or rites that have been followed for a long time and that are common to a specific group or community. Making them explicit informs the design and the understanding of requirements.

2.6 Alexander – The Oregon Experiment

In 1980, Alexander explained how patterns should be applied and how they lead to a process of participatory design, the Oregon Experiment (Alexander et al., 1980). The foundation for this process lies in the philosophy of Alexander, which becomes clear in (Alexander, 1979):

“... people can shape buildings for themselves, and have done it for centuries, by using languages which I call pattern languages. A pattern language gives each person who uses it, the power to create an infinite variety of new and unique buildings, just as his ordinary language gives him the power to create an infinite variety of sentences.”

(ibid., p. 167)

This vision was institutionalized in the planning process of the campus of the university of Oregon (an updated version of the planning process that however still includes the basic principles outlined by Alexander is currently introduced – cf. (Ramey and Thompson, 2005)). The process defines six basic principles:

**Participation** ensures that the end-users will be part of the planning process and, therefore, participate in shaping their environments. Alexander defines participation on different levels, ranging from acting as a client for an architect to actually building their environment. ((Alexander et al., 1980), p. 39)

In the Oregon Experiment, participation led to a very successful campus design. The university established a user group with students, faculty members, and staff. The user group decided, which projects should be built in the next phases. In a refinement phase, special focus groups (users with specific interests) concentrated on specific aspects of the building (focussing on one aspect at a time). In both, the user group and the focus groups, the prospective users started thinking about the various aspects of the space and how each part should be improved.

**Coordination** links together the different development initiatives made by the users in the user group. This ensures that the design activities will stay coherent at the whole campus. Instruments to achieve coordination are a permanent
planning office that coordinates the activities in the user group and brings different users with overlapping plans in contact.

**Organic Order** turns out to be one of the most important principles although it did not clearly emerge in the Oregon Experiment. For that reason, we will not discuss it now but postpone the discussion of organic order until section 2.8 where this principle forms the basis of Alexander’s current design theory. For now, it is important that order defines a space in which the requirements of the users are satisfied and conflicts between users are minimized.

**Piecemeal growth** is the technique for reaching organic order. It argues to concentrate on one aspect at a time. This includes identifying, and solving, one concrete problem.

**Diagnosis and repair** is the process of analyzing the existing campus regarding aspects that work, and aspects that do not work. This includes a phase of reflection: during the use of the environment, users are encouraged to step back and ask themselves, whether or not the environment serves their needs. If not, they are asked to mark the deficits and state change requests for the environment. The planning committee uses a map to catalogue all patterns that are not working (and the places where patterns are working well).

The results of the reflection process are captured in a health map of the campus. This health map shows areas of the environment that work well and areas that need to be fixed (cf. fig. 2.8).

**Patterns.** To empower the end-users so that they can find working solutions, they necessarily need a way of accessing established proven solutions. By means of patterns, one can describe expert knowledge as rules of thumb. These rules include a problem description, which highlights a set of conflicting forces and motivate a proven solution, which helps to resolve the forces.

In this manner, patterns can serve as an educational resource in the user group: they make the core of the problem and the solution explicit by explaining what happens to the inner forces.

Additionally, patterns are the essential language applied in the user group: the patterns (taken from a pattern language (Alexander et al., 1977)) act as a basis to communicate on a high level of abstraction the key properties of design.

Critics of the pattern approach have often argued that the pattern language would suggest an eternal truth. The style in which patterns are written (direct imperative voice) emphasizes this impression. But if this would be the case, the patterns would have the quality of dogmas, which, in turn, would mean that pattern authors describe unchangeable truth. In the Oregon Experiment, this impression is relativized (p. 137-141) by explicitly stating that pattern languages should evolve:
Participants should be encouraged to modify the published pattern language, which means that new patterns can be added and obsolete patterns can be removed.

Patterns with an impact on the whole community have to be evaluated and adopted by the whole community. This role can be delegated to a planning board.

In an annual review, the community checks the validity of all patterns.

New or modified patterns can only be accepted on the basis of observations or experiments.

As the culture of the community evolves, the pattern language that is used in the community is intended to evolve as well. Alexander further argues that all stakeholders should be encouraged to experiment with the patterns and propose improvements. In the context of the Oregon Experiment, this affected, e.g., teaching staff or students. This means that not only the environment of a user’s actions should be adapted, but also the ways of improving the environment should evolve.

All the mentioned principles are in use at the University of Oregon for nearly 30 years. New patterns are still brought in by the users. For each new project, there is...
a user group that comes up with the requirements. Participation did not go as far as Alexander envisioned in the way that users draw schematic designs before hiring architects. Rather, they propose special patterns and design concepts as a basis for the architects, who then continue the refinement together with the user group.

Another problem reported by the planning office was the application of piecemeal growth. The main reason for this was that “all campus projects must go through the Oregon State Legislature, which tends to favor larger building projects.” (University of Oregon Planning Office, 2005) Small projects in contrast receive fewer attention by the institutions in charge of funding.

2.7 The Scandinavian Approach – Participatory Design

End-user involvement is the main goal of participatory design (Schuler and Namioka, 1993). It has its origins in the debate between Scandinavian industry and trade unions regarding the relationship of work and democracy (Bjerknes and Bratteteig, 1995). The goal was to amplify the user’s voice and thereby improve working life. In 1977, this attitude was included in the Norwegian act relating to worker protection and working environment (AML):

“The employees and their elected representatives shall be kept informed about the systems employed for planning and carrying out the work, and about planned changes to such systems. They shall be given the training necessary to enable them to learn these systems, and they shall take part in designing them.”

(Section 12, §3 of the AML (AML, 1977))

Bjerknes and Bratteteig (1995) provided three reasons, why participatory design would lead to successful design decisions:

– Users who participate during the design process will help to improve the developers’ knowledge about the system under development, and in turn the users will gradually learn about the system that evolves. While the first aspect leads to a system that is shaped to the users’ needs, the second aspect will ensure that the users will be expert users of the system. The hope is that both aspects will lead to a higher acceptance rate.

– Expectations of participating users will be in line with the system under development. This means that users will adapt their expectations during the development and reduce their resistance to change from a former context to the new context imposed by the design.

– Workplace democracy is increased since the users who are affected by the system can influence the design decisions.

Bjerknes and Bratteteig (1995) identified four different levels of user participation according to the scope of the organizational unit in which design activities take place.
At the work situation level, users shape the tools that they need to perform their work tasks. Cooperative design is one technique to reach participation at this level. Users cooperate with designers and bring in their knowledge about the specific tasks. This view on participatory design is related to the theory of situatedness. Since the user is contributing experiences from the situation, the developed tool is likely to fit into comparable work situations.

The workplace or organizational level focuses on interaction between users. Organizational processes are shaped by the designed tools such as communication technology. The socio-technical approach of design provides a background for participation at this level (e.g., Hirschheim et al., 1995, p. 36). It argues to address both organizational infrastructures and social systems found in organizations. It has its roots in early action research projects of the Tavistock Institute with the focus on improving work conditions of British coal miners (in the year 1951). In the late 1960s and the early 1970s, trade unions successfully raised the awareness for a socio-technical view on organizational changes (cf. Mumford, 2000, for a history of socio-technical design). The main message was that social processes and technical systems that are involved in the processes should co-evolve. All members of the social system should have an interest in improving the system and thus participate in its development.

Design at an inter-organizational level shapes the relationship between the organization and its context. The participation process is more difficult at this level since the participants often can not be clearly identified. These relations have to be considered again from a socio-technical perspective.

Finally, the social or working life level focuses on the work situation at the level of society. Factors that can be designed are, e.g., political constraints. Participation should be reached on a democratic basis.

In the context of groupware development, the workplace or organizational level is of special importance. End-users should be provided with social (process) structures in which they can express their needs. One option is, e.g., user groups at the organizational level that have to be consulted whenever the designed technology is modified. In former times, this role was played by guilds, nowadays, e.g., works committees fill this role.

However, even though the end-users are in tight interaction with designers, the designers typically do all design activities. Participatory design stresses communication with users throughout the design process, but assumes that the user can play the role of skilled worker. Transfer of design knowledge to users was not in the main focus of participatory design.

2.8 Alexander – The Nature of Order

The most recent works of Alexander summarize the earlier approaches and describe the role of patterns and their application on a level that is no longer bound to
architecture.


Wholeness defines how well the different parts of a region in space interact. If a space is whole, the different parts of the space help each other, instead of imposing conflicting requirements on the space. This definition of wholeness is in line with the definition of (Smuts, 1936) who first defined it in the context of Holism. Holism describes the tendency in nature to form wholes that are greater than the sum of the parts. The parts and the whole co-evolve through creative evolution. Each part is connected to other parts by a field of forces. The more a part is embedded in this field of forces, the larger is its contribution to the wholeness of the organism.

Consider a social system as an example for the holistic perspective (cf. (Esfeld, 2003)): Alice is a member of the social system. Alice in turn consists out of bones and skin. But these components are no parts of the social system. In the context of the social system, these parts are out of focus. What makes Alice a part of the social system is that she is capable of thought, action, and interaction with other members of the social system. These are the properties that constitute the social system. Since Alice’s parts lack these properties, they are no longer considered as being parts of the social system. Moreover, Alice can only interact with other members of the social system if she is situated in the social system. Therefore, the full set of capabilities will only emerge if Alice is considered as part of a whole.

If the parts support each other, they evolve to centers. A center is a part in space that is in relation to its context (and since the context is not restricted to the nearest points in space, it is in relation to all other centers). Centers may be strong, which means that they are in line with other centers that are close to them. Centers can, however, only be understood in the context of the wholeness: since centers constitute the whole, the whole shapes the view on centers. The whole evolves from the synthesis and interaction of its parts.

For Alexander, wholeness is always bound to spatial structures. Holism, in contrast, is applicable to all structures regardless of any spatial properties. The only important property is that the structures carry a semantic distance. This is the reason why we consider the concept of wholeness applicable also for socio-technical systems (like groupware) as well. In socio-technical systems, the wholeness is composed of technical and social components (technical artifacts and users). And (as the above example suggests) the nature of the components is defined by the context of the socio-technical system. A user can, for example, play a specific role in a group process. The role defines the users’ actions that shape the socio-technical system. At the same time, the role is defined by the socio-technical system.

Wholeness can now be defined as the strength of its contained centers and the relation of the centers. At the same time, the wholeness defines the centers. Centers always become centers because of their position in the arrangement as a whole.

Transferred to the context of socio-technical systems, centers are structures that satisfy a specific requirement of the system. Relations between the different centers map to the relation between different requirements. If two requirements are
compatible (e.g., if one user of the system has the goal to teach another user and the other user in turn has the goal to learn from the first user), then structures to support the requirements also support each other and thereby create strong centers and wholeness. On the other hand, if the requirements are conflicting, the structures supporting the requirements will not support one another and can even destroy one another. In the first case, we speak of coherent requirements, while in the second case, the requirements are incoherent.

On a mathematical level, wholeness $W$ can be defined as follows:

1. Let $S$ be a socio-technical system addressing $n$ requirements $R = \{r_1, ..., r_n\}$. Represent a subsystem $S_j$ of $S$ by means of a bit vector $j = j_1, ..., j_n$ with $j_i = 1$ if the requirement $r_i$ is considered as element of $S_j$, 0 otherwise (we will interpret $j$ as a binary number).

2. Let $c_j$ be the coherence of the subregion $S_j$ with $0 \leq c_j \leq 1$. Alexander remarked that $c_j$ is a subjective measure. In our context of socio-technical systems, $c_j = 1$ if and only if all requirements of $S_j$ are satisfied and none of the requirements are in conflict. Note that this is interesting only from a theoretic perspective since the behavior of human beings is non-deterministic, which makes it impossible to capture and satisfy all requirements for a socio-technical system (since these systems represent wicked problems). In real socio-technical systems, $c_j$ will thus be smaller than 1.

3. Call the subregions $S_j$ with $c_j > t$ (with a given threshold $t$ that is close to 1) centers of $S$. Note that with a low value for $t$ each single requirement $r_i$ can represent a center although the individual value of $c_j (j \in 2^{\{1, ..., n\}})$ will be quite low. In an hypothetic ideal case, the whole system $S$ could emerge to a center. In this case, all requirements would be satisfied and emphasize one another.

4. Then, the wholeness $W$ is defined as the system $S$, together with the measure $c$ and all those subregions that have a measure more than some threshold and thus qualify as centers.

(cf. (Alexander, 2003a, p. 447))

The problem is to find a good measure $c$ for part (2) of the definition. In The Timeless Way of Building, Alexander (1979) called this measure the quality without a name. This quality emerges if the interaction of forces can take place in the environment without conflicts.

In The Phenomenon of Life, Alexander proposed a more concrete definition by means of 15 fundamental properties that help the collection of centers to constitute the whole. Examples in spatial systems are:

Levels of scale that help to perceive the center as part of a larger whole. To still perceive the coherence of the whole, the different levels of scale may not be too different in scale.
Strong centers that ask to look for the most important parts in the design. Alexander provided the example of a fire-place, which served as a strong center in ancient buildings. It brings together the inhabitants since it is the center for various purposes: warming up, preparing food, telling stories, etc. In modern societies, it is much more difficult to identify strong functional centers.

Boundaries that help to better identify the center. It focuses attention on the bounded part and it situates the center in the context.

Deep interlock and ambiguity that argues to create centers that share parts of their surrounding. The intersections of the surroundings will form a new intermediate center that helps to strengthen the two original centers.

Contrast that stresses the need for differentiation. A center has to differentiate itself from the surrounding and from other centers.

Gradients that explains how centers with different goals can be connected. The solution is to establish intermediate centers that help to change one quality slowly across space to another.

Simplicity and inner calm that argues to remove all centers that do not actively support other centers. It could be considered as the demand for creating the simplest solution that could possibly work.

Not-separateness that reminds the designer to consider the designed artifact as a situated artifact. As in his earlier works, Alexander follows the Heiddegerian notion of situatedness. The goal is to create a design that lacks abruptness or sharpness. Such a design minimizes the probability of breaks that distract the user of the space from his initial task. This is the reason why Alexander considered this property as the most important of the 15 properties.

Other properties are Alternating Repetition, Positive Space, Good Shape, Local Symmetries, Roughness, Echoes, and The Void. Alexander argues that the 15 properties constitute living structures in diverse sciences like architecture, biology, or physics.

Applied to socio-technical systems, these properties are still important. If we consider again a group process, the above properties can be interpreted as follows:

Levels of scale affect the different levels of interaction ranging from single user actions (without considering other users) over small group interaction up to the interaction in a large community.

Strong centers, for instance, can be kick-off meetings that are considered as important events by all group members.

Boundaries help to shield the group members from other group members and thus support better interaction in the group.
Deep interlock and ambiguity are omnipresent in group processes since group members never belong to only one group. Instead, each person is a member of many groups in different social contexts.

Contrast can be observed when a group creates an identity that makes them different to other groups.

Gradients are, e.g., important if group members gradually take responsibilities in the group or build up trust.

Simplicity and inner calm argues, e.g., for group processes that reach the group’s goal without much distraction.

Not-separateness reminds the group members that the group is again part of a larger social system and that the exchange with this system is important for the group’s success.

In The Process of Creating Life, Alexander (2003b) discussed why design can lead to configurations with less or more wholeness. According to Alexander, the key factor lies in the process: If the process unfolds in a way that relevant centers are empowered and unimportant centers are reduced it is very likely that the resulting design creates a living structure.

Each modification of the designed space should focus on strengthening the wholeness of the addressed subregion in space. And since the 15 properties contribute to wholeness, each transformation should focus on making one or more of the properties stronger. Alexander calls these modifications structure preserving transformations.

He defines a process with the following steps

1. The process starts by paying attention to the current state of wholeness of a focal region $R$.

2. Then the weakest subregions are detected that lack wholeness.

3. These subregions are scanned for latent centers. One of these centers is the focus of the next steps.

4. One or more of the 15 structure preserving transformations is applied to the latent center.

5. This results in the strengthening of centers and the birth of new latent or matured centers. Especially, larger existing centers should be strengthened and the overall coherence of the region should be improved.

6. A first test checks whether or not centers really improved.

7. A second test checks if the improvement was the simplest solution that could possibly work.

8. If one of the two tests fails, the modification is undone.
9. The process continues in a next cycle by looking at weak centers again.

Applied to a design task, the process creates a sequence of structure preserving transformations. It is obvious that each step in such a sequence depends on the step taken in the previous iteration. Having a sequence of $n$ steps would lead to $n!$ possible sequences, of which only a few are successful. This would not be feasible if there were no aids in structuring the sequence so that it leads to a structure preserving sequence.

The key factor here are patterns again. Considered in the context of wholeness, patterns are concrete instances of structure preserving transformations in a specific context. They describe how centers can emerge in a system of wholeness, and therefore serve as generic centers. Their validity comes from empiric observations: If successful sequences applied these patterns, the probability that they will help to create a successful sequence in another project is high. Since it is possible to name the effect on the different properties, it is also possible to approximate the effect on $c_j$ of a pattern (not in absolute values but in positive or negative tendencies).

From this perspective, patterns can be considered as strong generative rules that play a comparable role as memes do. Dawkins, who coined the term explained it as a replicator for ideas.

"Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation. If a scientist hears, or reads about, a good idea, he passed it on to his colleagues and students. He mentions it in his articles and his lectures. If the idea catches on, it can be said to propagate itself, spreading from brain to brain. " (Dawkins, 1989)

A meme, according to Gil-White (2005), carries several properties that allow them to be part of a Darwinian process; namely inheritance, mutation, and selection. A meme is a representation of an idea that is passed on from mind to mind in a human culture.

Inheritance is bound to replication. Memes are passed on from brain to brain. Transferred to the idea of patterns, the patterns are best practices that are imitated by other designers to create a good design. They are passed on from designer to designer and experienced by the users (who again pass them on, though from a different perspective).

As Gil-White (2005) argues, memes mutate in every act of transmission, since they are combined with the receiver’s current set of memes (his mental state). Transferred to the concept of patterns, a pattern is interpreted by the user who reads (or learns) the pattern. This user combines the practices proposed in the pattern with his own mental state, thus creating a mutated version of the pattern. This is what Alexander described when he said that “each pattern describes a problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” (Alexander et al., 1977, p. x). It is thus not intended that each solution is an exact replication of the pattern. Instead, the solution has to have small adaptations to the concrete context and the designer’s views (e.g., of aesthetics). But the adaptations should not be too large, otherwise the meme – or the pattern –
would not survive; this means that it could no longer be identified and the problem addressed by the meme would no longer be solved.

Whether or not a pattern survives depends on the probability that it will be passed on. Strong patterns that generate strong centers are very likely to be passed on since they carry on a subset of the fundamental properties. Patterns with a large value for \( c_j \) thus represent fit memes.

Gil-White (2005) pointed out that memes can only survive in the context of other (supportive memes). He provided the example of the meme of Beethoven’s fifth symphony, which requires that the memes of playing musical instruments like the violin or the trombone are passed on. The same is true for patterns. They require that supportive (and more widely accepted) patterns are passed on as well. The full set of required patterns then forms a pattern language.

Alexander proposes that each project constructs its own pattern language. This language can be invented from scratch (but still based on an empirical basis), composed from existing patten languages, or constructed by appropriating patterns to the context of the project. It can even be valid to invent patterns without an empirical basis. In this case, the application of the patterns (in different sequences) has to prove the validity of the patterns à posteriori.

The knowledge of pattern sequences helps to learn how to speak a pattern language. Compared to natural language, pattern sequences are examples of great literature that makes the best possible use of the language.

In summary, compared to The Timeless Way of Building and The Oregon Experiment, three aspects of patterns became more important: (1) the view of patterns as generic centers which means that they have to support the 15 properties, (2) the appropriation of patterns that reaches from the recomposition of a pattern language up to the invention of new patterns, and (3) pattern sequences that steer a process of structure preserving transformations.

2.9 Takeuchi and Nonaka – The Rugby Approach

The question is how a holistic design can be supported by means of a design process. Alexander already provided the idea of structure-preserving transformation, but this does not yet focus on group interaction in design. From Heidegger’s understanding of situatedness, the Oregon Experiment, and Schön’s concept of reflection in action, we highlighted the need for end-user involvement where end-users act as designers to shape their own environment. But still, the interaction in the team and the sequence of steps performed in the team (the workflows) need to be appropriated for innovative design.

Takeuchi and Nonaka (1986) use a sports metaphor for describing a creative holistic design approach. They compare traditional design with a relay race: Concrete responsibilities and tasks are assigned à priori to each team member. The tasks are ordered in a sequence and each team-member depends on his predecessor completing his task. Communication between the team members is limited to the
interfaces between the different tasks (e.g. passing a design document from a design
group to an implementation group).

In contrast to this approach, Takeuchi and Nonaka propose a rugby approach: A
team of players with different (multi-disciplinary) skills interact from the start to the
end of the game. The ball is passed between team members frequently depending on
the current situation in design. Different activities take place simultaneously (e.g.
the design evolves while at the same time, the implementation already investigates
the applicability of earlier versions of the design). The goal however is to move the
whole team downfield.

The point is that this approach – although innovative and extremely flexible re-
garding changing requirements – is vulnerable to wrong decisions. If, e.g., a require-
ment was wrong, the initial design will have flaws, and a fixed requirement will cause
a re-design. But as we discussed in section 2.3, this problem is inherent to any de-
velopment activity. Thus, facing it in a process of iterative experimentation showed
to be appropriate in contexts addressing very wicked problems. Takeuchi and Non-
aka provided case studies of innovative product development in larger multi-national
companies, like IBM, Xerox, or Honda, to support their claim. They identified six
characteristics of these teams:

Built-in instability argues to intentionally state the problem as a wicked problem
and thereby challenge creativity in the team.

Self-organizing project teams argue to give an interdisciplinary team the most
possible freedom to act with a focus on in-team communication and exchange
of ideas and knowledge within the team.

Overlapping development phases describe the rhythm of activities within the
team in a way that different phases of development, like design and implemen-
tation overlap to a large degree. The merits of such an approach are shared
responsibility and cooperation in the whole team. Obvious problems include
an increased need for communication; which may limit the application of this
approach to projects with few members. In addition, overlapping phases re-
duce the possibility of traditional division of labor for the benefit of shared
responsibility.

Multilearning focuses on the importance at learning on different levels:

– the individual level on that a team member gains new insights regarding
  a task (e.g. by means of books),
– the group level on which the team performs learning as a group activity
  (e.g. by means of an excursion), and
– the corporate level (e.g., by means of a corporate education program in
  which all members learn a shared view on quality control).

Subtle control, where managers act like trainers of a rugby team. They select
the players (and exchange them if appropriate), create a work environment,
establish training plans and reward schemes, or encourage the team to make mistakes and learn from mistakes.

**Organizational transfer of learning** that argues to propagate best practices experienced in the development team to other teams.

### 2.10 Requirements for a Process that Supports End-User Centered Groupware Development

In this chapter, we discussed different approaches to design. The discussion was intentionally kept on a level that was not specific to software and groupware design. The reason for that was that the design of groupware needs to address the design of social processes as well as the design of technical support systems. A discussion of design methods needs to take into account that the design is not only targeted on creating technical solutions following a mechanistic view of design. Instead, situatedness and non-determinism of the users has to be taken into account.

---

**Figure 2.9:** Influences between the different approaches to design.

Figure 2.9 provides an overview of the relations between the different design
theories discussed in this chapter. The different approaches to design suggest requirements for an ideal development process in order to solve the problem of this thesis. The processes provided provided answers to the question why finding appropriate requirements is a difficult task. Strategies that help to involve end-users are of special importance when designing a socio-technical system. The different design approaches provided hints how the interaction between the user and the development team may look like.

We will now summarize the requirements and relate the requirements to the different design approaches.

The most comprehensive requirement defines that the resulting product should be as good as possible. This may sound obvious, but in the context of the concept of wholeness, it is a very concrete and difficult requirement to reach.\(^3\)

**R1: Wholeness**
The designed groupware system should have a high level of wholeness.

We then derive requirements that describe how a high level of wholeness can be reached:

**R2: Structure-Preserving Transformations**
Each step performed in the design process should contribute to the groupware’s wholeness.

Since resources are limited, designers and users should, preferably, perform those steps that lead to large positive changes in wholeness. This implies that the users and developers are empowered to detect the most important issues at each step in the process. Due to the wickedness of the problem space, requirements analysis needs to take place as an integral part during the whole development. Three requirements support this goal:

**R3: Reflection**
The process should encourage users to reflect on their activities and adapt their group environment so that the group task can be better supported.

**R4: Process Awareness**
Users need to be aware of the current process and the current ranking of most urgent problems.

\(^3\)Requirements will be references by their number throughout this thesis (e.g., R1). Refer to the index for a mapping of requirement names and the pages on which the requirements were addressed.
R5: Iterative Design
The process needs to support iterative extension and modification of functionality in small iterations so that it can adapt to changing group processes.

Implicitly, the above requirements looked at the problem of finding the right problem to solve from two different perspectives: R3 put constraints on what problem should be addressed while R5 demanded that not too many issues are addressed in one iteration. To be more concrete, we can even further restrict the size of iterations:

R6: One Problem at a Time
The process should limit the development to one problem in each iteration.

This does not mean that only one force can be addressed in isolation. Rather we consider the problem space of groupware development as a highly related problem space. Each solution should thus take the context of the problem and the solution into account.

R7: Field of Forces
Each problem and solution has to be investigated in its context taking the field of forces imposed by the context into account.

R8: Explicit Forces
Designers and users should make these forces as explicit as possible.

The process of reflection already considers designers and end-users. Actually, both groups should participate during the whole life cycle of the application.

R9: End-User Involvement
End-users should be involved in the design process and find appropriate interaction structures so that they can express their needs.

To play the role of an informed designer in the process of design, education is an important prerequisite. The same is true for developers and designers (especially if they are new to the field of groupware development).

R10: Education
The process needs means to make end-users and developers eloquent enough to express and exchange their personal needs, group needs, or design views.
The latter requirement implies that end-users have to be empowered to understand and assess the different design decisions in a way that the benefits and drawbacks become clear to the user.

R11: Explicit Consequences
Solutions should be stated in a way that the benefits and drawbacks of the solution become clear to the users.

Understanding the solution is the first step to successful reuse. To allow the developer to apply the solution in his situation, he must be able to adapt the solution to the context.

R12: Integration
The solutions should be adaptable to the socio-technical context like the group structure, the programming language, the communication protocol, or the system environment.

R13: End-User Development
Wherever possible, users should be empowered to perform these adaptations on their own.

Depending on the degree of adaptation, the applied solution can already be the solution to a different problem. In this case, the solution should be fed back to the group so that future comparable problems can be solved more appropriate.

R14: Sharing of Best Practices
Successful adaptations and new designs have to be made available for all users so that they can be used for solving future problems.

R15: Quality Review
The quality of such new solution needs to be ensured by a sound review process.

Table 2.2 shows how the requirements relate to the different theories of design discussed in this chapter. If an entry in table 2.2 is marked (△), it means that the requirements is also part of the corresponding design theory.

In the next section, we will review existing software and groupware design approaches and show how they fulfill the requirements.
Table 2.2: Relation between the different design approaches and the requirements.
Chapter 3

State of the Art

We will address the state of the art in two phases: first, we will discuss existing software development processes and techniques, and show how they relate to the requirements. We will, especially, discuss the impact that these methods have regarding the understanding of wholeness. In a second phase, we will have a look at how these approaches have been applied in the context of groupware development.

3.1 Software Development Processes

In the last decades, many processes for the creation of software development have raised attention. These reach from early life-cycle models (in times where a sequential order of activities was the only dominant development method) up to iterative and agile processes.

From the context of groupware development, it is important to understand up to what degree the different process models support the requirements elicitation in uncertain context of wicked groupware problems. Development processes can be categorized according to their level of evolution (ranging from non-iterative processes to processes with many short iterations – requested by R5) and their level of end-user involvement ranging from the involvement as a customer up to the involvement as a developer (R9 and R13).

As shown in fig. 3.1, current schools of software development can be classified as follows:

- sequential non-iterative processes that argue for a requirements definition only in early stages of the project (non-iterative, user acts as customer),
- iterative processes that are a sequence of instances of non-iterative processes (with varying iteration lengths in the different process models) (iterative, user acts as customer),
- agile methods that argue for close interaction and communication between end-users and developers in short iterations (very iterative, user acts as customer interacting with the developer), and
– participatory design and end-user development that focus on empowering the users to act as designers or developers (iterative, user acts as designer or developer).

In the next sections we will discuss representative development processes from these classes. The discussion will be complemented with additional techniques for requirements elicitation and a view on how patterns have been utilized in the field of software development.

### 3.1.1 Sequential Non-Iterative Development

Alexander’s theory of decomposition was adapted by the software development community leading back to the NATO conference on software engineering (Naur and Randell, 1968).

“... software designers are in a similar position to architects and civil engineers, particularly those concerned with the design of large heterogeneous constructions, such as towns and industrial plants. It therefore seems natural that we should turn to these subjects for ideas about how to attack the design problem. As one single example of such a source of ideas I would like to mention: Christopher Alexander: Notes on the Synthesis of Form.” (ibid., p. 35)

Later on in the conference, Naur suggested that the process of clustering requirements and organizing them in a tree structure would be very promising to software development:

“... make a tree structure of the decisions, so that you start by considering together those decisions that hang most closely together, and develop
components that are sub-systems of your final design. Then you move up one step and combine them into larger units, always based on insight, of some kind, as to which design decisions are related to one another and which ones are not strongly related.” (ibid., p. 45)

Top-down and bottom-up development, introduced in the early 1970s, follows the same ideas: The set of functional requirements is refined and divided into (independent) sub-problems. The problems are translated into software when they can no longer be divided so that the individual parts would make sense in isolation. The parts are then combined (bottom-up) to larger structures until all requirements are met. So, software engineering adapted the process for decomposing the problem space into smaller, more feasible problems. The decomposition led – as Naur suggested – to a tree structure.

To this respect, the proposed decomposition shares all problems identified by Alexander in his essay *A City is not a Tree*. More specifically, the requirements may not overlap, which, however, is often the case in reality.

Contemporary development concepts generate structures that often do no longer comply to trees: object-oriented systems, for instance, allow to model centers (objects) and their relations to other centers (other objects). Different objects may communicate using message exchange. These messages can be exchanged concurrently, which leads to a graph structure of interacting objects (with no restrictions to the relations in the graph).

Therefore, decomposition can thus no longer be applied to the whole structure of a development project. Instead, its importance lies in contexts where one small sub-problem is addressed for that the requirements are well defined and the focussed set of objects interact in a sequential order.

Fig. 3.2 shows the *Waterfall* process initially proposed by (Royce, 1970) which is a prominent software development methodology that is based on decomposition. It arranges software projects in different phases. It starts with the design of the requirements, and, finally, reaches the operation of the implemented product. The problem with waterfall processes is that all requirements are collected at the beginning of the project and changes to the requirements should be avoided later on. According to Alexander this is a problematic understanding of design. Especially, it is in conflict with R5.

The original version of the waterfall process included two iterations: after the preliminary program design a preliminary system should be built. This serves as a prototype for collecting first user experiences and assessing the technical behavior of the system. The design of the preliminary system follows, again, a fully-fledged waterfall process except that it does not include a separate requirements analysis since the requirements were already gathered in the main waterfall process.

Royce intended that the customer is involved in four phases of the project: He (1) defines the requirements, (2) reviews the prototype built in the preliminary iteration, (3) performs critical software reviews between design and coding, and (4) performs an acceptance test before the system enters operation. The reason for including the end-user and for creating a prototype implementation was to reduce risk due to
vague or incorrect initial requirements. In relation to the requirements of section 2, it addresses the following requirements:

- R3 (Reflection) takes place when the user uses the preliminary system to gain usage experiences. The problem is that the usage of the preliminary system is limited to the time before the analysis phase of the main waterfall process.

- R4 (Process Awareness) is achieved since the process is formally structured. At each point in time, one can determine in which phase one currently is. The problem here is that the phases are very long and address different functional aspects (e.g. all requirements have to be addressed in the design phase, which makes it difficult to determine what functional aspect is currently investigated, and how this aspect relates to other functional aspects).

- R5 (Iterative Design) is addressed by creating one prototype. However, this corresponds to exactly two iterations and these iterations do not detect the requirements independently. Since the number of iterations is fixed (and low), we cannot see R5 satisfied.

- R9 (End-User Involvement) takes place during reflection. However, the waterfall process rather speaks from customer involvement, which does not mean the same: the customer does often not have to use the system in daily work contexts.
3.1. SOFTWARE DEVELOPMENT PROCESSES

Table 3.1: Relation between the waterfall process and the requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>▽</th>
<th>Requirement</th>
<th>▼</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Wholeness</td>
<td></td>
<td>R6: One Problem/Time</td>
<td></td>
</tr>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td></td>
<td>R7: Field of Forces</td>
<td></td>
</tr>
<tr>
<td>R3: Reflection</td>
<td>△</td>
<td>R8: Explicit Forces</td>
<td></td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td>△</td>
<td>R10: Education</td>
<td>△</td>
</tr>
<tr>
<td>R11: Explicit Conseq.</td>
<td>△</td>
<td>R12: Integration</td>
<td>△</td>
</tr>
<tr>
<td>R15: Quality Review</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 summarizes the requirements addressed by the waterfall process. The waterfall process was widely cited. Notably, most citing authors reduced it to a linear sequence from analysis to design that did not include the preliminary system development and the steering points for the end-user (e.g. (Davis, 1997) or (Malotaux, 2001)). This has the effect that most “waterfall” projects do not address the requirements R3, R5, and R9. Large projects especially often rely on formal requirements that are defined before developing the system and that serve as a contract between customer and the development team.

![Figure 3.3: Cost of change from a waterfall process perspective (adapted from Beck (1999, p. 21) and Kruchten (2003, p. 13)).](image)

The reason for this is to reduce the cost of change since many developers assume that changing requirements become more expensive the later they are detected in the development process as shown in fig. 3.3 (cf. also the second requirement paradox in section 2.3).

Table 3.1 summarizes the requirements addressed by the waterfall process.¹

¹If a requirement is violated, the table includes a ▽, if it is partially addressed, it shows a ▽, and if it is supported, it shows a △. In cases where the process supports a requirements very well, ▲ is shown. Note that in this and the following sections only requirements that are addressed or explicitly hurt by the process are mentioned. Requirements that are not applicable will not be mentioned in the discussion. The cell in the table will stay empty in this case. A comparison of the different process models discussed in this chapter can be found on page 77.
The waterfall process still plays an important role in the widely used V-Model. Recently, this model was extended with respect to requirements analysis and project planning and evolved to the V-Model-XT (KBSt, 2005) shown in fig. 3.4. The V-Model-XT was developed by the German administration and has the goal to structure the relationship between software developers (contractors) and public authorities who act as customers. Although it was intended for contracts in public administration, it is widely used in industrial settings as well.

![Figure 3.4: The V-Model-XT.](image)

It defines how requirements documents should be specified (from a customer’s perspective). Contractors subsequently create an information system that complies to the requirements (note that the V-Model does not prescribe how this construction has to take place). The customer finally compares the specified requirements with the delivered solution and accepts the solution if all requirements are satisfied. The main focus of the V-Model-XT thus lies between the customer and the contractor. Requirements are formulated in a specification document, which is the basis for the contract with the contractor. The customer intends not to change these requirements since any change results in a complicated change request procedure (in conflict with the requirement for short iterative design R5). Therefore, participation (R9) is very limited and uncertainty is not addressed.

![Figure 3.5: The V-Model-XT appropriated to agile processes.](image)

The V-Model-XT argues for appropriation in each process instance. Notably, the documentation for the model also includes appropriations for agile process models (cf. section 3.1.3) shown in fig. 3.5. However, the sequential nature of the V-Model gets lost in such appropriations. Steps are mixed up and iterations are added.
3.1. SOFTWARE DEVELOPMENT PROCESSES

fact that such appropriations can be found in the official V-Model documentation shows that a linear sequence of steps is often not appropriate in real-world development projects.

But, as discussed in section 2.3, the design of software and especially the design of groupware often leads to changes in the requirements. This means that a project that follows a single waterfall process can easily fail. The question thus is how to “embrace change” (Beck, 1999). Iterative software development processes provide a solution to this problem.

3.1.2 Iterative Development Processes

The basic idea of iterative software development processes is to reduce the length of the process and perform the process multiple times. According to Larman and Basili (2003), it has its roots in early development project at IBM in 1957. In 1985, it was made more prominent with Boehm’s introduction of the spiral development process framework (Boehm, 1988). One prominent example of contemporary iterative processes is the Rational Unified Process (RUP) (Jacobson et al., 1999). The iterations in RUP can be compared with small Waterfall processes (including the full set of documents that is normally produced in a waterfall process).

![Figure 3.6: Connecting multiple instances of a waterfall process to reach an iterative process.](image)

Although the RUP was not the first iterative process in the context of software development it received much attention. Key concepts are, compared to the waterfall process, much shorter iterations and explicit roles and workflows for activities performed in the development team (cf. fig. 3.7).

According to the progress in the project the different iterations will have a different composition regarding the importance of the individual phases: the inception phase, the elaboration phase, the construction phase, and the transition phase. The inception phase mainly focuses on requirements elicitation and the definition of the overall idea of the project. The elaboration phase focuses on making the use-cases more concrete. It includes parts where first ideas are implemented in a prototype system. The construction phase focuses on the implementation of the system. Requirements, analysis, and design are still part of each iteration in the construction phase, but their impact on the implementation task will not be as large as in previous phases. This means that most of the time is spent in implementation.
In later editions of the RUP (Kruchten, 2003), the development centered view was complemented with additional supportive workflows for the management. These are:

**Deployment** focusing on creating a system that can be passed on to the customer (including the required documents).

**Configuration- & Change-Management** that provides workflows for managing change requests and configuring the different versions of the product.

**Project Management** focussing, e.g., on risk assessment, the definition of product measures, or the management of the team’s work schedule.

**Environment** with the goal of equipping the development team with the right tools (e.g., software development environments).
Deployment becomes important in late stages of the construction phase. Configuration- and Change-Management is an important issue throughout the construction phase. The transition phase concentrates on shipping the project, and installing it at the client’s site. In this phase, deployment, configuration management, change management, and the implementation of required changes are in the main focus.

The core process model focuses on the management of and the interaction in the development team. One role, the system analyst, is supposed to interact with the customer. But the customer is still in a position where he states requirements and tries to limit the interaction with the development team afterwards. Unless the customer takes one of the roles defined in the workflows (and thus acts as a member of the development team) his influence will be limited. The focus on formal change request procedures in addition slows down the reaction to change of the context.

Ambler et al. (2005) extended the RUP with workflows for later phases in the product life-cycle – the production phase and the retirement phase – and called this extended process the Enterprize Unified Process (EUP) (see also Ambler, 2005b).

![Diagram](image-url)

**Figure 3.8:** Importance of disciplines in different phases of the EUP (cf. Ambler, 2005b).

The production phase provides guidance on keeping the product running. Thus, “operations and support” play an important role as additional discipline. Change- and configuration management additionally focus on registering user change requests and scheduling these requests for a future redesign of the system. In the production phase, only minor bug-fixes are performed, which implies that major change requests from a user will have to wait for a new project.

The retirement phase describes how the system should be replaced by another system or removed. Change and configuration management is extremely important in this phase.

---

2Ambler (2005a) argues that the customer should be trained to fill roles such as the Business Process Designer and Requirements Specifier. This, however, requires that the customer moves away from his original profession towards an analyst’s profession.
Fig. 3.8 shows the importance of the different disciplines in the context of the life cycle.

The EUP (as a more comprehensive version of the RUP) better meets the requirement of iterative design that we expressed in R5. Compared to the waterfall process there are many iterations and the iteration length is reduced. The overhead of documentation and formal activities is also reduced since EUP asks developers to focus only on documents that make sense during a specific iteration. Nevertheless, it is still a document-centered approach in which communication is structured by means of pre-structured documents (e.g. class diagrams, formal use cases, or interaction diagrams (Fowler, 2000)). All participants have to learn to understand and write such documents, which makes it hard for end-users to participate in the design and implementation phases.

For each iteration the developers perform an iteration planning. This includes that the tasks for the current iteration are defined. Process awareness (R4) is thus established, although an iteration focuses on a set of problems rather than on one single problem (in conflict with R6).

The involvement of end-users is coordinated by the system analyst in the early analysis phases. He interacts with the customer and creates use cases. Use cases re formal scenarios of system use. Unlike user stories, they follow a predefined structure with the goal of describing the system behavior as exact as possible. In the production phase, the customer has two interaction partners in the development team: the customer service representative is responsible for communicating with the end-users and assisting them during system use. “Problems” that have their origins in the user’s “misunderstanding” of the system are solved by the customer service representative since he helps the users to learn how to use the system. The system support representative is responsible for cases where the system misbehaves. He escalates these defects or enhancement requests to the actors in the configuration and change management discipline so that they can be scheduled for a future release. Other involvement requires, as discussed above, that the user becomes an analyst or a developer. This is why end-user involvement (R9) is still limited, especially in the context of system use, where the user should be encouraged to reflect (R3). The process does not further guide how the users should be educated to act as designers (R10) or even perform changes on their own (R13).

Table 3.2 summarizes the requirements addressed by the Unified Process.

---

3Note that Cockburn (2000) proposed a more prosaic notion for use cases that is very suitable for end-user involvement.
A tighter integration of the customer during all phases is desirable for groupware application development. Agile methods that will be discussed in the next section address this problem.

### 3.1.3 Agile Methods for Software Development

Agile software development (e.g. Cockburn, 2002) focuses on tighter interaction within the development team and between the team and the customer. At the same time, it argues to reduce the creation of artifacts used in a process and the number of formal workflows of the process. This is expressed by the *Agile Manifesto*:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.”

*(Beck et al., 2001, original emphasis)*

The motivation for agile methods is that the requirements will change over the project life cycle. Instead of minimizing risk through more elaborated analysis and design (before doing any development), it focuses on creating simple solutions that are easy to modify and adapt to changed requirements. If the software is kept simple, the exponential cost of change (fig. 3.3, p. 43) will be reduced so that it is no longer exponential (fig. 3.9).

![Figure 3.9: Cost of change from an agile process perspective (adapted from Beck (1999, p. 23)).](image)

A large set of agile methods has evolved since the mid 1990s. These methods can be understood as templates for concrete processes that are tailored to a specific

---

4Refer to Abrahamsson et al. (2003) or Boehm and Turner (2004) for an overview, a genealogy, and a comparison of current agile methods.
project (since the agile manifesto explicitly states that the focus should be on the individuals that are participating in the process, rather than on the process itself).

Agile methods have been perceived quite controversial, since they argue to travel with as few as possible formal workflows and documents. Tom DeMarco, e.g., expressed his preference for agile methods in a personal communication with Grady Booch as follows (Booch, 2001, footnotes not part of the original quote):

“I think the strongest trend at work today is toward light methodology. In this class I include Booch Lite\(^5\), Kent Beck’s XP\(^6\), Scrum\(^7\), DSDM\(^8\), Crystal Methods\(^9\) (by Alistair Cockburn), and Jim Highsmith’s Adaptive Development.\(^{10}\) Probably the most prominent of these is XP (extreme programming). It is the best of a new breed of approaches that thumb their noses at CMM\(^{11}\) and all other forms of fundamentalism. The focus has to be on building skills into the people, not building regimentation into the process. It requires no great leap of intellect to see that everything has to go faster in this age. You don’t go faster with bigger methodology; to get faster we have to move toward much lighter methodology. The CMM makes us wonder \textit{What else shall we add?} while the light methodologists are always asking \textit{What can we take away?}”

The main argument against agile methods is the lack of empirical evidence. Instead of being synthesized using scientific measures and tools, most of the methods have evolved from subjective personal experience. However, the body of empirical scientific evidence is currently growing for all agile methods (cf. Abrahamsson et al. (2003) for a list of recent studies and (Cohen et al., 2003) for an overview of current agile methods). The results of such studies help to point out the advantages and the limitations of agile approaches.\(^{12}\)

In the following section, we will discuss three agile methods regarding their appropriateness for groupware development. We selected (1) eXtreme Programming

\(^{5}\)Booch (1992)
\(^{6}\)See page 51.
\(^{7}\)See page 55.
\(^{8}\)Dynamic Systems Development Method (DSDM Consortium, 1997) is a development method organized in five phases comparable to the RUP. It emphasizes activities related to project management, especially methods for prioritizing requirements. Customers classify the requirements in four distinct categories (Must have, Should have, Could have, and Want). As in the RUP, the strong focus ion planning should help to reduce risk.
\(^{9}\)The Crystal Methods define a process framework for development processes that focus on communication and people (Cockburn, 2002). It is intended that the methods are tuned to match the organizational context and the concrete problem.
\(^{10}\)Adaptive Software Development (Highsmith, 2000) – a process model that argues for a high level of flexibility in the team so that it constantly adapts plans and actions to the current situation.
\(^{11}\)CMM (or the most recent version CMMI – (CMMI Product Team, 2002)) is a framework for evaluating the maturity of a software process. It describes on more than 700 pages requirements for a software process and the documents that need to be created in the process. Together with Boehm and Turner (2004, p. 187), we consider the CMM as a framework that generates heavyweight processes.
\(^{12}\)It is needless to say that the same evidence is still needed for newer methods that are based on more traditional approaches (like the EUP). Thus, we observe a problem of credibility for most new development process models.
(XP) since it is the most widely known agile methodology, (2) Scrum since it is related to Rittel’s concepts of wicked problems and based on Takeuchi’s and Nonaka’s rugby approach, and (3) Feature-Driven Development that is closely related to UML-based approaches like the RUP and thus bridges the gap between agile and traditional approaches.

### eXtreme Programming

eXtreme Programming (XP) was disseminated to a broad public with the publication of Beck (1999). It is based on four values that should shape development activities:

1. **Communication** that argues for as much face-to-face communication as possible and needed to achieve a shared understanding in the team and between the team and the (on-site) customer. Communication is the most important value. Since the teams often create software without having a formal (specified) design, it is much more important that an agreement in the team about technical solutions is reached and that this agreement is communicated to other team members.

2. **Simplicity**, so that only the simplest solution that could possibly work is developed. This value is based on the observation that in traditional processes many features or extension points are built but never used.

3. **Feedback**, so that all participants always have an impression about the current state of development.

4. **Courage** that demands of all participants to accept failure and fix it (rather than creating work-arounds) or to throw code away if it shows to be insufficient.

   The principles match with the values of the agile manifesto. Communication supports interaction and customer collaboration. Simplicity supports working software (the system does exactly what users want and not more). Feedback and courage support responsiveness to changes.

   From the core values, Beck derived 12 practices that should be applied in a XP project (cf. fig. 3.10). Coding standards, collective ownership, pair programming, and the 40 hour week are very specific to the activities of the developers (the programmers) in the team. The other practices are more important with respect to the requirements of chapter 2.

   Testing using a test-first approach and refactoring describe a process comparable to diagnosis and repair of the Oregon Experiment (cf. p. 22). Before any modification is applied to the system, developers and users are asked to think about tests that check the system for the break that should be fixed (reflection – R3). This test should describe the conflicting forces and how these forces should be resolved by the system. After the test is written, the developers and users check whether the test really fails (to “show” that the test is correct). They then implement a solution that resolves the conflicting forces (and thus satisfies the requirements). After the solution is built the system is checked again to prove that the modification really solved
the problem. To check whether or not all other forces are still resolved according to the users’ needs these tests are collected and re-executed when integrating the solution with the whole system. This ensures that the system continuously evolves and that existing centers are improved (supporting structure-preserving transformations – R2).

Refactoring implements diagnosis and repair on a non-functional level. In test first design, the users reflect and report on break situations. In refactoring, developers report on breaks from a technical perspective – so-called code-smells. At this level, a break is produced by code that is hard to understand. Whenever developers encounter such code that smells, they are asked to refactor the code using a transformation guideline (the refactoring) from a refactoring catalogue (Fowler, 1999). Each refactoring is a micro-pattern that describes how to detect bad code and how to transform it in order to make it more readable and flexible for future extensions.

Simple design supports both refactoring and test first development. The idea behind this practice is that code is designed without any unneeded structures and that the designed artifact has a good shape. With this respect, simple design can be seen as a practice that helps to bring the properties Simplicity and Inner Calm and Good Shape into place (cf. section 2.8).

To create good shape, developers and end-users need to establish a shared vision of the built system. They need an idea of the shape, comparable to a sculptor who needs a vision in order to transform a stone into a sculpture. Finding and communicating such an idea can be a challenging problem in XP.

Figure 3.10: Practices in XP in relation to requirements for a groupware development process (adapted from Beck (1999, p.70))
Beringer (2004) describes the problem in the interaction between developers and end-users as *expertise tension*: End-users may lack the expertise needed to build, appropriate, or apply tools in a specific context. Among others, one approach to overcome this problem is the use of *metaphors* to map problems of an unfamiliar domain to a more familiar problem space. According to Lakoff (1993), there are several aspects that define a metaphor:

- Metaphors serve as mappings across conceptual domains.
- Metaphors serve as conceptual means not as linguistic constructs.
- Metaphors are asymmetric and partial, which means that only important parts of the source domain are captured by the metaphor and projected to the target domain.
- The mapping works by focussing on correspondence in experiences connected to the entities in a source domain and entities in a target domain.
- The correspondences help to apply patterns of use that are familiar in the source domain to the target domain.

From this background, the process of software development can be understood as the process of creating metaphors. Patterns of use from a familiar domain have to be mapped to another domain (with a potentially different context). The software developer helps to find or invent metaphors while communicating with the end-user. The problem is that good metaphors are hard to find and that XP does not provide guidance in this search process.\(^\text{13}\) Good patterns (that describe strong centers), however, could fill the role of a system metaphor in XP (more details on this thought will be discussed in section 4).

The second shaded area in the upper part of fig. 3.10 contains practices that are relevant for involving the end-user and addressing his needs.

End-user involvement is one of the most important but also most difficult issues in XP (R9). XP argues that a customer representative should live with the development team as an on-site customer. It mainly focuses on participation during requirements engineering related tasks. The user acts as a member of the development team and steers all activities. The advantage of an on-site customer is that there is always a person at hand, who has experienced the requirements and can provide more detail on the requirements whenever needed (cf. (Paetsch et al., 2003) for a deeper discussion of requirements engineering in agile processes). The customer’s special focus is on the *planning game*. The planning game is a lightweight tool for requirements elicitation and prioritization. The users create stories that describe the

\(^{13}\text{Khaled et al. (2005) wrote a pattern language to support the detection of a metaphor in XP. Their guidance is to examine the semantic network addressed by the system in a so-called metaphorscape. Areas of special focus are then identified in the metaphorscape. The most important metaphors are further investigated in a group inspection with the goal of identifying good and weak points of the metaphor.}
system behavior (informal success stories comparable to those described by Cockburn (2000)). In addition, the users think of a test that would convince them to accept the solution (the acceptance test). The stories are collected on story cards and estimated regarding the required efforts to implement each story estimated by all members of the planning game.

In a next step, the customer “shops” user stories until the available resources of one release are spent. He also prioritizes the “bought” stories since it is a rule of XP that estimates can be wrong. The result of the planning game is an ordered list of story cards that will be implemented in that order (and thus supports a good process awareness – R4).

Note that the customer can change his mind, which results in new stories, removed stories, or a new order for the stories. This requires that the stories are small. One iteration (lasting one or two weeks) should provide enough resources for addressing several stories. It also requires that the customer accepts to take responsibility for the implemented system. Small releases and continuous integration help the customer to reduce risk, since he always has a working piece of software. A predefined requirements specification that serves as a contract is, on the other hand, not applicable for XP projects.

Instead, *time boxing* is more appropriate (Malotaux, 2001). This approach describes a situation where time and costs are fixed. In the three-dimensional process of design that considers time, costs, and features, features are the only variant to react to changes in the plan (cf. fig. 3.11).\(^\text{14}\)

![Figure 3.11: The set of features varies during the development process (adapted from Malotaux (2001))](image)

One often neglected point in the planning game is that the customer needs to be able to create stories. As we discussed it chapter 2, this is difficult for wicked problems without any additional guidance regarding orientation in and identification of the problem space. The process of reflecting and determining conflicting requirements is difficult and not supported in XP’s core process model. Hazzan and Tomayko (2003) proposed to combine interaction of the planning game with Schön’s concept of reflection in action (to satisfy R3). Discussions in the planning games

\(^{14}\)Note that Beck and Fowler (2001) add a fourth value, quality, that should also be fixed.
should be structured in a way that the customer moves through different levels of reflection (cf. ibid.):

- **Designing**: The customer feels the need of a requirement and is asked to think about comparable situations he experienced before.

- **Description of designing**: The customer identifies comparable problems and reshapes the design ideas of the first step by applying lessons learned from the comparable design problem.

- **Reflection on description of designing**: The developer reflects on his understanding of the two descriptions provided by the customer and together they appropriate the solution to the current context.

- **Reflection on reflection on description of designing**: After the design is envisioned, both developers and customers reflect on their process of appropriating current requirements and comparable problems. This results in process learning.

One can consider this extended XP approach as one step towards educating the customer in better expressing his needs (R10).

A second type of planning game focuses on a specific iteration. In such an iteration planning game, the developers identify tasks that are stories more focussed on the implementation of the system. These tasks refine the user stories. During implementation, the developers pair up to solve one specific task. Thus, developers always address only one problem at a time (R6).

In short development iterations all aspects of a traditional waterfall process have to take place (cf. fig. 3.12). But, assuming that an XP development iteration lasts one or two days (producing releases of, e.g., 7 user stories every fortnight), the aspects are addressed almost in parallel (cf. Beck and Fowler (2001, chapter 5)).

The different iterations in XP planning games are shown in fig. 3.13. The smaller the focus of the iteration gets (and the shorter the iteration becomes), the more technical issues will be considered in the iteration. Thus, the developers will act as main drivers (providing the directions and most of the ideas) in the development iterations, while the customer provides most ideas in the release-sized iterations. One could think of a context where the customer is empowered to act as a main driver in the development iterations as well. But since the steps in these iterations are very technical, new tools for addressing technical issues are required to satisfy the requirement of end-user development (R13). We will discuss research in this field in section 3.1.4.

Table 3.3 summarizes the requirements addressed by eXtreme Programming.

### Scrum

The rugby approach of Takeuchi and Nonaka (1986) laid the ground for the Scrum software development methodology (Schwaber, 2004). The methodology focuses on
Figure 3.12: Extremely short iterations let all activities in the design process take place almost concurrently.

Table 3.3: Relation between eXtreme Programming and the requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>XP</th>
<th>RUP</th>
<th>Waterfall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

small projects with teams of up to 10 members. The project’s objective is to address a wicked problem that implies that the requirements are hard to define and that the problem area is not well known.

As XP, Scrum is based on different kinds of iterations with different levels of scale (fig. 3.14).

At the beginning of the project, the Scrum team lays out the project’s vision and defines requirements. These requirements are kept in a requirements log (the product backlog). The product backlog is owned by the product owner. An important role of Scrum is the Scrum Master. He acts as consultant for the developers and assists the customer in maintaining the product backlog.

Before each iteration, the team performs a sprint planning. The goal of this phase is to define a prioritized list of requirements that should be addressed within the next development iteration. Scrum calls this list the sprint backlog.
A project is composed from 6 to 8 development iterations, the sprints, that last approx. 30 days. The requirements in a sprint backlog may not be changed externally (e.g., by the customer) after the team started the sprint.

The team self-organizes during a sprint. The only fixed point is a daily Scrum meeting in which the team members report on solved tasks, obstacles, and plans for the next day. Comparable meetings are also part of XP.

At the end of a sprint is a release. This is passed to the customer who can then evaluate the product and adapt the requirements in the product backlog. In addition, the team reflects in a post-sprint retrospective on the sprint and identifies best practices that emerged in the sprint.

Scrum Masters can form echelon teams to scale the process for larger projects. Each Scrum master becomes a member of the global echelon team. This ensures that communication between different product teams is maintained and that conflicting product development will be minimized.

Although based on the rugby approach, Scrum neglects an important aspect: Overlapping development phases were part of the rugby approach. This is only partially the case in Scrum, since the requirements analysis phase is preceding the sprints and during a sprint there is no way to switch back to requirements analysis.
CHAPTER 3. STATE OF THE ART

Figure 3.14: Iterations in Scrum.

Table 3.4: Relation between Scrum and the requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Scrum</th>
<th>XP</th>
<th>Requires</th>
<th>Supports</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Wholeness</td>
<td>△</td>
<td>△</td>
<td></td>
<td>△</td>
</tr>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td></td>
<td>△</td>
<td></td>
<td>△</td>
</tr>
<tr>
<td>R3: Reflection</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R4: Process Awareness</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R6: One Problem/Time</td>
<td>△</td>
<td>△</td>
<td></td>
<td>△</td>
</tr>
<tr>
<td>R7: Field of Forces</td>
<td>△</td>
<td>△</td>
<td></td>
<td>△</td>
</tr>
<tr>
<td>R8: Explicit Forces</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R9: End-User Involvmt.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R10: Education</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R11: Explicit Conseq.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R12: Integration</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R13: End-User Dev.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R14: Share Best Pract.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>R15: Quality Review</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
</tbody>
</table>

Compared to the requirements for groupware development processes, Scrum does thus not add additional value over an XP approach. It rather only poorly addresses the requirement of end-user involvement (R9) and only partially addresses the requirement of education (R10) in the post-sprint retrospectives. The requirement of structure-preserving transformations (R2) that was addressed partially by XPs test-first approach is also not supported in Scrum.

Table 3.4 summarizes the requirements addressed by Scrum. Note that Scrum mainly focuses on social aspects and team interaction. It can be combined with XP’s development practices in order to support the interaction in the sprint.

Feature-Driven Development

Although widely accepted as an agile method, Feature-Driven Development (FDD) (Coad et al., 1999) is not as flexible as XP or Scrum regarding changing user requirements. FDD organizes a project in the following stages (cf. fig. 3.15):

Develop an Overall Model: Domain experts (users and customers) and developers form a modeling team. They have the task of building an informal features list and examining different modeling alternatives to satisfy the features. Finally, a chief architect merges the created alternatives into one coherent model that serves as a system model for future phases of the process. Note that FDD
assumes that one genius architect takes the role of a chief architect. He will shape the designed system according to his beliefs and is assisted by the rest of the modeling team.

**Build a Features List:** Like in top-down approaches, a feature list for the system is built up-front. Developers and domain experts collaborate in identifying and scoping required features. Note that this phase is not intended to run in iterations. We can thus observe a waterfall-like requirements elicitation in FDD.

**Plan by Feature:** Chief programmers, the project manager, and the heads of the development team discuss possible sequences for the required features and plan milestones for coherent subsets of features.

**Design by Feature:** For each feature, an individual iteration is started. A chief programmer investigates the feature and thinks of possible affected classes. All owners of these classes are invited to the feature team and will from then on assist the chief programmer in designing the feature.

**Build by Feature:** After having created a design for the feature, each class owner implements the part of the feature that touches his class. The chief programmer observes the work of the individual class owners and ensures that the parts of the individual developers fit together.

Most concepts of FDD have been found in earlier contexts: The concepts of chief architects and chief programmers have already been discussed by Baker (1972). Prioritization of requirements is also common to requirements engineering approaches. The different phases are present in waterfall process and the transitions between the phases are – like in the RUP – supported by UML documents like class models or interaction diagrams. Actually, FDD relies heavily on modeling techniques rather than on programming as it was the case in XP. These indications suggest that the FDD model is a rather conservative approach to software development.

The interesting part of FDD regarding the requirements for groupware development lies in the focus on one feature at a time (in line with R6).

Table 3.5 summarizes the requirements addressed by Feature-Driven Development.
3.1.4 Participatory Design and End-User Development

In participatory design for software systems the end-users closely interact with the developers to shape the system (Muller and Kuhn, 1993). In the context of single-user software development, participatory design has focused on the work situation level (cf. section 2.7). The field of participatory design lies between the end-user domain and the professional developer’s domain. Thus, it forms a “third space in design” (Muller, 2003). In terms of Alexander one could say that participatory design aims at deep interlock and ambiguity (cf. p. 28) between the different domains of the end-user and the professional developer.

We distinguish (1) participatory design, where an artifact is designed by the developers who are assisted by the users, and (2) end-user development, where the users build or appropriate their application on their own. The first variant results in a product which is probably closer to the users’ need, but still ready-made. The second variant results in an application that is individualized for one specific user (although the user’s capabilities and the tool’s context and variability limit the options for appropriation).

Muller et al. (1993) propose a taxonomy for organizing practices used in participatory design. They distinguish practices regarding the time in the development cycle and the type and context of participation (fig. 3.16):

- Practices can be used early or late in the product’s life cycle. An early application focuses on early phase in the traditional life cycle and addresses the elicitation of requirements or the design of user stories. Practices for late phases focus on collaboration during implementation or participation in usability tests.

- The type and context of participation can be classified regarding the question whether the user moves to the developer’s world or the developer moves to the user’s world. In the first case, the users participate in activities that are traditionally related to software development. In the second case, the developer participates in activities that are domain-related.

The taxonomy results in a diagonal layout of the different practices. In early stages most approaches focused on involving the developer in the users’ work situation, while late stages focus on involving the user in the software design.

Muller (2003) provides a detailed overview of the state of the art in participatory design. In this section we take a course-grained look at different practices with the
3.1. SOFTWARE DEVELOPMENT PROCESSES

Figure 3.16: Taxonomy of PD Practices (adapted from (Muller et al., 1993, p. 27) – The original version did not include the clustering and showed additional practices).

goal of introducing terms and techniques that were also applied in the design of collaborative systems (cf. section 3.2.3).

Ethnographic Methods have been widely used in participatory design. An ethnographer is embedded in the workplace and observes the users’ actions. Contextual Inquiry (Holtzblatt and Beyer, 1993) modify ethnographic methods by allowing the observer to interrupt the user’s work at any time. The observer asks questions regarding the users intentions to complement his outsider’s view of the process. In this way, the interviewer assists the user in making the forces that determine his behavior explicit.

Scenario-Based Design (Carroll, 1995) focuses on describing scenarios of system use in a narrative form to ease the communication between developers and users. The scenarios can be attributed with an explicit description of benefits and drawbacks for each scenario participant. Unlike ethnographic methods, they extend the observed interaction by visions of future interaction and thus help to envision future solutions. Scenario-based approaches are very close to use case design (if understood in the way propagated by Cockburn (2000).
where use cases are considered as narrative descriptions of an envisioned system).

Jarke (1999) classifies scenarios at two different levels: scenarios describing the current behavior of a system and scenarios describing the desired behavior which satisfies new goals and requirements (fig. 3.17). The two forms of scenarios are integrated in a design process where the description of the current state is located in early phases of design and the envisioned scenario is created after analyzing both the new forces and the current scenario.

Interviews (Wood, 1997) have a comparable goal to scenario-based design. Current work practices and a future modified practice should be envisioned. Unlike scenario-based approaches, semi-structured approaches use guiding questions to find out details about the different abstraction levels of the analyzed system. They emphasize the role of a facilitator who steers the reflection in order to create different formal models of the system.

Low- and High-Fidelity Prototyping: Prototyping serves as a means for getting a more realistic impression of the system in use. We distinguish between different levels of fidelity: low fidelity prototypes use simple technologies (like paper based prototypes or graphical mock-ups) to let the user test the user...
### Relation between Participatory Design and the requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Wholeness</td>
<td></td>
</tr>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td></td>
</tr>
<tr>
<td>R3: Reflection</td>
<td></td>
</tr>
<tr>
<td>R4: Process Awareness</td>
<td></td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td></td>
</tr>
<tr>
<td>R6: One Problem/Time</td>
<td></td>
</tr>
<tr>
<td>R7: Field of Forces</td>
<td></td>
</tr>
<tr>
<td>R8: Explicit Forces</td>
<td></td>
</tr>
<tr>
<td>R9: End-User Involvmt.</td>
<td></td>
</tr>
<tr>
<td>R10: Education</td>
<td></td>
</tr>
<tr>
<td>R11: Explicit Conseq.</td>
<td></td>
</tr>
<tr>
<td>R12: Integration</td>
<td></td>
</tr>
<tr>
<td>R13: End-User Dev.</td>
<td></td>
</tr>
<tr>
<td>R14: Share Best Pract.</td>
<td></td>
</tr>
<tr>
<td>R15: Quality Review</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.6** Relation between Participatory Design and the requirements.

Interface. A set of screen “pictures” can be arranged on a story board to represent dynamic behavior of the system. Wizard-of-Oz studies (Dahlbäck et al., 1993) provide a higher fidelity. They let humans simulate the computer processing in order to let users experience the system behavior. Finally, working prototypes provide the highest level of fidelity.

**End-User Development** moves the user in the position where he can create his software without the help of a designer. This will be discussed in the next section.

Table 3.6 summarizes the requirements addressed by Participatory Design.

### End-User Development

End-user development (cf. (Sutcliffe and Mehandjiev, 2004) for an overview of current approaches to end-user development) extends participatory design by empowering end-users should to modify their software on their own (satisfying R13). One technology for end-user development is the provision of tailorable software that allows end-users to make adaptations to the software while they are using it (Kahler et al., 2000) (satisfying R3 and R12).

The challenge is to provide opportunities for tailoring that are appropriate for the people who need to perform the changes. There are three different levels on which tailoring can take place. On a high level, appropriation of content structures serves as tailoring since it can modify the interaction with the content. On a lower level, enabling or disabling functionality serves as a means for reshaping the interaction of a user with the tool. Yet on a lower level, composition of components (e.g., (Slagter et al., 2001) or (Teege, 2000)) allows the user to build new combinations of tools. It is obvious that the higher abstraction levels are closer to the users’ work context, whereas the tailoring actions on the lower level require that the user learns how to use the tailoring mechanisms in order to perform modifications that exceed the possibilities of tailoring actions at a higher level.

As with other design problems, tailoring requires insight in the problem at hand, the various conflicting forces, and possible solutions that are provided by the tailored application. The more technical the level of tailoring becomes the more design knowledge is required by the user. How to educate the users to act in the desired tailoring level is a challenging question (R10).

Fischer and Giaccardi (2004) identified different types of users regarding their level of influence to the system (fig. 3.18). At the consumer-end of the spectrum
stands the passive consumer who does not influence the received information (e.g., when watching TV without any control over channels). On the other end of the spectrum stands the meta-designer, a user who is capable of influencing the way how the design is shaped.

The closer a user comes to a technical level, the more programming skills are required. Finally, the user will reach the level of end-user development. At this level, the user uses domain-level programming languages to codify his actions in software. A well known example is the use of scripting languages in Microsoft Office. The user can perform activities and let the system record these activities. The recorded program can be replayed but also edited by the user. The example shows how the user can be introduced to a domain-language leveraging him from his current user level to the level of a programmer. Learning takes place by presenting examples to the user and asking the user to modify the examples according to the desired context.

Wulf and Rohde (1995) proposed to embed tailoring in the development process 3.19. The extended the STEPS software development process initially proposed by Floyd et al. (1989): The user interacts with the developers in the project planning, the release planning, and the system design. The system design results in a system specification that is used as input for the software realization performed by the development team and the embedment preparation (the preparation that the system can be used at the workplace) performed by the user. The resulting system version is then used in parallel to the maintenance work until a new release is planned.

As one can see in fig. 3.19, the user is supposed to tailor his application during system use. Developers provide assistance if the tailoring is too complicated for the user.

Table 3.7 summarizes the requirements addressed by End-User Development.
3.1.5 Patterns in the Context of Software Development

The process models investigated so far have focussed on developer-customer interaction and the analysis of requirements. Educational aspects, the sharing of best practices, and the use of structure-preserving transformations have, however, not yet been addressed. This is the reason why we will have a closer look at the application of patterns in software development. Although they do not represent process model, they can serve as tools to inform design and therefore complement the design methods discussed so far.

History of Object-Oriented Design Patterns

Patterns have been discussed in the software development community since OOPSLA 1987, where Kent Beck presented a case study of the development of a semiconductor testing environment (Kerth, 1988). The examples that Beck provided showed patterns on a user interface level (e.g., a pattern demanding the use of verbs in menus). The idea was initially accompanied with a high level of scepticism. Adele Goldberg, e.g., commented on the pattern approach in relation to software development problems that “we have merely replaced our old programming problem with a new one.” (ibid.)

In 1992, Peter Coad published the first widely distributed article discussing the
relation between Alexander’s work and the field of software construction in the Communications of the ACM. He introduced patterns as tools to standardize small piecework “into a larger chunk or unit. Patterns become the building blocks for design and construction. Finding and applying patterns indicates progress in a field of human endeavor.” (Coad, 1992, p. 152) Coad argued that the main contribution of patterns to object-oriented analysis and design is that they shift the focus away from small-scaled objects and classes to the interplay between objects. Patterns can serve as tools to better understand the interaction of objects with other objects and the whole system context. Consequently, he defined an object-oriented pattern as “... an abstraction of a doublet, triplet, or other small grouping of classes that is likely to be helpful again and again in object-oriented development.” (ibid, p. 153)

Compared to Alexander’s patterns, Coad did only focus on the static and dynamic object structure. He did not state explicit problem statements, nor did he discuss benefits and drawbacks of the pattern. Each of Coad’s patterns was illustrated with a class diagram, a trend that was later on imitated in other pattern languages.

At the same time, a group of object-oriented software designers discussed the tradition of design-books that were known from other engineering disciplines (e.g. mechanical construction). The possibilities for such a design book were discussed at a workshop at OOPSLA’92 (Anderson, 1992). This workshop was important since it resulted in the first appropriation of a pattern structure that included all aspects of Alexandrian patterns. A pattern should contain

- TITLE (punchy, carefully chosen)
- DIAGRAM (symbolic, suggestive)
- MARGINAL ICONS (for the quick flipper)
- PROBLEM (brief statement)
- SOLUTION (brief statement)
- REQUIREMENTS/DEPENDENCIES
- DETAILS OF SOLUTION
- FURTHER CONSEQUENCES
- DISADVANTAGES
- SPECIFIC ATTRIBUTES
- VARIATIONS & RELATED ENTRIES
- AVAILABILITY
- EXAMPLE
- IMPLEMENTATION
- CODE FOR EXAMPLE
- AUTHOR, DATE, ACKNOWLEDGEMENTS
- REFERENCES

(ibid, p. 111)

Another result was that Gamma et al. (1993) – the Gang of Four (GoF) – extracted good object-design structures that were frequently found in the code of object-oriented GUI frameworks. These patterns matured to the most influential publication on software design patterns (Gamma et al., 1995).

The rationales behind design patterns for the GoF were to
provide designers with a shared vocabulary to discuss and comment design alternatives,

provide designers with micro-architecture building blocks that they can compose to generate more complex architectures,

ease the learning for using frameworks by referring to design patterns in the framework’s description, and most important,

discuss the trade-offs that are related to a specific design decision. (Gamma, 2002)

As for the patterns of Coad, the audience of the GoF patterns were software designers. This justified the used pattern format that again included many design diagrams and concrete code examples. A good explanation why object-oriented design patterns should be formulated in a more technical way was given by Gabriel (1996, p. 33):

“The first place where I think I differed with others’ interpretation of Alexander’s work was in defining the users or inhabitants of a piece of software as its coders or maintainers. At least one computer scientist identified the ‘user’ of a piece of software as the end-user. This appears to make sense at first, but when you read Alexander, it is clear that a ‘user’ is an inhabitant – someone who lives in the thing constructed. The thing constructed is under constant repair by its inhabitants, and end-users of software do not constantly repair the software, though some might want to.”

While Gabriel is right for low-level technology issues, his interpretation needs to be questioned for high-level issues that should be of relevance both for the developer and the end-user. Borchers (2000a) goes so far that he states that the GoF patterns were no patterns since they did not bridge the gap between lay people and professionals. If we consider environments that are tailorable, the inhabitants are end-users. For that reason, high level aspects of software that address requirements and user interface issues need to be explained with patterns that have the developers and the end-users as target audience.

Therefore, there is the need for a broader view on patterns that moves from the low level of abstraction, introduced by the GoF patterns, to a holistic view on the designed artifacts that replaces the artifact in its system context. However, Coplien (2003) may be right with his observation that “the majority of programmers are entrenched in the Gang of Four patterns and have difficulty moving beyond those to system concerns.”

Table 3.8 summarizes the requirements addressed by Software Patterns.

From Object-Oriented Design Patterns to Patterns for HCI Development

Maybe, the most important contribution of the GoF publication was that it gave other software designers a template to capture best practices as patterns. It laid the ground for the hillside group, an organization dedicated to collect new patterns
and organize conferences with the goal of raising the quality of the growing body of
pattern literature.

As demanded in the previous section, patterns in the context of analysis required
a more user-centered format. Many authors moved back to a more literate form that
was closer to Alexander. Especially the problem and solution statements that were
not quickly accessible in the GoF patterns emerged as most important sections of a
pattern.

A first approach to this direction was the book on *Analysis Patterns* (Fowler, 1997). Although the presentation was still focussed on designers (including class di-
agrams and modeling advice), the problem statements addressed real user problems
like trading in the context of business workflows or the management of plans.

An important landmark regarding end-user centered patterns is the evolution
of patterns for shaping human computer interaction. These patterns frequently
follow the Alexandrian form and address problems from the perspective of a user
while still addressing questions of a developer. Erickson (2000) argued that patterns
could play the role of a *Lingua Franca* for design that helps end-users and developers
in communication.

Examples are pattern languages addressing web design. Martijn van Weijlje’s
published a collection of Web design patterns (van Welie, 2005). These collections
focus on the user and are intended to be used during requirements elicitation and web
site design. The latter collection has been further used to evaluate existing museum
web sites in the Netherlands (van Welie and Klaassen, 2004). The collection of found
patterns suggests a sequence of structure-preserving transformations that could lead
to the web site. However, the real process of design was not addressed by the study.

The pattern format used by van Welie is very brief and supports quick access.
An example pattern – the BREADCRUMBS pattern – from the web design pattern
collection is shown in fig. 3.20. It lists a brief problem statement, followed by the
context of the pattern (labeled *Use When*). The next part is a bold short form of the
solution and a longer (non-bold) part that provides details of the solution. Finally,
the solution is explained (the section labeled *Why*) and additional visual examples
are provided.

BREADCRUMBS is an interesting example since it also appears in other pattern
languages for web design. (Duyne et al., 2002, p.565) formulates the problem of
LOCATION BREAD CRUMBS as follows:
3.1. SOFTWARE DEVELOPMENT PROCESSES

Figure 3.20: An example pattern from web pattern language. Reprint from http://www.welie.com/patterns/showPattern.php?patternID=crumbs.
Problem: It is easy for customers to get lost on Web sites, losing track of where they are in relation to other pages on the site.

Solution: Provide bread crumb links that show how to get from the homepage to the current page and back. Use a string of back links and separate them by a ‘pointing’ (>) character.

(ibid, p.567)

Van Duyne et al add an additional related patterns section that links to other patterns which should be considered as well. Thus, the patterns in the van Duyne collection are better linked.

Another instance of Breadcrumbs can be found in Ian Graham's web usability pattern language (Graham, 2002). He also uses an Alexandrian presentation format.

Problem: How can users see where they are relative to the site’s home page, which probably offers more navigation options then other pages?

Solution: Do not rely solely on breadcrumbs for navigation unless you are very short of space. Breadcrumbs should be complimented by a NAVIGATION BAR (25) and/or other navigational devices. Some navigation however may only be available from the home page, but breadcrumbs need to go on every page. Put breadcrumbs near the navigation bar and always at the top of the page. Make it clear that they a secondary form of navigation, perhaps by using a lighter or smaller font. Highlight or embolden the current location. Separate them with a > symbol or other pointer-like device. Clarify their function by saying ‘you are here’. Don’t use them in place of a well chosen page name.

Other known descriptions of the bread crumbs pattern are the ACTIVE REFERENCE pattern (Rossi et al., 1995).

Breadcrumbs also appear in other interaction design guidelines. For instance, the usability glossary at www.usabilityfirst.com includes an entry for breadcrumbs:

on websites, a form of navigation where the current location within the website is indicated by a list of pages above this page in the hierarchy, up to the main page. For example, if you were browsing the products at a department store, you might see the following hierarchy when you’re on the Sneakers page:

Home > Products > Clothes > Shoes > Sneakers

Each of the categories above the current page is usually a link to the corresponding category page.

Compared to the pattern descriptions, the description in the glossary only elaborates on the solution. So, the main advantage of the pattern description is that the problem is made explicit.

All proposed patterns provide a short discussion of the implications when applying the pattern. The repetition of the pattern already shows that the solution is prominent. Additional rational, why the solution works (e.g. the section in the pattern by van Welie) further helps to convince the reader.

However, the examples raise two problems regarding patterns in HCI:

1. *How important is novelty for patterns?*
3.1. SOFTWARE DEVELOPMENT PROCESSES

The three patterns appeared in a time frame of 8 years. All were slightly different in the way they stated the problem and the solution, but the core of the pattern was identical for all instances.

Novelty is a dilemma for patterns. Patterns should be mature which implies that the solution was applied frequently. Patterns should be part of a consistent pattern language which implies that they have a consistent structure throughout the language. On the other hand, having replicate patterns makes it more difficult to value progress in the pattern movement. Especially, if the other instances are not referenced, this is problematic for the community as a whole and limits their scientific credibility.

A rephrasing of a pattern can be valid in a case where the problem statement becomes more clear. Since patterns are educative tools, better formulations of problem statements are a significant contribution (this was the case from Rossi’s version to the version by van Duyne). The other cases should find ways to reference the patterns rather than rewriting them. This, however, requires that pattern authors agree on a common pattern structure and referencing scheme.

At the CHI-03 workshop on HCI Patterns: Concepts and Tools (Fincher et al., 2003), (Fincher, 2003), the participants\textsuperscript{15} created a first version of a pattern template that can be used to unify the appearance of the different pattern languages that evolved so far. The need for this step was seen because of a larger growing set of so far isolated patterns in the area of HCI. Connecting the different pattern languages is a challenging task and the translation of the different collections into a unified representation was one first step to solve this problem.

The pattern template was described by means of an XML DTD called PLML (Pattern Language Markup Language - pronounced “PellMell”), which is shown in fig. 3.21.

Tools that present patterns to readers can interpret pattern documents according to the DTD and visualize them using their own structure. The context could, e.g., be displayed after the problem statement (as it is the case with van Welie’s version or in front of the problem statement (as it is the case in van Duyne’s version).

2. How much evidence does a pattern need?

3. patterns!evidence

Alexander argued that a valid pattern should be observable over and over again. The GoF made this rule less restrictive by only asking for three known uses. The rationale behind this decision is obvious: since computer science is still a young discipline, it is difficult to find dozens of known uses. But even

\textsuperscript{15}Sherman Alpert, Jan Borchers, Sally Fincher, Janet Finlay, Ashraf Gaffar, Sharon Greene, Scott Henninger, Javier Hernández, James Linn, Pedro Molina, Till Schümann, Daniel Sinnig, John Thomas, Jenifer Tidwell, and Martijn van Welie.
Figure 3.21: PLML developed by the CHI2003 Pattern Workshop Participants.

if many known uses exist like in the example of Breadcrumbs, does this mean that the pattern is good?

Even what may seem obvious can be used differently by real users. For the case of breadcrumb navigation, Lida et al. (2003) conducted a study in which the use of this navigational aid was measured and compared to other navigational techniques:

We found the overall usage of the breadcrumb in site navigation to be low.Breadcrumb users were not found to be more efficient than users who did not use the breadcrumb. Participants used a variety of navigational means, such as the Back button, left and top navigation bars, and searching to find the information instead of or in addition to the breadcrumb tool.

Patterns for HCI should thus provide scientific evidence – or at least discuss possible limitations.
3.1. SOFTWARE DEVELOPMENT PROCESSES

Table 3.9: Relation between HCI Patterns and the requirements.

<table>
<thead>
<tr>
<th>Pattern Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Wholeness</td>
<td>Wholeness</td>
</tr>
<tr>
<td>R6: One Problem/Time</td>
<td>One Problem/Time</td>
</tr>
<tr>
<td>R7: Field of Forces</td>
<td>Field of Forces</td>
</tr>
<tr>
<td>R12: Integration</td>
<td>Integration</td>
</tr>
<tr>
<td>R3: Reflection</td>
<td>Reflection</td>
</tr>
<tr>
<td>R8: Explicit Forces</td>
<td>Explicit Forces</td>
</tr>
<tr>
<td>R13: End-User Dev.</td>
<td>End-User Dev.</td>
</tr>
<tr>
<td>R4: Process Awareness</td>
<td>Process Awareness</td>
</tr>
<tr>
<td>R9: End-User Involvmt.</td>
<td>End-User Involvmt.</td>
</tr>
<tr>
<td>R14: Share Best Pract.</td>
<td>Share Best Pract.</td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td>Iterative Design</td>
</tr>
<tr>
<td>R10: Education</td>
<td>Education</td>
</tr>
<tr>
<td>R15: Quality Review</td>
<td>Quality Review</td>
</tr>
</tbody>
</table>

Jan Borchers published another treatise on interaction patterns (Borchers, 2001). He used an Alexandrian style for the pattern presentation. However, his patterns are consciously not bound to large empirical evidence. Instead, he uses patterns to convey research ideas, especially new ideas in the field of input devices and ambient computing.

All provided examples highlight the need for collecting patterns in a pattern language to form a coherent representation of the design space.

Table 3.9 summarizes the requirements addressed by HCI Patterns.

Experiences with Patterns in the Context of the Software Development Process

The pattern community found technical design patterns very useful (Beck et al., 1996):

“We have found that design patterns: 1) provide an effective ‘shorthand’ for communicating complex concepts effectively between designers, 2) can be used to record and encourage the reuse of ‘best practises’, 3) capture the essential parts of a design in compact form, e.g. for documentation of existing software architectures”

The main focus of software pattern use has been on the creation of software artifacts during the design and implementation phases. Patterns were used to guide the design of class models. Patterns can be understood as complementing tools for modern software development processes (Schmidt, 1996).

There are only few software development processes that explicitly focus on the use of patterns. However, several authors have highlighted the applicability of patterns for different tasks in the development process:

Requirements elicitation: Kovitz (1999) claimed that patterns are always in the designer’s mind when looking for requirements. “Rigorous research and definition of requirements is possible only in relation to a specific design pattern.” (ibid, p. 23) He proposed to use the patterns as a tool for raising the right questions (cf. Helm (1995) and Schmidt (1996)). The resulting requirements document should provide answers to these questions but still be connected to the pattern. Beck et al. (1996) also report on a case where the study of patterns in a pattern catalogue revealed requirements that were considered as important for the project but were not part of the requirements specification so far.
Transition between analysis and design: Helm (1995) saw the importance of patterns at the transition from analysis to design. The patterns should inform the developers when mapping requirements to class models. Schmidt (1996) pointed out that software design patterns especially help to keep aware of non-functional requirements such as flexibility of the created model.

Documentation and evaluation: One example of the use of patterns for analyzing existing systems has already been reported: the analysis of museum web sites (p. 68). Wesson and Cowley (2003) report on the application of patterns in the analysis of an E-Commerce site. They used the pattern collection of van Duyne et al. and described sequences that they followed during analysis. Vlissides reported on the use of patterns for design reviews. Again on a technical level, the patterns are used to compare existing design with good shape (and perform improvements) (Beck et al., 1996).

Communication: A better mutual understanding between developers who were familiar with the same set of patterns has been reported by all authors cited above. However, learning the pattern is an effort that should not be underestimated.

Especially structural software patterns evoked ambiguous reactions in the context of agile processes. The existence of patterns may tempt developers to include more and more patterns in their software for the purpose of higher flexibility and internal “beauty” of the resulting system. However, this can get in conflict with the agile directive that the software should do exactly what the customer wants and nothing else. (Gamma, 2002) thus said that it is also important to find ways to remove (unneeded) patterns from a design. This problem is less important in the context of high-level patterns since these patterns help to map user requirements to a design.

Borchers (2000a) proposed a pattern-oriented usability engineering life cycle. He identified the need of pattern use for all phases during this life cycle:

1. In inceptive iterations, patterns should be written to capture knowledge about the application domain. This will help developers to learn more about the application domain and users will be forced to structure their expertise and make it explicit.

2. Patterns should also be written in the competitive analysis of existing product so that good UI solutions are captured as HCI patterns.

3. The definition of usability goals leads to forces for abstract HCI patterns.

4. In a phase of parallel design, different teams use patterns as design guidelines while creating initial independent designs (note that the variety of designs will be reduced by the use of patterns).

5. Borchers proposed to support the following phase of participatory design by an exchange of the pattern languages to foster mutual learning and understanding. This, however, implies that users and developers have a pattern language.
3.1. SOFTWARE DEVELOPMENT PROCESSES

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td>R7: Field of Forces</td>
<td>R12: Integration</td>
</tr>
<tr>
<td>R3: Reflection</td>
<td>R8: Explicit Forces</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.10: Relation between Pattern-Oriented Usability Engineering and the requirements.

6. During coordinated design, system consistency is achieved by connecting the design of the designed artifact with the existing design elements in the artifact’s context. The use of a low level HCI pattern language can support this phase.

7. The next phase in the life cycle is the application of guidelines and standards. Borchers proposed to formulate these standard and guidelines in a pattern format to make them easily readable.

8. Prototyping is the phase, where software design patterns come into play. They help designers to better communicate on the created technical components.

9. During empirical testing of the prototype, patterns can be attached to the found problems.

10. The prototype is improved in a phase of iterative design. As in the prototyping phase, software design patterns and HCI patterns can inform the design in this phase.

11. Finally, when the system is in production, feedback is collected from the users. This feedback should, according to Borchers, be used to improve the patterns (i.e., if a solution is not working, the pattern should be rephrased).

Table 3.10 summarizes the requirements addressed by Pattern-Oriented Usability Engineering.

Dearden et al. (2002) discussed the use of pattern languages in the context of participatory design. They proposed a process that evolved in three phases. During all phases, a designer-facilitator assisted users in understanding and applying the patterns or creating their own design solutions. The phases are:

1. Introduction of patterns, where the facilitator encourages the user to look into the patterns of the level of abstraction that is currently relevant for the design task.

2. Users do early prototyping using storyboards or paper-prototypes and sketch their own ideas and adaptations of the patterns.

3. Iterative development in which the prototypes become more realistic in each iteration until they are the final application.
In a complementary study, the same group of authors reports on the use of pattern languages in their process (Finlay et al., 2002). Results showed that pattern languages can successfully be applied in the context of participatory design. Users especially relied on the examples and the bold-faced solution statements of the patterns. They managed to adapt patterns to their contexts (although some users regarded the patterns as truths and discarded their own ideas of design). They also brought up new pattern ideas or found flaws in the pattern language.

Discussion with respect to the Groupware Development Requirements

The experiences from different application areas of pattern languages show that patterns can serve as supportive means in the design process. When focusing on end-user involvement, high-level patterns are especially important. These patterns have to be written in a format that is readable by both end-users and developers. Patterns from the field of HCI seem more suitable at this level of abstraction than technology centered patterns in the GoF style.

The connection between patterns and processes is still very limited. The most comprehensive approach in this direction has been proposed by Borchers. But since he focuses on a usability design process, he does not address the needs of very uncertain requirements and especially not the problem of changing requirements during system use. The process proposed by Dearden et al. shows that parts of the participation laid out by Borchers can lead to success in participatory design. However, if we consider participation at the level that was envisioned by Alexander, an ideal participatory process should also address situations where the users act as implementors of their environment.

3.1.6 Discussion of Software Development Processes

In this section, we addressed the problem of groupware development at an abstract level: the development of software in general and especially the development of interactive systems.

The process models discussed in the first part of this section showed to be more or less flexible to changing requirements (ranging from waterfall processes to XP). End-user involvement took place at different levels, ranging from participation in the requirements specification (in the traditional waterfall process) up to a continuous interaction between development team and the end-user (in XP).

Participation of the end-user also varies regarding the phase in which the participation took place and the technique used for structuring the interaction with the user.

Patterns – although often used as technical tools for fostering communication between designers – can be successfully applied in user-developer interaction. However, if they are not combined with a flexible process model, they have only limited impact and support the user only in small phases of the overall development process.

Table 3.11 summarizes the contributions of the different approaches to the re-
3.1. SOFTWARE DEVELOPMENT PROCESSES

Table 3.11: Relation between the software development approaches and the requirements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall Processes</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Unified Process</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>eXtreme Programming</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Scrum</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Feature-Driven Dev.</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Participatory Design</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>End-User Development</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Software Patterns</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>HCI-Patterns</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Pat.-orient. usab. eng.</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
</tbody>
</table>

requirements for a groupware development process. The table shows that the sum of all approaches provides a satisfying coverage of the requirements but no single approach can solve the problems of groupware development alone.

The table shows that a combination of an agile method like XP with HCI patterns comes very close to the full set of requirements. Weak points of XP (e.g., end-user development and structure-preserving transformations) are supported with a pattern approach. In addition, aspects regarding education and sharing of practices are not addressed by the development processes but sufficiently covered with the use of HCI patterns. The combination of patterns and a participatory design approach as proposed by Borchers (2000b) is a good step towards combining iterative design with educational aspects strengthened by the use of patterns. While it reasonably addresses the solution of small design problems, it still needs to be extended with better support for coordination in order to design groupware solutions with larger user communities.

The state of the art presented up to now considered groupware development as a development activity that is related to software development only. The next section will discuss to what extent the different development methods are found and modified in groupware development.
3.2 Groupware Development Processes

In this section, we will address the state of the art from a groupware development perspective and show how ideas from software development have been applied to groupware development.

3.2.1 A critical review of Dewan’s waterfall process model for groupware development

Dewan (2001) presents a traditional design process for groupware: He combines a waterfall process with additional steps for analysis. His model is shown in fig. 3.22. It is primarily intended as a development model in research contexts. Although Dewan explicitly states that the waterfall process can be executed several times, the focus of the waterfall process comprises the whole product. Therefore, we can not consider the process as iterative.

However, the different steps in the waterfall process are appropriated to the problems in groupware design.

In the first functional decomposition step, the application’s problem space is related to the different areas of groupware research. Examples for functional units are multi-user undo, access control, or group awareness. Having such a phase as an initial step in the process is very problematic: it addresses technology-driven questions before examining scenarios of use and finding out what the user really wants. This leads to technology-driven groupware-design that may lack situatedness.

However, the process model of Dewan includes steps of analysis. It suggests that scenarios should be imagined by the developers or that a CSCW designer observes real-world collaboration and derives scenarios for groupware support. End-users are, according to Dewan, not capable of performing this step since they lack the required insights in the field of CSCW (ibid., p. 86). This attitude is problematic for the process. As we have shown in chapter 2, groupware support should be situated in the user’s context. An imagined scenario of use could work, but it is more or less a question of luck since the imagined context does not necessarily match the real user’s context.

The requirements step is again described by Dewan from a technology centered view. Rather than focussing on functional requirements, he discusses technology constraints that emerge in groupware systems such as the reuse of existing applications in a shared window system or the provision of group awareness and different modes of coupling.

In the next step, the interaction model is defined. It describes the ways how users interact from a high level perspective. This includes finding a metaphor (like the metaphor of a ROOM) for collaboration. Dewan only vaguely describes who performs this step, but from the context, one can conclude that he intends that the designers or CSCW researchers are in charge of finding the metaphors. Again, this is problematic since it does not ensure that the chosen metaphors are consistent with the user’s metaphors.
3.2. GROUPWARE DEVELOPMENT PROCESSES

Figure 3.22: An adapted waterfall process for groupware development (Dewan, 2001).
Table 3.12: Relation between Dewan’s Waterfall process and the requirements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td>▽</td>
<td>R7: Field of Forces</td>
<td>▽</td>
<td>R12: Integration</td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td>▽</td>
<td>R10: Education</td>
<td></td>
<td>R15: Quality Review</td>
</tr>
</tbody>
</table>

The next proposed step is the design of a user interface. According to Dewan, this “does not have much to do with collaboration” (ibid., p. 92). This argument should be carefully questioned: there are many aspects in which the user interface for groupware applications differs from traditional user interfaces. Examples have been provided in the research by Gutwin and Roseman (1996) or Greenberg and Roseman (2003). It is thus important to consider the visualization of other user’s activities and presence.

The next three steps focus on the provision of the technical infrastructure for the groupware system. They include, e.g., discussions on distribution schemes. Again, this shows a technology centered view on design.

The realization phase consists of a two phased implementation: first, developers implement an infrastructure that is subsequently used to build the domain-specific application.

Finally, different stages of evaluation prove the appropriateness of the resulting groupware system. Activities on a technical level include inspection regarding the requirements, simulations (mainly of inputs and critical phases to prove the functionality of the underlying technical infrastructure), the application of standard problems known in the field to compare the resulting system to other research prototypes, and self studies. On a user-centric level, lab studies and field studies help to better understand the effects of the groupware system regarding test subjects or the real user in the field.

The process meets only very few of the groupware development process requirements: It provides process awareness (R4) to a limited degree since it has clear steps and helps the user to understand where he should direct his focus. It fosters limited reflection (R3) in the different stages of evaluation, the reflection, however, is not demanded from the user but performed by the developer. It also has no consequences on the daily use of the tool.

The main reason, why the process does not sufficiently cover the requirements is that it is intended as a means to structure the development of research prototypes. But since the field of CSCW is still young, many systems originate from such research prototypes. Thus, the process reflects a large part of the state of the art in groupware development.

Table 3.12 summarizes the requirements addressed by Dewan’s Waterfall process.
3.2.2 Agile Processes for Groupware Development

Rittenbruch et al. (2002) report on the application of eXtreme Programming practices for the design of a social portal (also (Mansfield et al., 2002)). The most important observation made in this project was that the whole user community could be empowered to write user stories (compared to a user representative in XP). For that reason, the system had an electronic feedback mechanism that allowed users to write stories. However, this change to the planning game required additional roles:

**Ad-hoc customer:** Before the early prototype could be released, there have to be users who play the role of the customer without being real users. These ad-hoc customers help to shape the first prototype until real users can create stories.

**User-evaluation customer:** In traditional XP, the customer is a representative of future users (or business). The problem with this role is that the customer stays together with the design team and can thus gradually become a designer. Although this is good from a participation perspective, it bears the danger that the customer starts to think like a designer. Problems that are encountered by new users can thus stay undetected. The user-evaluation customer has the task of observing new users in their system use, look for interaction breakdowns, and create user stories for these users.

**Design customer:** In XP, only customers are allowed to create new cards. This can have the effect that the customer designs the system from his personal background and experience. New innovative technologies that are not known by an average customer, will often not be considered. Since many areas of groupware are still in their infancy, average users may design inadequate solutions.

**Gardener:** Since all users were allowed to create user stories the number of stories grew very fast. Therefore, users and developers had difficulties to coordinate the proposed stories. The gardener was responsible for clustering and maintaining the user stories.

This example shows how agile methods can be appropriated for the design of collaborative applications. The main point is that the real user community of the system should be encouraged to participate in perfecting their environment.

The main difference to XP is a better involvement of the end-user (R9): While the user involvement of XP focuses on interacting with one customer representative, the extended XP process proposes to involve all users. The role of the gardener helps to improve process awareness (R4) among all users: Since the story cards are stored in a groupware system, all users can browse and modify the cards.

Table 3.13 summarizes the requirements addressed by Extended XP process.

The only questionable role is that of the design customer. It is a work-around for a lack of experience with new technology at the user’s side. However, if the user would be educated (R10), he could be trained in these new trends and in insights.
of CSCW research. Then users could create user stories that take the benefits and drawbacks (the forces) into account (R7, R8, and R11).

### 3.2.3 Application of Participatory Design Methods

Participatory design methods have become quite prominent in groupware design, e.g. (Grudin, 1994). They argue that groupware systems have to be designed for modifications to suit evolution of use. An important issue of tailoring groupware is the impact of tailoring operations: if a tailoring operation impacts other participants, they may, as a group, perform the tailoring. In that case, the users need a method to decide on a common tailoring style and a process, how tailored groupware systems are reintegrated in the group (Fernandez et al., 2002) and thus encourage sharing of best practices. Most tailoring environments do not consider this.

All levels of tailoring are relevant for groupware contexts. However, tailoring actions on a high level (modifications of the content structure) can have a much higher impact since the content is “inhabited” or used by many users. However, as Grudin stated “using tailorable systems is a good step to providing flexibility, but to tailor effectively is a challenge, because people are not conscious of detailed organizational functioning and how changes affect other people.” (Grudin, 1994, p. 99)

The use of collaboration scenarios has been applied to different development projects in the field of CSCW. Examples are the PoliTeam project (Cremers et al., 1998) that had the goal of fostering collaboration between federal ministries in Bonn (the former German capital) and Berlin (the current capital) and the SAIK project (Bardram, 2000) that applied a scenario approach to hospital information systems in Denmark. Especially the narrative nature of scenarios served as a good means to bridge the gap between analysis and design. In the next two subsections, two concrete process models that are based on participatory design will be discussed in detail: **Organizational and Technical Development (OTD)** and **Seeding, Evolutionary Growth, and Reseeding (SER)**.

### Organizational and Technological Development

With OTD (Integrated Organizational and Technological Development), Wulf et al. present an evolutionary approach to appropriate group processes and technology to support the processes during system use (Wulf and Rohde, 1995; Wulf et al., 1999). This approach takes into account that three aspects need to co-evolve in a
work-setting: technology that supports group processes, organizational structures, and qualification of the group members. One has to decide for each problem which factors need to evolve to solve the problem. The OTD model is shown in figure 3.23.

Figure 3.23: The OTD model (Wulf and Rohde, 1995).

After the problem is established, a group of stakeholders investigates the current state of the organization. The group should include all people who are affected by the problem. If the investigation comes to the conclusion that the problem is relevant, they create alternative options. They utilize different techniques from participatory design (stories, software prototypes, etc.) to foster mutual understanding.

According to the type of problem, the group determines technology, organizational, and qualification factors that should be changed. In case of a technology issue, the users appropriate the system by means of tailoring or developers perform a redesign (including the possibility for removing the system). In case of an organizational factor, the group processes are changed (which does not mean that technology also has to change). Finally, if the problem is a result of qualification
Table 3.14: Relation between OTD and the requirements.

| R2: Struct.-Pres. Trans. | R7: Field of Forces | △ | R12: Integration |
| R5: Iterative Design | R10: Education | △ | R15: Quality Review |

deficits, a qualification program for the stakeholders is developed. Note that all three aspects can co-occur.

The result is a modified system that can be used until the next breakdown.

The OTD process comes very close to satisfying all requirements. It fosters reflection (steps evaluation of current work and stating a problem – R3), process awareness (R4), and iterative design (R5). By letting the stakeholders analyze the current state, the forces of the current situation become explicit (R7, R8). The end-user is involved throughout the process R9 and performs modifications through tailoring (R13). Finally, qualification for participation is considered as an integral part of the process, which satisfies R10.

Open issues are the sharing of best practices, the ways how changed artifacts are embedded in the environment R12 and the question whether or not the process creates coherent system designs (R1 and R2). The size of the iterations is not defined by OTD (R6).

Table 3.14 summarizes the requirements addressed by OTD.

**SER - Seeding, Evolutionary Growth, and Reseeding**

Fischer et al. (2001) propose a process model for collaboratively designing user’s environments (especially collaboratively used information systems – IS). They propose three different stages for supporting design groups:

**Seeding.** At the beginning of a cycle, environment developers and future users capture their domain knowledge and requirements for the IS infrastructure. The resulting system serves as a seed for the next phase. It is a rather small collection of domain knowledge that is designed to evolve over time.

**Evolutionary growth** takes place when users use the IS as a tool to perform their work. Two main purposes are served by the seed:

1. it serves as a tool for solving domain related problems, and
2. it serves as a knowledge store since the solutions of former domain related problems are captured in the IS.

**Reseeding.** Whenever the use of former knowledge (obtained from results of the evolutionary growth phase) becomes complicated, environment developers and domain designers can start a phase of reseeding. The goal of this phase is to
Table 3.15: Relation between SER and the requirements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: Struct.-Pres. Trans.</td>
<td>R7: Field of Forces</td>
<td>R12: Integration</td>
</tr>
<tr>
<td>R5: Iterative Design</td>
<td>R10: Education</td>
<td>R15: Quality Review</td>
</tr>
</tbody>
</table>

reorganize, formalize, and generalize the information collected during evolutionary growth.

The main difference to other participatory design approaches lies in the emphasis of mutual learning. Users let the knowledge of all stakeholders in the design process evolve during use. This is why the authors of the SER model call the involvement of users informed participation. Participatory design instead focuses on the construction of artifacts.

According to Fischer et al. (ibid.), informed participation is supported by referring to comparable problems of the past. How these problems and solutions should be represented to support quick access to captured knowledge is not discussed in SER.

Table 3.15 summarizes the requirements addressed by SER. The strengths of SER are in the process of reflection (while appropriating the system – R3) and in the sharing of reflection results (R14). The reseeding phase ensures that the shared knowledge will be structured and assessed (R15).

3.2.4 CSCW Patterns

First patterns for the design of groupware applications focused only on technical aspects like replication, session management or concurrency control (Guerrero and Fuller, 1999), (Schüller et al., 1999), (David et al., 2003). They were written in a developer-centered format (comparable to the GoF patterns). Thus, these patterns can support developers in finding out how to apply a specific solution but are not suited to support communication between end-users and developers.

In ethnography-centered CSCW research, patterns have been used to categorize observations in field work (Martin et al., 2001; Crabtree et al., 2002). In this context, patterns have a descriptive goal. They thus do not state problems and solutions but rather report on recurring behavior in the observed social system.

An example pattern from an ethnographic pattern language is provided in fig. 3.24. The pattern states what the recurring behavior is (Essence of the Pattern section). The following part speculates on reasons why a specific behavior recurs and why it could be considered as helpful. Illustrations in forms of Vignettes provide descriptions on ethnographic studies where the behavior was found. Again, the illustrations only describe and do not prescribe behavior. In the last section labeled dependability implications, the pattern informs the reader about possible problems.
CHAPTER 3. STATE OF THE ART

Figure 3.24: An example pattern from the pointer pattern language. Reprint from http://polo.lancs.ac.uk/patterns/WorkWithInterruptions.

in the observed context. This relates to a liabilities section of traditional pattern formats.

Martin et al. (2002) discussed the application of ethnographic patterns. They primarily serve as triggers for discussion in the phase of requirements elicitation and early design. They help users to reflect on their specific situations. However, these patterns do not suggest a specific design.

Several authors from the design patterns community have focussed on describing social processes with patterns. Frequently recurring themes are pattern languages describing software development processes (e.g., a pattern language focussing on the delivery phase of a software development process (Prince and Schneider, 2004)) or pattern languages discussing group management issues (e.g., the project on Organizational Patterns (Coplien and Harrison, 2004 (to be published)) or the pattern...
language by Manns and Rising (2004) that discusses the process of changing organizational structures and traditions). These patterns can be considered as social patterns. They are very valuable for understanding social processes in organizations. However, they do not assist the user in applying the proposed solution in a computer-mediated context.

Since the CSCW2002 workshop on Socio-Technical Pattern Languages, a series of workshops focussed on the appropriation of the pattern approach to the high-level design of groupware systems. The workshops helped to bring together the organizational and the technical view of patterns for group support. The fact that the workshops took place at different conferences also helped to gain different perspectives on the groupware patterns theme.

The workshops had the following foci:

- The CSCW2002 workshop on socio-technical pattern languages (Thomas et al., 2002) focussed on factors that create a pattern language from individual patterns and on attributes of socio-technical patterns. It brought together researchers from ethnographic, sociologic, economic, and technical backgrounds. It became clear that the interpretation of the nature of a pattern differed greatly. The ethnographic interpretation (see above sections) focuses on describing interaction without prescribing a design. Sociologic interpretations focussed on the human interaction and the question of how to shape processes. There was only a limited use of technology aspects. Technology-centered views approached the field of socio-technical patterns from the perspective of technical possibilities. In the discussions it became clear that both technology and social aspects need to be addressed in socio-technical patterns.

Another discussion regarded the form of patterns: Again, the participants presented different approaches ranging from literal Alexandrian presentations up to technology oriented presentations (including technical diagrams with a notation that the reader has to learn before being able to understand the pattern). One consensus was that the patterns should be readable by end-users. The role of narrative descriptions was appreciated by most of the participants.

- A sequel of the CSCW2002 workshop was the ECSCW2003 workshop “From good practices to patterns” (Herrmann et al., 2003b). The goal was to better understand the process of finding patterns in the research area of CSCW. Participants submitted initial versions of patterns and descriptions focus areas that could be interesting for pattern mining. During the workshop, these focus areas were further examined and pattern candidates as well as first prototypical patterns were detected. Again, the participants came from different disciplines including education, ethnography, software development, and socio-technical research. Although mostly convinced of the possibilities for the application of patterns in the context of CSCW, the participants stayed sceptic about the current practical implications. Two issues were considered as problematic: (1) the community did not recognize patterns as tools for groupware development yet and (2) there is still a lack of a comprehensive pattern language for
groupware development.

- At CHI2004, the organizers of the workshop on “Human-Computer-Human Interaction Patterns” (Schümer et al., 2004)\(^{16}\) approached the field of groupware patterns in the CHI context. The workshop built on experiences from former pattern-related workshops of CHI and tried to bring together experts in HCI patterns and groupware experts.

The discussions focussed on four different hot topics for the development of collaborative systems: (1) design issues for large scale communities with a focus on e-democracy, (2) the design of physical collaboration environments, (3) problems and perspectives for computer-mediated communication, and (4) technology issues regarding groupware service composition. The topics were chosen to gain a holistic view on the full spectrum of computer-mediated human-human interaction ranging from large scale political issues over medium scaled issues for small group interaction up to issues regarding the supportive technology.

![Diagram of patterns for supporting computer-mediated communication](image.png)

**Figure 3.25:** Patterns for supporting computer-mediated communication. Results of the CHI2004 workshop sub-group that I moderated.

Participants split into four groups and created a pattern landscape for each area. The structure of one result is shown in fig. 3.25 showing patterns in the context of computer-mediated communication.

3.2. GROUPWARE DEVELOPMENT PROCESSES

Besides drawing a landscape of relevant areas for pattern languages, the focus was on how to ensure the quality of a pattern language for human-computer-human interaction. Participants practiced Writer’s Workshops, a review technique that is widely applied at PLoP conferences\textsuperscript{17} but not common in the HCI patterns community.

- A focus group at EuroPLoP 2004 also stressed the process of mining Human-Computer-Human-Interaction patterns as well (Schümer and Zdun, 2004). Due to the expertise of the participants, it primarily focussed on patterns for supporting collaborative learning. While the other workshops primarily focussed on pattern representations and techniques for improving pattern quality, the EuroPLoP workshop put special attention on the described subject, namely, the collaborative learning process. The result was a small set of patterns regarding role assignment in CSCL.

- Finally, a workshop at INTERACT2005 ("Workshop on HCI patterns: Mapping User Needs into Interaction Design Solutions")\textsuperscript{18} will merge the results of the above workshops under the perspective of process integration of patterns.

Besides the workshop activities, two major publications appeared recently addressing the topic of groupware patterns. At CRIWG 2004, Lukosch and Schümer (2004a) discussed the applicability of patterns for groupware from a technical, but still narrative, perspective. The paper reported on low-level parts of the pattern language used in this thesis. Thus, we will not discuss the contribution of this work to the state of the art now, but rather have a look on how these patterns integrate with the rest of this thesis in chapter 4. The second recent publication argued for the use of socio-technical pattern languages in CSCW (Herrmann et al., 2003a). The authors captured discussions from the CSCW2002 workshop and proposed to combine the pattern approach with their socio-technical modelling technique SeeMe (Herrmann et al., 2000) and applied this combined pattern format to the research area of knowledge management. The problem that we see with this approach is the use of diagrams. Although diagrams can be successfully used as illustrative vehicles, they require learning effort regarding the notation. A pure Alexandrian format of patterns that focuses on narration could thus be more suitable.

The workshops provide a comprehensive picture of current activities in the collection of patterns for collaborative systems. Several researchers have made first experiences with collecting socio-technical patterns, but the sizes of the collected pattern sets only provides anecdotal evidence for the existence of patterns in CSCW. Unlike in the HCI patterns community, the groupware patterns community has not yet reached the state where a widely accepted pattern format is reached. Regarding

\textsuperscript{17}PLoP = Pattern Languages of Programs. The PLoP conferences have the goal of collecting patterns of various domains and bring together researchers from different fields related to the use or development of software. Unlike at other conferences, the focus is not mainly on innovation of new technologies but on the art of presenting patterns so that they have the highest possible outcome for an intended readership (e.g. for end-users).

\textsuperscript{18}organized by Franca Garzotto, Symeon Retalis, Till Schümer, Uwe Zdun, and Stephan Lukosch
When considering patterns as tools, they help to support the requirements of chapter 2 do different extents. Technology patterns (summarized in table 3.16) focus on supporting the developer. They help to develop the system in structure-preserving transformations (R2) from a technical point of view. When applying a technology pattern, the developer concentrates at one specific problem (R6) in its context (R7, R8) keeping in mind the consequences described in the pattern (R11). The pattern serves as an educative means between developers (R10). Technology patterns are however not focusing on the end-user (R9, R13). The PLoP community has established review and shepherding processes that help to raise the quality of technology patterns (R15). All major collections of CSCW technology patterns went through such a process or are currently shepherded. Thus, we can assume that they are of high quality.

Ethnographic patterns (summarized in table 3.17) are intended to be used in the discussion between designers and users. They serve as a tool for reflection (R3) by providing access to ethnographic studies in comparable contexts. They describe the interaction in the observed scenario (R7) and try to argue why a specific behavior took place (R8). However, they do not intend to communicate a solution. They are not helping the user how to solve a specific problem (violating R10) and can therefore be considered as means for sharing descriptions of practices but not necessarily of best practices (violating R14). Since they do not provide solutions, they as well do not help to understand consequences of a specific solution (violating R11).

The degree to which socio-technical patterns can help to meet the process requirements depends on the way they are used by the users and developers. The CSCW community still lacks an approach for supporting the application of patterns in a groupware design process. However, if we assume that the patterns are used in a participatory iterative process, they can capture design knowledge in a way
Table 3.18: Relation between Socio-Tech nical Patterns and the requirements.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wholeness</td>
<td>R6: One Problem/Time</td>
</tr>
<tr>
<td>Structural-Pres. Trans.</td>
<td>R7: Field of Forces</td>
</tr>
<tr>
<td>Reflection</td>
<td>R8: Explicit Forces</td>
</tr>
<tr>
<td>Process Awareness</td>
<td>R9: End-User Involvement</td>
</tr>
<tr>
<td>Iterative Design</td>
<td>R10: Education</td>
</tr>
</tbody>
</table>

that is easily accessible by end-users and developers. They help the developer and the user to create structure-preserving transformations (R2). When used by the end-users in a break situation, they also support reflection (R3) since they report on comparable recurring problems of other users and provide solutions to these problems. The reader of the pattern will concentrate on the pattern’s problem (R6) and investigate its context (R7) and forces (R8). Since the patterns are written for end-users, they help to educate the user (R10) so that he can make informed design decisions (R9) that take potential drawbacks into account (R11). The patterns address both the technical and the social parts of the problem. In many cases, the solution can be implemented by a changed social process. In such cases, the users will change their work style and in this sense implement the pattern on their own (R13). Finally, users of a collaborative application should be encouraged to share their socio-technical patterns (R14 and R15). To what extent this is practiced depends on the climate of knowledge sharing in the users’ context.

3.3 Summary of the first part

Table 3.19 provides a summary of the first part of the thesis. We have seen that different approaches to design suggest 15 requirements for a groupware development process (cf. chapter 2). The different approaches to design provide the theoretic and methodic background for such an ideal groupware development process.

As shown in the upper part of table 3.19, the different approaches influenced one another but no theoretic model meets all requirements. The Oregon Experiment is very close to an ideal model but it still lacks properties addressing wholeness and structure-preserving transformations. The theory from Alexander (2003a) can extend the Oregon Experiment to fill this gap.

The considered state of the art for software development support shows two nearly complementary clusters:

1. Design methodologies (the waterfall process, the Unified Process, eXtreme Programming, Scrum, and Feature-Driven Development) support those requirements that focus on analyzing the problem and structuring the solution process in a way that the problem is simple enough to be addressed in the project. Agile methods have strengths in their way to break down the problem in tasks and prioritize the tasks in order to build a system that meets the users’ most important requirements. Especially eXtreme Programming puts
<table>
<thead>
<tr>
<th>Situatedness</th>
<th>△</th>
<th>△</th>
<th>△</th>
<th>△</th>
<th>△</th>
<th>△</th>
<th>△</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis of Form</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>A City is not a Tree</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Wicked Problem</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Reflection-in-Action</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>PDSA</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Gen. Centers &amp; Memes</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oregon Experiment</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Participatory Design</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Holism &amp; Wholeness</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Structure-Pres. Transf.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Rugby Approach</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Waterfall Processes</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Unified Process</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>eXtreme Programming</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Scrum</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Feature-Driven Dev.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Participatory Design</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>End-User Development</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td></td>
</tr>
<tr>
<td>Software Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>HCI-Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Pat.-orient. usab. eng.</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Dewan’s Waterfall</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>extended XP</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>OTD</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>SER</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>CSCW Techn. Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Ethnographic Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>Socio-Techn. Patterns</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
<td>△</td>
</tr>
</tbody>
</table>

Table 3.19: Relation between development approaches and the requirements.
a focus on the end-user and considers him as the most authoritative instance for structuring the development activities. It is the customer who decides on the next steps for each very short iteration.

2. Educative tools that focus on empowering the end-user to make informed design decisions (the pattern approaches). Users should learn best practices and share these practices with other users. It is important that the users find a way to communicate the best practices. The pattern approaches address this issue. Especially HCI patterns focus on teaching the end-user in designing his system. Participatory design and end-user development provide a process context in which patterns can be applied by end-users.

An ideal groupware process should combine techniques and tools from both clusters. However, we can see the same set of clusters in the state of the art considered for groupware development support. In cases where traditional software development processes were appropriated for groupware development, little focus has been on the education of the end-user and the sharing of best practices. Groupware specific processes (OTD and SER) approach the user involvement by either offering opportunities for tailoring or by capturing best practices. But still, they have no systematic approach for capturing and sharing the knowledge.

As in the broader context of software development, a pattern approach that focuses on socio-technical patterns can fill this gap. If combined with an agile process model (i.e., the extended XP model) and enriched by an environment that supports tailoring and sharing of best practices, it can probably meet all requirements.

This is the approach that we will follow in the second part of this thesis. We will present the Oregon Software Development Process model that combines the extended XP model with the use of socio-technical patterns.
Chapter 4

A Pattern Approach for Groupware Development

In the remaining part of this thesis we will propose a new development process that combines patterns with aspects of agile processes in order to satisfy the requirements of chapter 2. The remaining part is structured as follows: In this chapter, we will discuss the process model (section 4.1) and the pattern structure for socio-technical groupware patterns that support educational aspects of the process (section 4.2). Chapter 5 will present a small set of patterns for supporting group formation based on increased group awareness. The application of these and related patterns is the subject of chapter 6. We will present observations from a field study where the process and the patterns were applied to build a collaborative learning platform (section 6.2 ff.). This study will be complemented with theoretic considerations regarding pattern sequences (section 6.4.3) and experiences from smaller student projects (section 6.6).

4.1 The Oregon Software Development Process

The Oregon Software Development Process (OSDP) intends to foster end-user participation, pattern-oriented transfer of design knowledge, piecemeal growth of the system under development in the form of short iterations, and frequent diagnosis or reflection that lead to an improved application.

Figure 4.1 shows the different phases of the OSDP. It suggests three different kinds of iterations – conceptual iterations (section 4.1.1), development iterations (section 4.1.2), and tailoring iterations (section 4.1.3) – denoted by the three circles. After introducing the iterations, we will relate the whole process to the requirements (section 4.1.5) and compare it with other processes (section 4.1.6).

4.1.1 Conceptual Iteration

The role of the conceptual iterations is twofold: they should help the users to form a project group and norm their goals in the participatory process.
The first conceptual iterations will have a special focus on group formation, as it was proposed by Tuckman and Jenson (1977). They proposed that a group’s life cycle runs through five different stages beginning with the first meeting of the group up to the end of group work. The first three stages describe how the group lays the foundation for future collaboration. These three stages are:

**Forming** is the stage where group members become aware of the other user’s identities. They will also get an idea of the group goals and activities.

**Storming**, as the name suggests, is the stage where initial conflicts are fought out. Examples for such conflicts can be an unequal participation in the group or the fight for group leader roles.

**Norming** represents the stage of the group process where the group members have found their position in the group. Group members start to constructively design their group interaction and define group goals.

Conceptual iterations are initiated from within the organization. Users or management bring in ideas for change in the organization. These ideas serve as seed for the interaction between end-users and developers in conceptual iterations. The ideas should evolve from concrete needs, but still leave an open area for participation of the users. Examples are the support of computer-mediated distance education in a traditional university, the creation of a support community for users of an industrial product, or the support of a community of circumstances for teenagers hitting puberty.
4.1. THE OREGON SOFTWARE DEVELOPMENT PROCESS

The extent to which real users can be involved in early conceptual iterations depends on the existence of the user community. If the goal of the project is to establish a user community that did not exist before, it can be very problematic to find initial users that can be involved in the group process (cf. the discussion of XP in the context of groupware development – section 3.2.2). In this case, it can be reasonably to look for people who are close enough to the future users. Taking the example of the university aiming at computer-mediated distance education, it makes sense to find users from the following groups: students studying at a traditional university, students from distance teaching universities, teaching staff, employed people who for time reasons cannot participate at a traditional university but feel the need for additional training.

These users are invited to participate in a focus group that runs the conceptual iterations. Such an iteration consists out of four main activities (numbers in the text refer to the numbers in fig. 4.1): (1) the collection of forces, (2) the envisioning of scenarios, (3) the selection of patterns and the creation of low-fidelity prototypes, and (4) the reflection on the created scenarios and prototypes.

In the first step, users exchange stories from their workplace that are related to the seed ideas. In the example of an industrial support community, a technician would for instance describe how he managed a phone call from a user. The user, on the other hand, would describe a situation in which he needs support and how he normally acts to receive help. The stories are an important source for finding initial (high-level) forces (1).

A special attention is put on break situations. Two questions are asked: What is difficult in the current practice of interaction? What are the reasons for these difficulties? The first question focuses on reflection and the detection of a break while the second question focuses on better understanding the break and detecting the forces that led to the break.

For each story, a scenario interest group (SIG) is formed including those users, who brought up the story, and the development team. The SIGs contrast the stories with a scenario for future use (2). This scenario envisions the future structure of interaction at a very high level of abstraction. SIG members then relate the scenarios to forces and discuss which forces are addressed in the future scenario. The extent to which the forces can be made explicit depends on the certainty that users have regarding the domain task. In most cases, this will still be very vague.

In an implementation phase of the conceptual iteration (3), the users look for patterns that can support the envisioned scenarios. The patterns, in return, reshape the scenarios since they can bring up related or new aspects of technology use that were unfamiliar to the participants who created the initial scenario. The special focus is on the scenario part of the patterns since it tells a related story to the user. The pattern scenarios are on the same level of abstraction as the stories created by the users (in the first phase). An example for a pattern scenario is shown below:

Consider Maria, an inhabitant of Munich. She knows many people living in her

---

Note that we prefer the term forces instead of requirements. In the Alexandrian sense, forces exist in space. They may be at different levels of attention and emerge to requirements.
CHAPTER 4. APPROACH

vicinity. One day, she decides to invite all of them to her birthday party. But since she knows the other people only by name, she has to look up all their addresses. She takes a look at the telephone book and looks up her first contact called Alfred Meier and is shocked by the 1280 people called Meier in Munich. It takes her some time to find Alfred but even then she finds 11 people called Alfred Meier in Munich. Thus, the number of unimportant Meiers is much too large and Maria spends a long time to look up all neighbors that she wanted to invite to her party.

The scenarios of the patterns inform the design of the future scenario. Through analogy with their stories, the users may detect that additional forces are addressed by the pattern. Thus, adding the pattern solution to the scenario can help to address these up to then ignored forces. The patterns also propose a concrete solution. Users consider the solution and think about how the solution matches with the invented scenario. The scenarios are finally attributed with the patterns that are relevant for a specific the individual steps of the scenario. In addition, these patterns are added to the group’s pattern language (together with additional patterns that are closely related).

The patterns and the scenarios are used to create low-fidelity prototypes of the system. Users are supported by the examples provided in the patterns that illustrate how the individual patterns have been reflected in the user interface of some applications.

The prototypes and the pattern-attributed scenarios are then discussed in the whole group. The group performs a ranking of the scenarios and selects a specific number of scenarios that should be implemented in subsequent development iterations.

Besides the detection of first functional forces, the conceptual iteration will set the stage for future development. This includes that a system metaphor (as demanded by XP) is found and used in the scenarios. It also requires that the users get a first impression of efforts and possibilities of the new or changed system. The prototypes provide a very vivid idea of how the system could be changed. The methods used in the conceptual iterations are related to methods employed in traditional participatory design and complemented by the use of high-level patterns.

4.1.2 Development Iteration

In the development iterations, the scenarios are further refined by the SIGs. All SIG members meet for a planning game. They discuss the most obvious forces that appear in the scenario. These forces are written on story cards together with the specific part of the scenario that is in conflict with or supporting to the force. The result is a set of user story cards (as in XP).

Users again refer to high-level patterns to inform their process of decomposition and solution design. They provide answers to the questions stated in the pattern.

\footnote{From the Buddy List \textsuperscript{227} pattern, initially published in (Schümmmer, 2004c) – A note regarding the pattern notation: References to patterns show the pattern name and the page number on which the pattern is explained set in a subscript font.}
4.1. THE OREGON SOFTWARE DEVELOPMENT PROCESS

As long as the pattern language is, like the original Alexandrian example, organized using different levels of scale, the decomposition can follow the connection between the different levels of scale. At the same time, developers refer to low level patterns to answer more technical questions raised by the user story (6). Users and developers can propose additional patterns at this stage of the process. These will be considered as preliminary patterns that have not yet shown their validity. However, they should be documented by means of patterns since the pattern language serves as the community’s design memory and vocabulary.

The patterns are connected to the stories in order to inform later implementation of the story. Developers further attribute the stories with a rough effort estimate (in pair programming hours).

As in XP and Feature Driven Development, users can then “go shopping” and “buy” user stories. At each shopping step, they are only allowed to buy a fixed, but small, number of hours (in the field study we used 16-24 hours = 2-3 days). This results in an ordering of the user stories comparable to Scrum’s backlog.

At any time in the process, users are encouraged to provide feedback regarding stories that are on the backlog and reorder or modify the stories. Unlike in Scrum, the users can modify stories until a developer begins implementing them. This has the effect that the user has continuous control over the direction of development actions taken by the developers.

On the other hand, this has the drawback that fixed milestones (containing functional and temporal fixtures) are no longer possible. The users thus have to agree with the development team that functionality will always be developed regarding the backlog, and that the development team is paid regarding the real efforts. Trust between developers and users is an important factor when working together on this basis. The developers have to trust the customers that they will be able to state their most urgent wishes. Customers in return have to trust that the developers will create a system that satisfies exactly those most urgent wishes and that they put efforts in solving these wishes that is worth paying for.

Since different SIGs can work on different scenarios in parallel, the creation of different backlogs has to be coordinated. We propose four different types of backlogs: project backlogs, SIG backlogs, end-user backlogs, and non-functional requirements backlogs. They will be discussed below.

The most important backlogs are the SIG backlogs. They hold the task cards from the SIGs that were formally appointed by the user community. One problem with XP is that it may be difficult to convince the customer of the need of a refactoring. This is the reason why the OSDP adds an artificial SIG, the system maintenance SIG, that is responsible for detecting design smells in the system. The system maintenance SIG has the same rights as the other SIGs. Members collect and prioritize tasks in the system maintenance SIG backlog.

Stories that are brought up by users without the context of a special SIG are collected in an end-user SIG backlog. They are rather difficult to prioritize since they emerged from different users with potentially different goals. Thus, the end-user backlog will be addressed on a first-come first served basis.
As proposed by Rittenbruch et al. (2002), the team member with the role of the gardener is responsible for coordinating the different backlogs. She observes the different backlogs created by the teams and creates a scheduling that is comparably fair for all SIGs (a round-robin strategy can already be sufficient). The SIGs are informed of the merged schedule for story cards and can comment on this schedule, e.g., to convince the gardener that two cards of this merged schedule should be swapped.

Figure 4.2 shows an example of a project with two scenario-specific SIGs, the artificial system maintenance SIG, and the end-user SIG. All SIGs have their backlogs. The gardener takes cards from these backlogs and adds the cards to the project backlog.

Instead of adding all stories to a set, the gardener can look for comparable stories (or overlaps in stories). Whenever she detects redundancy, she proposes a rewriting of the stories. The SIGs or the gardener then proposes a new set of merged stories. In fig. 4.2, the first task of the star SIG overlapped with the first card of the system maintenance SIG. The gardener therefore took both cards and created one merged card. The third task of the star SIG was identical with the fourth task of the cross SIG. Even though the cross SIG’s previous cards were not yet scheduled, the gardener can decide to combine a less important cross task card with a star task card rated as important by the star SIG.

As long as no stakeholder disagrees, the development team will implement the most urgent task from the backlog.

In the implementation step (7), developers provide a software solution for the first card from the backlog and an automated test (8) to confirm the solution’s conformance to the users’ wishes. Developers typically implement a pattern by means of application frameworks or developer-centered component frameworks. This may involve the development of new software components. Such components can be
4.1. THE OREGON SOFTWARE DEVELOPMENT PROCESS

built using frameworks or other base technologies. The developer uses the answer to the questions provided in the check section and the information provided in the collaborations section to inform his design. Note that the order of testing and implementation can be reversed (like in XP).

After the first user story is implemented the users get access to the system and start using it (as far as the system supports the user in performing his tasks). The system use will result in modifications of the backlog as well as in tailoring iterations. The latter are described in the next subsection.

4.1.3 Tailoring Iteration

In the tailoring iteration end-users use the application for the desired purpose. While using the system end-users with pattern-based design knowledge are encouraged to reflect on their activities whenever they encounter a breakdown (9). A breakdown leads to an entry in the groupware’s health map (in the simple case, a note that a specific group need could not be satisfied with the groupware system). In cases where the user does not detect this breakdown (i.e., if the user feels uncomfortable but thinks that this feeling cannot be changed), an evaluation user (cf. section 3.2.2) can expose the breakdown, discuss it with the user, and initiate a reflection process together with the user.

Users then take a closer look at the detected shortcoming. First, they analyze the forces that are in conflict (10). High-level groupware patterns help in this process by describing frequently occurring issues, the various forces, and a proven solution in a way that is appropriate for tailoring end-users.

The solution provided by the high level patterns informs the successive groupware tailoring design (11) and the execution of the tailoring (12). Tailoring actions can take place at different levels. In content level tailoring, the users change the artifacts that are managed by the groupware system in order to solve the problem. At this level, the pattern assists the user in using a tool. At the functional level, users appropriate the functionality provided by the tool. They activate needed functions and deactivate functions that are in the way. At the component level, the users perform more extensive tailoring actions: they compose functional groupware components in order to create new configurations of applications that support the team in an unanticipated way.

To support tailoring as a group level, a pattern scout looks for solutions that work well. As the evaluation user, the pattern scout observes users in the system with the goal of finding recurring successful system use. The found best practice is then discussed with the users and documented in the pattern format. Such new best practices then find their way in the pattern catalogue. Note that these patterns are in most cases very domain specific (e.g. patterns for supporting customer relationship management in the context of a support system). On the other hand, these patterns will be used most frequently in the user community since they are appropriated for the interaction that typically takes place in the community.
4.1.4 The relation between the different iteration types

During project life, the different kinds of iterations will be executed in different frequencies as shown in figure 4.3. Fig. 4.3 does not forecast exact frequencies over time. Rather, typical relationships between the frequencies of the different iteration types are shown.

At project start conceptual iterations are the most important iterations. Prototypical development accompanies the gathering of use cases and patterns in development iterations. While the project evolves the frequency of development iterations grows and conceptual iterations are not as important anymore. Since the development iterations produce more sophisticated prototypes users will start using the system more frequently and reflect on their system use. Thus, the number of user-triggered tailoring iterations increases towards the end of the project while the other two kinds of iterations become less important.

Users can decide to switch from a tailoring iteration to a development iteration if the needed action seems to require a more specialized treatment. In this case they create a story card and add it to the end-user backlog. The gardener will examine the story card and propose an insertion point in the project backlog.

Tailoring and development iterations are expected to focus on the intended scenarios of system use. If new scenarios evolve that are not variations of the initial scenarios, the group should perform conceptual iterations.

As suggested by figure 4.3, the different iterations can take place at the same time in the project. A part of the development team may perform iterations of one kind while another part of the team concurrently performs iterations of another type. The different sub-teams coordinate by means of the project backlog that is maintained by a gardener. It is the gardener’s task to maintain consistency between the different sub-teams.

4.1.5 Requirements addressed by OSDP

The OSDP aims to satisfy all requirements for groupware development as discussed in chapter 2. The relation between requirements and practices is discussed in the
4.1. THE OREGON SOFTWARE DEVELOPMENT PROCESS

following list on a theoretical level (the practical comparison will follow in chapter 6):

**R1: Wholeness.** The project backlog addresses the development of the system from a holistic perspective. This, however, does not ensure that the resulting product will have a high level of wholeness. Automated tests of built system components contribute to the requirement of wholeness. Since they check whether old requirements are still met, any change in the system will extend the system by keeping old relations intact. The more important aspect regarding wholeness, however, is the contribution of each step. Each step has to enhance wholeness, which means that it helps to eliminate break situations encountered by the user otherwise. Patterns provide proven solutions for modifications that try to solve a specific problem. This implies that the use of patterns makes the changes more likely to enhance wholeness.

**R2: Structure-Preserving Transformations.** The sequence of applied patterns should be structure-preserving. Most important modifications (that resolve the largest problems regarding wholeness) thus need to be performed first. The process of ordering task cards on the SIG backlogs and on the project backlog contributes to this estimate of most urgent problems.

**R3: Reflection.** Reflection is an integral part of the OSDP. Users are encouraged to reflect on the reasons for breakdowns and share the solutions that they designed to overcome the breakdown (in the future). They can also escalate the conflicting forces to a development iteration, which encourages them to report even problems that they cannot fix on their own. The other iteration types include a reflection element during acceptance testing and the discussion of created mockups and selected patterns.

**R4: Process Awareness.** All participants can access the project’s backlog at any time. The developers, in addition, compare their estimates with the real effort needed to complete a user story.

One potential problem with process awareness is that all iterations can run in parallel. Although the participants of a specific iteration are aware of their progress in this iteration, the process awareness regarding the whole project can be problematic. On the other hand, since all development activities are steered by the project backlog, this is the measure regarding the delivered product, which is the most important result of the project.

**R5: Iterative Design.** Iterations are one of the basic principles of OSDP. Users iterate on different levels, and changes are lightweight at every stage in the process.

**R6: One Problem at a Time.** This requirement is addressed at different levels:

In the conceptual iterations users create scenarios that in most cases comprise many problems. This is compensated by the use of patterns, since patterns always focus on one specific problem.
In development iterations the scenarios are decomposed into smaller user stories that should address only one problem at a time. The use of patterns again fosters the process of scoping the user stories to a size so that they address exactly one problem.

In tailoring iterations users reflect immediately after a breakdown. They analyze the problem that led to the breakdown and try to solve this problem. The breakdown could be caused by different problems. However, the user is guided to focus on the most obvious problem first and fix this problem.

**R7: Field of Forces.** The process, per se, does not support this requirement. However, the patterns serve as a tool for understanding the forces of a specific problem and the forces of related problems (through the relation to other patterns).

**R8: Explicit Forces.** Users are encouraged to reflect on the forces that caused problems. This reflection process is supported by patterns that list forces of often encountered problems.

**R9: End-User Involvement.** End-users play an important role throughout the process. They control the order of the backlog and thereby the whole development activities of the development team. They play an active role in the conceptual iterations where they envision scenarios of future system use together with the developers. Finally, end-users the main participants of tailoring iterations.

**R10: Education.** Patterns are the main means for educating the end-user so that he can act as an informed designer.

**R11: Explicit Consequences.** Again, the patterns play an important role to satisfy this requirement. They list positive and negative consequences so that a person applying the pattern can balance benefits and drawbacks.

**R12: Integration.** Integration happens on different levels: In the phase of project planning the different scenarios are integrated by the gardener, who looks for intersections between the results of the different SIGs. In the implementation phase the integration is supported by means of patterns. The patterns describe the context and make the relation of a solution to this context explicit.

**R13: End-User Development.** In tailoring iterations the users modify their environment.

**R14: Sharing of Best Practices.** Users who acted as a tailor are encouraged to share their tailoring results with other users. The role of the pattern scout is the key player in this context since he looks for recurring solutions and maps them to a pattern that is then shared in the community.

**R15: Quality Review.** Community practices that are captured by means of patterns should undergo a strict review process. The review process includes peer
reviews and iterations between pattern author and reviewer. The patterns are further discussed in the user group before they find their way into the community pattern language.

4.1.6 Comparison to the State of the Art

The OSDP borrows practices from different process models discussed in chapter 3. We will discuss the relation in the following section.

**XP.** It borrows the concept of user stories, the basic structure of the planning game, test-driven development, and very short releases (after each completed user story) from XP. Without modifications it can employ all other practices of XP except the on-site customer. The role of the on-site customer is split in two roles: we distinguish between members of a SIG and other users. The members of the SIG interact with the development team in face-to-face discussions. Other users get in contact with the development team by creating and discussing story cards. This can be supported using a groupware application.

**Extended XP.** The role of the gardener and the role of the evaluation customer are borrowed from the extended XP process as well as the concept of involving the end-user in the planning game. The members of the conceptual iteration could be considered as ad-hoc customers. The role of a development customer, however, seems artificial in the extended XP model. Instead, we decided to give the developers their own SIG, so that they are considered in the story card scheduling.

**Scrum.** The concept of backlogs is very important in the planning process of OSDP. In contrast to Scrum we decided not to block stories that are in an iteration of a sprint. This has the effect that the customer can not define deliverables for the iterations. The unit of deliverable is, instead, always limited to the most important story of the backlog. In return, the customer has the advantage that he can change the stories more frequently. This is more difficult for the development team but guarantees the highest possible level of flexibility, which is important for wicked problems.

**Participatory design.** The conceptual iterations use methods from participatory design. Users envision future scenarios and create prototypes of different levels of fidelity. The tailoring iterations make use of customization, which means that users participate with designers in design activities late in the process. In addition, the evaluation user observes how users use the technology to perform a specific task. This activity is related to activities in ethnographic methods. The main difference to the classification of Muller and Kuhn (1993) is that this observation takes place late in the process to initiate next iterations.

**End-user development.** Depending on the power of the groupware system, the users can tailor on a more or less technical level. This can result in a situation
where the user actually implements parts of the system using a domain-specific programming language. However, to which extent this is possible depends on the groupware system built.

**RUP.** The idea of different phases with different iterations in the project life cycle was influenced by the model of RUP. However, we propose only three different iteration types and much fewer phases in the involved iteration steps. A second difference is that the OSDP encourages the synchronous execution of iterations at different levels.

**Extended STEPS.** The idea of embedding tailoring as an integral step in the iterations during system use was already part of the extended STEPS model. OSDP emphasizes the need of tailoring even more since it includes an evaluation user who actively triggers tailoring steps.

**OTD.** The organizational and technological development approach proposes a comparable group of stakeholders that do initial designs. Reflection and end-user involvement is an important component of both processes. OTD and OSDP propose that both process knowledge and process support should co-evolve. In OSDP process knowledge is coded in patterns. These can be used to foster organizational learning and improvement of practices. However, OSDPs level of sharing for best practices exceeds the level proposed by OTD.

**SER.** The idea that best practices should be transferred in a community knowledge base is propagated by SER and OSDP. Other parallels are the focus on reflection and piecemeal growth.

**HCI and groupware patterns.** Patterns play an essential role in OSDP. Especially the prosaic nature of patterns advocated by the HCI patterns community fits the intended use of patterns in the OSDP. The integration of patterns in the development process proposed by Borchers (2000b) shows many parallels with the conceptual iterations of OSDP.

### 4.2 Appropriating the Representation of Groupware Patterns

While patterns were so far only discussed regarding their narrative nature, we did not yet discuss the structure of the patterns and the structure of a collection of related patterns that form a pattern language. This is the theme of this section.

#### 4.2.1 Single Pattern Representation

We have discussed different formats for presenting patterns in section 3.1.5. Formulating a pattern description as much as possible compatible to the PLML DTD
4.2. THE PATTERN REPRESENTATION

brings the advantage of being able to share and merge patterns from different languages. However, the DTD has several missing parts that should be addressed in the context of patterns for groupware design.

The following description lists the different sections of a pattern with their role in the pattern format and an example (set in small italics) how these sections appear in a concrete pattern.

**Name.** The name of the pattern serves as a handle for the pattern. All names constitute the set of available patterns and will be used when users of the language communicate about the problem domain. It is important that the pattern name is easy to understand and reveals the core of the solution. It should conjure images that convey the essence of the pattern, since these are easy to memorize. Pattern names should be set in a consistent but recognizable typography, e.g., using capital letters. The name is complemented with other well known names if they exist.

*An example is the Buddy List pattern with the alias names roster (in Jabber), contact list (in MSN or ICQ), and address book.*

**Sensitizing Picture.** Next to the title is a picture that should help to better remember the pattern. It should match to the title and already tell a story about the context of use or the involved participants and interactions.

The Buddy List had, for example, the picture of kids holding their hands (fig. 4.4). This picture suggests attributes like trust and connectedness between buddies.

![Figure 4.4: The sensitizing photography of the Buddy List pattern (photography by Ronald Raefle, Berne, visipix.com).](image)

**Context Diagram.** When patterns are presented in a pattern language (cf. section B.1), a context diagram helps the reader to orient in the language. It shows a miniature version of the pattern language diagram where patterns appear as nodes. Relations between patterns appear as edges.
CHAPTER 4. APPROACH

Figure 4.5: The context diagram of the Buddy List pattern.

Intent. The intent section of the pattern captures the core of the pattern’s solution in one sentence. It is used for creating overview lists of the pattern language that contain only name and intent of all patterns.

Show only selected known users.

Context. The context describes the situation in which the pattern is intended to be used. It may refer to other patterns (of a higher level of scale) that use the current pattern in its solution. The goal of the context description is to help the reader understand which context the pattern was designed. Nevertheless, it does not restrict the application to such a context. Users can decide to apply the pattern in other contexts as well, which however complicates the process of adapting the pattern to such a different context.

You are using an interaction space like a communication channel, a groupware application or a collaborative virtual environment together with many other users.

Problem. The problem description describes the most important aspects of the problem that occur over and over again. It should be stated in a way that the tension of conflicting forces (or the pain of the break) becomes obvious to the user. The reader will at least read this section when investigating the pattern.

When many users are able to interact in the interaction space it is hard to maintain an overview of relevant interaction partners since the number of users exceeds the number of relevant contacts for a specific user. User lists grow very large and it is hard to find people who the local user knows. On the other hand, the local user is often only interested in those people he already knows.

Scenario. The scenario is a narrative description of a situation in which the pattern could be applied. It may describe a context that is comparable to the context of the pattern. It may also describe a different context in order to make the range of possible transfers explicit. An example is the use of a real-world scenario for a pattern that describes interaction in the virtual world. In this case the
scenario helps the reader to reflect on real-world experiences and understand the forces in the real world. It is then up to the user to find the same forces in the virtual context again.

The scenario for the Buddy List was already provided on page 97.

**Symptoms.** This section describes ways how to detect the conflicting forces. It has the form of a list of sentences that start with the phrase “You should consider to apply the pattern when . . .”.

- users spend a long time searching for another user.
- the system may be used by more than one user with a specific name who only differ in their address or login. Remembering logins on the other hand is considered difficult by the users.

**Solution.** The solution resolves the conflicting forces mentioned in the problem statement and the symptoms section. It is the second part that is most probably read by the user of the pattern. Thus, problem and solution together with the context make up a brief version of the pattern for fast access that provides more information than the short description in the intent section.

Provide buddy lists where a user can enter other users who are of interest to him. Whenever the local user browses other users, first show only users from the buddy list.

**Collaborations.** This section names the actors and components of the pattern. It also explains how the different parties collaborate to resolve the patterns forces. It can be considered as a high level description of the proposed use-case, although it is still in a prosaic form.

Whenever the local user interacts with another user the local user can add the other user to his buddy list. The buddy list is a set of user objects. A user can be added to the buddy list by selecting his representation in the user interface and executing a command (e.g. a menu item associated to the user object).

**Rationale.** The rationale section provides the first part of the pattern’s “proof”. It explains how the pattern supports its objectives, gives an explanation why the whole pattern works, and shows why it is appropriate in the given context.

The main reason why this pattern works is that it eases the process of finding other users by storing these users in a personal list. Compared to public directories of users the personal list only contains the users that are important for the local user. This will reduce the number of name conflicts and enlarge the hamming distance between two names in the buddy list (it is less likely that two users with close names like Alfred Meier and Alfred Meyer are both buddies of the same user).

Connecting the means for adding users to the buddy list with the user’s representation (or the interface elements that are used to interact with the other user) makes the process of adding a user to the buddy list intuitive and reminds a user to consider adding the user.
Check. The transfer of the pattern to a concrete context requires that the developer (or the user) maps the abstract actors and relations named in the pattern to his concrete application domain. To ease this process the check section provides a list of questions that can be answered comparable to a checklist. The answers document the mapping between the application domain and the pattern domain.

- How are users represented in the application?
- Can the user representation be connected with a context-sensitive command to add the user to a buddy list (and how)?
- How does a user select other users in the application?
- Where is the connection between selecting other users and looking them up in a list?
- Will the buddy lists be bound to a specific context (i.e., will users use different buddy lists if they interact with different communities)?

Danger Spots. This section elaborates problems that may occur when the pattern is actually integrated in a system. It includes pitfalls or technical caveats that might hinder the inexperienced developer from successfully applying the pattern.

If users only consider buddy lists for maintaining contacts to other users they will hardly find new users in the system. Thus, you should ensure that users can also browse other users who are not on their buddy list (e.g. by providing a User Gallery → 197).

Known uses. The known uses section is the second part of the pattern’s proof. It provides examples of systems that implement the pattern. The authors of patterns should try to find at least two examples from different groups (or, even better, other problem domains). The example shown below lists only one of several known uses of the Buddy List pattern.

Instant Messaging Systems like MSN Messenger, icq, AIM, or Jabber all provide buddy lists. Presence information is retrieved from the presence server only for those users, who are on the buddy list. Figure 4.6 shows how a contact can be added during a conversation with icq. If the local user allowed other users to contact him, they can directly talk to the local user by sending him a message. Note that the remote user has to retrieve the address of the local user by other means than the buddy list since the local user is not yet on the remote user’s buddy list. The remote user will be shown as a user “Not in List” (cf. fig. 4.6-A) on the buddy list for the time of the chat. If the local user feels that the contact is valuable, he can add the remote user to the buddy list by pressing the “Add” button, which will open the “Add User” dialog (cf. fig. 4.6-B). The local user can provide a nickname for the remote user under which the remote user will appear in his buddy list. Note that the the “Add User” dialog by default checks an option so that the remote user will be able to add the local user to his buddy list as well. This is important to maintain reciprocity → 238 although it can be bypassed (by unchecking this option).
4.2. THE PATTERN REPRESENTATION

4.2.1. THE PATTERN REPRESENTATION

Figure 4.6: Adding a contact to the buddy list in icq.

From then on, the remote user will be shown on the buddy list with his current status (fig. 4.6-C). The local user can from then on get in contact with the remote user easily without looking up this user’s icq number.

Related patterns. Lists patterns that should also be considered when the reader finds the current pattern relevant. This section links to other patterns in the larger context of the pattern language. For each related pattern it explains how the pattern is related and how the patterns differ.

USER GALLERY → 197 provides means for browsing all users in the system. As outlined in the safety rules, a USER GALLERY complements a BUDDY LIST and opens the opportunity to meet new community members.

RECIPIROCITY → 238 is important if buddy lists reveal much personal information. In this case, a user should be able to control who can put the user on his buddy list. On the other hand, if the local user adds a remote user to his buddy list, the remote user should also be allowed to add the local user to his buddy list.³

The main difference to PLML is that our pattern format provides a more detailed structure of each pattern. This allows the user to selectively read only specific parts of the pattern regarding his current situation. While PLML distinguishes between solution and rationale, we argue that the pattern should provide more details sections for the solution. Examples are the check section or the danger spots. Representing

³The original BUDDY LIST pattern contains additional references to additional patterns.
a pattern from the groupware pattern collection by PLML would mean that the sections solution, collaborations, check, and safety rules need to be part of the solution tag of PLML.

4.2.2 Pattern Abstraction Scale

Throughout all iterations the users participate and work with a shared groupware pattern language. This language consists of patterns on different levels of abstraction. High-level patterns describe issues and solutions typically targeted at end-users. Low-level patterns describe issues and solutions typically targeted at software developers on a more technical level.

High-level patterns focus on the system behavior as it is perceived by the end-user. They empower the end-user to shape his software system to meet his demands. Thus, high level patterns need to be more descriptive and prosaic than low level patterns that focus on how to implement that behavior. Low level patterns focus on the implementation of the system and thus include more technical details.

Both low-level and high-level patterns can be positioned on a seamless abstraction scale. This scale can be used to compare two patterns. Given two patterns $p_1$ and $p_2$, we can say that $p_1 < p_2$ if $p_1$ focuses on more technical issues than $p_2$ and uses a more developer-centered description language than $p_2$. Note that the order will not be a complete order. It is rather relative with respect to the current focus of analysis. The goal of the abstraction scale is to provide the reader with a first impression, e.g., whether or not the pattern requires software developers to implement the solution.

Software design patterns that deal with class structures, control flow, or coding techniques form the lower bound on the abstraction scale of patterns. Patterns that focus on human interaction and address computer systems just as tools to support the human interaction (e.g. groupware patterns) would be placed near the upper bound on the abstraction scale. Patterns at a very high level would describe how the end-user can compose off-the-shelf components and embed them in his work process. This would then mean that the software developer would no longer need to assist the end-user in implementing the pattern.

Additional techniques for structuring pattern languages and supporting the reader and the author in orienting in the pattern language are discussed in appendix B.
Chapter 5

The Pattern Language

Patterns are the most important tool of OSDP. The groupware pattern language currently contains 130 patterns. Since it is intended as an evolving language, the patterns are at different levels of completeness and the different topics of groupware development are covered to a different degree. The pattern development was guided by the users of the pattern language.

For space reasons, we will not present the full pattern language. We will instead focus on one aspect of the pattern language that is very characteristic for the development of groupware applications: group formation by means of enhanced group awareness.

We will first discuss why this family is important for groupware development and provide a theoretic background on group formation issues (section 5.1). Section 5.2.1 will provide an overview of the presented pattern language. The following sections provide the full descriptions of eight patterns from the selected family. Short versions of the other patterns from the groupware pattern language can be found in the appendix. Note that the full catalogue of groupware patterns is scheduled for publication at the end of 2006. More details on this can be found on the authors’ home page http://www.pi6.fernuni-hagen.de/p4cmi.

5.1 Importance of Group Formation

In line with Heidegger’s concept of situatedness we can state that group interaction evolves out of a current social situation. The need for starting a group life cycle is driven by the context of each prospective group member. However, if the users work in single-user modes, they often do not notice opportunities or needs for interaction.

The question to answer when forming a group is whether or not the current situation fits the need for collaboration between users. This question addresses temporal issues (when is the right point in time) as well as situational issues (what is the right work context to shift users’ attention to the process of group formation). While many aspects of tool support in collaborative applications have their counterpart in single-user applications (e.g., a collaborative diagram editor provides the same tools for manipulating diagram elements as a single user diagram editor) group formation
is only important in group contexts. It is thus a good example for investigating the role of groupware patterns.

Supporting users in detecting opportunities for interaction has been a theme of the research area of group awareness in the field of CSCW. Dourish and Bellotti (1992) defined group awareness as “an understanding of the activities of others which provides a context for your own activity”. Providing awareness within a formed group helps to coordinate the group members’ own activities. Examples are workspace awareness (Gutwin and Greenberg, 1996) mechanisms where the users of a shared workspace are informed about other users’ activities within the workspace. Awareness of others can serve as an indication for collaboration in situations where no group has formed yet and trigger group formation.

The awareness of other users is one of three important aspects of a user’s context that can trigger group formation. The current position within a work process is another aspect. A third aspect is the current interaction setting of the user.

![Figure 5.1](image.png)

**Figure 5.1:** Aspects for group formation support based on enhanced group awareness.

Figure 5.1 shows the three aspects in a group formation classification cube.

**Activities** is a dimension that describes the state of the activities which are supported by the system. Activities can be arranged on the levels *past*, which means that the user has performed all steps of the activity and moved his focus to another activity, *current*, which means that the user is currently performing steps that are needed to complete the activity, and *planned*, which means that the need for the activity is already identified but that no user is currently participating in the activity. Past and planned activities occur in asynchronous work (from the context of the current point in time) while current activities map to synchronous work.

**Artifact** as a dimension expresses the relation of the user and his activities to the set of shared artifacts. We can identify the levels *focus*, *nimbus*, and *nimbus*
complement. The focus represents the set of artifacts that are in the focus of the activity. An artifact is in the focus if it is perceived and/or modified by the user in the activity. The nimbus contains artifacts that are closely related to the artifacts in focus. Finally, the nimbus complement contains artifacts that are not part of the nimbus. These are artifacts that have a long semantic distance to the focal artifact. The model of focus and nimbus has been discussed in depth by Rodden (1996) although the nimbus complement was not considered by Rodden.

Nevertheless, the nimbus complement is relevant for situations where activities on complementing artifacts are an indication for group formation or awareness information. If, for instance, the community could benefit from a shared understanding of the full artifact space, users should especially be made aware of artifacts in their nimbus complement.

Interaction, finally, is the dimension that defines how tight two people currently work together. Users can work in a single-user mode without explicitly taking other user’s activities into account. The actions of other users are recognized only by changes in the shared data. Loose interaction focuses on the provision of awareness about other users. Users still perform the activities as individuals but are informed of other user’s activities. Thus although no direct interaction between the users takes place, there is an understanding of other users in the system. Tight interaction focuses on how collaboration can be supported between users. Manipulation of shared artifacts is only one issue in this context. Other important issue are the awareness of other group members in the workspace (workspace awareness (Gutwin and Greenberg, 1997)) and the awareness of the group process.

The effects of group formation can be illustrated using the classification cube. Group formation mechanisms help the group to change their work context from one segment of the cube to another (fig. 5.2). The closest form of collaboration takes place in the shaded segment of fig. 5.2. In this case, users interact in a group at the same time with a shared focus on the artifacts.

In general, we can distinguish four directions that should be considered in a group formation process (represented by the numbered arrows in fig. 5.2).

1. Users move their focus from single-user human-computer-interaction to tight group interaction. Consider as an example a workshop at a conference. Potential participants prepare position statements (single user interaction) and publish their position statements on a web server. Other interested participants become aware of the position statements and review the collection of position statements (loose interaction). Finally, interested users register for the workshop and meet in a meeting room to discuss the position statements (tight interaction).

2. Users narrow their corridor in time so that asynchronous interaction raises the need for synchronous interaction. For the example of the workshop, this
means that the users schedule the interaction for a time slot at the conference (using planned activities to inform current activities) or collect the state of the art as a background for the discussion (using past activities to inform current interaction).

3. Users have to broaden their focus in order to find other compatible users. Consider a researcher working on collaborative gaming research. His special topic is that of competition in jump-and-run games. At a conference like CHI that has a very broad focus he may however not find users who share this special interest. The researcher will have to consider more general topics (like research on games) to find a workshop that is relevant for him.

4. The process of group formation combines all three movements discussed above and extends them by establishing a shared focus again.

5.2 A Pattern Language for Group Formation by Enhanced Awareness

5.2.1 Overview

The patterns presented in the next sections describe groupware mechanisms that support the users in moving from one segment of the cube towards the direction of the shaded segment.

Figure 5.3 shows a selection of patterns from the group support family addressing questions of group awareness. The shaded patterns will be discussed in the next section in depth. Although the other patterns cannot be presented in full length,
Figure 5.3: A section of the pattern language showing patterns used in the CURE group awareness sequence and their related patterns.
the overview allows us to catch a glimpse of the connectivity and complexity that we will find in a full-fledged pattern language for groupware development.

Figure 5.4: Transitions supported by the patterns presented in this chapter (numbers link to the pattern description in the text).

We can relate the patterns to the transitions shown in fig. 5.4. The patterns included in full length are:

**From Shared Data to Shared Work**: Foster group formation and spontaneous collaboration by bringing together users who share an interest in the same data and providing them with means for communication and collaboration.
Users focus on an artifact in single user HCI. They become aware of other users who work at the same artifact at the same time and start tight interaction (fig. 5.4-1).

**LOCAL AWARENESS** → Provide awareness information in the context of the artifact, which is in the local user’s focus.

The pattern describes how to move from single user interaction to loose interaction (fig. 5.4-2).

**PRESENCE INDICATOR** → Indicate that remote users look at an artifact, which is visible on the local user’s screen.

The pattern helps to move from single user HCI to loose interaction (fig. 5.4-2).

**USER LIST** → Show who is currently participating in a session.

This pattern focuses on tight interaction. Users should stay aware of collaborating users in the same session (fig. 5.4-3).

**GAZE OVER THE SHOULDER** → Detect the users’ interaction with shared artifacts.

The focus of this low-level pattern is detect current activities. It is essential for providing any awareness on activities (fig. 5.4-4).

**ELEPHANT’S BRAIN** → Store information on the users’ activities in a log to allow the users to understand (other) users’ activities and the artifacts’ evolution.

This low-level pattern supports to move from past to current activities. Users who are currently active can become aware of past activities, understand their work, and establish contacts (fig. 5.4-5).

**DAILY REPORT** → Inform the user about changes on relevant artifacts in a user defined frequency (e.g., once a day).

This pattern is a high-level counterpart to the ELEPHANT’S BRAIN pattern. It informs the local user about other users’ past activities on relevant focal artifacts (fig. 5.4-6).

**HELLO HELLO** → List new members of a group or a community at a prominent place and introduce them to other members.

One aspect of the HELLO HELLO pattern is that users move their focus from the focal artifact to the nimbus complement in order to become aware of other new users who have not yet established contacts for loose or tight interaction (fig. 5.4-7).

The remainder of this section will present the individual patterns in their full form.
5.2.2 FROM SHARED DATA TO SHARED WORK

**Intent** Foster group formation and spontaneous collaboration by bringing together users who share an interest in the same data and providing them with means for communication and collaboration.

**Context** Users are working with an application that provides access to shared data. They consume or manipulate the shared data using their personal clients, which may be at diverse locations.

The shared data may be, for instance, a file, a set of files that are used for a project, a cluster of related objects in a distributed application, or a set of records in a database that is accessed by all users.

The interaction with the shared data is not easily predictable, since the possible activities do not follow a fixed set of workflows.

**Problem** Although many users work with the same shared data, they may not recognize the other users’ work. This results in parallel or conflicting work and a lack of collaboration and learning from one another.

**Scenario** Imagine a scenario in the automobile industry: Many employees work on the design of new car. Some experts design the electronics, while others work on the layout of the control panel. In each group there are different responsibilities and roles. While the designer arranges the tachometer, the ecologist considers how the tachometer may be recycled. All engaged employees work on the same shared data – namely the design documents of the whole car.

During the design process different modes of collaboration (Schümmer and Haake, 2001) may take place: First the designer works on his own to create a first draft. If the material for the tachometer is to be specified, the situation demands a close cooperation with the ecologist. But the ecologist is not the only person the designer needs to collaborate with. Together with a
mechanic, he has to investigate, whether or not the tachometer can be created with the material considered. These are only two examples of collaboration on shared artifacts, which is needed in a complex design process.

The more complex and dynamic the environment (respectively the shared data) gets, the harder it is to predict all valuable constellations for collaboration by means of a strict work flow.

Consider as a less obvious situation in the car example the design of safety seats for babies and the development of an airbag security system. In most modern cars that have a co-driver airbag, one can no longer place the safety seat for babies on this seat since the inflation of the airbag may kill the baby sitting in the safety seat. In this example, two designers did not collaborate although they worked with the same space in the car. The result is that some requirements were not communicated among the two designers.

The examples show that collaboration opportunities are often not recognized and collaboration does not take place. This may have two consequences. The most obvious consequence is that the work of the different participants will result in conflicts. If the designer does not contact the electrical engineer and the mechanical engineer, it might be that the design of the tachometer does not provide any space for the needed cables that lead the signals to the device. In this case, the designer obviously didn’t know much about the technical constraints of his design, which leads to the second consequence: The designer would not be able to learn from the electrical engineer. If they had collaborated, they could have discussed their different experiences and won new insights.

From this scenario, one can see that it can be very important to foster collaboration, when two or more users work on the same part of the shared data.

**Symptoms**

*You should consider to apply the pattern when ...*

- managers consider it as difficult to predict who should collaborate to reach a specific goal.
- team members notice that they worked on the same artifacts after the work is finished.
- parallel work causes conflicts that could have been resolved when the team members had worked together.

Users are unaware of the potential for collaborative activities that could improve the overall result or minimize conflicting work.

The users are working with the shared data without much knowledge of their colleagues (fig. 5.5 shows the user as box (a) who accesses shared artifacts (box b)). The only effect that lets
them deduce that other users are working on the shared data is that the data changes from time to time. Uncoordinated collaboration in a shared filesystem is an example of this kind of work.

One of the forces in the context section stated that it is difficult to know all useful points of collaboration in advance. But even if this knowledge exists, it is still challenging to establish groups. Users have to coordinate collaborative episodes before they actually take place (cf. fig. 5.6).

![Diagram of user-driven group formation](image)

**Figure 5.6:** User driven group formation.

First, the users have to identify other users who should be part of the team and contact them to propose a collaborative session (cf. fig. 5.6-1). We call this kind of group formation as *user driven group formation*. If all participants agree, they start a collaborative session (fig. 5.6-2). The topic of the session sets a new focus – the group focus – that all participants need to share. Establishing group focus may be difficult if the group members were focussing on unrelated activities when the group started its session.

Depending on the desired strength of collaboration, group members need to be aware of each other’s activities during the collaborative session. The users’ activities need to be tracked (fig. 5.6-3) to provide awareness (fig. 5.6-4). Awareness information is mainly used to coordinate the activities in the collaborative session and to avoid conflicting activities in the same workspace.

**Solution**

Therefore: Model dynamic groups that are built from the set of users of the shared data who share a common interest (e.g. detected by patterns Local Awareness→ or Change...
5.2. A PATTERN LANGUAGE

Allow these groups to be established on the basis of their members’ activities (again Local Awareness→127), so that they are likely to have a common interest. Provide them with collaborative tools that support communication (Talk first →242) and collaboration (like shared editing in a Room→182).

Use the patterns of the related patterns section to detail each of the actions proposed by the above paragraphs.

Collaborations

User: The user works with an application on shared data. His work initially takes place in a single user mode. After becoming aware of other users, he starts a group with these users to enter closer collaboration.

Activity: Activities are objectifications of the actions the user performs. They represent a semantic entity formed of low-level interactions, such as key-strokes or mouse movements. Examples for activities could be

- navigating from one web page to another,
- performing a search query,
- working in the same shared space (and in this case interpreting the content of the shared space as shared data), or
- changing a method of a source file in a programming environment.

As the examples suggest, activities should be at a semantic level, which make sense for the end-user.

Artifact: A set of artifacts represents the shared data. Activities focus on artifacts.

Group: When users decide to collaborate, they form a group. This group then interacts using shared tools.

The participants (User, Activity, Artifact, and Group) collaborate in four main use cases that are addressed by related patterns:

Tracking Activities: Users perform activities on artifacts, which have to be tracked to be able to provide awareness (Gaze Over the Shoulder→142).

Providing Awareness: Based on tracked activities, the system calculates awareness information, which is indicated to users (Local Awareness→127, Change Warning→228).

Establishing Contacts: Users perceive awareness information. This means that they know, which other users they can contact to form a group (Talk first →242).
Accomplishing Collaboration: After users have formed a group, they interact in the group. Activities in the group are tracked as well in order to provide group awareness and to provide non-group members with awareness of the group’s activities.

Rationale

The pattern From Shared Data to Shared Work emphasizes the importance of providing group awareness (cf. fig. 5.7). The group formation process starts in the phase of single-user work. While the users work independently (a) on shared artifacts (b), the system tracks their activities (1). Based on this information, the system can provide awareness information about the users’ single-user activities (2). This information can, for instance, reveal whether other users currently work on the same artifacts. If a user detects another user who works in the same area, she can establish a contact with this user (3). We call this kind of group building data and activity driven group formation. Compared to the user driven group formation, data and activity driven group formation ensures that the users share a common focus, which makes focusing the group and accomplishing collaboration (4) much easier. In this model, awareness information is an important means to detect collaboration opportunities. Note that it also plays a role within tightly coupled collaboration, where awareness information can be implemented in the same way, as if the group formation had been user driven.

Figure 5.7: Data and activity driven group formation.

One might ask, how this solution differs from the application of a traditional version management system, like CVS (Price, 2000) or Envy (Pelrine et al., 2001). Version management systems also provide means for coordinating parallel work on shared data. If two or more users work at the same time with the same artifact, most version management systems create parallel editions of the artifact and leave the integration of the two editions to the end-user. That is actually the benefit that would make a user asking for the pattern From Shared Data to Shared Work: Knowing in advance, who works with an artifact, can make the user more
5.2. A PATTERN LANGUAGE

cautious because he knows that all work that he does may conflict with other users’ work. Collaboration at this point in time combines the integration aspect with the actual work – and puts the efforts for integration on the shoulders of all co-workers.

Check

Before applying this pattern, you should answer these questions:

- What are the artifacts that are used by more than one user?
- What concurrent activities do take place and what effect do these activities have on the artifacts?
- Are there any pre-defined groups to which the activity tracking should be limited? If yes, how will conflicts be resolved that were caused by concurrent groups?
- What kind of awareness visualization will you apply (cf. the related patterns section)?
- Is there already a mechanism that tracks activities or do you have to build such a mechanism?
- Do users agree that their activities are tracked? What happens if a user requests not to be tracked?

Danger Spots

Providing mechanisms for dynamic artifact-based group formation may lead to a practice that disregards formal team management at all. The pattern does not intend that formal work flows should be discarded, but that they need to be accompanied by dynamic group formation.

When designing the systems, you should take into account that users are often reluctant to change their current work environment. Thus, it is likely that totally new applications will not be used only for the sake of better group awareness (cf. e.g. (Sohlenkamp et al., 2000)). You should integrate the group awareness and team formation mechanisms as tightly as possible. The ideal case would not require any user actions for setting up the service, and users would only notice the changed application, when collaboration opportunities arose.

Known Uses

**TUKAN** (Schümer and Schümer, 2001) is a collaborative programming environment, which supports dynamic groups in eXtreme Programming.

A typical development cycle starts in a single user mode (or with two developers working together at one machine). The developer selects an appropriate task and starts coding, as if he was working alone. At the same time, other developers also start their work. All parties ignore one another until they become aware of each other because they are browsing or changing semantically related methods. The developers use
the built-in communication support to get in contact with each other and finally decide that they should solve those parts of the task together that affect both of them. The system provides means for synchronous collaboration such as a collaborative code editor.

I2I (Bradshaw et al., 2002) is a system that provides awareness on other users’ activities on the web. Whenever two or more users browse the same or a semantically related artifact, the system visualizes this as a Presence Indicator. Users are provided with means for communication to establish contacts to users working on related artifacts (Talk first).

GAMA Mall (Schümmmer, 2002) is a design study of a collaborative shopping application, which extends the web pages of the Amazon.com™ bookstore. Users become aware of other users browsing the store. The system informs the users if other users browse content, which is semantically relevant to the local user. Users can then initiate tighter collaboration, such as chatting or collaborative browsing.

Related Patterns

Related patterns are grouped into four sections: patterns for providing awareness, patterns for tracking activities, patterns for establishing contacts, and patterns for accomplishing collaboration.

You should first look at the pattern Local Awareness. This pattern outlines the most basic way of providing awareness to users of a system with shared data. It can be refined by extending the notion of locality. The pattern Active Neighbors provides a starting point for this.

All patterns that aim on the provision of awareness rely on other patterns, which track the required data from the user actions. The starting pattern for this section is Gaze Over the Shoulder.

If you feel that the application provides enough awareness, you can figure out how contact is established (cf. Talk first) and which means you may use to support collaborative episodes (cf. Collaborative Session).

The Pattern Adhoc Meeting (Coldewey, 2003) describes, how people should meet dynamically whenever an important issue arises. Although it does not relate to any technical issues or distributed team settings, it shares the same idea.
5.2.3 Local Awareness

Alternative name(s): Contextual Awareness, Document Awareness (e.g. (Cohen et al., 2000))

Intent

Provide awareness information in the context of the artifact, which is in the local user’s focus.

Context

Many users are working on a set of shared artifacts. You have already decided to use the pattern From Shared Data to Shared Work to encourage the formation of dynamic teams. The groups should be focussed on a shared current interest, which matches their current actions.

The application, which is used to access the shared data provides means for manipulating or accessing the data and for keeping it consistent only.

The users’ current state is not yet reflected in the user interface. Especially, there is no means to find out whether or not other users are available who share a common interest.

Problem

Although most systems that work on shared data provide support for coordinating shared access, they often don’t tell the user, who is working on a specific artifact. Such information is needed to establish ad-hoc teams that share a common focus. Without such information, users assume to work alone – and do not see the possibility or urge for collaboration.

Scenario

Imagine a real world plaza, where people meet for social interaction. This plaza is popular, since it offers many services, such as cafés to hang around, notice boards where citizens of the community post their private adverts, and shops where one can buy, for instance, food or books. Since the plaza is crowded, it motivates talking or resting in the community.
Now consider a virtual community web site. It also offers bulletin boards, chat-rooms, and links to virtual shops. All these facilities are arranged on a central portal page. Although all communication means and services are available, the users often feel isolated at the portal page. The community web site looks as if no users were using it and does not motivate people to rest and talk in the community. Especially, the users don’t see a way to meet by chance, which is one characteristic of a real-world plaza.

**Symptoms**

You should consider to apply the pattern when . . .

- users cannot say, whether or not other users work with the same artifacts.
- users always have the impression of being on their own.

The background of the problem lies in the different nature of real-world interaction and virtual interaction. If people interact with real-world-artifacts, they have to be physically co-located with this artifact. In the case where more than one user works with the artifact, these users have to be co-located as well since all of them have to be co-located with the artifact. When users interact with artifacts in a virtual space, co-presence is not needed since all interaction is computer-mediated (and thus can be performed by means of computer networks). The users still interact with and have a relation to the artifact, but they don’t have to get in contact with the other users anymore.

**Solution**

Therefore: Provide awareness in context. This means that the system tells the local user, who else is currently interested in the local user’s focussed artifact and what they do with this artifact. Show this information whenever the artifact is shown on the screen. The information should contain details about the user drawn from his user profile, the artifact, and details on the activity, which the user is performing. Ensure that the information is always valid.

**Collaborations**

**Artifact:** The unit of shared data, the users work on. The granularity of the artifacts has to be determined by the application designer. Examples are web sites, single pages of a web site, or more application specific artifacts, such as classes and methods in a programming environment.

**Activity:** The representation of a user’s interaction with the shared artifact. It may be a modifying activity such as changing a web page or a reading activity, where the artifact is perceived by the user. In many contexts, it is not trivial to detect which activities take place. The GAZE OVER THE
SHOULDER-145 pattern points out how activities can be detected.

**Local User / Confocal Users**: The local user interacts with the system and wants to get informed about confocal users who interact with the system from a distant location. *Two or more users are considered as confocal, if their current focus is on the same artifact.* This term was chosen to clearly distinguish between physically co-located users who work at the same physical location and confocal users who work with the same virtual artifact.

The set of confocal users of an artifact contains all users who have started an activity with the artifact, which is not yet finished. Several known uses don’t distinguish between local and remote users when looking for activities. This implies that the set of confocal users for a specific artifact that is currently watched by the local user always includes the local user. Note that all users are local users at their machine. On the other hand, making the distinction reduces awareness information to awareness on remote users (only activities performed by remote users are considered).

**Awareness Information**: Awareness information represents the status of remote users and is displayed to the local user. The information may be at different levels of detail. For a small set of confocal users, it may provide information on the other user’s identity.

**Rationale**

By explicitly telling the user that also other users are working with the artifacts (or more general, work at the same virtual location), these users get aware of each other, which is the basis for establishing contacts.

As with FROM SHARED DATA TO SHARED WORK-120, one might ask how this solution differs from the application of a traditional version management system. If the only goal would be to avoid other users from overwriting their colleagues work, this might well be reached by using a version management system, which avoids parallel work on the same artifact (locking) or highlights conflicts before they can be committed to the version management system (such as in CVS (Price, 2000)).

But the important point in this pattern is that the user gets aware of activities that currently take place on the artifact – not activities that are completed, as it would be the case when a user decided to check in a new version. This opens the opportunity to collaborate *within the activity* or adapt the activity to the changed situation as soon as possible.
Another difference to version management systems is that the Local Awareness pattern considers a broader range of activities – not only modifying accesses like in version management systems but also reading accesses to the shared data.

**Check**

*Before applying this pattern, you should answer these questions:*

- What artifacts do users use?
- Can you identify the artifacts at different levels of scale? If yes, what is the right level of scale for the awareness calculation?
- How do you store the activity information? Can you make use of a central activity database (e.g., an Elephant’s Brain)?
- Will you be informed when new activities take place or existing activities terminate? If no, can you think of an implementation of the Gaze Over the Shoulder pattern in your context?
- Are there situations in which Local Awareness should be turned off?

**Danger Spots**

The granularity of artifacts and the size of the user community determines, whether or not users will be aware of one another.

If the system is very crowded (i.e. many users are working on a small set of artifacts), then the number of confocal users will be large at each point in time. Displaying all confocal users as distinguishable persons may produce an information overload and distract the local user from his task. In this case, it is better to simply tell that there are some or a specific number of confocal users without revealing their identity. On the other hand, displaying only the information that users are sharing a focus on the artifact without providing numbers for the size of the crowd does not provide any useful information in this case. This indicator will always be active.

If the system provides access to many artifacts and the group of users is small, the probability that two users work on the same artifact will be very small. In this case, you can either reduce the granularity of artifacts or apply the pattern Active Neighbors, which extends the presence to other semantically related artifacts.

An important issue for the acceptance of the pattern is trust and privacy. User monitoring does only work if it is mutually accepted by all participants. Otherwise, you will soon get the effect that users complain about being monitored or don’t use the awareness features anymore. The pattern Masquerade addresses this problem.

When using this pattern you should not mix up tasks and activities: This pattern does only propagate current foci of users in
terms of the artifact that they are currently interested in. It does not reveal information on the users’ current task besides the interaction that can be monitored from the activity itself. The question that this pattern solves is not how to bring together two or more people with a related task, but how to bring together people who use the same artifact to fulfill their potentially independent tasks.

Notifying clients about the users’ current activities can be technically difficult if the communication cannot be initiated by both, the producer and the consumer of the activities. In this case, ensuring that the awareness information is always valid requires an additional communication channel or polling requests of the consumer of the notifications. Especially, traditional request-response communication channels as used, e.g., in the HTTP protocol are insufficient to let the producer of the activities inform the consumer (assuming that the consumer plays the role of a client for the activities). The local user’s system thus needs to act as a receiver for activity notifications from other users and as a sender of activity notifications from the local user.

**Known Uses**

**Textual Indication on Community Websites** Many community web sites tell how many confocal users are currently on the site. The notion of focus is very broad at most of the sites. A user can have a focus on the site, but not on individual pages. An example for this kind of LOCAL AWARENESS is the home page of phpWebThings, an open source portal system (http://www.webservicenet.com/wt/news.php). It shows how many other users are currently connected to the site as a text message (fig. 5.8).

![Users Online](http://www.webservicenet.com/wt/news.php)

**Figure 5.8:** Confocal users at the phpWebThings community site.

This kind of awareness information is the easiest variant of the pattern. All users are interacting with the same artifact (namely the web site). It is not distinguished between local and remote users. Users are able to surf the web site anonymously as guests but not without leaving traces.

Some indicators at web sites extend the awareness information to tell on which page the other users currently are. An example for this is the community page www.mvnforum.com (fig. 5.9). It shows the duration that they spent on the page and how long they spent on other pages of the same web site.
CHAPTER 5. THE PATTERN LANGUAGE

Figure 5.9: Confocal users at the mvnForum community site (http://www.mvnforum.com/mvnforum/listonlineusers).

Odigo (Odigo, 2001) is an instant messaging client, which tracks the users while they are navigating the WWW. It shows confocal users in an external view (fig. 5.10). The user can chose whether he is interested in the current page or in the whole web site.

Figure 5.10: Odigo (Odigo, 2001) showing confocal users at www.amazon.com.

CSCW3 The CSCW3 prototype (Gross, 1999) is a system that shows for a given URL who else is browsing this URL. It is a special kind of browser that logs user activities and displays confocal users in an external window (fig. 5.11). The local user can see, who is currently on the page and who has been on the page recently. The latter is a behavior that goes beyond the pattern’s proposed functionality. It is described in the TIME COMPRESSOR pattern.

CoCoBrowse (Ter Hofte et al., 1997) is a specialized browser,
5.2. A PATTERN LANGUAGE

Figure 5.11: The CSCW3 prototype (Gross, 1999).

which shows in an attached window who else is currently viewing or editing a specific web page.

Gaze Over the Shoulder \(^{145}\) tracks user activities. Information on users’ activities is needed to calculate the set of cofocal users.

Time Compressor \(^{242}\): Use the time compressor pattern, when users don’t work at the same artifact at the same time and you still want to bring together users who worked on the same artifact successively.

Presence Indicator \(^{135}\): If screen space is limited, you can use a Presence Indicator \(^{135}\) to present information on the cofocal users in an iconic representation that saves space.

Change Warning \(^{228}\): For simplicity reasons, the current pattern assumed that all activities have the same quality. This is not the case for systems where users can change artifacts.
In these systems, you should consider to highlight possibly conflicting activities using the Change Warning.

User List: User Lists have been applied frequently in shared workspace systems. Although these lists have many commonalities (the visualization can, for instance, be the same), there is a crucial difference in the intent. A traditional User List intends to provide awareness on participants who have already joined a collaborative session. Group formation is not a goal of User Lists.

Publisher-Subscriber (Buschmann et al., 1996) can be used to keep the calculation up-to-date.
5.2.4 Presence Indicator

Intent
Indicate that remote users look at an artifact, which is visible on the local user’s screen.

Context
You have generated awareness information on confocal users using the Local Awareness pattern or on peripheral users with the Active Neighbors pattern. You have also decided to use an In-Place Awareness View to ease the process of connecting awareness information with those artifacts that are affected by the awareness information. This means that you are showing the awareness information in a close proximity to the original artifact’s visualization.

Now you are thinking about how to visualize the awareness information gathered.

Problem
The In-Place Awareness View makes it easy to connect other users’ activities with focussed artifacts. But the surrounding of the artifact provides only limited space for information. Awareness information thus competes with application data.

Scenario
Think of several users who are looking for books in a virtual bookstore. They want to be aware of one another to discuss the books, if they are interested in the same book. The virtual bookstore provides Local Awareness and Active Neighbors to calculate confocal users or users who work in a semantic nearness of the displayed artifacts.

Figure 5.12 shows how awareness information was displayed on the screen. The developers chose to provide the awareness information next to the artifact’s representation on the screen. This information can be found next to the mouse pointer in figure 5.12.

The explanations of the awareness information clutters the whole page layout because it uses too many words. In addition,
the textual representation of the awareness information is not easily distinguishable from the domain information (in this case the information on books). That makes the information hard to perceive and disturbs the user from his actual task (to look for books).

**Symptoms**

*You should consider to apply the pattern when...*

- the available information channels to transmit information to the user don’t have much more capacity than what is needed to transmit the shared data.

- in GUI systems there is not much space for showing information next to the artifact so that the awareness information destroys the screen layout.

- external awareness views, such as a **Radar View**[^1], do not work because the artifacts cannot easily be described without their original context. For that reason, users find it hard to associate awareness information with the corresponding focussed artifact.

**Solution**

*Therefore: Limit the size of the awareness information’s representation so that it uses only a small part of the*
available information channels. For a GUI system, this means that you should represent the confocal or peripheral users as a single icon instead of a long textual form. Focus on telling that there are other users, rather than providing much information on the other users’ identity or task. Ensure that the indicator differs from the other artifacts representing application data.

**Collaborations**

The main participant of the pattern is the *indicator*, which represents the awareness information. Each indicator displays awareness information for activities of confocal users or active neighbors that are referencing one specific artifact. The indicator is placed right next to this artifact. The indicator can be an icon in GUI systems, a special sound in audible interfaces, or any real-world object in *tangible interfaces*. It should be distinctive from the application’s shared data.

Wherever possible, the indicator has built-in behavior, which reveals more clues on the awareness information. For instance, if *tool tips* are available in the user interface, the indicator can explain itself with a tool tip when the mouse pointer touches it.

The sets of *confocal users* or *active neighbors* can be characterized by different group aspects:

- the cardinality (i.e. the number of confocal or nearby users),
- the distance to the artifact (this is always 0 if the users are confocal; for active neighbors, it represents the semantic distance of the user to the artifact), and
- the users’ identities (which is uncomplicated if it is only one confocal user).

It is up to the application designer to map the different group aspects to the indicator. In general, size, color, and shape can be used as dimensions to represent the above aspects.

A *mapping function* can – in most cases – automate the projection of group aspects to the different dimensions of the indicator.

**Rationale**

The indicator has two main advantages that help the user to notice the awareness information without being disturbed by it: Firstly, it does not use much space (resp. bandwidth of the information channel) and therefore it can in most cases be integrated with the representation of the artifact. Secondly, the indicator is visual different to the rest of the displayed information. By carefully keeping the indicator distinguishable from the artifact’s visualization, users will no longer mix the awareness information with the shared artifacts.
Figure 5.13 shows how the awareness information can be displayed using presence indicators. In this design, only one aspect of the confocal user group is reflected in the icon (next to the cursor in figure 5.13): the color represents the distance of the user (or the user’s) to the artifact. In the above case, red (dark) icons represent users who browse very related books whereas green (light) icons represent users who browse more different books.

By placing the icon next to the artifact one directly connects users to artifacts. Especially in settings where the users do not collaborate in the same workspace this is the only reference point for collaborative activities. This gets even more important if one addresses the asynchronous case (as stated by Dourish (1997)).

**Check**

Before applying this pattern, you should answer these questions:

- What is the focus of the visualization? Is it to represent users or to provide awareness on activities?
- What icon shape best represents the (groups of) users?
- Is cardinality of users/activities important? If yes, can you vary the size, the color, or the brightness of the icon?
- Are there different levels of importance for different activities or users? If yes, can you reflect the importance of the most
important activities or users in the presence indicator’s color or shape?

- Can you provide a tool-tip? If yes, what details will you show in the tool tip?

**Danger Spots**

Ensure that the icon’s visualization does not conflict with other visual elements on the screen. Conflicting means in this context that the application itself uses icons of a comparable shape or color. Use a **Distinct Awareness Info** in this case.

**Known Uses**

**TUKAN and GAMA-Mall** both use little figures to represent confocal or nearby users. GAMA-Mall (Schümmcr, 2002) was already shown and discussed in the motivation and rationale sections of this pattern.

![Figure 5.14: Presence indicators in the TUKAN team programming environment.](image)

The programming environment **TUKAN** (Schümmcr, 2001b) uses colored figures that are attached to methods to show that other users are working on the same or related artifact (fig. 5.14).

**PoliAwaC** (Sohlenkamp et al., 2000) is an awareness client for the PoliTeam collaborative workspace application that was used by members of the German government ministries in Berlin and Bonn. Members exchanged and organized documents using PoliTeam, which were formerly circulated in folders (when the whole ministry was in Bonn). In PoliAwaC, the users can browse shared artifacts using an explorer-like view (fig. 5.15). If an artifact is currently viewed by another user, the artifact’s icon is changed: the size is enlarged to make these documents more prominent and parts of the icon are blended with a color that represents the user who is currently viewing the artifact. A smily icon is finally added to the enlarged icon to visualize the process of viewing. The system was evaluated in the ministry (on daily use) and the awareness icons were regarded as useful. The only problem
that users reported was the mapping between icon-color and user color.

**Flying flags** as the one shown in the introductory picture are real-world representations of presence indicators. Whenever the sovereign arrives at the castle they fly the flag. Thus, anyone can see whether or not it makes sense to go to the castle and contact the sovereign.

The **Presence Indicators** in the PoliAwaC awareness-client (Sohlenkamp et al., 2000).

**Figure 5.15:** Presence Indicators in the PoliAwaC awareness-client (Sohlenkamp et al., 2000).

---

Related Patterns

**Change Warning**→228: When users manipulate the shared data, it might be more important to report on other user’s modifications to the shared data than reporting on other user’s presence. The Change Warning pattern does so, by indicating confocal modifications of other users.

**Color-Coded Distances**→228 can be used to visualize the fact that the presence indicator displays the presence of a user at a related artifact and not at the artifact to which the indicator is attached. Use this pattern if active neighbors should be displayed.

**Interactive User Info**→199 describes how to design the awareness info so that it can be used as a means for establishing communication or collaboration sessions.

**Distinct Awareness Info**→229 discusses how the visualization can be designed so that it does not disturb too much from the user’s current task.
5.2.5 User List

Intent
Show who is currently participating in a session.

Context
You have managed to establish a Collaborative Session between several users. These users start to interact. Now you are thinking about ways for communicating group membership awareness.

Problem
Users who only share common data have no mutual awareness of each other. Users don’t like to perform activities if they don’t know who observes them. Thus, they will not act freely with the shared artifacts.

Scenario
Fred and Thelma frequently meet in the room at noon and start chatting about problems with an architecture they are currently designing. They start to discuss about how to convince the lead architect Steven about necessary changes to his proposed architecture and at the same time avoid conflicts with Steven. However, Steven got conspicuous and entered the chat room at a quarter to noon. Unfortunately, Fred and Thelma could not see that Steven was already there. The consequences are up to your imagination.

Symptoms
You should consider to apply the pattern when . . .

- effective interaction requires that other users can be identified.
- session membership may change over time.
- not all users of a system collaborate in all sessions.
- distributed users cannot see one another.
- users frequently broadcast requests like “Hi, who else is present here?”.

Solution
Therefore: Show the names of all users who are in the same session in a user list and ensure that the list is always valid.
Collaborations

The user’s name is added to the session’s user list as soon as he enters the session. The user list is a visual representation of all users who are currently participating in the session. When a user leaves the session his visual representation is immediately removed from the user list. The user list acts as a subscriber that gets informed whenever the underlying model (the set of participating users) changes, e.g., using the Publisher-Subscriber pattern.

Rationale

By adding a user list, all participants can see who else is part of the group. They can be sure that only the users on the user list can see their activities. This is the first condition to establish trust in the group. If the local user trusts all other present users, he will act freely.

Check

Before applying this pattern, you should answer these questions:

- What is the context of the User List? Does it make sense to show a user list for the whole application or should you rather distinguish between session-centered user lists?
- How many users do you expect in your user list?
- Can you use graphical representations in the user list (Virtual Me→197)?

Danger Spots

When the users enter the stage – namely that they perform an activity in a shared workspace – the system adds them to the User List. The users are removed when they leave the shared workspace by moving on to another workspace. This works fine if all users always move between workspaces or explicitly log-off the system. But what happens if users simply switch off their computer or loose their network connection? Their user representation will be part of the User List forever. To detect whether a user is still available use timeouts for the execution time of a specific function or use server-side or client-side mechanisms.

Not all visualizations of the user list do scale for large numbers of users. If you consider, e.g., to list all names in a text field, the number of users is limited by the length of all concatenated user names, which is constrained by the length of the text field.

Known Uses

MSN Messenger shows a user list in each chat window. Each user is shown with a user picture (cf. Virtual Me→197). In contrast to other systems it only shows remote users in the user list (in the right middle of fig. 5.16). The local user is shown in a separate place (cf. Who Am I→191).

COAST (Schuckmann et al., 2000) generates user lists by visualizing the set of interested users of a shared application model.
5.2. A P A T T E R N L A N G U A G E

1. Select the local application model (e.g. ChatAppLocal) of the application that needs a user list.
2. Add a read-only COASTInputfield to the windowSpec using the user interface builder.
3. Name the aspect for the input field userlistText.
4. Open a methodBrowser on the method ChatAppLocal>userListText (use theclassnames according to your application model).
5. In the getValue-block retrieve the names of the interestedUsers for the sharedApplicationModel and combine them into one string. The code may look like this:
   ```
   [userString := ''. 
   self sharedApplicationModel interestedUsers 
   do: [:aUser | userString := userString, aUser name printString] 
   separatedBy: [ userString := userString, ', '].
   userString].
   ```
6. Clear the setValue-block to ensure that no value can be assigned to the field.

UNIX who: The who command in Unix can be used to show all users, who are connected to the same machine.

```
max% who
schuemme pts/2 Aug 4 10:59 (pd9e9c14c.dip0.t-ipconnect.de)
```
Although the users do not necessarily collaborate, they share the same resources in their session.

**Related Patterns**

- **Collaborative Session** and **Room**: Sessions or rooms in most cases include a user list.
- **User Gallery** lists all users who are known by the system regardless of their current collaboration status.
- **Buddy List** limits the set of shown users to known users. This is especially useful if a large number of users share the same session.
- **Local Awareness** shows the presence of users who focus on a specific artifact. It provides detailed information regarding the users’ interests (and often shows less users). The **User List** instead shows all users who are in the same session, regardless of the artifact that they are focusing on.
- **Virtual Me** supports users in generating an electronic representation of themselves. These representations can be used in the user list to make the entries of the list more personal.
- **I Am Alive** (Saridakis, 2003) requests the client to send notifications showing that it is still available. If these notifications fail to appear, remove the corresponding user from the list.
- **Are You Alive** (Saridakis, 2003) When using this pattern, the sites regularly check whether another site is available. If not, remove the corresponding user from the list.
- **Session Timeout** (Sørensen, 2002) uses upper bounds for the time in which a session object has to be accessed before it is marked invalid. Similarly, one can use upper bounds for time between two activities of a user shown in the **User List**. As soon as the upper bound is passed, remove the corresponding user from the list.
- **Publisher-Subscriber** (Buschmann et al., 1996) can be used to keep the calculation up-to-date.
5.2.6 Gaze Over the Shoulder

Alternative name(s): Event Sensors (Prinz, 1999)

Intent
Detect the users’ interaction with shared artifacts.

Context
Users interact with shared artifacts using proprietary tools and want to log activities for future reference. The tools modify the shared artifacts. The results of the modifications are visible within the tools and in the changed shared artifacts.

Problem
Many proprietary tools are not designed for extendability. They do not provide means to modify the application’s behavior. This makes it difficult to add automatic tracking of user’s activities, which you would need to provide awareness.

Scenario
Imagine a seminar where Alice, Jane, and Sally, a group of students, are preparing a talk. They decided to do literature research independently using the web. Each of them signed up for a section of the topic and starts her search. After two weeks, they meet again and merge the results. Each of the students has written down some URLs, which she had rated relevant. During the discussion, Jane noticed that one of the pages found by Alice is very related to some information that she encountered during her research but did not rate relevant enough to add it to her URL list.

Unfortunately, she does no longer remember the URL of that page. Since remembering a page required that Jane actively set a bookmark in her browser, she only did this for those pages that she expected to be relevant. This information was her final result of her single user task – namely the research in her section of the topic.

If she had also remembered the path that led her to the solution, her team mates could have benefited from this information. On the other hand, it would have been a large overhead to manually save each page.
Solution: Add an additional layer in the communication between the application and the shared data to monitor user actions. Allow other parts of your application (e.g. a Elephant’s Brain or a Local Awareness) to subscribe to monitored activities.

Collaborations

Request: The user performs activities on shared data. Activities are internally represented as requests from a user interface element (in the broadest sense) to a shared object. An example for a request could be the request to save a file in a text editor.

Listener: The listener observes the communication channel and the shared object and forwards information on activities to a subscriber such as the Elephant’s Brain. The listener seems to be transparent. That means that the user (resp. the source of the request) does not notice that the request passed a listener when it was transmitted via the communication channel.

Communication Channel: Requests are transmitted using a communication channel. To apply the pattern, one needs to be able to monitor this communication channel.

Since the data is accessible by more than one user, there has to be a communication channel by which the users access the shared data or by which they are informed about state changes. All communication that takes place via this channel can be monitored (if it is not encrypted) by the listener by either observing state change that results from communication activity or by directly grabbing the information from the communication channel.

In a point-to-point communication, this requires that the listener acts as a man in the middle. It receives all requests from the application and forwards these requests to the shared object (thereby playing the role of the original sender of the request).

In a broadcast communication infrastructure, the listener can just act like any other receiver (and listen for the same requests as the intended receiver).

Where the communication channel is located depends on the application’s communication architecture:

- In WEB applications, it can be the TCP-connection that is used to transfer HTTP requests. TCP connections can by definition have many intermediaries such as proxy hosts, routers, or gateways. To attach to this communication channel, a listener could act as a proxy. It would perceive a user requesting an URL. On the other hand,
WEB applications model most interaction within URLs. The listener would in this case need to parse the URL (or even the response as well) to retrieve more information on the user’s activity.

- In applications that work with files on a shared file system, the communication channel is the file access API, which is often again a network protocol (e.g. in NFS (Sun Microsystems, 1989)).

It can also be a state change in the file system that indicates that an activity takes place. If users, for instance, start working with a MS-Word document called MyText.doc, MS-Word creates a temporary file named ~$Text.doc, which contains, among others, information on the user who opened the document. When the user closes the document again, this file is removed. A listener would in this case monitor the directory, where the shared files reside and track the creation and the removal of temporary files. It would perceive

- a user starting to work with a document,
- the creation of a new version of the document, and
- a user ending his work with the document.

Rationale Since the proprietary tool does not allow other applications to engage in the internal application logic, the only way to distribute state information is by observing external communication channels.

If the observer of the channel notices communication, it can redistribute this information to subscribed users (using, for instance, a shared repository that holds information on all activities, such as the ELEPHANT’S BRAIN→150). In this way, it is possible to calculate awareness information without changing the original application.

Check Before applying this pattern, you should answer these questions:

- What is the communication channel in your application?
- Can you identify the user from messages observed at the communication channel? If yes, how is the user represented? If not, could the application of the WHO’S THAT GIRL→245 pattern help?
- Can you identify the accessed artifact from the observed communication? Is the identifier unique?
- Can you store the identifier directly in the activity or do you need a PROXY OBJECT→237?
- Will you observe the communication channel or will you act as a man in the middle (i.e., as a proxy)? Will the communication be affected by the listener?
CHAPTER 5. THE PATTERN LANGUAGE

Ensure that the user knows that he is monitored. Users have to be able to decide on their own whether or not they want to be monitored (cf. MASQUERADE→196). And they have to feel that being monitored adds value to their activities (e.g. because of the principle of RECIPROCITY→238). Otherwise, they will in most cases find a way to do their job bypassing the monitoring mechanisms.

NESSIE: The NESSIE framework (Prinz, 1999) provides event sensors, which are comparable to the listener in this pattern. The sensors are placed next to specific artifacts. Whenever an activity is performed on the artifact, the event sensors create an activity object, which is then stored in the NESSIE awareness server (see the description in ELEPHANT’S BRAIN→150).

User-Centered Push (UCP): Underwood et al. (1998) proposed an architecture that monitors the user’s interaction with the WWW and makes this information available to other users. Listeners (called information sleuths) monitor TCP streams through which a user accesses content on the web. Whenever content is accessed, information on this access (referred as fact) is sent to a shared access log (called board).

Proxy-Logger: The Proxy-Logger (Hong and Landay, 2001) is a web proxy, which tracks the user’s browsing activities to create diagrams that visualize the browsing actions involved in a specific task. The listener is modeled as a proxy. Users enter the URL, which they want to visit, in the proxy’s welcome page. The proxy then modifies all delivered HTML content, so that all subsequent URL’s are also retrieved by the proxy.

GAMA Mall (Schüffer, 2001a) also uses a proxy approach to monitor web browsing activities.

Related Patterns

ELEPHANT’S BRAIN→150 provides means for storing monitored activities.

MASQUERADE→196 provides the user with means to switch the monitoring mechanisms on and off.

RECIPROCITY→238 is an important issue when users are asked to provide personal information. Make sure that the user perceives a personal benefit by providing personal information.

Proxy: The most obvious method for implementing the GAZE OVER THE SHOULDER pattern is to use a PROXY pattern (Rohnert, 1995) in a way that the communication channel is redirected through the proxy. But the proxy approach is only one possibility to keep track of actions. As discussed in
the participants section, there are different points where the listener can be attached to the communication channel. The most appropriate point depends on the nature of the communication channel.

**Navigation Observer** (Rossi et al., 1995) solves a comparable problem in the context of hypermedia applications. As in **Gaze over the Shoulder**, actions are recorded for future reference. The main difference is that the **Navigation Observer** implements a full Observer pattern (Gamma et al., 1995) with the implication that update messages are sent from within the application directly to the observer. The application (i.e. the controller) thus needs to know its observers, which are notified on state changes. This is in conflict with one constraint of the **Gaze over the Shoulder**’s problem: *the application should not be changed*. The observation mechanism in **Gaze over the Shoulder** is in contrast independent of notification mechanisms of the observed application.
5.2.7 Elephant’s Brain

Alternative name(s): Event History, Activity Log

Intent
Store information on the users’ activities in a log to allow the users to understand (other) users’ activities and the artifacts’ evolution.

Context
Users perform parallel activities on shared artifacts without being totally sure about the effects. They tend to try out solutions without knowing whether or not the effects will be correct. Another issue is that they are not always able to tell which artifact they will use for their activity, and often activities conflict with other users’ activities.

Users work with the shared artifacts for a long time and it may be necessary to review artifacts that were manipulated in a former work episode (e.g. the programming work of the previous week, month, etc.).

Problem
Merging two users’ (past or current) work is a difficult task. It requires that the activities are transferred to the same context and that the goals are aligned. But many applications don’t provide access to the artifact’s history, its use, and its evolution. Thus, merging is vulnerable to errors and often collaboration does not take place since the merging efforts exceed the estimated gains of a collaboration.

Scenario
Imagine a single user software development environment. The programmer is developing an application using a development approach, where testing and development are two alternating phases. She begins with writing a test for the desired functionality – assume that she is looking for an algorithm that calculates all prime numbers. In the test, she checks, whether the algorithm finds the prime numbers 3, 5, and 7.
She then continues to write software that fulfills the test - let’s say that the algorithm just checks whether the number is larger than 1 and odd.

In a next step, she creates a second test that ensures that 9 is not a prime number. Writing code to pass the test brings her to the idea of changing the code in a way that it checks whether the number is odd and between 3 and 7.

A next test demands to also accept 11 as a prime number. At this point, the programmer notices that the last change was not really correct. Thus, she has to revert the change. But unfortunately the development environment does not support her. She has to remember the change that she applied and try to recover the previous code version from her memory. It may be simple if the action that needs to be reverted is the last action that she performed. But often one does not see the problems of a specific activity directly after the activity was performed. It is rather often the case that one identifies the effects of wrong activities a long time after the activity was performed.

Now consider a scenario where two programmers work on the same project. Each programmer solves different problems, but both problems involve the same shared artifact. Even if the first programmer does still remember which changes she performed, the second programmer cannot revert these activities, because he does not know what his colleague did.

Symptoms

- users often notice that an artifact changed, but could no longer remember how it looked before.
- users cannot detect who performed incorrect changes, and what these changes were.
- users cannot find out what another user X did to the artifact Y yesterday.
- users don’t understand the changes of other users.

Solution

You should consider to apply the pattern when . . .

Therefore: Remember all activities that users perform on shared artifacts – not only modifying accesses, but also read accesses. Provide access to the activities, so that a user can understand (and merge) other users’ activities with his own activities.

Collaborations

Two main participants are involved in the pattern: the log and the activity. A log is a container that holds the activities. Logs should be persistent to allow the users to refer to activities of previous sessions.
The activity includes information on

- the type of the activity, such as reading, editing, creating, removing, etc.,
- the artifact that was touched by the activity (often in two versions: the version before the activity and the version after the activity),
- the time at which the activity took place (often as a set of two timestamps representing the start and the end of the activity), and
- the user who performed the activity.

In several systems, activities further include a user comment, which provides more information on the activities' intent. Activities should comprise one semantic user transaction with the system. Examples could be the selection of a menu command, the insertion of a new paragraph in a text document, or the viewing of a web page in a web browser. Activities should be updated immediately when a user interacts with the application.

Clients interact with the log by adding and consuming or querying activities. The log should be accessible from all clients who need to perform calculations on it or who submit activities for log storage. In collaborative applications this implies that it can be accessed using a network protocol.

For the display of up-to-date collaboration information, the log can serve as a Publisher of new activities according to the Publisher-Subscriber pattern (Buschmann et al., 1996). Clients who use the log to provide awareness information, subscribe to activities. They provide an activity pattern that describes relevant activities. The publisher will inform the subscriber whenever activities matching the pattern change.

Rationale

By logging all activities that users perform, one can inspect all activities later. This helps to understand the evolution of a specific artifact (by looking at all activities that took place on the artifact) or a specific user’s work (by looking up all activities, which were initiated by a specific user).

Since the log is persistent, the information will be remembered also when the user has forgotten it.

The Elephant’s Brain goes beyond version management functionality in two main points:

1. It also remembers accesses that did not change the artifact. These accesses are not mandatory for restoring the system at a specific point of time. But they help to understand the user’s
background for a specific activity, and in some cases they help to reveal conflicting activities.

2. The Elephant’s Brain provides means for notifying interested users on activities, which take place with a specific artifact (note that some version management systems support this, which is the reason why CVS is included in the known uses section, although it does not remember non-modifying accesses as it was discussed in the previous paragraph).

Check

*Before applying this pattern, you should answer these questions:*

- Do you have a central server component on which you can place the log? If not, how do you distribute the logs between clients?
- Who will create the activity object, the client or the server? What information is needed to create and initialize the activity object?
- What information will you transmit to the server if you generate the activities at the clients?
- How do you support efficient queries of the activity log? How do you handle large numbers of activities? Can you use a database that to store the activities?
- Does it make sense in your context to create an activity log for each artifact (i.e., store the activities together with the artifact)?

Danger Spots

Ensure that many users can add activities to the log at the same time. One way for achieving this is to model the log as a bag to which users can add activities in any order producing the same result. Since each activity is unique (distinguishable user and time), there will be no conflicts in adding activities to the log.

When referencing artifacts in the activity description you have to make sure that the right version of the artifact is referenced. In environments, where no versioning is available, you should include sufficient information to restore all different versions of an artifact from inspecting all activities that took place on the artifact.

Prinz (1999) argues that the activity log should be decoupled from the application, which generates events. This allows an easy extension to support a large variety of events generated by different client applications.

Note that the Elephant’s Brain can only store those activities that it gets informed of. For off the shelf tools like CAD packages, this might just be the creation of a new version. You should therefore carefully examine the application to monitor as
much information as possible since this information is crucial for a
detailed (and intelligent) group support. Refer to GAZE OVER
THE SHOULDER→145 for hints on how to gather such information.

**CVS version logs:** The CVS version management system
(Price, 2000) maintains a file *history*, where meta-
information on all activities is stored. This information in-
cludes the user, the touched artifact, and the type of the mod-
ification. The version information can be obtained using the
cvs *history* command. An example output looks like this:

```
cvs history -c
M 2003-02-27 13:09 schuemm 1.2 README test == ~test2\test
A 2003-02-27 13:14 schuemm 1.1 short.txt test == ~test2\test
M 2003-02-27 13:16 schuemm 1.2 short.txt test == ~test2\test
```

Each line in the history output describes one user action within
the CVS repository. It shows a type-code of the activity, its
time, the user who performed the activity, and the artifact
that was modified.

One can configure a cvs server to run any software when arti-
facts are accessed. This can be used to allow distributed clients
to subscribe to changes according to ELEPHANT’S BRAIN. A
subscription mechanism as it was proposed by the PUBLISHER-
SUBSCRIBER pattern, however, is not part of the standard cvs
system.

CVS only logs accesses to the repository. This is sufficient
to support a CHANGE WARNING→228. For the provision of a
PRESENCE INDICATOR→135, it is too few information. Pres-
ence indicators would need information not only on those ac-
tivities that were performed to copy shared artifacts to a local
workspace. It would also need the information on what the
users do in their private workspace.

**VisualWorks Change Set:** The VisualWorks Smalltalk environ-
ment (Cincom, 2001) writes all changes to a change file. This
change file is mainly intended to recover from system crashes,
but an important additional use-case is the inspection of chan-
ges of an artifact.

VisualWorks provides a special changes browser for this, which
shows all changes of a specific source artifact.

Since the programming environment is a single user environ-
ment, it does not provide any multi-user access to the change
file.

**NESSIE Awareness Server:** The NESSIE Awareness Server
(Prinz, 1999) is a general purpose awareness server, which
stores events. Each event carries information on the originator, the action, the touched artifact, and the time at that the event took place. The events are implementations of ELEPHANT’s BRAIN’s activities.

Client applications can add activities to the server using an HTTP-based interface. They can also subscribe to changes by specifying the type of the activity and a desired context (i.e. the touched artifact’s location).

**BSCW** (Bentley et al., 1997b) is a shared workspace system that logs all activities, such as reading, editing, or moving documents, that are performed in the shared workspace. The logfile captures events that look like this:

```
User:[52, 'Alice']
object:[134, 'Home']
Type:CopyEvent
Time:1030181775.14
Path:[[63, '&NewHome']]  

User:[51, 'Bob']
object:[124, 'BusinessPlan.doc']
Type:CutEvent
Time:1030182090.44
Path:[[119, 'Workgroup']]  
```

The example shows parts of two activities. First, a user Alice has copied an artifact called Home to a folder NewHome. In the second activity, Bob cut the artifact BusinessPlan.doc.

**TUKAN** (Schümmmer and Schümmmer, 2001) is a collaborative programming environment that extends VisualWorks Smalltalk. It logs all activities that users perform in the environment (e.g. reading source code or modifying a class file) and stores these activities in an activity log.

The activity log is stored as a shared object, which is replicated to all clients using the COAST framework for synchronous groupware (Schümmmer et al., 2001). The replication mechanisms also include a distributed version of PUBLISHER-SUBSCRIBER to trigger view updates or other actions when any log entry changed.

---

**Related Patterns**

**Proxy Object** → 237: If the artifacts cannot be accessed directly in the application one has to use a meta representation in the activity description.

**Gaze Over the Shoulder** → 145: When the application’s internal control flow cannot be extended to monitor activities, use the
Gaze Over the Shoulder pattern. It will seek for activities, which are then forwarded to the Elephant's Brain's log for future reference.

Local Awareness: Use Local Awareness to inform users of different activities that currently take place at the same artifact to avoid conflicting work. After the activity was performed, remember it to support users in resolving previous conflicting activities.

Model-View-Controller: If the application is implemented following the Model-View-Controller pattern ((Krasner and Pope, 1988) and (Buschmann et al., 1996)), one way of determining activities is to hook into the control flow of the controller. Whenever the controller receives a startUp message (i.e. when it starts its work), an activity is created. The activity is then filled with artifacts that are accessed while the user uses the controller. When the controller's control-flow terminates it fills the end time of the activity.

Command: One can interpret the activities as Commands according to the Command design pattern (Gamma et al., 1995). The main difference is that commands should be able to execute (and undo) themselves, which exceeds the simple logging purpose of the activities. Commands are therefore more tightly bound to the application. In contrast to activities, they are potentially active and not just descriptive. If the application uses Commands, one can reference commands as activities and store them in the log.

Change Log: A change log (Anderson, 2000) stores different states of an object or an object's attribute together with additional information on the originator of the change. It is comparable to Elephant's Brain because it also stores old states of the artifact and thus provides the information, which is needed to reconstruct past activities. But the focus of both patterns is different: A change log is mainly solving the problem of restoring or accessing old states of an object whereas the Elephant's Brain focuses on logging all activities (not necessarily modifying accesses).

Edition: The edition pattern (Anderson, 2000) shows how the change of an object's state can be associated with the event that caused the change. It directly binds the activity to the object, which was affected by the activity. For cases where the shared objects can be manipulated this is an alternative approach to store the activities. The decision, whether to store the activities (on any shared artifact) in a repository (Elephant's Brain) or directly with the artifact (Edition), de-
pends on the access patterns for the information. If activities are mainly accessed by time or user they are easier to find in a repository. If they are accessed for a specific artifact they can also be stored directly with the artifact. For the collaborative setting of Elephant’s Brain, one should in any case ensure that not only modifying accesses are stored (as it is the case in the original Edition pattern).

**Shared Repository:** The log should be modeled as a shared repository (Mularz, 1995) to keep it open for future additional applications as well as future additional awareness views.
5.2.8 Daily Report

Alternative name(s): Daily Summary, Change Report, Newsletter

Intent Inform the user about changes on relevant artifacts in a user defined frequency (e.g., once a day).

Context Users collaborate asynchronously by modifying shared objects.

Problem Changes in indirect collaboration are only visible by inspecting the changed artifact. Users want to react to other users’ actions but they cannot predict when these actions take place.

Scenario German banks have to inform their customers on transactions on their bank account at least once a month. The reason for that is that customers need to be able to object to transactions that they did not approve. But since the bank can not undo transactions that are too long ago, they have limited the period in which a customer can object to 30 days. Normally, customers use the account statement printer to get the information on transactions. But if the customer did not use the system for a long period, the bank sends an account statement by postal mail.

Symptoms You should consider to apply the pattern when . . .

- Users rely on each others’ activities but cannot predict when the activity will take place.
- Users frequently scan for changes but rarely find changes.
- Collaboration takes longer than needed since users do not scan for changes as frequently as they appear.
- The community performs many modifications each day so that direct notifications for each change would consume too much of the user’s attention or would be too expensive.
5.2. A PATTERN LANGUAGE

Solution Therefore: Inform the user periodically about changes that took place between the last notification and the time of the current report.

Collaborations The user defines an interest profile manually or automatically based on his access rights in the collaborative system. He also defines a notification interval and a communication channel by which he would like to be notified.

After the interest interval has passed (in most cases at night), the system checks if artifacts matching the interest profile have been modified within the last time interval. If this is the case, the system puts meta-information on the change into a daily report. The daily report with information on all matching modified objects is sent to the user using the provided communication channel.

Meta information can, e.g., contain a short description of the artifact, information regarding the person who modified the artifact, and the time and type of the modification. It should include a quick reference to the changed artifact to ease the process of accessing it.

The check for changed artifacts can take place in two alternative ways: (1) the system can query the time stamps of all objects and search for those time stamps that fall in the notification interval. This has the advantage that the artifacts only have to carry a time stamp and no additional data structures are needed to track changes. Information regarding the performer of the change needs to be stored with the artifact if such information is desired in the daily report. Alternative (2) is to track all changes in an Elephant’s Brain and query the Elephant’s Brain for activities that took place in the notification interval. Since the activities carry all required meta information (performer, time stamp, and type of the activity), this information does not need to be part of the artifact. However, the number of recorded activities may soon grow and slow the system down.

Rationale Users get informed on changes. This allows them to react to other users’ actions. The fixed interval of change notifications ensures that other users can predict when the local user will read the changes. Compared to immediate change notifications the interval reduces the number of interruptions. Since users are able to tailor their report it will only contain relevant information. For that reason, the user will probably read the report.

Check Before applying this pattern, you should answer these questions:

– How frequently do changes occur and how long do users accept to wait until first reactions?
– How can you represent artifacts in the report medium (e.g. in
a mail)? Can you provide URLs that let the user access the specific artifact by just one click?

– What is the best time to send your report? Is there a period of low system use in which the report can be sent?

– Should users have to register for the report or do you send it automatically (consider SPAM concerns)?

– How can users turn the report off?

– How do you support your users in specifying their interests?

– Is the report personalized (because of different interest profiles) or will all users share the same report?

**Danger Spots**

The report can be considered as spam. Make sure that the users know how to tailor the report to their needs.

In most cases it is advisable to avoid empty reports. However, users could think that the report got lost if they don’t receive any report.

Make sure that the report is structured in a way that can be easily grasped. Provide enough information on the artifacts to allow a user to filter irrelevant changes without looking at the specific artifact in the system.

**Known Uses**

**BSCW** (Bentley et al., 1997a) sends reports by e-Mail after each day on that the content of a workspace to which the user has access changed or users were active in the workspace (reading).

The user can define which types of events (e.g., move events, read events, or create events) should appear in his daily report and which events should be sent immediately (by individual e-mails).

**e-Commerce Web Sites** often allow users to store their interests and their e-Mail address on the server. Whenever a new item is added to the system or an old item is changed, the system sends a notification mail informing the potential customer that there might be a an opportunity for a customer-vendor interaction (e.g., buying a house as shown in the report extract in figure 5.19).

**Mailing lists** like the Yahoo Groups \(^1\) often provide options for controlling the frequency of messages sent to the user. The user can decide if he wants to receive individual messages or daily reports. The reports can contain all message contents or just the headers with links to the individual messages in the web-based mail folder.

\(^1\)http://groups.yahoo.com
5.2. A PATTERN LANGUAGE

Figure 5.18: A daily report in the BSCW Shared Workspace System.

**Related Patterns**

**Attention Screen** → 195: An attention screen filters notifications and contact requests in order to ensure a user’s privacy. It can be combined with the Daily Report to ensure that the user stays informed on the activities in the collaborative environment. It can also enhance the acceptance of the Daily Report since it allows the users to define which information should reach them via the daily report.

**Change Warning** → 228: The change warning provides information on changed artifacts in the same context as the artifact itself. The notification that the artifact has changed is attached to the artifact. In contrast, the Daily Report externalizes this information and transmits it to the user’s work context outside the system (e.g., the user’s mail box).

**Elephant’s Brain** → 150: The Elephant’s Brain keeps track of all activities in the system. The Daily Report can be generated from the Elephant’s Brain by querying it for activities that took place on relevant artifacts in the period of the last report interval.
Figure 5.19: A daily report at an e-commerce site showing new offers.
5.2.9 Hello Hello

*Alternative name(s): Welcome Area*

You say yes, I say no. You say stop and I say go go go, oh no. You say goodbye and I say hello. Hello hello I don’t know why you say goodbye, I say hello.

John Lennon, Paul McCartney

**Intent**
List new members of a group or a community at a prominent place and introduce them to other members.

**Context**
You have established a group and initial users have created social bonds. By this way the group has found its identity. The group members distinguish themselves from people who are not part of the group.

**Problem**
If the group wants to progress it is often needed that they integrate new members. But since the group members are very focused on their internal interaction they may fail to notice new potential members and ignore their possible contribution.

**Scenario**
Consider a scientific conference in the area of computer science. More than hundred researchers gather for several days to present their research results and exchange ideas. Besides the exchange of new ideas, the main objective of the conference is to maintain existing and extend new research contacts.

But will all participants be able to exchange ideas? More experienced visitors are eager to maintain existing contacts and get the latest news from colleagues they met in the years before. This takes time, and time is limited. It is often only by accident that returning visitors get in contact with researchers who participate at
the conference for the first time. And – since the community does not know the newcomers’ faces – it is very likely that the ideas of the newcomers will only have limited space for discussion and exchange.

In contrast, one can often observe that newcomers and returning participants form two distinct groups – with distinct parties in the evening. This makes the exchange of ideas and learning inside the community difficult.

**Symptoms**

*You should consider to apply the pattern when...*

- long time community members have established tight links and are eager to communicate within the community.
- long time members share a large collective history, in which new members did not play any role.
- the community is large enough to allow the formation of sub-groups so that long time community members form a closed group.
- new members find it hard to enter the community.
- new members are ignored by long time members.
- fresh ideas that bring the community forward are provided by new members but ignored by existing members.

**Solution**

*Therefore: Provide a prominent place in the community’s interaction space where new members and their ideas are introduced. In a computer mediated group, this can for instance be a special section on the group’s home page. Whenever a new member joins the community ensure that the existing group members notice the new member.*

**Collaborations**

*Welcome Area:* The welcome area is a prominent region in the community’s interaction space. It can be a special spatial area in a physically co-located community, a special page in a virtual web-based community, or a special topic or thread on a discussion board. While the first two examples are bound to a spatial dimension, the last example is bound to a temporal dimension. It denotes a specific time span where the community members use their interaction space as a welcome area.

*Newcomer:* The newcomer joins the community. He introduces himself in the welcome area and explains why he wants to be a part of the community.

*Veteran:* A veteran shares a long interaction history within the community. He is an accepted member and keeps many social links to other people. The veteran commits to visit the welcome area frequently. In cases, where the welcome area is a
5.2. \textit{A P A TTERN LANGUAGE}

specific point in time, the veteran commits to participate in
the community at this point of time.

\textbf{Rationale} Since the newcomer is asked to introduce himself, he is provided
with a forum where he can articulate his ideas and thoughts that
drove him to join the community. Since all newcomers act this
way the newcomer does not have to fear that his introduction could
disturb existing members.

Due to the commitment of the veterans, they will notice when
the newcomer introduces himself. Thus, the newcomer will be rec-
ognized by the existing community members.

The benefit of using a designated welcome area is that the in-
troductions do not interfere with other group interaction. Veterans
can decide consciously when to visit the welcome area (in cases
where it is a special place) or the whole group will focus on the
welcome area at the same time (in cases where the welcome area
is a specific time frame). This ensures that there is no overlap be-
tween the introduction of new members and the interaction within
the group. Whoever is present in the welcome area concentrates
on the welcoming phase. This way, veterans and newcomers con-
struct a common experience that can serve as the basis for creating
a collective history.

\textbf{Check} \textit{Before applying this pattern, you should answer these questions:}

- How can you technically detect newcomers? Is there a regis-
tration procedure?
- How long should newcomers be part of the group before they
  are asked to introduce themselves in the welcome area?
- Does it make sense that users decide on their own when they
  will introduce themselves (if yes, how do they decide)?
- Does it make sense to introduce users who have not yet pro-
  vided personal information?
- Can you connect the welcome area with other interaction areas
  that are of high relevance for the veterans (e.g., by interleaving
  the welcome area and an alumni event)?

\textbf{Danger Spots} Newcomers may not wish to attract much interest at the first time
they join the group. They may need some time to look passively at
the group and see how the group members interact. In this case,
you should move the welcoming ceremony to the point of time when
the member actively decides to participate in the group. Until then,
newcomers can act with a \textsc{Masquerade}\textsubscript{196}.

Another problem can be to convince veterans to commit them-
selves to pay attention to the welcome area. Veterans may just
not see the need for investing efforts for newcomers. This is an indication that the community is resistant to growth. Considering participation in the welcome area as part of the metric that calculates the users’ ranking in the HALL OF FAME can help to solve this issue.

**ChiPlace** (Girgensohn and Lee, 2002) was used at CHI2002 to foster networking between attendants of the conference before the conference took place. Users could provide a description of themselves. When entering the people page (cf. figure 5.20, left), a list with new members was shown in a prominent section (at the beginning of the page). Clicking on the user’s name opened a user profile (cf. figure 5.20 right).

![Figure 5.20: New members at CHIplace.](image)

**www.visualbuilder.com** is a community of software developers. New members are listed in a designated welcome area on the entry page (shown in the left part of figure 5.21). Clicking on the user’s login opens another page that shows the user’s profile (right part of figure 5.21).

**www.communities.com** (cf. also the known uses section of User Gallery) allows to query new members can be
queried by clicking on the “New Members” link (in the right part of figure 5.22). This will show the list of users that is shown in figure 5.22. By clicking on the user’s login, one can see the provided information.

**Games at EuroPLoP** are special slots in the conference schedule where the participants meet for cooperative games. The games have the purpose of getting in touch with each other. Name games help to learn the participants’ names and other games help to find similarities between the participants.

**Mentor**→**236**: Newcomers, who introduced themselves in the welcome area, can be paired with a mentor who accompanies the newcomer in his first steps and personally introduces him to relevant veterans.

**Masquerade**→**196**: To allow newcomers to interact within the community without exposing themselves, they can move through the community anonymously. The MASquerade pattern discusses this form of interaction in depth.

**Hall of Fame**→**232**: The **Hello Hello** pattern focuses on the introduction of newcomers to veterans. The opposite is done in the **Hall of Fame**: here, honorable community members are presented to the community to provide newcomers with an orientation, who influenced the community most.
Big Jolt (Manns and Rising, 2004) proposes to invite a knowledgable expert to the community who will tell about his experience in the field. Inviting the expert to the welcome area can serve two means: First, the welcome area receives a higher reputation since it is not only a place to meet newcomers but also to get in touch with experts. Second, newcomers will learn more about the ideas that the community propagates. If the expert is a community member himself, he will probably be one of the top members in the Hall of Fame.  

User Gallery: The welcome area is often a special section of the User Gallery.

Introduction Session (Fricke and Völter, 2000) proposes a welcoming phase at the beginning of a seminar. It shows how the problem of learning more about newcomers (in this case all seminar participants are considered as newcomers) can be resolved in a co-located seminar. The solution is to “take time at the beginning of the seminar to let everyone introduce him-/herself to the others.”
5.3 Summary

In this chapter we discussed parts of a pattern language for supporting group formation by means of enhanced awareness mechanisms. The following factors influence the context for group formation: the focus of the users’ current activities (artifacts), the type of interaction (ranging from single-user hci to tightly coupled group interaction), and the distance in time ranging from past activities over synchronous interaction up to planned activities.

The presented patterns ease the transition in the direction of tightly coupled synchronous interaction with a shared focus. They rely on observing the users’ activities and providing activity awareness on related activities.

The patterns are examples for patterns at different abstraction levels. Technical patterns like the Gaze Over the Shoulder or the Elephant’s Brain address issues of data management and interaction tracking. At the other end of the spectrum social patterns like the Hello Hello pattern describe a process of group interaction. It can be supported by technology but does not need the technology to work.

The next chapter will report on the experiences made with groupware patterns in the context of the Oregon Software Development process.

\[^2\text{Note again that additional thumbnails can be found in the appendix.}\]
Chapter 6

Experiences

The goals of the validation are twofold: First, we have to see to what extent a team of developers and users is able to follow the OSDP and create software that matches the users’ needs. Second, we have to discuss the validity of the groupware patterns since they are the most important tool in the OSDP.

After these two fundamental issues have been studied, we need further evidence whether or not the OSDP really satisfies the requirements outlined in section 2. Finally, we need to check if groupware created with a process that satisfies the requirements really is accepted by the users and eases their interaction.

6.1 Observation Method

Validation of software processes in general is problematic since one cannot perform a development task twice with the same preconditions. And even if this would be possible, the preconditions would be hard to define. Traditional controlled lab experiments, as one could use, e.g., for studying the usability of a software system are therefore very inappropriate for the evaluation of software development processes. They would place the observed participants in an artificial environment, which assumes that one can define all external forces of the development process. But since, by definition, groupware development is a wicked problem, it is especially not possible to define all external forces of groupware development beforehand. Another argument against lab studies is that realistic software development projects are often long-term projects lasting more than one year. Controlling the lab setting for such a long time frame is in most cases not possible.

A different evaluation method is thus needed to better understand the effects of the OSDP and the value of the pattern language. McGrath (1984) has classified strategies for group studies along different dimensions (fig. 6.1):

– The unsophistication of the setting (inner circle of fig. 6.1). Observations can be
  
  ▶ embedded in natural systems,
Figure 6.1: Research strategies for group studies (McGrath, 1984) and their applicability for assessing the OSDP (darker areas better fit the needs of the OSDP evaluation).

- performed in contrived and created settings,
- independent of a concrete setting, or
- unimportant since the evaluation takes place on a theoretic level.

- The observed behavior can be universal and thus applicable to many contexts or it can be particular, which makes it hard to generalize (horizontal axis of fig. 6.1).

- The method can be obtrusive or unobtrusive (vertical axis of fig. 6.1). If the research method is obtrusive, the subject will be aware of the fact that it takes part in a research study. This, however, can affect the subject’s behavior.

McGrath pointed out that none of the research strategies leads to a perfect result that would support three goals shown in fig. 6.1:

1. generalizable over populations of actors,

2. precise in the sense that exactly the variables in which the researcher is interested are studied, and

3. realistic with respect to the context.
6.1. OBSERVATION METHOD

As shown in fig. 6.1, the optimal configurations for each of the three goals forms a triangle. Perfecting one goal thus weakens the other goals. For that reason, McGrath proposed that an evaluation should combine different strategies in order to paint a valid picture of the evaluated process.

In the context of the OSDP evaluation, this means that we have to select different evaluation strategies. The results will provide partial but complementary evidence.

To reach maximum precision, a comparative laboratory experiment would be the preferred evaluation method. However, such a study would require that we know exactly what the previous knowledge of each team member is. We would further need to compose two teams with exactly the same previous knowledge and ensure that one team follows a traditional approach and another team follows the new approach.

Both teams would also need the same explicit and implicit goals. To be able to compare the teams, we would need two teams with the same attitudes and wishes regarding a design problem.

These factors make comparative studies rather difficult and, above all, artificial. The realism and context would be insufficient. So, we decided not to include contrived and created settings in the evaluation.

The maximum realism in context can be reached with field studies. Using this method means that the observer lives with the team for a longer period of time. The group process is not manipulated by the observer. Occasionally, the observer can ask questions to better understand the observed behavior. For the OSDP, this would mean that the behavior of a group is observed that applies the OSDP. The project context and the project goals need to be natural settings and the manipulation of the groups should be limited.

The main advantage of this approach is that also external factors that were not considered in an experimental layout can be observed during the study. We therefore decided to apply the OSDP in the field and to perform a field study in which we used the OSDP for the development in a large software project (i.e., the CURE project). We will present our observations in sections 6.2–6.5.

A problem with the CURE study was the expertise of the members of the development team. All of them were experienced software developers. Thus, the learning aspect (R10 and R11) for members of the development team could not be validated in the CURE project. We thus complemented the results of the CURE study with a field experiment. Again, the evaluation took place in a naturalistic setting, a development project. The difference to the CURE project was that the project goal was more clearly defined, that the project duration was shorter, and that the developers were less experienced. We observed the cases of three student theses. In each case, a single student was asked to create a collaborative application using the patterns as educative means. The observer played the role of the customer who defined the overall goals of the application. The student had to perform all design activities (the customer did not participate in the design). The selection of patterns and the transformation of patterns to the concrete application context was monitored by the observer. The observations will be presented in section 6.6.
The field study of the CURE project together with the field experiments of the student projects cover the evaluation made in a natural setting. To achieve generality, these settings are combined with findings from theoretical considerations and expert assessments (falling in the category of expert studies).

The theoretical considerations are based on Alexander’s theory of wholeness. The question is whether or not a sequence as it occurs in the field study leads to wholeness. Alexander proposed 15 fundamental properties that describe systems with a high level of wholeness. A specific sequence is described in section 6.4.3 with a special focus on the wholeness created by the sequence.

The expert assessment focuses on how the written descriptions of the OSDP and the patterns have been reviewed by peer experts. A short summary of this aspect will be given in section 6.7.\(^1\)

A summary of the findings of these four ways of gathering experiences will conclude this chapter.

### 6.2 Context of the CURE Project

The CURE project was a two year project with the goal of creating a collaborative learning environment for the FernUniversität in Hagen (Haake et al., 2004a).

The FernUniversität in Hagen is the German distance learning university. Teaching at the FernUniversität includes different forms of learning: courses, seminars, and different forms of practical problem solving in lab courses. Course material and accompanying individual exercises are sent to distributed students via surface mail or the Internet. Course-specific newsgroups and direct e-mail communication with professors and teaching assistants support asynchronous discussions. Students might also use the telephone to contact staff directly. In addition, students can meet tutors and other students at one of the 60 learning centers, if one exist near their location.

Since 1996, the FernUniversität is active in developing new technical infrastructures for supporting distance education. The *Platform 2000* was an integration project with the goal of combining different systems that were in use for the support of learning management at the FernUniversität (Feldmann-Pempe et al., 1999; Krämer, 2000; Feldmann and Schlageter, 2001).

The emergence and ubiquity of computer mediated communication and interaction has, however, changed the expectations of the students at the FernUniversität. Regarding interactivity, these expectations exceed traditional course or content management that was offered by the *Platform 2000*. Although the traditional ways of distributing printed material to the students are still very important for the students, they demand tighter interaction with and better support from teaching staff. Especially since the studies are no longer free of charge, the students concentrate

---

\(^1\)Borchers (2001) used a comparable combination of expert assessment and the application of patterns in student projects in order to validate his pattern collection for human-computer-interaction design.
on receiving qualified help and finishing their studies as soon as possible. Teaching staff on the other hand has the goal of establishing learning communities that foster mutual help between students, improve the learning results, and relieve teaching staff in times of short financial budgets.

In 2002, the FernUniversität initiated a project group with the goal of examining opportunities for collaborative learning.

6.3 Conceptual Iterations in CURE

From summer 2002 on, a group of teachers from different faculties, members of the university’s data processing service center, student representatives, and researchers from the chair for distributed systems gathered and exchanged different scenarios for collaborative learning. In an initial session, all group members reported on their experience with collaborative episodes in higher education. The group collected ten different scenarios. All scenarios were discussed with a broad user audience in the university’s discussion board. Based on the students’ feedback, four of these scenarios were rated as most important. They were collaborative exercises, tutor-guided groups with collaborative exercises, virtual seminars, and virtual labs.

A scenario interest group was formed for each scenario with the goal of refining the scenarios and creating first sketches for computer support in order to provide a common ground for discussing a potential development of a collaborative learning environment. The scenario interest groups had four to seven members.

We concentrate on the SIG that considered the area of virtual seminars. It had in total five members from the department of psychology (with the goal of supporting virtual seminars), the distributed systems department (a member of the future development team who had the goal of transferring the scenario into development iterations), and students who contributed their experiences with seminars.

As a first step, the group members met in a co-located session and discussed forces of seminars (the goals of the participants) and created a scenario of traditional seminars with talks, each presented by two students. The following text is a fragment of the first story. It considers the phase of a traditional seminar in which all seminar participants participate at the same time (presentation phase).

The students Marks and Alice prepared a seminar lecture on the topic of cooperative software development. 15 minutes before the scheduled start of the seminar, they enter the seminar room. They connect their portable computer to the provided LCD projector and check if the slides are readable. More or less on time, the other seminar participants enter the seminar room. Speakers and the audience notice one another and greet each others. The speakers are glad to see that two external visitors came to join their talk. This encourages them to give an even better presentation. The visitors were pleased that they were given the opportunity to join the talk of which they heard in the local newspaper.

Due to a traffic jam, the teachers arrived 15 minutes late. The speakers waited for them and all participants chatted on arbitrary themes. The seminar leader provides an introduction and welcomes the speakers. Then he hands over to the speakers. The speakers present their prepared talk. They make use of modern media like video
and audio clips to make their presentation a bit more thrilling. Karl, a member of the audience, raises his hand since he would like to ask an intermediate question. However, Marc and Alice ask him to wait until they finished their prepared talk. Karl is annoyed but waits.

After the presentation, Mark and Alice ask for questions. Karl, the teacher, and three other students have an interesting discussion. The other students do not take part in the discussion. Mark and Alice think that the lecture was a success and are motivated to prepare an essay that summarizes their talk.2

The development team member then introduced the other SIG participants to patterns for groupware development. The introduction was kept on a very informal level. The main focus was on the known uses and the scenarios of the patterns. The development team member took the role of moderating the process according to the needed steps of a conceptual iteration.

Together, the SIG members discussed the phases of the story, converted it to more formal scenarios that take the transition to a virtual context into account. Each step of the scenarios was attributed with patterns that could support the step. The SIG members also discussed opportunities for a different dramaturgy of the scenarios. The story resulted in five scenarios on different levels of abstraction. The first three scenarios are shown below:3

**Scenario: Run a Virtual Seminar**

**Main actor:** Students

**Stakeholders:** Teachers, Students

1. Students register to the seminar.

   Relevant patterns: Virtual ME→197, Hello Hello→163.

2. Students introduce themselves to the other participants of the seminar.

   Relevant patterns: Birds of a Feather→199, Opposites Attract→236, Shared Folder→182.

3. Students select a desired topic and create groups. The seminar support system offers recommendations for potential co-learners. The system provides the group with a shared workspace to store their documents.

   Relevant patterns: Centralized Objects→228, Shared Folder→182.

4. The group defines and publishes their goals and milestones for participating in the seminar.

   Relevant patterns: Centralized Objects→228, Shared Folder→182.

5. The group prepares their seminar talk and discusses overlapping aspects with other seminar groups. The teachers control the compliance with the milestones.

   Relevant Pattern: From Shared Data to Shared Work→120.

6. Groups give their talk in a synchronous presentation phase (cf. detailed use case).

7. The groups submit their final essays to the teachers. The teachers read and comment the essays and finally provide a clearance for publication.


---

2 Note that the scenarios and all other examples were translated from German to English.

3 A previous German version (without pattern attachments) can be found at [http://www.fernuni-hagen.de/LVU/intern/gremien_niederschriften/CSCL_Anwendungsfall_virt_Seminar.pdf](http://www.fernuni-hagen.de/LVU/intern/gremien_niederschriften/CSCL_Anwendungsfall_virt_Seminar.pdf).
8. The groups publish their essays and inform other group members about the submission.
   Relevant patterns: Shared Folder→182, Change Warning→228.

9. During the wrap-up phase, the students discuss and annotate the essays.

10. The teachers ask the students for feedback regarding the seminar.
    Relevant pattern: Letter of Recommendation→233.

Scenario: Synchronous Presentation Phase in a Virtual Seminar
Main actor: Speakers
Stakeholders: Students (including Speakers), Guests, Teachers

1. Users (speakers, students, guests, and teachers) log on to the presentation system. The presentation system provides information about other connected users and their roles.
   Relevant patterns: Login→181, User List→141, Role→240, Virtual ME→197.

2. The speakers transfer their prepared material to a workspace of the presentation system.
   Relevant pattern: Shared Folder→182.

3. The teacher grabs the right to speak (his role allows him to do so) in the integrated communication channel (chat).
   Relevant pattern: Floor Control→231.

4. The teacher transmits his greetings using the chat.

5. The teacher transfers the right to speak to the speakers.
   Relevant pattern: Floor Control→231.

6. The speakers present their talk (cf. the additional scenario below).

7. The speakers release the right to talk (resp. type in the chat) and open the discussion.
   Relevant pattern: Floor Control→231.

8. Students ask questions. The speakers or other students provide answers to the questions.
   Relevant pattern: Moderator→236.

9. The users disconnect from the presentation system.
   Relevant patterns: Login→181 (Logout), Lights off→234.

Scenario: Speakers Present Their Talk
Main actor: Speakers
Stakeholders: Listeners, Speakers

1. The speaker receives the right to speak.
   Relevant pattern: Floor Control→231.

2. The speaker moves to an illustrative page from his prepared material. The presentation system ensures that this page is shown on all listeners’ screens.
   Relevant pattern: Travel Together→243.

3. The speaker provides a comment on the shown material using the chat channel.

4. If desired, the speaker transfers his right to speak and present to his co-speaker.
   Relevant pattern: Floor Control→231.

5. Until the end of the talk, the speakers repeat steps 2 to 4.
6. The speakers release their right to talk to the group.
Relevant pattern: Floor Control → 231.

The other three SIGs worked in a comparable way (with comparable results). The results were discussed in a broader group. Teachers throughout the university and students could comment the scenarios using discussion boards.

As a result, the FernUniversität established a research project aiming at testing initially five different forms of collaborative learning in teaching practice. In addition to the four scenarios identified during the conceptual iteration, university members demanded an additional scenario, the collaborative exam preparation.

Instead of providing a different learning environment to each scenario (and to make adaptations of these for different usages), it was decided that a single collaborative learning platform, which would support all five scenarios, and which would be tailored in tailoring iterations by the teachers and students to the needs of their actual course, would be beneficial.

The initial metaphor for the system was that of a shared document workspace system. This was close to the metaphor the users knew from BSCW (Appelt and Mambrey, 1999). The users thought about supporting the scenarios with BSCW but felt that the required level of flexibility could not be reached with this system. This was the main reason why the development of a new solution was preferred.

6.3.1 Results from the Conceptual Iterations

The project followed the conceptual iteration of OSDP to a large extent. The only difference was in the acquisition of the patterns. The patterns were passed on in a narrative way. While this emphasizes the need of illustrative parts of the patterns, it differs from the original idea that the project group takes the pattern catalogue as a written guideline. However, this may be different in other projects, especially, when the user community is not co-located in the conceptual iterations.

Most of the requirements that were intended to be addressed in the conceptual iteration were met. Note that R2 (Structure-Preserving Transformations), R13 (End-User Development), R14 (Sharing of Best Practices), and R15 (Quality Review) are not addressed in conceptual iterations of the OSDP.

Users addressed different forces in their stories and found a way to integrate the stories in a coherent approach. The result, an integrated learning environment with a special focus on tailiorability, has a high level of wholeness since it integrates the different needs of a larger user population (R1).

The process of narrative unfolding of user stories fostered reflection (R3). The users told their story in the group, while other group members questioned parts that were not clear or illogical.

The users focussed on important scenarios right from the beginning of the conceptual iterations. The ranking performed in the user group helped to focus on the most urgent scenarios for the university. Communicating these rankings to other users (via the discussion boards) further improved the process awareness (R4).
During the early phases of the project, users performed several conceptual iterations (R5). The products of the SIGs (scenarios and mock-ups) were refined and discussed in the larger user group. The discussions helped to gradually meet the users’ needs.

Each SIG focussed on one scenario at a time. Scenarios however comprised several problems. Thus, the focus was not yet on one problem at a time. The only point in time, where the SIG members reached this intensive focus was while attributing the different steps of each scenario with pattern candidates. In this phase, the SIG members successfully focussed on one problem at a time (R6).

The understanding of forces (R7) was not as exhaustive as intended. Users tried to describe the field of forces of their situations. But since the narrative was not directly situated in the task of the seminar, the forces were not as easy accessible as if it would be the case for an analysis that takes place during the seminar. Consequently, the requirement for making forces explicit R8 and the requirement for better understanding the consequences R11 was not met sufficiently.

The conceptual iterations involved a large user community. Especially the process of discussing the scenarios in public discussion boards led to good end-user participation (R9).

Education of end-users (R10) took place by passing on the ideas of the groupware patterns. The integration of the solution with the user’s daily context (R12) was ensured by basing development activities on stories brought in by the users from their daily teaching background.

6.4 Development Iterations in CURE

The development team consisted of four senior employees and three students who contributed to the development. Except the technical project leader, none of the developers has been involved in the development of community systems before.

The end-users were university teachers, who wanted to use one or more of the above learning scenarios in their teaching, and students with the goal of improving their learning conditions. They had already participated in the first conceptual iterations and stayed in the SIG when their topic entered a development iteration. Some of the SIG members have tried different collaboration technology before (especially BSCW and Newsgroups). Others had only limited experience with using the internet for distributed teaching.

The SIGs met for an initial planning game to further investigate the forces of the scenarios discovered in the conceptual iteration. The result of these SIG meetings were three sorted SIG backlogs. They contained task cards related to the steps in the scenario. Each task card was rated by the development team regarding its efforts and by the users regarding the importance. In early iterations, the content of the task cards was still quite vague. A first set of top-rated task cards e.g. included the task of creating a shared workspace at the system’s server. The shared workspace metaphor, which was chosen by all SIGs in the first iteration, thus became the
system metaphor and was implemented with highest priority.

![Image](image1.jpg)

**Figure 6.2:** The project backlog of early phases.

The technical project leader took the role of the gardener in the early phases of development. He merged the task cards from the different SIGs and created a project backlog for the first iterations (containing approx. 30 cards). The cards were put on a whiteboard that was accessible by all development team members (cf. fig. 6.2). Other SIG members had difficulties accessing the cards since they worked at other locations.

This problem was solved as soon as the CURE environment supported the creation and manipulation of pages. From then on, the planning cards were managed within CURE in a development room and users could access and modify cards. In early phases, users made not much use of this option, but later on, they started to contribute to the planning cards in CURE. Especially, many users followed the activities that were published in the development room.

Figure 6.3 shows an example of the backlog used in March 2004. Each entry in the left list maps to a planning card (shown in the right window). In this example, the team member Britta Landgraf has implemented a card that addresses the possibility of public keys for rooms. After completing her task, she reflected on her activities and wrote relevant insights to the lower part of the card. She also tracked the real effort in order to improve the estimation of task cards.

**6.4.1 A sequence shaping the collaboration environment**

The first card was the Login card. In compliance with the **LOGIN** pattern, it stated the goal that users who had an account should provide information regarding their
6.4. DEVELOPMENT ITERATIONS IN CURE

Figure 6.3: The virtual project backlog in later phases.

Identity before being allowed to access the resources on the server.

LOGIN

**Problem:** You are developing an application that requires non-anonymous interaction.

**Solution:** Provide a Login screen that requests users to identify themselves by entering a login name and a password before they can start to use the application.

**Known Uses:** www.ebay.com, www.amazon.com

The team implemented a servlet that generated the required login form and processed the provided login and password. Note that the SIG rated this task as more important than, e.g., the first step in the first scenario shown above (Students register to the seminar). The different importance received during the planning game revealed that some parts of the scenarios are nice to have, but not crucial for fulfilling the task of a virtual seminar.

In order to be able to present the solution, the next task card had to be implemented as well, which was the document workspace card. It was influenced by the Shared Folder pattern:
CHAPTER 6. EXPERIENCES

Shared Folder

**Problem:** Different collaborating groups share a lot of data. Due to the amount of data from different groups, users cannot find the data they need for collaboration.

**Solution:** Allow groups to organize their shared data in composites that are modeled as Centralized Objects.

**Known Uses:** BSCW (Appelt and Mambrey, 1999), FTP-Server, WebDAV

The resulting system could be used already. Users could log in to a system and browse material stored in a shared folder. First experimental use revealed that the users missed the opportunity to also communicate about the content. This made the SIG members reconsider their system metaphor of a shared document workspace. They examined the pattern space again and found the ROOM pattern:

Room

**Problem:** When distributed users want to collaborate, they have to deal with many technical issues to initiate the collaboration. They have to set up communication channels and make the shared material available to all group members. Results of the shared collaboration often need to be captured for later reference. All these tasks are error-prone and require technical expertise, which users often don’t have.

**Solution:** Model a virtual place for collaboration as a room, which can hold documents and participants. Ensure that users, who are in the same room at the same time, can communicate by means of a communication channel that is automatically established between all users in the room. Make sure that all users can perceive all documents that are in the room and make these documents persistent. That includes that changes on these documents are visible to all members in the room.

**Known Uses:** VITAL (Pfister et al., 1998), Lecture2000 (Schlichter et al., 1998), LambdaMOO (Curtis, 1998), CollegeTown (Guernsey, 1996), TeamWave (Greenberg and Roseman, 2003), DIVA (Sohlenkamp and Chwelos, 1994)

The pattern conceptually helped the users to understand the difference between structuring the virtual learning environment and interacting within the environment. Users who are in the same room can access all pages that are contained in the room. Changes of these pages are visible to all members in the room.

The room metaphor served as the most important metaphor during the following development. The pattern was not implemented with all aspects in the first iteration. First aspects included a set of pages and tools to manipulate the pages.

Note that the conceptual understanding was sufficient for the users at this time in the project. They now knew where their communication mechanisms would be anchored and how communication channels would be managed in the system. The implementation of communication features was postponed to later phases in the project. Four months after the development of the first room, a mailbox was added to allow asynchronous communication. Another three months later, the rooms were equipped with a persistent chat that was always visible to users present in the room.
6.4. DEVELOPMENT ITERATIONS IN CURE

Figure 6.4 shows a typical room in CURE after the communication features were added. It contains documents (in the example, two users discuss the concept of entropy for which a document has been created in the room *Verteilte Systeme*). It provides two room-based communication channels: a mailbox and a chat.

The ROOM pattern opened up a design space that was not considered before. As discussed in the ROOM pattern, rooms could be connected to form a virtual learning environment. A task card to implement the DOOR pattern was therefore added to the project’s backlog (and placed on a high-priority position since the users wanted to be able to connect several rooms).

**DOOR**

*Problem:* In virtual rooms, objects and characters can only interact with peers in the same room what limits their possibilities. Especially, interaction between different groups cannot easily be supported.

*Solution:* Use doors between rooms as a means for navigating between rooms. Model different states of the door to reflect the mutual accessibility of the related rooms.

---

4Note that the screen shots were taken at different points in time of the project life (to reflect the state after each pattern was applied). This, however, implies that some user interface elements evolved over time and will not be consistent throughout all screen shots of the CURE project.
Users could now move between rooms. After connecting to the system, the users started in the entry hall. From that room, there were doors to other rooms where the learning took place (e.g. a room for the distributed systems chair or a room for linear algebra).

By providing a plenary room, sharing and communication in a whole class or organization can be supported. By creating new rooms for sub groups and connecting those to the classes’ or organization’s room, work and collaboration can be flexibly structured.

After the number of rooms increased (after approx. 3 months of system usage), the test users detected that they had problems to find their way through the rooms. They had problems to remember how the rooms were positioned and connected.

Thus, a graphical room-map was added to ease the orientation in the system following the MAP pattern.

**Map**

*Problem:* To orient in space, users have to create a mental model that represents the space and the contained artifacts. This is a very difficult task.

*Solution:* Create a visual representation of the spatial domain model by means of a map.

*Known Uses:* GIA (Group InterAction) (Manhart, 1999), CVW (http://cvw.sourceforge.net/), CollegeTown (Guernsey, 1996)

The map was generated from the underlying room structure. One problem with generated maps was – as mentioned in the pattern – the stability of relative positions. Two simplifications of the room structure were needed to provide this stability:

- Rooms were limited to have only one entry point directed to the hall. This means that the rooms could be visualized with a tree topology. Without this limitation, the resulting map could possibly be no longer planar and thus not drawable as a two-dimensional map with non-crossing links. Other links between rooms were nevertheless possible by placing a hyperlink on any of the source room’s pages. But these connections were not understood by the users as doors but rather as traditional hyperlinks without any spatial meaning.

- Rooms were automatically ordered (alphabetically) and aligned in a clockwise layout. This means that rooms still could change their positions but that relative order was maintained.

Figure 6.5 shows an example room structure for a distributed software development lab. An entry room is used to support interaction within the whole course (called Learning Gadgets in fig. 6.5). Each project group additionally has a group room in which the specific group project is run. In the example, these rooms are called efa, FOW, and GoGoGadgeto, which are the names of the project groups. A
discussion room serves as a discussion space for aspects that are of interest for more than one group. To aid navigation between rooms, CURE offers a room directory and a room map. While the room directory shows the rooms accessible to the user, and their nesting, the room map shows a graphical layout of the room structure.

![Room directory and room map in CURE.](Image)

The short sequence reported so far – Login, Shared Folder, Room, Door, Map – shows how the initial scenario gradually evolved from a document centered approach to a user centered approach in which users interact in collaboration spaces. The users learned – informed by the use of patterns – how to design collaborative environments and how to move away from traditionally experienced interaction in single user systems (the folder metaphor known from single-user operating systems).

Figure 6.6 shows how the different patterns influenced the development of the system. The patterns are arranged as events on horizontal timescales. The letters at the bottom of each event denote the thematic sequence in which the pattern can be seen. We identified five major sequences. The first sequence focussed on defining the space for interaction and was already described above. The second sequence (B) addressed the interaction with the content of the interaction spaces. It co-evolved with the evolution of the first sequence (A). The third sequence (C) focussed on providing information on the other user’s activities (workspace- and group awareness) and the fourth sequence (D) addressed issues on privacy (of work groups) and group formation (getting to know each others and establishing a group environment). Finally, the last sequence (E) focussed on aspects needed for synchronous interaction.

In the following, we focus on the sequences (C) and (D) that are addressing rather socio-technical issues (compared to more technical issues in the sequences (B) and (E)). We expect to see clear evidence of OSDP’s value in these sequences.

### 6.4.2 A sequence addressing group awareness

Group awareness was rated as an important issue after the users made their first experiences of editing in the system. Forces that were rated as problematic were:

- Users felt as if they were using the system alone.
This had the effect that users had problems to establish synchronous communication sessions.

Users spent a long time waiting for their collaboration partner to respond. Frequent polls of the shared documents were time consuming for the users and frustrating. At some point in time users decided not to browse for changes so frequently anymore.

Users tried to get in contact with other users but the other users did not respond. This had the effect that some users thought that their group members had no interest in participating anymore.

New users had a hard time entering the community since there were too many registered users in the system and potential collaboration partners were hard to find.

Most of the patterns of sequence (C) have already been presented in chapter 5.2. We can thus concentrate on how they were applied in the context of CURE.
6.4. DEVELOPMENT ITERATIONS IN CURE

The Elephant’s Brain pattern was different to the patterns mentioned so far since it was demanded by the developers, not by the end-users. Nevertheless, the forces that led the developers to the Elephant’s Brain were brought up by the users.

Users regarded it as important to know who else is currently connected to the system. They hoped to create a feeling of co-presence that could lead from Shared Data to Shared Work. An appropriate means for supporting this sense of co-presence was to provide Local Awareness. But in order to calculate and visualize confocal users, the system needed to track the users’ activities (Gaze Over the Shoulder) and store tracked activities for future analysis in an Elephant’s Brain.

![Figure 6.7: A User List in CURE.](image)

Users considered it important to see those users, who are currently in the same room. Whether or not the users were also on the same page was not so important for the participating users. Thus, although the Elephant’s Brain proposed to record activities on the level of artifacts (pages), the definition of confocal users was done on the level of the room in which the artifacts reside.

The confocal users calculated by the Local Awareness pattern were shown in a User List. In a first iteration, this list was a textual list of user names. In later iterations, the user names were replaced by the user’s picture (cf. Virtual Me). The most recent visual layout of the user list in CURE is shown in figure 6.7.

The above example shows how the pattern language supports the decomposition and the composition of design problems (cf. fig. 6.8). Patterns on a high abstraction scale help the user to understand the overall problem and envision scenarios of system use. Supportive patterns on lower levels of abstraction provide alternatives and guidance for understanding the detailed interaction in the problem context. The user moves down to a level of abstraction that can be used as development task by the developer. The developer will move further down and employ technical patterns (e.g., patterns for structuring databases or patterns like the Composite pattern of the GoF language (Gamma et al., 1995)). For the user, the low-level patterns define
how the bricks for the application are shaped. These bricks will then be combined to reach larger goals. In the example, the solution of the ELEPHANT’S BRAIN provided an infrastructure for filing activities. The activities were detected using the GAZE OVER THE SHOULDER. The interplay between ELEPHANT’S BRAIN and GAZE OVER THE SHOULDER allows the detection of confocal users and these users should be shown in a User List in order to provide users with a sense of co-presence.

Although the initial use cases focussed on both synchronous and asynchronous learning, it soon became clear that most interaction in CURE will take place asynchronously. Synchronous awareness mechanisms like the User List were important but asynchronous awareness was even more important (even though the users did not recognize this right from the beginning of the project). New and modified artifacts need to be easily found.

The first approach to this problem was a “What’s new” page (a combination of Change Warning and External Awareness View). On this page, all artifacts are listed that have been changed since the last login.

The problem with this approach was that it still required users to actively request information on changes. Users had to log in frequently and the login rate decreased after they did not find any changes for several days. Unfortunately, this also led to a situation where users did miss important news.

The solution was the Daily Report pattern. The system periodically checked for news and sent this news to the users by e-mail.

The following example is an excerpt of such a report. It lists new and changed pages in rooms that are accessible by the receiver. For each modified page it provides information on the page’s location and on the user who performed the change. Embedded links to the pages ease the process of having a closer look at the change if the page was relevant to the user.

A problem encountered in larger group rooms led to the application of the Presence Indicator. If too many users were present in the room the list of pictures
Changes in CURE after the last report (you can find an interactive version of this report at http://teamwork.fernuni-hagen.de/room/308?method=showNews)

========
New Pages
========

targetURL for login
Room: Entwicklungsraum
Author: Britta Landgraf
Address: http://teamwork.fernuni-hagen.de/room/353/68431

=============
Changed Pages
===============

Awareness User Interface
Room: Development Room
Author: Britta Landgraf
Address: http://teamwork.fernuni-hagen.de/room/353/7521

Performance Elephant’s Brain
Room: Development Room
Author: Mohamed Bourimi
Address: http://teamwork.fernuni-hagen.de/room/353/34218

Figure 6.9: A Daily Report in CURE.
consumed too much screen space. The solution for this problem was to limit the number of users that are shown in a user list so that the maximum screen space was known in advance. If more users are present in the room, the visualization changes to a presence indicator, a single image for the whole group of users. The image is context sensitive: if the user moves his mouse cursor to the presence indicator, he can read the tool tip that lists the names of all co-present users.

Notably, one problem regarding group awareness became apparent for that no pattern in the pattern language matched: Since the users mainly worked asynchronously, it was problematic to maintain an overview of who was participating in the group. Daily Reports reported on changed artifacts and the users who performed these changes. This information was used to track activities within the report interval. But still these clues were not sufficient to find out if the group had a problem regarding free riders or inactive users.

![Figure 6.10: Life Indicators as they were invented in CURE.](image)

The solution was to provide all users with a virtual flower (fig. 6.10). Whenever a user logged in to CURE, his flower was virtually watered and flourished. Every day when the user stayed away from the system, his flower wilted.

In figure 6.10, the users Ma, Schümmer, and Singh-Pannu were active in CURE within the last 12 hours. Thus, their flower is colorful and upright. The user Landgraf has been active within the last two days. Her flower has lost parts of its color. The flower of the user Lofin went limp (indicating activities within the last week) and the flower of user Meyer shows that he has not been in the system within the last week.

The great success of the flowers (users were talking about their need to connect to the CURE system in order to water their flower) made the developers contribute the idea as a pattern:

**Life Indicator**

**Problem:** If users work mainly asynchronously they can lose track of their own activity level compared to the activity level of other users. Especially, they project their own level of activity on other users. But if these levels don’t match, expectations will differ from reality and users will be frustrated because of long periods of silence from their interaction partners.
**Solution:** Show a *Life Indicator* together with the user’s virtual representation. If a user participated in the system recently, use a picture for the indicator that looks very alive. Use gradually more dead pictures to represent the period of inactivity. Make the picture look like a toy for which the user can take responsibility.

**Known Uses:** *Flowers in CURE*

Another case where no pattern could be found in the pattern language was the problem that users complained that they did not know anymore with which account they were connected to the system. One user proposed to show the login of the connected user on each page. He provided the Amazon™ bookstore as an example.

The development team took this example as guideline and searched other occurrences of the solution. The result was the *Who am I* pattern:

**Who am I**

**Problem:** After users connect to a system the system keeps track of the identity of the connected user (e.g. by means of the *Who’s that Girl* pattern). But the user may forget under which account he connected to the system. Misuse of accounts can be the result.

**Solution:** Show the name of the connected user on each page. Make this name distinct from other user’s names.

**Known Uses:** *Protected Web Sites, Unix Shells, COAST UML Editor*

The strategy to examine existing solutions prior to creating the own design helped in better understanding the problem domain. The process of generalizing known uses to the pattern format thus contributed to the design that finally satisfied the users’ requirement.

At the user interface level the *Who am I* pattern was implemented by adding a thick border around the local user’s image in the *User List*. In figure 6.7 (p. 187), e.g., the right user picture has a thick border indicating that this user is the local user.

The last pattern of the group awareness sequence is interesting since it was first implemented in a tailoring iteration (cf. section 6.5.2). It was the *Hello* pattern. The need emerged from the fact that users should socialize after they registered in the community. While first versions of this pattern have been implemented by tailoring users, the user community asked for a global welcome area in the system.

In a first development implementation, recently registered users were shown up-front in the *User Gallery* of CURE (cf. p. 198). So, whenever a user wanted to browse or search other users, he was directed to the newcomers first (fig. 6.11).

However, showing newly registered users showed to be problematic. These users had only provided their name and the information was not sufficient to get to know each other. The development team modified the welcome area so that only those new users are shown, who had provided a mini home page. This made the welcome area more interesting to browse.
6.4.3 A sequence addressing privacy and group formation

This third sequence extended the last two sequences in order to provide privacy (and group formation support). It is a good example to show how the evolutionary application of the patterns resulted in structure preserving transformations that increase the system's wholeness.

We can differentiate the patterns of this sequence into two subsequences:

1. The sequence addressing privacy concerns and
2. the sequence addressing group formation.

Both sequences are based on the ROOM→182, the DOOR→183, and and the LOGIN→181 pattern. It therefore makes sense to have a closer look at how the two subsequences changed the level of wholeness. If we use the fundamental properties outlined by Alexander we can observe a certain degree of wholeness that was achieved with the initial three patterns:

The CONTRAST and LEVELS OF SCALE property is strengthened as the ROOM distinguishes between users inside a room and outside a room and rooms are perceived as small part of a larger whole. Additionally, the BOUNDARIES property is
enforced in two different ways. On the one hand, privacy for the collaborating users is introduced, and on the other hand access to the shared objects is limited.

Communication facilities like a mailbox for asynchronous communication and a persistent chat for synchronous collaboration strengthened the Strong Center of collaboration and increased the Not-separateness of the users.

By applying the Door\textsubscript{-183} pattern the Boundaries property was weakened. However, this transformation also strengthened the Not-separateness of the users, the Levels of scale and the Gradient as transition between global and local rooms are possible. Additionally, the Strong Center of a place for collaboration is enforced as the doors increase the experience of working in rooms.

The Login\textsubscript{-181} pattern strengthened the Boundaries property as only users with a valid identification could enter the system.

The subsequence addressing privacy

**LOCK AND KEY**

**Problem:** People feel the need to distinguish public from private places and thereby identify private objects that are located in the private place. They may want to restrict access to private places and the contained private objects so that only familiar users can access private places and objects.

**Solution:** Protect private places with locks that can only be opened with matching keys. Provide the users with keys that can be used to open locks. Make these keys first-class objects that are visible to and portable by the user. Place private property inside locked rooms.

**Known Uses:** Moo, SSH, CURE

The Lock and Key pattern was applied to add access control to the rooms. Before a user was allowed to enter a room, the system checked whether or not he holds a virtual key to this room. If the user does not have a key, he could ask the key holders of the room to replicate their key and pass on a copy to the requesting user. By adding this kind of access control the Not-separateness of the collaborating groups was decreased. However, in contrast to this the Contrast and the Boundaries property of CURE was strengthened as the separation of the collaborating groups was increased.

To structure group interaction and to reflect the roles of users in a group the users requested different keys with different rights, such as the rights to enter a room, create sub rooms, edit pages, or to communicate within the room. CURE displays the different rights by assigning different colors to the private keys of the users (see fig. 6.12). Due to the different rights a user can have in a group the Levels of scale, Contrast, and Gradient properties were strengthened. The Levels of scale property increased because it was now possible to model different kinds of interaction between group members. The Gradient property increased as users could create keys with gradually increasing rights. User could control how group members gradually take more responsibilities. In the beginning of a group
interaction a user might, e.g., just be allowed to read the content of a room while
she later might edit it. The CONTRAST property was strengthened by the LOCK
AND KEY pattern as each group in the system could now establish its own way of
interaction.

The LOCK AND KEY pattern together with the ROOM pattern models
the representation of groups in CURE. A group interacts in a room and the group
members are all users who have a key for this room (Haake et al., 2004b). Thus group
formation is supported by means of creating rooms and distributing keys for these
rooms. A user can have keys to different rooms and thus can be a member of different
collaborating groups. Thereby, the LOCK AND KEY pattern finally strengthens the
DEEP INTERLOCK AND AMBIGUITY property, as groups get connected.

As described above, users without a key could ask the owners of a key for a
personal copy. This could probably be modeled without any additional tool support.
Since the user can see other users who have a key with sufficient rights for the room,
they could send these users a mail that explains their request. The key owner would
receive this mail and check if the request makes sense. In a positive case, he would
create a key that matches the user’s needs and inform the requesting user by mail
that his wish was fulfilled.

The problem is that these activities require a complex workflow that has inher-
ently several media breaks. Users thus requested an easier way to ask for a key,
which was modeled using the BELL pattern.

**BELL**

**Problem:** When latecomers want to join a collaborating group, they may intrude
the group’s privacy. The group may feel disturbed by uninvited visitors, or simply not
notice that someone wants to join.

**Solution:** Provide the meeting place with a bell that allows latecomers to raise
attention regarding their will to participate in the group. Model the bell as an interface
element that is associated with the representation of the shared space where interaction
of the group takes place. Signal the activation of the bell to the members of the group
and let them decide whether or not they allow the asking user to join the group. In
both cases, inform the asking user and – if possible – guide the user in the next step of entering the session (or staying out).

**Known Uses:** ISDN, NetMeeting (Summers, 1999), FUB (Haake and Schämm, 2003)

The bell pattern supports **Not-separateness** on several levels: (1) on a technical level there is no longer the need to switch between different tools and media for performing a key request, (2) on a human-human interaction level it structures the way how users interact in a recurring situation (and thus helps to better understand this process), and (3) on a group level it connects users in a group with users outside the group. The bell belongs to both spaces, the space in the room and the space outside the room. It is thus a region with **Deep Interlock and Ambiguity**.

In early 2004, many users had access to different rooms. But not all rooms were equally important for them. For instance, a member of the distributed systems department had access to a room of the psychology department for assisting the psychologists in shaping the room’s content from a technical perspective.

But when the seminar in psychology started, the computer scientists was informed about all changes to content in the psychology room. Since the developer was not interested in this information, it distracted him from his other work. The user could have canceled his membership in the psychology room, but this was not desired by the psychologists.

The **Boundary** of his awareness was too weak so that relevant as well as irrelevant awareness information reached this user.

The CURE developers added a list of accessible rooms from which the user could select those rooms for that he would like to be informed about changes in the daily report.

**Attention Screen**

**Problem:** Every request for attention needs to be processed by the user. Thus, it already takes some of his attention. But in situations, where the user needs to focus his attention on other things, this is disturbing.

**Solution:** Enable the user to filter the information which reaches him. Use meta-information (e.g. sender details) or content information (e.g. important keywords) to distinguish important information from not so relevant information. Collect the less important information at a place where the user can process it on demand and forward relevant information directly to the user.

**Known Uses:** Siemens S55, Mozilla Junk Mail Filter, TeamSpace (Fuchs et al., 2001), WebWasher, Instant Messaging Systems like MSN Messenger, ICQ, AIM, or Jabber

The **Attention Screen** adds another **Gradient** to the **Boundaries** of each user. The properties mutually contribute to one another. With the **Attention Screen**, the user can gradually block information to which he still has active access (the user still can go to the room and look for new pages).
MASQUERADE

**Problem:** User monitoring is required for providing awareness information to remote users or associating work results with a specific user. On the other hand, users often do not act as confident if they know that they are monitored as they would act in an anonymous environment. Especially openness and the courage for taking risks may be much lower.

**Solution:** Let the user control how much interaction information he provides to the system. This means that the user should be able to filter the information which is revealed from his personal information. Remember to consider Reciprocity → 238.

**Known Uses:** NYNEX Portholes (Lee et al., 1997), TUKAN (Schümmer and Haake, 2001), Anonymous access in web-based community systems

The MASQUERADE pattern again contributed to the Boundary of the masked user. By moving through the system with a protective mask, users can explore the system safely without being noticed by anyone else. The MASQUERADE adds an additional Level of Scale for interaction. Without the MASQUERADE, users could either interact as a full member in the system or not interact. The third level of browsing material without being part of the group (and without being noticed) is positioned between the two former levels.

In CURE, the MASQUERADE was implemented by providing an anonymous login account. Room owners could pass on keys to this anonymous account. Users acting with an anonymous account were also able to request a key (and thus demand from the room owner that the specific group opens up for anonymous interaction). The only interaction that was not possible for anonymous users was the delivery of e-Mail messages since this required that the user revealed his identity. As demanded in the pattern, the implementation needs to take Reciprocity → 238 into account (and thus respect the Local Symmetries property). The owners of the place (the users having keys for the room) have to be able to control whether or not anonymous users should be allowed to enter the room.

---

**Figure 6.13:** Attention Screen for Daily Reports in CURE.
The subsequence addressing group formation

The patterns discussed in the previous subsequence focused on providing users with a private and safe place. However, users should also be supported in finding one another in order to initiate collaborative work. The following patterns weaken the privacy to better support group formation.

Since CURE is a learning platform it is often important that the users act in an authentic way. Authenticity is an important issue when building trust. Trust is important if the users take the risk of failure. And failure is an important aspect in learning activities. Building an authentic and trusted relationship between users was therefore considered as an important issue in the design of CURE.

**Virtual Me**

**Problem:** Account names used by users of a collaborative application are often not very expressive. Especially, an account name is only a short textual representation of the user. In a large user community, account names look similar. But users need to communicate their identity in order to interact with other users.

**Solution:** Allow the users to play theater! Provide them with means to create a virtual identity that represents them while they act in the system. Show the virtual identity when the user is active.

**Known Uses:** Textual characters in MUDs, VITAL (Pfister et al., 1998), WEB Homepage

The Virtual Me pattern only partially addressed this need since users should act in a realistic way. Their theater play should not be too far away from reality. However, the other parts of the pattern apply to CURE. Users should be able to present themselves to other users. To foster authenticity the users could provide a photography. They could also present a short text describing themselves.

Up to now, 38% of the user population (515 of 1341 users) made use of either a textual description or a photography. Notably, female users showed a higher willingness to present themselves (41% vs. 37%). The difference is even more significant if we compare the percentage of users who contributed both a picture and a text. Here, 25.7% of all female users provided both kinds of information while only 20.5% of the male users provided both a picture and a text. Male users were more eager to provide only a picture (6.3% male vs. 1.9% female).

The Virtual Me pattern established the users as additional Strong Centers besides the rooms and the edited content. Having a picture and/or a textual description helped to focus on the users.

With this in mind, the SIG members demanded additional means for connecting the users. The asked for an interesting way to find out more about the users of the system.

**User Gallery**

**Problem:** If more than one user interacts with shared data, it is hard to coordinate the interaction - especially with strangers. Without knowing who is using the system, it is hard to establish collaboration or to become aware of other users’ activities.
**Solution:** Provide a list of all users who are members of the community. Let the members provide personal information in this list that is related to the community’s task. Design this list in a way that it is interesting to browse.

**Known Uses:** VITAL (Pfister et al., 1998), ChiPlace (Girgensohn and Lee, 2002), Community web sites like www.communities.com

In CURE, the User Gallery was modeled as a special room, the Student Gallery (fig. 6.14).

![Student Gallery in CURE](image)

**Figure 6.14:** Parts of the Student Gallery in CURE.

Unlike other rooms, the student gallery does not list wiki pages but shows the self descriptions of the registered users. It was important that all users were shown (especially in the early phases of system use where the user community was relatively small). The Student Gallery should bring together students from different faculties and foster exchange between any students of our university. With respect to the properties of wholeness this emphasized **Deep Interlock and Ambiguity** and increased **Not-Separateness**.

Another option for moving the individual user into the focus was the **Interactive User Info** pattern:
INTERACTIVE USER INFO

**Problem:** Users are able to associate the awareness information with the artifact, but it is still difficult to find out how to launch appropriate collaborative tools (e.g. for opening a chat tool with the confocal user – cf. TALK FIRST → 242).

**Solution:** Make the user representation interactive so that another user can reveal more information on the shown user or start tighter collaboration with the user by acting on the user’s representation.

**Known Uses:** IM Buddy Lists

This pattern was implemented by linking all visualizations of user names to their Virtual Me. Static information like the Change Warning → 228 at the bottom of each page thus became active. Content was related to real users. The visualizations of the self descriptions were additionally enhanced by providing mail links to the users. Establishing a contact with the author of an artifact was thus eased.

The Interactive User Info helped to increase Not-separateness and Strong centers.

One problem with the User Gallery is that it does not scale well. If the number of users grows too large, the list of the User Gallery will also grow accordingly. When the system had approx. 300 users, the users had problems to find relevant peers in the Student Gallery. They asked for filtering regarding their interests and selected the Bird of a Feather pattern.

BIRDS OF A FEATHER

**Problem:** If people don’t know one another it is hard to decide who could be a good partner for a collaborative activity. For co-located situations, humans have developed intuitive strategies (based e.g. on visual clues) that help them to select whom they should contact if group formation is needed. In distributed work environments the presence of other users is often reduced to their user name. This makes it hard to find another user for a collaborative activity.

**Solution:** Compare user profiles or interaction histories to identify two users who share big parts of their history. Propose these users as collaboration partners.

**Known Uses:** MEMOIR (Pikruksis et al., 1998), Autonomy CEN (Autonomy, 2002), Yenta (Foner, 1996), BoF-Sessions at Conferences, Collaborative filtering systems like the book recommendations at Amazon or the music recommendations in Ringo (Shardanand and Maes, 1995).

The pattern was implemented based on the rooms to which a user has a key. A user can select that he only wants to see users that share a common key with him (fig. 6.15). This ensures that there is at least one group in which both users participate.

This pattern helps to connect users with comparable interests. It extends the center of each user to an even Stronger Center in the social network of users. It further helps to increase Not-separateness since it brings together users with shared interests and it helps to define Boundaries encapsulating members of the
same interest groups. However, we should note that these boundaries are personalized and therefore allow **Deep Interlock and Ambiguity** of each individual user.

The last pattern of this sequence is the **Visa** pattern:

**Visa**

**Problem:** While the protection of group interaction by explicit logins protects the community it also makes it difficult for non-group members to interact with group members. Especially, if there exist some (e.g., social or business) reasons why those non-group members of the system cannot become permanent group members, interaction is not possible.

**Solution:** Offer the users of the system a possibility to grant visas (temporary user accounts) to non-users of the system.

It bridges the gap between group formation aspects and privacy aspects by allowing temporary access to group resources. In CURE, the Visa pattern was modeled by means of time keys. Group members can decide to pass a time-restricted key to another user. They can define a start date and an end date. Within these dates the key will allow the key holder to access the associated group room.

Such keys were, for instance, used for presenting results of a work group. The members passed keys for the presentation week to other interested users. Before that week, they cleaned up their room in order to have only documents in the room that they could show to other users. The visitors were then able to inspect the room. After the keys became invalid again, the group could continue the internal activities.

The **Visa** pattern helps to increase **Not-separateness**. It also provides **Gradients** since one can provide limited access to specific users. Finally, it helps to keep **Boundaries** intact since the room will be shielded again after the keys became invalid.
6.4. DEVELOPMENT ITERATIONS IN CURE

Table 6.1: Estimated values for wholeness in the considered pattern sequence.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Levels of Scale</th>
<th>Strong Centers</th>
<th>Boundaries</th>
<th>Deep Interlock &amp; Amb.</th>
<th>Contrast</th>
<th>Gradients</th>
<th>Not-Separateness</th>
<th>Local Symmetries</th>
<th>Wholeness Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Room</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Door</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock and Key</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bell</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Attention Screen</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Masquerade</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual Me</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Gallery</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive User Info</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Birds of a Feather</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Visa</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Wholeness</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td></td>
<td>34</td>
</tr>
</tbody>
</table>

Evolving Wholeness

The two subsequences described in the above sections let wholeness emerge. We have shown that each pattern helped to strengthen some of the fundamental properties defined by Alexander. Using Alexander’s measure of wholeness (p. 27), we can provide a rough estimate for the total level of wholeness after each pattern was taken into place (cf. table 6.1).

These estimates help to compare the structure before and after the pattern was applied. As Alexander pointed out, there is no exact measure for wholeness. Instead, we take a simple approach where we consider each distinct functional increase of wholeness as an increment of one. For the example of the Login→1 pattern, we stated that it increases the Boundaries. We count this as an increment of 1. The Room→1 pattern also increased the Boundary property. But in this case the Boundary was enforced on two levels; the artifact level and the interaction level. We therefore place an increment of 2 in table 6.1. In two cases, properties were weakened which is shown as a decrement in table 6.1.

The last column of table 6.1 shows the increment of wholeness added by each pattern. The last line of the table shows the total wholeness after the application of all patterns from the sequence. The evolution of wholeness is shown in figure 6.16. The assumption was that the wholeness measure would increase in a structure preserving transformation. This could be observed in the analyzed sequence.
However, the estimates are based on interpretation of the sequence. They cannot serve as an exact scientific measure. Instead, they are one step in understanding how patterns contribute to good design. It is also not appropriate to ask for a threshold above that a system is complete or whole. Whether or not a system is whole enough still needs to be decided by the end-users.

![Graph showing the evolution of wholeness in the considered pattern sequence](image)

**Figure 6.16:** The evolution of wholeness in the considered pattern sequence (different colors/shadings represent different levels of wholeness in the graph).

### 6.4.4 Results from the Development Iterations

The experiences from the development iterations show that the development team and the end-users were able to perform the iterations as described in the OSDP. One problem was the test coverage. The developers were eager writing code that implements the patterns and often wrote only partial test cases. Especially at the end of the first project period (after 12 months), the developers tried to implement end-users’ task cards in order to provide the most comprehensive system functionality. The problem with this approach was realized in the second half of the project. Especially in the last quarter of the project missing tests showed to be critical. In this part of the project, the persistence layer of the system had to be changed in order to satisfy external requirements for the production phase of the system.
The adaptation of the persistence layer showed to be one of the most problematic changes during the project life cycle. The group responsible for running the system in production mode had the requirement that the data, which was stored in a file system before, should be stored in a database. To change the persistence, all persistent objects had to be mapped to the database system. Performance dropped dramatically since objects were kept in memory before and the developers made heavy use of object associations, which are not as cheap when mapping objects to a relational database.

The problems with the persistence layer could have been avoided if the members of the group responsible for running the system in the production mode were participants in the SIGs. Unfortunately they were not involved in the teaching activities and therefore not very interested in participating in the early phases of the project. Insisting on participation of these users would have been important for the project.

Another problem overseen by both, developers and other SIG members, was the performance of the communication mechanisms. One reason for this is that the development team initially tried to make use of a java servlet framework that is not tuned to synchronous message exchange (with a high message rate).

These two technical problems – persistence and performance of synchronous communication channels – slowed down the team in the last third of the project. User stories could not be delivered as fast as in the first two thirds of the project. This had the effect that users also reduced the number of stories that they brought in. We can thus observe that a fast response to user requirements motivates users to think about possible improvements more frequently. The fast delivery of functional increments is vital for a high end-user participation.

The requirements of chapter 2 were addressed in the following ways:

- Wholeness (R1) was demonstrated in section 6.4.3. The pattern sequences led to wholeness regarding the properties proposed by Alexander. In addition, the existing tests ensured that existing functionality remained intact when adding extensions. However, as discussed above, the tests were not as comprehensive as needed.

- Structure-Preserving Transformations (R2) were achieved by applying sequences of related patterns. The effect of one structure-preserving transformation was demonstrated in fig. 6.16.

- Reflection (R3) took not place to a large extent since it was rather an issue in the tailoring iterations. The only phase where reflection played a role was when the backlog was reordered and the relation between backlog and the existing system was examined.

- Process awareness (R4) was limited in the early phase of the project since the end-users of the SIGs had only limited access to the project’s backlog. However, it increased as soon as the end-users could access the backlog from within CURE.
– Iterative Design (R5) took place since the development team frequently released new versions of the system. One example, where the iterations changed early design decisions, was the shift from a Shared Folder metaphor to the Room metaphor.

– The task cards of the backlog helped developers and users to focus on one problem at a time (R6). Actually, when most task cards were completed, the development team released a new version of the system.

– Forces were examined with the help of relations between patterns (R7 and R8). The patterns often helped the developers to also consider related problems.

– End-user involvement (R9) was most intensive in early development iterations. The users actively contributed to the design and proposed the application of specific patterns. In the case of the Life Indicator→Life Indicator, they even brought in new patterns.

– Education of users and developers (R10) was an important issue throughout the development iterations. The patterns were used as one means for transferring design knowledge. This was complemented with traditional forms of knowledge transfer, especially the exchange between the experienced developer and the users or the peer developers. If no experienced developer would have been in the team, the role of the patterns would have been different.

– The explicit consequences (R11) of the patterns triggered discussions regarding the involved pitfalls of a solution.

– Integration (R12) was addressed by using a combined backlog of the different SIGs. In addition, the CURE system was integrated with the current learning management system of the university.

– End-User development (R13) was not addressed in this type of iterations. The development tasks were all performed by the developers since they required programming. However, the basis for end-user tailoring was laid by the development team (since this was requested as an orthogonal requirement by the users).

– Finally, new pattern ideas were brought in that were shared between the users (R14) and which found their way into the group’s pattern catalogue. The quality of these solutions is still evaluated (R15).

The resulting system was widely accepted by the users. The user community grew constantly from the first voluntary use by students. Students and teaching staff spread the word about the system and other students and teachers started to use the system as interaction platform. After 18 months of regular use, the system has reached a user population of 1393 users (all of them used the system on a voluntary basis).
6.5 Tailoring Iterations in CURE

The role of the tailoring iterations became important in the latter parts of the project. After users used the system for a longer time, they started to reflect on system use and on interaction patterns that recurred in their interaction. The time span between the first login and the first tailoring action varies. Some power users have started to tailor their room after approx. one month of system use while other users never tailored the system.

We could observe tailoring at two levels:

- Users established interaction sequences that supported a group process in a sub-optimal way. The users did not change the underlying technology but rather the pattern of use. These instances of tailoring were the starting point for development iterations. An example that will be discussed below was the implementation of the Hello Hello pattern.

- Users modified both the process and the underlying technology in order to establish a new way of interaction. These tailoring actions were also triggered by from patterns of use. But in contrast to the first type of tailoring actions, the users also thought about modifying the technology using the tailoring facilities offered by the CURE system. Examples are the CURE planning game and the support for the creation of diploma theses.

We will present the three examples of tailoring in the next two subsections.

6.5.1 The Hello Hello pattern

The department of psychology had longer experiences in teaching virtual seminars using “traditional” news-group technology. They had good experiences with a round of introduction: At the beginning of the seminar the users should introduce themselves and explain their motivation for the seminar.

In order to conduct a virtual seminar in CURE the psychologists thought about how they could reach the same goal in CURE. They understood that the basic form of interaction that they intuitively followed in their news-group seminars was that of the Hello Hello pattern.

Therefore, they had to define a special welcome area in which the introduction could take place. The welcome area was defined both by its time (in the early phases of the seminar) and by its location (a special initial page on which the participants should introduce themselves). By providing the users with a location for introducing themselves and with a phase of the seminar, the social process (how to interact in CURE) was adapted to the specific goal (that of the Hello Hello pattern).

After running more than one seminar it became clear that some parts of the description were required for all users. Examples are the user’s name and his picture. The idea was born to have a system-wide representation of these user specific attributes. The Virtual Me pattern fulfilled this goal. The problem was that
the users were not in control of a system wide space in which such descriptions could be stored and managed. This was the reason why the Virtual ME pattern was escalated to a development iteration.

The example shows how tailoring of social processes, modifications of patterns of system use, and the emergence of new requirements went hand in hand in a tailoring iteration. Patterns played an informative role here. Note that the HELLO pattern was later on also introduced on a system wide level (cf. section 6.4.2).

6.5.2 The emergence of CSCL patterns during tailoring iterations

Another example for end-user driven adaptations of system use was the emergence of patterns for collaborative learning. Again, the adaptation of a social process was the first step.

The context was the process of supporting students in their first larger research project, the diploma thesis, which ends their diploma studies. Up to then, the students have often not experienced in depth research processes that involve the definition of a research agenda, the search for and understanding of relevant literature to capture the state of the art, and the adaptation of the state of the art to the specific research problem. Especially the search for and the understanding of the current state of the art is often not experienced to a satisfying degree by the students.

This was the point in time when domain specific patterns came into play: The users needed more advice in solving a pedagogical problem, not a technical problem. Several patterns of the pedagogical patterns movement (Eckstein, 1999) can contribute to the problem of superficial understanding of literature research. An example is the Read, Read, Read pattern of Bergin (2002):

**Read, Read, Read**

**Problem:** To do well in a dissertation you need to go deep, but you need enough breadth to know how your research relates to the rest of the field and to the rest of human knowledge.

**Solution:** Therefore, read everything relevant to your research. Read for both depth and breadth. Take notes as you read. Pay special attention to research methodology as you read. What questions were asked and HOW were they answered.

A first translation of a comparable approach to the domain of computer-mediated learning can be found in (Häfele and Maier-Häfele, 2004, p. 167), where the authors provide a method for collaborative text summarization:  

**Goals:**

---

5Note that Bergin initially proposed the pattern for PhD education. However, the teachers at our university demand the same practice for diploma students.

6Translated from German by the Author.
6.5. TAILORING ITERATIONS IN CURE

- The participants look into the subject.
- The participants provide feedback. This enables the teacher to assess, what has been understood by the participants.
- The participants act as mutual experts.

Group Size: 1

Process: The teacher provides learning material in the WIKI.

The teacher asks the students to read the material and create annotations of the following type:

- 5 annotations with links to related material,
- 5 questions focussing on the provided material,
- 3 issues that the participant accepts (with reasoning),
- 3 issues that the participant cannot accept (with reasoning), as well as
- 3 answers to questions stated by their colleagues.

Compared to the model proposed in the READ, READ, READ pattern, the collaborative text editing introduces a collaborative aspect since it asks the participants to state questions and answer other people’s questions.

With this background, the teacher started to think about literature search in the context of diploma theses. His first idea was to better structure the process of literature search (initially without any relation to pedagogical patterns). He suggested that the student creates a research room in which the student can collect relevant literature. In order to receive support, the student could invite the teacher so that the teacher could monitor the found literature and supplement the literature list.

The users created pages for found literature but still had problems to structure their process. For that reason, a second approach to the problem included a tailored environment for literature collection that explicitly suggests a process. The teacher provided page templates that structure the relevant information needed for found literature, for creating a summary of the paper, and for managing a collection of papers.

In the context of this thesis it is not important to go into details of the template definition. The important aspect is that the users define the structure of the content by providing XML tags that describe the fields that are on a page (cf. appendix C for more details). In the context of the literature review example, the teacher created a template that guides the student through the process of collecting relevant information about the paper. So, in contrast to the use of standard pages that allow free-form input, the student now has to fill specific fields that are relevant for his task. These fields will be visualized in a form that is close to the intended form of the school (regarding citation formats). By that the student also learns how related work should be cited.

Figure 6.17 provides an example of a literature summary page in the edit view (left) and in the display view (right). To embed these new pages in a group process, the teacher did two additional modifications: Firstly, he added a comment field to
the literature summary in order to foster discussion between students and assessment by the teacher.\footnote{This is not visible in fig. 6.17. Refer to appendix C for a full version of the template with a visible comment field.} Students were asked to comment each literature summary. The teachers finally provided suggestions for improvement. Secondly, he created an overview page on which the students should collect their summaries.

After the solution has proven its validity (e.g., after a group of students successfully collected literature with this template), it should be shared with the rest of the community. The pattern scout looked for the solution and discussed its rationale with the author. The combination of technology (the tailored environment) and the pedagogical process (in this case modeled as a group activity) was then captured by means of the \textbf{Literature Pool} pattern:

\textbf{Literature Pool}

\textbf{Problem:} Students need to read research literature in order to relate their ideas to the state of the art. But in their previous studies, students were rarely confronted with research literature. Instead, they received pedagogically enhanced material that clearly stated questions, methods, and results. Researchers, however, are often not as good in communicating their ideas in an educative way. Students have to learn how to read literature.

\textbf{Solution:} Ask the students to fill a literature pool. Let them search, summarize and
6.5. TAILORING ITERATIONS IN CURE

Comment the literature. Provide the teacher with access to the literature pool and let him comment on the student’s summary.

**Known Uses:** Literature research at the distributed systems department (in the context of diploma theses).

Note that the current status of the CURE project is in a phase where first patterns are detected by the pattern scout (the Literature Pool pattern actually was the first pattern that was found this way). We observed that it requires a longer time until the users begin to tailor and even longer experience until the users are confident enough to share their solutions within the whole user community. However, we are optimistic that the growing collection of user-made patterns will motivate other users to share their experience as well.

### 6.5.3 Results from Tailoring Iterations

We observed that tailoring took place according to tailoring iterations. We have also shown sequences in which the tailoring operations triggered new tasks for development iterations. The assumption of the OSDP was confirmed that the iterations contribute to each other.

The opportunity to tailor motivated users to reflect on their activities (R3). Users created new domain-specific patterns (R14) and provided implementations using the tailoring mechanisms of CURE (R13). Some users referred to patterns to support their tailoring, but in most cases the tailoring operations were rather based on intuition. This may be one reason why many users reported that tailoring was still difficult.

Propagating the collection of domain-specific patterns to the users is thus still a challenging task to educate the end-users (R10) and to support better tailoring actions. In cases where patterns were used the forces were also made explicit and discussed between the tailoring users (R7, R8, R11). In other cases, the forces did not play an explicit role.

We could observe that some users reflected on their tailoring actions and that their tailoring practice evolved. An example is the case of the Hello Hello pattern. However, we could not yet find a general trend for iterative improvement of tailoring practice.

Process awareness (R4) was not yet achieved regarding tailoring. Users instead tailored on demand. This is part of the tailoring idea. The tailoring user should fix a problem of use during use. This implies that the problem will not be scheduled or planned.

The other requirements were not addressed since they are not applicable in tailoring iterations.
6.6 Student Projects

To prove the educational quality of the patterns they were used in student thesis projects. Students selected a groupware development task and received a copy of the groupware pattern language. They were then asked to structure their application development by means of pattern sequences. The projects were:

CURE expert finder: The student was asked to enrich the collaborative learning environment CURE with mechanisms for finding potential co-learners. The selections of co-learners should be based on the other learner’s expertise. If a learner has a question the co-learner should be able to answer the question.

The task was performed by a Bachelor of Computer Science student at the Distance University of Hagen. It had a duration of 6 months. The student had experiences in software development in Java and knowledge in Servlet programming that helped him in the practical implementation.

During the development, the student applied the following pattern sequence: Show the Expert \( \rightarrow \) Pay Back \( \rightarrow \) Letter of Recommendation \( \rightarrow \) Birds of a Feather \( \rightarrow \) Gaze Over the Shoulder \( \rightarrow \) Control Your Privacy \( \rightarrow \) Hello Hello \( \rightarrow \) Hall of Fame \( \rightarrow \) Presence Indicator \( \rightarrow \) Who’s Listening

The resulting system is described in (Cullmann, 2004).

Being asked to reflect on his learning process, he stated that the initial pattern (Show the Expert) was obvious for him and that he had the same idea without knowing the pattern. However, he also stated that the other patterns helped him to reflect on the problem area. He provided a mapping of the patterns to his concrete context and finally implemented an expert finder system.

The expert finder system can be considered as a rather large development project (compared to other student projects like the SCOPE example presented on page 212). The student based his work on the CURE project that had at this point of time 383 classes with 64405 lines of code.\(^8\) He modified 17 classes by changing/adding 512 lines of code. Considering only the modified classes, he performed changes at 4.4% of the code. He added a new package with means for earning and spending expert points and visualizing expertise throughout CURE. This package contained 34 classes and 3484 lines of code. In total, the student modified or added 4000 of 67889 lines of code which is a ratio of 5.9%.

We could observe that the changes weaved through the project (changing only small methods of existing classes). The patterns helped the student to understand which changes he had to perform.

The examiner of the project rated the student’s solution as a good design. The modifications reflected the essence of the patterns. The solution was

\(^8\)The metrics presented for the project shall only provide an impression on the project’s size and complexity.
well integrated with CURE. So, we can conclude that the student learned to understand and apply the patterns.

**Collaborative Web research:** The student was asked to integrate mechanisms in CURE that support a group of students in finding research literature on the Web. He was a diploma student of computer science at the Distance University of Hagen. The student was familiar with java technology. He was not familiar with the concept of design patterns.

![Figure 6.18: A student’s mock-up of a system supporting collaborative Web research.](image)

In a set of conceptual iterations the student selected high level and low level patterns that he considered important for his task. The patterns
CHAPTER 6. EXPERIENCES

were: Awareness Proxy $\rightarrow$ Elephant’s Brain $\rightarrow$ Remember to Forget $\rightarrow$ Gaze Over the Shoulder $\rightarrow$ Local Awareness $\rightarrow$ Time Compressor $\rightarrow$ Semantic Distance $\rightarrow$ Semantic Net $\rightarrow$ Active Neighbors $\rightarrow$ Presence Indicator $\rightarrow$ Swarm and Collect $\rightarrow$ Distinct Awareness Information $\rightarrow$ In-Place Awareness View $\rightarrow$ Letter of Recommendation $\rightarrow$ From Shared Data to Shared Work.

At the end of the conceptual iteration the student created mock-ups that visualized his transfer of the patterns to the context of CURE (cf. fig. 6.18). At the time of writing of this thesis, the student is still performing development iterations in order to implement the functionality outlined by the mock-up. We can, thus, not report on the complexity of his solution. However, he also based his project on CURE which had 601 classes and 132756 lines of code.

The examiner was asked to assess the results of the conceptual iterations. He was convinced that the proposed extensions of CURE could help to support the task of collaborative Web research. Especially the selection of recommendation mechanisms was considered as promising. The student reported that the patterns helped him in better understanding the task of groupware development.

The collaboration-aware software development environment SCOPE:

The student was asked to enrich the Java software development environment Eclipse (OTI, 2003) with mechanisms for providing group awareness and reducing conflicting work. The Eclipse environment includes a connection to the version management system CVS (Price, 2000). It also allows to compare the local workspace with the versions stored in the CVS. The problem is that differences between the group members’ workspaces cannot be detected until one user decides to check in his changes. And even then, other users are not informed immediately after the check in that newer versions of the code were available. The student was asked to enrich the interaction between developers so that they become aware of other users’ changes and activities as soon as they take place.

The task was performed by a diploma student of computer science at the University of Dortmund. The student had superficial knowledge in Java development technologies and was employed to get familiar with the Eclipse environment before he started to work on his diploma thesis. He had no experience with the development of distributed systems. The student worked on the project for 12 months (of which approx. 8 months were spent on trying to understand the Eclipse platform). The customers were developers of the Distributed Systems Department of the FernUniversität in Hagen. They used Eclipse on a daily basis and felt the need for more group awareness to avoid conflicting work.

During the project, it became clear that the student had problems in managing the task of software design. Especially the mapping of discussed design structures to code was difficult for him.
He applied the following patterns (without an obvious pattern sequence):

- **From Shared Data to Shared Work**  
- **Local Awareness**  
- **Gaze Over the Shoulder**  
- **Elephant's Brain**  
- **Proxy Object**  
- **External Awareness View**  
- **Change Warning**  
- **Centralized Objects**  
- **Mediated Updates**

From an expert’s perspective, the student did a good job in selecting the patterns. But due to missing experience in software development, a second developer (acting as a mentor) would have been needed during the implementation.

Finally, he implemented 17 classes with 2115 lines of code to visualize change warnings and track changes. In addition, he modified three classes of the Eclipse CVS client in order to implement the **Gaze Over the Shoulder** pattern.

The resulting system described in (Jaabari, 2004) provided first support mechanisms for visualizing **Local Awareness** in the Eclipse environment.

### 6.6.1 Results from the Student Projects

The student projects all showed that the patterns can be used to educate developers in the context of developing collaborative applications (R10). All three students reported that the patterns supported them in their process of interaction design.

The patterns were applied very differently by the different students. The reason for this was mainly that each student had to embed the patterns in different application contexts. However, the patterns showed to provide guidance for all different application contexts (R12).

The focus of the projects laid in the development iterations (although the last project spent a long time performing conceptual iterations). The patterns helped the students to focus on one problem at a time (R6) although the capability of understanding the interplay of different patterns (R7) depended on the developer’s level of expertise. Especially trained software developers profited from the patterns, while the untrained developer had problems in creating a design based on the pattern’s advice.

All students managed to provide working software systems that included the selected patterns although the students performed differently well regarding to their expertise in software development.
The experiment was focussed to only investigate the educational value of the patterns. Therefore other requirements addressing user involvement, wholeness, and reflection were not analyzed.

6.7 Observations Made by Peer Reviewers and Shepherds

The OSDP has been commented and reviewed by reviewers of the XP2004 conference (Schüller and Slagter, 2004). The feedback was positive with regard to the possibility that the process could foster end-user participation. One anonymous reviewer, e.g., commented the focus on end-user development as follows:

\[I\text{ think that end-user participation is an incredibly important idea that few will have considered. [...] If developers gave up “considering” the needs of real users and instead brought those users into the process we would have fewer terrible systems and more good ones. In particular we might avoid many of the attacks on personal privacy and autonomy inherent in many large systems. I applaud this effort.}\]

Critical comments focussed on the additional structure that the process requests. Since agile methods argue to put people over processes and tools, this attitude could have been expected. Other reviewers asked for more details regarding experiences with the process, which shows that an expert rating of only the process is very difficult without having an example of how the process was applied. However, the experience described before in this thesis provides evidence that OSDP can be applied successfully.

The patterns of the groupware pattern language have partially undergone a rigorous review process. They were submitted to annual EuroPLoP conferences in the years 2002–2005. The reviewing process of EuroPLoP is different compared to that of other conferences with respect to the interaction between reviewers and authors. After the author submitted an initial version of the pattern collection, he is paired with a shepherd, an experienced pattern author. The shepherd has the task of asking his sheep (that’s how the author is called in the process) challenging questions in order to make the pattern submission more clear and valid. The sheep’s role is to rewrite the paper in a way that the questions do no longer emerge from the paper. Instead of defending the own work the author should think about changing the representation (and sometimes also the content) to make it clearer for the audience. A pattern language for shepherding that helps to structure the interaction between sheep and shepherd is used by most shepherds (Harrison, 1999).

At the end of the shepherding process, the pattern collections have to pass a quality gate. The shepherds and associated program committee members discuss whether the pattern collection is mature enough to be discussed at the conference.

For EuroPLoP 2005, e.g., the program chair Andy Longshaw proposed the following questions that a pattern collection has to pass before it can be accepted for a writer’s workshop:
What is the target audience for this paper? Is it clearly identified?

Does the problem feel like a real problem? If not, why not?

Is the context well framed, and does the pattern identify the key aspects (forces) associated with the problem? Are there sufficient forces to support the problem (guideline of four simple statements or two “…but…” statements)? Do the forces pull in opposite directions? What else ought to be discussed?

Does the solution match the problem? Does it convince? After reading the paper, are you better prepared to tackle the problem?

Do the patterns relate to each other? Are sequences described where several patterns are involved before a satisfying outcome is reached?

Does the format help to communicate the key pattern elements?

Is the writing clear and adequate for the intended audience?

Does the paper make a sufficient contribution to the patterns body of knowledge beyond what is already there?

Would the community benefit from finding this pattern in the EuroPLoP proceedings?

Have the authors been open to feedback? Did the communication during shepherding flow well, and induce significant changes?

Would you like to workshop this paper?

This list of questions reflects the understanding of quality in the EuroPLoP community. There is a large focus on the form. Patterns have to make a tension between forces explicit and they have to provide a solution that feels as if it would reduce the tension. This understanding is very close to Alexander’s view of quality and validity as a phenomenon that has to take personal feelings evoked by the proposed structure into account.

At the conferences, the pattern collections go through another review process. In writer’s workshops the patterns are discussed by a group of 5–10 other pattern authors. The difference to traditional conferences is that it is not intended that authors defend their work. Instead, other authors who have read the paper comment on aspects that the author should keep and issues that would require improvement. The author becomes a fly on the wall and carefully listens to the advice given concurrently by the other members of the writer’s workshop. Controversial perceptions of the pattern collection become obvious and conflicts with other participant’s experience are made explicit. As in the shepherding process, it is again the goal to better understand how users of the pattern language would understand and apply them. This provides evidence whether or not the patterns could result in the desired change of the context.

In a post-conference phase the author is requested to rewrite the pattern collection again in order to reflect the comments from the writer’s workshop. Finally, a decision on publishing the paper in the conference proceedings is made.

A comparable format was used at the CHI2004 workshop on Human-Computer-Human Interaction Patterns (Schimmer et al., 2004).

In detail, the following pattern collections were shepherded and reviewed in writer’s workshops (with increasing levels of abstraction):
Patterns for Managing Shared Objects (Lukosch and Schümmer, 2004b):

- Centralized Objects
- Replicate for Speed
- Replicate for Freedom
- Guess What I Need
- Update Your Friends
- Mediated Updates
- Believe in Your Group
- Don’t Trust Your Friends
- Detect A Conflicting Change
- Lovely Bags

Patterns from the Activity Awareness Family (Fernandez et al., 2003):

- Remote Viewport
- Telecursor
- Remote Selection

Patterns for Session Management (Lukosch and Schümmer, 2005):

- Collaborative Session
- User List
- Invitation
- Bell
- Don’t Disturb
- Lights off

Patterns for Virtual Places (Schümmer et al., 2005):

- Room
- Door
- Map
- Lock and Key
- Visa
- Locomotion

Patterns for Computer Supported Dynamic Collaboration (Schümmer, 2004a):

- Local Awareness
- Presence Indicator
- Change Warning
- Active Neighbors
- Gaze Over the Shoulder
- Elephant’s Brain
- Semantic Net

Patterns for Filtering Personal Information (Schümmer, 2004c):

- Masquerade
- Attention Screen
- Buddy List
- Birds of a Feather
- Reciprocity
- Who’s Listening

Patterns for Building Communities (Schümmer, 2004b):

- User Gallery
- Hello Hello
- Letter of Recommendation
- Activity Counter
- Find the Guru
- Pay Back
- Group Award
- Hall of Fame
- Travel Together

For these 47 patterns, we can assume a good maturity. The other patterns from the collection were not yet shepherd.

6.8 Summary

In this chapter, we have provided evidence for the validity of OSDP and the groupware pattern language using four complementary methods.

The applicability of OSDP has been demonstrated by conducting a field study in which a team applied the OSDP and used the groupware patterns for a two-year project. The resulting system was welcomed by the users and is currently used by more than 1300 users. Feedback from the users expresses a high level of acceptance. Users reported that the system well supported their interaction in collaborative teaching and learning.

We could observe that the conceptual iterations were performed as anticipated. The participatory nature of these iterations let four SIGs emerge that collaborated in

9The collection is currently in the shepherding phase.
the following development iterations. The SIGs continued to steer the development process by actively contributing to and observing the project backlog. Sharing the backlog over the internet was considered as helpful by distributed SIGs.

In tailoring iterations, the users created and shared adaptations of the system. The use of patterns in this phase was not as high as expected. However, the collection of tailoring patterns is currently evolving and distributed among users.

During the development iterations, several patterns were implemented in the system. New patterns were discovered by the SIGs and added to the groupware pattern language. The temporal sequence of applying the patterns provided hints on how structure preserving pattern sequences may look like. Several of these sequences were followed in parallel.

The requirements were addressed to a different degree in the different iterations of OSDP. The combined picture shows that all requirements can be satisfied in a project that is structured according to the OSDP.

Theoretic considerations provided indications that the sequences of patterns were structure preserving and contributed to the system’s wholeness.

The educative value of patterns has been tested in three student projects. While all students reported that the patterns helped them to better understand the problem domain, the ability to apply the patterns was different. Students with more experience in the design and implementation of software systems performed better than students without experience. Patterns should thus not be considered as a silver bullet for educating novices to act as software developers. As long as there is a development activity required, the developers need to be trained both in the pattern language and in the development language.

The maturity of the patterns as well as the potential of the OSDP have further been evaluated by different reviewers. 47 of the patterns went through a shepherding process and writer’s workshops in order to make them more mature.
Chapter 7

Conclusions and Future Work

7.1 Summary

End-user involvement is one of the most important but much too often neglected issues in groupware development. Within this thesis, we have presented a development process that addresses this shortcoming. From a theoretical view on design, we derived 15 requirements for an ideal groupware development process that is based on an active role of the users. The basis for the requirements was a holistic view on design, as it is currently propagated by many designers, especially by Christopher Alexander. In his view, a holistic approach to design has to be combined with an evolutionary process and focus on end-user education in order to empower the end-user to play an active role in the development process. Together with Heidegger, Alexander puts special attention on the situatedness of design. Users should reflect on their activities whenever their flow of action is disrupted. This situated reflection (as it was also propagated by Schön) provides the best access to the requirements and supports solutions that meet the requirements.

We showed how the theories of Alexander (most important pattern-driven design, end-user participation, and iterative development in piecemeal iterations) can be found in many contemporary software development processes. Especially agile methods provide a valuable contribution to iterative processes that preserve and enhance the wholeness of the system built. These processes advocate that the customer is the driving force in the development process and that he can control the progress of each small development increment. However, agile processes put only little interest on the question of how the end-user can be empowered to make sound design decisions.

Patterns fill this gap since they are educative tools for teaching non-experts to do good design. We discussed several pattern approaches ranging from a technical view on patterns to the human-compute interaction centered view of HCI patterns. The latter are presented in a form that is understandable by the end-user. They can therefore help the end-user while participating in the development process.

In a second step of this thesis we studied current groupware development processes in order to understand how they make use of software development techniques
in order to integrate the end-user in the development process. Although some authors have proposed to adapt iterative and agile processes to the domain of groupware development, we could not find processes that heavily focus on educating the end-user. Approaches, which meet many of the requirements, include an extended eXtreme Programming process that focuses on involving the user community in planning and development (instead of just a single customer representative in XP), and an iterative process (based on the STEPS process model) that puts special attention tailoring during system use. They demand that the end-user adapts the system in order to meet requirements that evolve from reflection in action.

The main deficit in the state of the art is that no approach meets all requirements. We identified two clusters of approaches: Firstly, a cluster addressing traditional process requirements like process awareness (R4), iterative design (R5), or reflection (R3). Secondly, a cluster addressing issues of end-user involvement (R9), education in design (R10), and sharing of best practices (R14). In order to support the end-user centered development of groupware applications we proposed to combine approaches from both clusters.

We proposed an agile development process – the Oregon Software Development Process (OSDP) – that combines techniques of eXtreme Programming and Scrum with a pattern approach. The combined process meets all requirements.

OSDP is tailored to the process of groupware development with respect to the style of the groupware pattern language used and the process of end-user involvement. Groupware patterns have to be written on a very high level of abstraction. Instead of focussing on technical details (which would lead to a technology driven groupware development) they have to focus on interaction between users and how this interaction is mapped to the computer-mediated context. The involvement of the end-user focuses on embedding the process of reflection in the daily use of the collaborative environment and resolving break situations in the groupware system. All users instead of just one user should be empowered to steer the flow of development of the groupware application. All users should be able to tailor their application and share the tailoring results with others. And all users should contribute to a growing body of knowledge regarding the tool construction and use for their problem domain.

OSDP structures the development of the application in three kinds of iterations:

- In conceptual iterations users and developers collaborate in scenario interest groups in order to collect and refine scenarios of system use. They make use of groupware patterns to inform their mapping of social processes to a groupware setting.

- Development iterations focus on the implementation of the scenarios from conceptual iterations. Users and developers collaborate to create task descriptions and to relate these descriptions to groupware patterns. These descriptions are collected and estimated regarding their costs and benefits. The development team continuously focuses on the most important tasks in order to implement the most critical aspects first. Important aspects of the development iteration include that the list of tasks – the backlog – is made public in the user com-
community and that the task wishes of different users or SIGs are ordered and merged.

- In tailoring iterations users reflect on their groupware use and fix conflicting forces by adapting the groupware system. Pattern scouts encourage the users to share their adaptations with other users. If the adaptation matured it is formulated as a pattern and added to the community’s pattern language.

We argued on a theoretical level that the process meets all requirements. We related each technique used in OSDP to the requirements and showed how it supported the requirements in other process models. OSDP therefore fills the gap of a combined process model that addresses traditional development process issues as well as end-user involvement, education, and sharing of best practices.

Patterns play an important role in the whole process since they are the means for communicating design knowledge between users, developers and between users and developers. For that reason we presented a still evolving groupware pattern language and provided example patterns from the pattern language for supporting group formation based on enhanced group awareness.

The validation of the approach in real-world projects as well as student projects showed that groupware was able to follow the proposed process and that the patterns play an important role in the communication between all stakeholders. We observed that users were able to play an active role in all phases of the process. They shaped their environment and developed new best practices (e.g., practices for tailoring in order to support literature research). First of these practices were captured as patterns.

However, the process and the patterns are no silver bullets.

Patterns with a high level of abstraction can often be implemented by tailoring the application. However, low level patterns still require development and design expertise by the involved software developers.

The involvement of the user is another critical aspect. The user takes a very responsible role in OSDP. He has the power of controlling most aspects of the development process. However, if he does not steer the process it will very likely flow into directions that do no longer fulfill all user requirements. In addition, the responsibility of the customer requires new ways of thinking about project contracts. Projects can no longer assume that the development team will implement a fixed set of features in a fixed time frame for a fixed budget. Instead, users have to agree to be in charge of all these aspects for each small increment.

Being aware of these danger spots we can, however, summarize from the experiences that all requirements are addressed by the combination of the OSDP and groupware patterns.

### 7.2 Main Contributions

It is now time to restate the main contributions. They are:

- the introduction of the *Oregon Software Development Process* that fosters end-user participation,
- a new *format for design patterns* that is especially focussed on end-users,
- a *pattern language for group formation based on enhanced group awareness* and the presentation of selected patterns in an elaborated format,
- a *collection of thumbnails of additional patterns* from a pattern language for groupware development,
- a *case study of applying the process and the patterns* in a participatory design process of the collaborative learning platform CURE, and
- the *evaluation of pattern languages regarding the role of structure preserving transformation* in the context of the CURE project.

It is up to the reader to be convinced by now. Time will show if the contributions put end-user centered groupware development forward.

### 7.3 Future Work

As in every research work, one study opens up many new questions. We conclude this thesis with some open issues that could be addressed in future work.

Most important, we hope that this work will help to focus the different activities in the field of groupware patterns research. The goal would be that a holistic groupware pattern language would be created by experts from the community. This pattern language should be completed in order to cover the whole area of computer-mediated interaction. However, since most groupware development experts are still researchers, their motivation to capture proven solutions in patterns is still low. A changed view of novelty and research contributions is needed to help the field to mature. This view would need to welcome activities that try to condense existing knowledge and provide a novel way of communicating the solution instead of looking for novelty only in the solutions that are built.

The evolving groupware pattern language should be connected to other pattern languages, especially for human-computer interaction. The CURE case was a good example of how user interface design can be neglected. A current redesign of the user interface using user interface design patterns looks very promising. If these collections were merged beforehand, the users would probably not create unreflected user interfaces for well reflected groupware applications.

The use of a groupware application for collecting and connecting different pattern languages looks promising. The problem is to motivate pattern authors from different domains to contribute their patterns to the environment.

One could further imagine that users discuss how the patterns shaped their context: Instead of simply consuming the content, users could comment the provided
7.3. FUTURE WORK

patterns. van Welie (2005) has made a first step in this direction by adding comment fields to each pattern of his collection. But we believe that a discussion structure comparable to that of the writer’s workshops or the shepherding process could further raise the quality of patterns and the involvement of the users. Users could be encouraged to report on their experiences when applying the pattern, provide additional known uses, or reflect on additional danger spots.

The OSDP should be applied in other contexts in order to better understand its applicability. Especially, it should be tried in non-groupware settings. The question how different users can be involved in the development of such applications may be comparable to the question of how to involve groupware users. Evidence for this is, however, still to be found.

At a technical level, it may be interesting to better observe how the functional high-level requirements of the groupware patterns are mapped to software structures. Since many of the patterns crosscut component boundaries it is very likely that another development approach than object-oriented development could better support the mapping from patterns to code. Aspect-oriented development seems to be a promising step in this direction.
Appendix A

Additional Thumbnails

Active Neighbors

Problem: The Local Awareness pattern only signals confocal users on the same artifact. If users work on related artifacts, they are not aware of each other, which implies that no collaboration will be established. On the other hand, especially collaboration on relate topics can support creative processes and mutual learning.

Solution: Provide awareness on peripheral activities that take place on related artifacts. Use a Semantic Distance to show how relevant those activity are. Rate activities on artifacts with a short semantic distance more important than activities with a long semantic distance. Ensure that activities on related artifacts do not distract the user’s attention too much from the focused artifact.

Activity Counter

Problem: In a collection of shared objects, there may be more and less important objects. Especially for a newcomer there is no easy way to distinguish important from less important objects. This may result in a situation where the newcomer gets lost.

Solution: Add an activity counter to the visualization of the shared artifact. Artifacts that are important for the community will have a high number of activities such as visits, downloads, or updates. Unimportant artifacts may not attract as many visitors and therefore have a low activity counter value.

Activity Status Bar

Problem: Users need time to perform a Unit of Work. From co-located settings users are used to perceive non-verbal signals such as movements or noises if another user is active. If the users are distributed, these signals are missing. Users have to wait for the end of another user’s action or until a Timeout can be detected. Such delays discourages interaction between users. If the users on the other hand do not wait, actions can be concurrent and conflicting.

Solution: Provide an user interface element that shows that remote users are
active and what they are currently doing. Ensure that the remote user's activity is shown in the user interface immediately after the user started the action. Hide the activity if no activity is detected from the user (e.g., no keyboard input for a specific period of time).

**APPLICATION SHARING**

**Problem:** You want to collaborate using a problem-specific application but this application does not support synchronous use of many users.

**Solution:** Use an application sharing system that replicates the view and the controller of the application to the users’ machines.

**ASK THE AUTHOR**

**Problem:** In textual communication, the reader of a message can only refer to the written message in order to understand the sender's message. However, the message may be ambiguous.

**Solution:** Provide means for the reader to ask the author for clarification.

**ATTENTION SCREEN → 195**

**Problem:** Every request for attention needs to be processed by the user. Thus, it already takes some of his attention. But in situations, where the user needs to focus his attention on other things, this is disturbing.

**Solution:** Enable the user to filter the information which reaches him. Use meta-information (e.g. sender details) or content information (e.g. important keywords) to distinguish important information from not so relevant information. Collect the less important information at a place where the user can process it on demand and forward relevant information directly to the user.

**AUTHORITATIVE INSTANCE**

**Problem:** Users contribute to a collaborative information space. They provide their knowledge and receive other users’ contributions. However, if the users lack expertise they may contribute information with errors or believe in faulty information provided by other users.

**Solution:** Select expert users as authoritative instances for a collaboration space. Provide them with a quality seal that they can place on information that is free of errors.

**AWARENESS PROXY**

**Problem:** Your application is a client server system. Clients communicate with servers by sending requests and waiting for a reply. You want to include awareness information in the transmitted content without changing the client or the server.

**Solution:** Add a proxy that modifies and monitors the communication between the client and the server. Log each object that a user accesses (by a request) using GAZE OVER THE SHOULDER←145. Also log the changes that a user performs by requests
using a Elephant’s Brain\textsuperscript{150}. Modify the domain object before delivering it to the client to add awareness information (the different options for modifications are described in the section How to be aware).

**BELIEVE IN YOUR GROUP**

**Problem:** You want to ensure consistency but you do not have much time, until you can perform these changes.

**Solution:** Perform the change immediately. If another client performed a conflicting change earlier (cf. DETECT A CONFLICTING CHANGE\textsuperscript{229}), undo or rearrange your change.

**BELL**

**Problem:** When latecomers want to join a collaborating group, they may intrude the group’s privacy. The group may feel disturbed by uninvited visitors, or simply not notice that someone wants to join.

**Solution:** Provide the meeting place with a bell that allows latecomers to raise attention regarding their will to participate in the group. Model the bell as an interface element that is associated with the representation of the shared space where interaction of the group takes place. Signal the activation of the bell to the members of the group and let them decide whether or not they allow the asking user to join the group. In both cases, inform the asking user and – if possible – guide the user in the next step of entering the session (or staying out).

**BIRDS OF A FEATHER → 199**

**Problem:** If people don’t know one another it is hard to decide who could be a good partner for a collaborative activity. For co-located situations, humans have developed intuitive strategies (based e.g. on visual clues) that help them to select whom they should contact if group formation is needed. In distributed work environments the presence of other users is often reduced to their user name. This makes it hard to find another user for a collaborative activity.

**Solution:** Compare user profiles or interaction histories to identify two users who share big parts of their history. Propose these users as collaboration partners.

**BUDDY LIST**

**Problem:** When many users are able to interact in the interaction space, it is hard to maintain an overview of relevant interaction partners since the number of users exceeds the number of relevant contacts for a specific user. User lists grow very large and it is hard to find people who the local user knows. On the other hand, the local user is often only interested in those people who he knows.

**Solution:** Provide buddy lists, where a user can enter other users who are of interest to him. Whenever the local user browses other users, first show only users from the buddy list.
Centralized Objects

**Problem:** To enable collaboration users must be able to share the data.

**Solution:** Manage the data necessary for collaboration on a server that is known to all users. Allow these users to access the data on the server.

Change Warning

**Problem:** While a user works on an independent copy of the artifact, the checkout frequency may be low. So he may work on an old copy, which leads to potentially conflicting parallel changes. The conflict is worse if two parallel modifications had contradicting intents.

**Solution:** Provide change warnings in context. This means that the system indicates to the local user whenever an artifact has been changed by another user. Show this information whenever the artifact is shown on the screen. The information should contain details about the kind of the change and access to the new version of the artifact. If changes can be complex, consider to provide a comparison view for the local user’s state of the artifact with the remote user’s state of the artifact.

Collaborative Filtering

**Problem:** To recommend activities, the system has to predict a user’s behavior. In cases where the user has recommended subjects (e.g. by a *Letter of Recommendation*), the system can assume that the user will act in a comparable way with related subjects. But this calculation restricts the scope of recommendations to objects within a *Semantic Distance* of the user’s experience. Recommendations will thus not introduce the user to interesting subjects of semantically unrelated topics. But the user may like these topics.

**Solution:** Apply collaborative filtering: Find users who made comparable recommendations and scan their set of recommendations for subjects that are not in the local user’s current focus.

Collaborative Session

**Problem:** Users need a shared context for their collaboration. But computer-mediated environments are neither concrete nor visible. This makes it difficult to find a shared context in such environments.

**Solution:** Model the context for collaboration as a shared session object. Visualize the session state and support users in starting, joining, leaving, and terminating the session. Automate the selection and start of tools.

Color-Coded Distances

**Problem:** You want to distinguish important information that is related to the user’s current focus from information that is less relevant.

**Solution:** Use color-coding to distinguish near events from distant events. Use warning colors (e.g. red) for events that should catch the user’s attention and comfortable colors (like green, black, or blue) for events that occurred further away.
CONTROL YOUR PRIVACY

Problem: Your application monitors the local user or maintains communication channels to the local user. The gathered information is used to provide awareness information to remote users and to facilitate the establishment of communication channels (initiated by remote users). While this is suitable in some situations, the local user may also feel the need of not providing any information to other users and not being available for communication.

Solution: Let the user control his privacy. This means that the user should be able to filter the information, which is revealed from his personal information. Additionally, the user should be able to filter the information, which reaches him. Make sure to establish Reciprocity → 238.

DAILY REPORT → 158

Problem: Changes in indirect collaboration are only visible by inspecting the changed artifact. Users want to react to other users’ actions but they cannot predict when these actions take place.

Solution: Inform the user periodically about changes that took place between the last notification and the time of the current report.

DETECT A CONFLICTING CHANGE

Problem: If two or more users change the same data at the same time, changes interfere. This can lead to inconsistent data or contradict the users’ intentions. If the users are not aware of this conflict, they will no longer have a common base for collaboration.

Solution: Let each client remember all changes that have not yet been replayed by all other clients. Whenever a change is received from another client check it against those changes that have not yet been replayed by the other client and affect the same shared object. If the performed operations will produce a conflict then undo one of these changes.

DISTINCT AWARENESS INFO

Problem: You want to embed awareness info in the application’s visualization channel. But the awareness info can easily blur with the visualization of the application’s domain data.

Solution: Make the awareness info visually distinct from the application’s domain data. Use colors with a high contrast to the color used for displaying the domain data or icons that are visually distinct from the application’s domain specific icon set.

DON’T DISTURB

Problem: To allow spontaneous interaction, users have to be open for contact requests. But each request disturbs the contacted user or group in their current task. In addition, the importance of contact request may make it vital that the
contact takes place.

**Solution:** Include a virtual don’t disturb sign in the application that signals that the user or the group should not be disturbed.

**DON’T TRUST YOUR FRIENDS**

**Problem:** You want to ensure that nobody messes up your changes on replicated data objects, but many users are working with the same objects at the same time. Each site performs its changes locally before informing the other sites. This can lead to different execution orders of the changes. If the changes are not commutative, i.e. changing their execution order does not lead to the same shared state, the shared state becomes inconsistent.

**Solution:** Let a site request and receive a distributed lock before it can change the shared state. After performing the change let the site release the lock so that other sites can request and receive it for changing the shared state.

The lock can have different grain sizes. The grain size of a lock determines how much of a shared data object or all shared data objects can be modified after getting one lock.

**DOOR → 183**

**Problem:** In virtual rooms, objects and characters can only interact with peers in the same room what limits their possibilities. Especially, interaction between different groups cannot easily be supported.

**Solution:** Use doors between rooms as a means for navigating between rooms. Model different states of the door to reflect the mutual accessibility of the related rooms.

**ELEPHANT’S BRAIN → 150**

**Problem:** Merging two users’ (past or current) work is a difficult task. It requires that the activities are transferred to the same context and that the goals are aligned. But many applications don’t provide access to the artifact’s history, its use, and its evolution. Thus, merging is vulnerable to errors and often collaboration does not take place since the merging efforts exceeds the estimated gains of a collaboration.

**Solution:** Remember all activities that users perform on shared artifacts – not only modifying accesses, but also read accesses. Provide access to the activities, so that a user can understand (and merge) other users’ activities with his own activities.

**EMOTICON**

**Problem:** In the physical space emotions can be easily communicated by nonverbal clues. In textual communication, it is expensive to express emotional connotations by words. Emotions are often not transmitted if the communication focuses on content.

**Solution:** Let users express facial expressions by means of special characters
like ASCII-art. Exchange the ASCII text with images expressing the emotion (e.g. smileys).

**EXTERNAL AWARENESS VIEW**

**Problem:** You want to provide Local Awareness but the application’s visualization cannot be changed since there is no hook for this in the source code or the communication protocol.

**Solution:** Provide an external awareness view, that shows the part of the Semantic Net, which is currently used by the local user. Include Presence Indicator or Change Warning.

**FIND THE GURU**

**Problem:** You know that other users have more expertise with the artifact, but you don’t know who they are.

**Solution:** Find the user who shares a long history with the artifact. Use the Elephant’s Brain to see who performed activities on the artifact. Sort the list of people according to the number and type of activities and/or the time that has passed since then.

**FLOOR CONTROL**

**Problem:** Users collaborate in a setting where parallel work would lead to confusion between users or conflicting changes.

**Solution:** Model the right to interact by means of a token. A user holding the token is said to hold the floor and is the only one who can perform activities.

**FROM SHARED DATA TO SHARED WORK → 120**

**Problem:** Although many users work with the same shared data, they may not recognize the other users’ work. This results in parallel or conflicting work and a lack of collaboration and learning from one another.

**Solution:** Model dynamic groups that are built from the set of users of the shared data who share a common interest (e.g. detected by patterns Local Awareness or Change Warning).

Allow these groups to be established on the basis of their members’ activities (again Local Awareness), so that they are likely to have a common interest. Provide them with collaborative tools that support communication (Talk First and collaboration (like shared editing in a Room)).

Use the patterns of the related patterns section to detail each of the actions proposed by the above paragraphs.

**GAZE OVER THE SHOULDER → 145**

**Problem:** Many proprietary tools are not designed for extendability. They do not provide means to modify the application’s behavior. This makes it difficult to add automatic tracking of user’s activities, which you would need to provide awareness.

**Solution:** Add an additional layer in the communication between the application
and the shared data to monitor user actions. Allow other parts of your application (e.g. a Elephant’s Brain or a Local Awareness) to subscribe to monitored activities.

**GROUP AWARD**

**Problem:** Often, when a result of a collaborative process is valuable, only some of the group members get rewarded for their participation. This results in a lack of motivation for those members who were ignored in the reward.

**Solution:** Ensure that the reward for the result is shared among all group members. Find a distribution key that is considered as fair for all group members.

**GUESS WHAT I NEED**

**Problem:** The response time of interactive applications has to be short. The network latency and delay wastes time in distributed systems. Full replication, i.e. every user maintains a replica, demands high communication costs for initialization and consistency. If network communication between the users is slow, the response time of interactive applications increases and makes collaboration hard.

**Solution:** Let a user only hold a replica for a shared data object which she currently accesses or is supposed to access in near future. The other shared data objects that also belong to the state of the application can be ignored without any adverse effect to the user interface.

**HALL OF FAME**

**Problem:** The motivation for participation in a community is related to the feedback that participants receive from the community. But often very active participants are not enough recognized by the community members. This decreases the motivation of the active participants to continue their efforts.

**Solution:** Provide a list of those participants who participated most. Calculate the participants’s participation level with respect to the degree that the participants helped others. Let each participant compare himself to those participants shown in the Hall of Fame.

**HELLO HELLO → 163**

**Problem:** If the group wants to progress it is often needed that they integrate new members. But since the group members are very focused on their internal interaction they may fail to notice new potential members and ignore their possible contribution.

**Solution:** Provide a prominent place in the community’s interaction space where new members and their ideas are introduced. In a computer mediated group, this can for instance be a special section on the group’s home page. Whenever a new member joins the community ensure that the existing group members notice the new member.
APPENDIX A. ADDITIONAL THUMBNAILS

IMMUTABLE VERSIONS

**Problem:** Users want to be able to work independently and make their results accessible to other users - regardless the state of the artifact they started their work with. If two users change the same artifact, this results in conflicting changes and one change is often lost.

**Solution:** Store copies of all artifacts in a version tree. Make sure that the versions stored in the version tree cannot be changed afterwards. Instead, allow users to store modifications of the version as new versions.

IN-PLACE AWARENESS VIEW

**Problem:** You want to provide awareness information that relates to specific artifacts. But explaining this relation explicitly puts an extra burden on the user.

**Solution:** Place the awareness information next to the artifact to which it refers.

INTERACTIVE USER INFO → 199

**Problem:** Users are able to associate the awareness information with the artifact, but it is still difficult to find out how to launch appropriate collaborative tools (e.g. for opening a chat tool with the confocal user – cf. Talk first → 242).

**Solution:** Make the user representation interactive so that another user can reveal more information on the shown user or start tighter collaboration with the user by acting on the user’s representation.

INVITATION

**Problem:** One user wants to interact with another user. But the other user may not be ready for the interaction.

**Solution:** Send and track invitations to the intended participants that include the topic, the scheduled time, and the (virtual) location of the collaborative session. Automatically add all users who accepted the invitation to the collaborative session.

LETTER OF RECOMMENDATION

**Problem:** When users don’t know potential interaction partners, they may fear interaction because they don’t trust the partner. This may result in a high inhibition threshold and thus non-existent interaction.

**Solution:** Let the users rate the interaction and display an analysis of all users’ ratings together with the users or artifacts the user interacted with.

LIFE INDICATOR → 190

**Problem:** If users work mainly asynchronously they can loose track of their own activity level compared to the activity level of other users. Especially, they project their own level of activity on other users. But if these levels don’t match, expectations will differ from reality and users will be frustrated because of long periods of silence from their interaction partners.

**Solution:** Show a LIFE INDICATOR together with the user’s virtual representation.
If a user participated in the system recently, use a picture for the indicator that looks very *alive*. Use gradually more *dead* pictures to represent the period of inactivity. Make the picture look like a toy for which the user can take responsibility.

**LIGHTS OFF**

**Problem:** After interacting in a **Collaborative Session**, users want to resume their collaboration with the already achieved results or want to review these, but the results are not available\(^1\).

**Solution:** Persistently store achieved results of a synchronous **Collaborative Session** on a central server. If shared data is replicated, use the **Mediated Updates** pattern to keep a master copy of the shared data and track all changes that are applied to the shared data. Let users access the master copy at the central server for reviewing or resuming purposes.

**LOCAL AWARENESS → 127**

**Problem:** Although most systems that work on shared data provide support for coordinating shared access, they often don’t tell the user, who is working on a specific artifact. Such information is needed to establish ad-hoc teams that share a common focus. Without such information, users assume to work alone – and do not see the possibility or urge for collaboration.

**Solution:** Provide awareness in context. This means that the system tells the local user, who else is currently interested in the local user’s focussed artifact and what they do with this artifact. Show this information whenever the artifact is shown on the screen. The information should contain details about the user drawn from his user profile, the artifact, and details on the activity, which the user is performing. Ensure that the information is always valid.

**LOCK AND KEY → 193**

**Problem:** People feel the need to distinguish public from private places and thereby identify private objects that are located in the private place. They may want to restrict access to private places and the contained private objects so that only familiar users can access private places and objects.

**Solution:** Protect private places with locks that can only be opened with matching keys. Provide the users with keys that can be used to open locks. Make these keys first-class objects that are visible to and portable by the user. Place private property inside locked rooms.

**LOCOMOTION**

**Problem:** Placing objects in the virtual space helps to receive a spatial impression. But static placing of objects only allows one fixed perspective on the virtual space.

\(^1\) Many groupware systems do not care about storing achieved results for reviewing or resuming a collaborative session. Well-known examples are IRC clients that do not allow to log and store the conversation.
Solution: Provide the user with means for navigating through the virtual space from one place to another.

LOGIN → 181

Problem: You are developing an application that requires non-anonymous interaction.

Solution: Provide a LOGIN screen that requests users to identify themselves by entering a login name and a password before they can start to use the application.

LOVELY BAGS

Problem: Access operations to shared container objects change the content of the container by adding or removing elements. Most of these operations are very bad regarding concurrency. Thus, synchronous collaboration on container objects often seems impossible.

Solution: Wherever a high level of concurrency is needed model your container objects by means of a bag. If the container’s entries need to be ordered equip the data entries with an order criterium that can be uniquely assigned by each client (e.g. the current time stamp together with a unique client ID) but still store the entries in the bag.

MAP → 184

Problem: To orient in space, users have to create a mental model that represents the space and the contained artifacts. This is a very difficult task.

Solution: Create a visual representation of the spatial domain model by means of a map.

MARKER

Problem: Textual message look uniform which makes finding remembered content difficult.

Solution: Allow users to select content and place a marker on this content. The content will be shown visually distinct from other content (e.g. with a graphical icon or another color).

MASQUERADE → 196

Problem: User monitoring is required for providing awareness information to remote users or associating work results with a specific user. On the other hand, users often do not act as confident if they know that they are monitored as they would act in an anonymous environment. Especially openness and the courage for taking risks may be much lower.

Solution: Let the user control how much interaction information he provides to the system. This means that the user should be able to filter the information which is revealed from his personal information. Remember to consider RECIPROCITY...
**Mediated Updates**

**Problem:** Clients want to propagate update messages to other clients who keep replicas of the same data. If they contact the other clients directly, they have to maintain information who those clients are and have to establish communication with these clients. This is complicated and error-prone. Especially if some clients may disconnect and reconnect in an unpredictable way (if the set of clients changes over time).

**Solution:** After changing a replicated object inform a mediator which will distribute an update message to all interested clients.

**Mentor**

**Problem:** Newcomers have problems orienting in an unfamiliar community. Especially, they don’t know how community members normally act in specific situations.

**Solution:** Pair the newcomer with an experiences group member. Initially let the newcomer observe the veteran and gradually shift control to the newcomer.

**Moderator**

**Problem:** Group members have problems to conform to interaction norms or quality standards. This offends other members of the group or distracts them from their current task.

**Solution:** Select one user as a moderator who administrates the rights to release contributions to the group’s interaction space. Let the moderator asses each contribution to ensure quality standards.

**More than 1000 Words**

**Problem:** Online communication often starts using text-based chat or audio conferencing. Graphical content, like a picture, figure, or sketch, is difficult to convey by words. If communication partners try to use words for describing graphical content, the description is always a subjective interpretation of the person who describes it. Text- or audio-based communication forces people to communicate solely by words.

**Solution:** Provide an appropriate means to exchange graphical objects. Relate these objects to the existing communication channel, to set them in context.

**Opposites Attract**

**Problem:** If users only meet users with the same ideas (as proposed by Birds of a Feather), they will have problems generating new ideas.

**Solution:** Pair users with different opinions whenever you want to initiate creative and innovative processes.

**Pay Back**

**Problem:** To keep the users spending efforts for the community, they have to be motivated. But often the efforts are not reciprocal so that users don’t see benefits
related to their efforts. Thus, participation declines.

**Solution:**  Provide each user with virtual money that can be used to purchase services within the community. Let users earn a specific amount of money, when they positively contribute to the community.

**Personal Annotation**

**Problem:**  While interpreting the received message, the human brain elaborates on the content. If communication goes on, the elaboration result may be lost.

**Solution:**  Provide users with means for capturing their thought regarding a received message. Link the comment to the message and provide the owner of the comment access to the comment whenever the message is shown.

**Personalized Attributes**

**Problem:**  When users work on shared data, they all have the same state. This implies that all views that are based on the shared model are showing the same content. Thus, the only way of interaction is tightly coupled interaction (WYSIWIS – What You See Is What I See). But there may be the need to relax this coupling so that users can for instance scroll to different screen regions (relaxed WYSIWIS).

**Solution:**  Model application specific attributes of the shared data as personalized value holders that store the value for each user.

**Presence Indicator → 135**

**Problem:**  The In-Place Awareness View makes it easy to connect other users’ activities with focussed artifacts. But the surrounding of the artifact provides only limited space for information. Awareness information thus competes with application data.

**Solution:**  Limit the size of the awareness information’s representation so that it uses only a small part of the available information channels. For a GUI system, this means that you should represent the confocal or peripheral users as a single icon instead of a long textual form. Focus on telling that there are other users, rather than providing much information on the other users’ identity or task. Ensure that the indicator differs from the other artifacts representing application data.

**Proxy Object**

**Problem:**  You want to capture activities on artifacts, but you cannot access these artifacts from within your logging mechanisms. Especially, it is not possible to modify the artifacts so that they include references to the activities or to other artifacts.

**Solution:**  Create a proxy object that includes a reference to the real artifact and stores additional information such as references to activities or semantic relations to other artifacts. Provide mechanisms to retrieve the real object for a given proxy object and to retrieve the proxy object for a given artifact.
RADAR VIEW

Problem: You are collaborating in a large shared workspace and want to stay aware of other users’ foci to maintain a mutual understanding of each user’s work. But since the users’ work areas do not overlap, you can no longer see the other users’ viewports within your viewport.

Solution: Display a shrunken view of the shared workspace as a radar view. Visualize the local user’s position as well as remote users’ position (e.g. by means of Remote Viewports) to ease orientation and determine shared portions of the workspace.

RECIPROCITY

Problem: It is easy to agree on participation, if the goal is beneficial for everyone. But in many work situations, some people benefit more than others from a reached goal. In the extreme case, the beneficiaries of the reached goal do not have to participate in the group efforts at all. This leads to a situation, where the people who have to spend efforts on the group result no longer see the need to participate since the results are not valuable for them.

Solution: Establish reciprocity. Ensure that all group members’ activities result in an improved group result that is beneficial for all group members again. Prohibit people to benefit from group results if they are not willing to help the group in return.

REHABILITATION

Problem: User ratings always reflect the impression of a user at one specific point in time (and in a specific context). When reading the ratings, users expect that the rated user will act in a comparable way in comparable interaction contexts. But users may change.

Solution: Provide a voting workflow (Vote) to allow a user to adapt his user’s rating (or letter of recommendation) to his current behavior. Ensure that many users, especially users with negative views and users with positive views, are involved in this workflow.

REMEMBER TO FORGET

Problem: You are using the Elephant’s Brain and Time Compressor to provide asynchronous Local Awareness. This ensures that users leave their traces. The traces stay persistent to allow future reference to a user’s activity. At some point of time, you detect that some activities are no longer remembered by the user, but still displayed as asynchronous context awareness. The users will no longer be able to understand, why they relate to a specific artifact, since they forgot that the have ever seen this artifact. Thus, references to these activities do no longer encourage communication or collaboration, but confuse the parties.

Solution: Remove remembered activities after a point in time, when an average user will no longer remember this activity and the activity is no longer important for
the application. Consider the type and length of interaction as factors that influence, how long an activity is remembered.

**Remote Selection**

**Problem:** In a collaborative system, selections may be different for each of the users to allow parallel actions. This is problematic in two aspects: First, it produces confusion in actions since the cognitive expression of selecting elements and then having control over these elements does not hold anymore (other users may have overlapping selections). Second, hinders communication because it is not possible to use selection as focus of communication (as it is done in single user applications).

**Solution:** Visualize the selections of remote users to the local user. Make sure that other users, who are interested in a specific artifact are aware of a distributed co-worker, who selected this artifact.

**Remote Subscription**

**Problem:** Clients that show objects maintained from a server assume that these objects are valid. But since the visualization may be shown for a long time, the visualization may be out-dated. Users consuming the information will receive incorrect information.

**Solution:** Let clients subscribe to shared data’s state and ensure that the server informs the clients about relevant state changes.

**Remote Viewport**

**Problem:** In synchronous groupware that allows users to work on different parts of the same document, there is usually the requirement of knowing (or coordinating) the location and extension of the document others are working on. This results in an overload of human communication.

**Solution:** Explicitly indicate the location and the scope of each user’s view to every other user. In the case of graphical documents this can be achieved by including labelled rectangles that represent – in position and scope – the windows of remote users. In the case of textual documents, this can be achieved by adding additional, passive scrollbars that represent the scroll position of remote users in the document.

**Replicate for Freedom**

**Problem:** Users may not have a permanent connection to the system, where relevant data is kept. Without a permanent or just a poor connection to the data, users will not be able to finish their work, if the data cannot be accessed.

**Solution:** Replicate the data to the user’s device. Update the replicas whenever two systems which hold copies of the data connect.

**Replicate for Speed**

**Problem:** The response time of interactive applications has to be short. The network latency and delay wastes time in distributed systems. Thus interactive
applications are inappropriate if the response time depends on client-server communication.

**Solution:** Replicate the shared data to the users’ sites. Let a user change its local replicas and ensure consistency by using the Update Your Friends pattern.

**Role**

**Problem:** Users have problems to structure their interaction in the group. Especially, some users act in a way that is not anticipated by other users. This results in chaos.

**Solution:** Define roles that describe what the owner of the role is supposed to do. Link the roles to users when they engage in the group process.

**Room → 182**

**Problem:** When distributed users want to collaborate, they have to deal with many technical issues to initiate the collaboration. They have to set up communication channels and make the shared material available to all group members. Results of the shared collaboration often need to be captured for later reference. All these tasks are error-prone and require technical expertise, which users often don’t have.

**Solution:** Model a virtual place for collaboration as a room, which can hold documents and participants. Ensure that users, who are in the same room at the same time, can communicate by means of a communication channel that is automatically established between all users in the room. Make sure that all users can perceive all documents that are in the room and make these documents persistent. That includes that changes on these documents are visible to all members in the room.

**Semantic Distance**

**Problem:** Your Semantic Net is very dense in a sense that artifacts have a semantic relation to many other artifacts. But not all artifacts have the same importance for the user. If the user sees only the semantic net, he might get lost in the diversity of relations.

**Solution:** Use weighted edges to describe the strength of the semantic relation. Interpret these edges as distances. If two artifacts are semantically strong related, ensure that the connecting edge in the Semantic Net represents a short distance.

**Semantic Net**

**Problem:** Detecting short semantic distances between artifacts based on a similarity measure often leads to ineffective and inexact results. It is time consuming, when there are many artifacts with large distances because this would involve much unnecessary computation. In addition it fails, if two artifacts are related by means of an intermediate artifact.

**Solution:** Produce a semantic net that contains artifacts and relations between artifacts. Relate two artifacts, if they have much in common (as in the Semantic Distance pattern). Define the distance between two artifacts as the length of
the shortest path between these artifacts.

**Shared Annotation**

**Problem:** When discussing specific content, it is difficult to target the comment on specific parts of a content. Users have to explicitly describe how to find the content.

**Solution:** Provide means for entering comments on specific messages/content. Collect all users comments and display them together with the message/content.

**Shared Folder → 182**

**Problem:** Different collaborating groups share a lot of data. Due to the amount of data from different groups, users cannot find the data they need for collaboration.

**Solution:** Allow groups to organize their shared data in composites that are modeled as Centralized Objects.

**Show the Expert**

**Problem:** Individual user’s knowledge may be useful for other users. On the other hand, exposing knowledge can mean that the knowledgable user is confronted with questions of the other users. So, how can people regardless the efforts be motivated to expose themselves as experts for a specific subject of the shared application domain?

**Solution:** Show experts for each shared object in the collaborative application. Consider various means for deciding, who is the expert for a specific artifact: e.g. the time spent modifying the artifact, the occurrences of consuming the artifact, or explicit assignment of users to an artifact. Show the expert’s name together with the artifact.

**Spatial Domain Model**

**Problem:** Users of a system using a Semantic Net have difficulties to understand what it means to work with a specific artifact. Especially, they have difficulties to understand that one object has a strong semantic connection to another object. On the other hand it is essential that the user understands these relations to cope with Active Neighbors.

**Solution:** Use the metaphor of a virtual space, which can be inhabited by the application’s users. Provide easy means to inspect two related objects in a sequence. Describe the change of focus from one object to another with terms that are commonly used to describe movements in the real world.

**Swarm and Collect**

**Problem:** Your group has to find information in a large information space. For each individual group member this would take more time than he has. Thus, the whole information space cannot be scanned individually.

**Solution:** Let the users swarm out in the information space and collect relevant information in a shared workspace. At the end of the session let them discuss and
consolidate the found information.

**Talk first**

**Problem:** You have encountered another user, who is working on the same or a related artifact. It is obvious for you that collaboration will help you to better achieve your goal. But this may not be obvious for the other user, since this user can have different goals. The other user may be offended by your disruption.

**Solution:** Communicate with the other user before you start a tightly coupled session. Use the communication to get to know one another and match each other’s goals. Also negotiate on the following process of collaboration.

**Telecursor**

**Problem:** In synchronous groupware applications, users simultaneously manipulate artifacts using diverse input devices. As the input devices of different users are decoupled they cannot longer be used to gesture or focus communication. Moreover, when users attempt to manipulate the same artifacts it results in a state that is unexpected to all of them involved. This hinders collaboration.

**Solution:** Represent the targets of other users’ input devices as telecursors. Telecursors are visualizations of these input devices that look comparable to the local representation of the input device (but different enough to distinguish them from the local input device’s representation). They are shown on remote users’ screens. Telecursors are strictly bound to the position of the input device.

**Tell Me a Story**

**Problem:** If users have not participated in the evolution of an artifact in a collaborative information space, they may have difficulties in understanding the artifact’s current state.

**Solution:** Replay the other users’ activities in a way that the user sees how the artifact evolved. Show old versions of the artifacts together with a description of what the users did. Include both navigational and modifying activities.

**Threaded Discussions**

**Problem:** If more than one user is allowed to make contributions to an interaction space (like a chat or a forum), it becomes difficult to structure the contributions in a way that their referral structure can be understood by the users. This leads to potential misunderstandings and chaos.

**Solution:** Use threaded discussions. Track, which contribution a user viewed when he decided to react to this contribution. Submit the contribution as a response to the viewed contribution and relate it the response visually to the responded contribution (e.g., arrange both contributions in a close spatial proximity or draw a visual reference between both contributions).

**Time Compressor**

**Problem:** You are using **Local Awareness** to inform the local user of
activities of his colleagues. The underlying Spatial Domain Model is large and the number of users is relatively small. Thus, two users will not likely be at the same place at the same time. Thus, the spatial domain model will look as if it was empty.

Solution: Collect and combine activities that took place at the same artifact at different points in time. Show the distance in time between two usages of an artifact (e.g. by using Color-Coded Distances). Let the user scale and limit temporal distances.

Timeout
Problem: In a group process, all participants assume that other participants do not stop interacting. However, in computer-mediated systems, the interaction means (namely the network or the computer system) can disconnect users from a group so that they cannot react in a timely manner. This is crucial for any group interaction.

Solution: Attribute crucial interaction steps with an upper limit for the duration of the interaction. If nothing happens in that time, restore a consistent system state with the assumption that the user will not provide the awaited data. This may also include the invalidation of information about the user.

Travel Together
Problem: When finding their way through an unknown environment, users can often get lost.

Solution: Browse through the information space together. Provide means for communication and collaborative browsers that show the same information at each client’s site.

Unit of Work
Problem: Users think on a higher abstraction level than computer systems. Normally, a user’s intended modification is split up in several commands modifying the shared data. If only a part of these commands is executed, this is different to the users’ intents and leads to confusion.

Solution: Execute the changes to the data model by means of Commands. Log each command on a Unit of Work in a way that the Unit of Work includes information about accessed and changed data. When the user’s transaction is complete, commit the changed data that was logged on the Unit of Work.

Update Your Friends
Problem: Users change their local copies of the replicated artifacts and the other users cannot notice these local changes. This makes collaboration impossible.

Solution: After changing a replicated object locally
- send an update message for this object to all clients that also maintain a replica,
APPENDIX A. ADDITIONAL THUMBNAILS

– take care that all clients receive this update message, and

– let these clients change their replica according to the information in the update message.

**User Gallery → 197**

**Problem:** If more than one user interacts with shared data, it is hard to coordinate the interaction - especially with strangers. Without knowing who is using the system, it is hard to establish collaboration or to become aware of other users’ activities.

**Solution:** Provide a list of all users who are members of the community. Let the members provide personal information in this list that is related to the community’s task. Design this list in a way that it is interesting to browse.

**User List → 141**

**Problem:** Users who only share common data have no mutual awareness of each other. Users don’t like to perform activities if they don’t know who observes them. Thus, they will not act freely with the shared artifacts.

**Solution:** Show the names of all users who are in the same session in a user list and ensure that the list is always valid.

**Virtual Me → 197**

**Problem:** Account names used by users of a collaborative application are often not very expressive. Especially, an account name is only a short textual representation of the user. In a large user community, account names look similar. But users need to communicate their identity in order to interact with other users.

**Solution:** Allow the users to play theater! Provide them with means to create a virtual identity that represents them while they act in the system. Show the virtual identity when the user is active.

**Visa → 200**

**Problem:** While the protection of group interaction by explicit Logins protects the community it also makes it difficult for non group members to interact with group members. Especially, if there exist some (e.g. social or business) reasons why those non group members of the system cannot become permanent group members, interaction is not possible.

**Solution:** Offer the users of the system a possibility to grant visas (temporary user accounts) to non-users of the system.

**Vote**

**Problem:** Users need to be aware of other user’s attitudes if they want to interact. But user profiles cannot provide an answer to all attitudes that a user has. And even if the attitudes are part of the user profile as modeled by the Virtual Me → 197 pattern, it is still hard to figure out the distribution of opinions in the community.
Solution: Present the community members with or let them organize polls on controversial questions in the context of the community’s topic. Show a virtual ballot on a prominent place in the community. After the user has voted provide him with the result.

Who Am I → 191

Problem: After users connect to a system the system keeps track of the identity of the connected user (e.g. by means of the Who’s that Girl pattern). But the user may forget under which account he connected to the system. Misuse of accounts can be the result.

Solution: Show the name of the connected user on each page. Make this name distinct from other user’s names.

Who’s Listening

Problem: Users are providing information for other users by means of shared objects. But making an object accessible does not ensure that the object was seen by other users.

Solution: Inform the author of a shared object when another user reads this object.

Who’s that Girl

Problem: Since received messages all look the same it is difficult to determine the originating user of received events.

Solution: Let the senders of messages include a unique attribute in the messages (e.g., a request field, a cookie, or a unique network sender address). At the receiver, map this attribute to a user.

You Are Here

Problem: When using a map users infer the perceived environment with its abstract representation on the map. They compare visual properties to find a match between the map and the environment in order to locate their current position. This is often a difficult task.

Solution: Track the user’s position (e.g., using the Gaze Over the Shoulder pattern) and visualize it in the map (using, e.g., a red dot).
Appendix B

The Authoring and Viewing Environment CoPE

B.1 Linking Patterns in a Pattern Language

Patterns need to collude to create sequences of structure-preserving transformations. Alexander (1979) described this regard by comparing pattern languages with natural languages. As speakers connect single words to form literate sentences a user of a pattern language can combine patterns in a pattern sequence in order to create wholeness.

But as learning a natural language is a difficult task the users also have to manage the learning process of a pattern language. It is thus reasonable to look for supportive means that help users in understanding relations in the pattern language.

Two obvious relations are already part of the pattern description. In the pattern context patterns from a higher level of abstraction are named. The inverse relation is found in the related patterns section. such a relation from high-level patterns to the patterns on the next lower level is in many cases bidirectional. But the example of the related pattern section already reveals a problem: Not all patterns named in the related patterns section are patterns on a lower level of abstraction. They can also be patterns on the same level that describe alternative solutions to the same problem.

We argue to support the user by illustrating a combined set of relations in a pattern graph. The relations can be modeled as (labelled) edges between two pattern nodes X and Y. We propose the following relations:

- X mentions Y in its context. This means that Y was applied before Y.
- X is related in its related patterns section to Y.
- X uses Y in its solution.
- X is a variant of Pattern Y.
- X specializes Y. This relation is comparable to inheritance in object-oriented development where the subclass specializes its superclass. The specialized pattern still solves the problem of its ancestor but adds aspects that make the solution more specific.

- X connects to Y as part of the sequence S. The label of the relation explains the step in the sequence that leads from X to Y.

Besides these relations between patterns we propose to add relations between patterns and other artifacts:

- X and Y are members of the same family. A pattern family is a set of patterns that are addressing a related design issue. Examples are patterns for providing group awareness or patterns for managing shared objects. A concrete example of a pattern family is the collection presented in chapter 5.

- X and Y involve a common participant P in the collaborations section of the pattern. In this case, X and Y are related to P. The label of the edge between the pattern and the participant describes the role of the participant in the pattern.

- X and Y can be found in the same known use U. This is modeled using an edge between the pattern and U. The label explains how the pattern is used in the known use.

All these relations have been part of the textual pattern description. But especially the latter relations between patterns and other artifacts were not made explicit (because neither the participants nor the known uses were considered as first class objects in current pattern structures). We argue that the relations should be modeled as explicit (typed) hyperlinks in the pattern description. With appropriate tool support (a hypertext environment) users can explore the different relations between patterns and thereby gradually learn how the patterns form a pattern language.

We created the authoring and reading environment CoPE\textsuperscript{1} to support the users and authors in reading, authoring and sharing the patterns. It is a collaborative application that provides means for creating and linking pattern descriptions (cf. fig. B.1). It presents the pattern map (in the left part of fig. B.1) and the hypertextual representation of the pattern (in the right part of fig. B.1). A pattern can be edited using a WIKI syntax (Leuf and Cunningham, 2001).\textsuperscript{2} Especially, the author can add links to other patterns that express the semantics of the relation.

The hypertext representation of the pattern is analyzed by the tool and a graphic representation is created. The different relations between patterns can be expressed using a graphic pattern diagram representation that is close to UML.

\textsuperscript{1}CoPE is an acronym for Collaborative Pattern Editor.

\textsuperscript{2}A WIKI syntax is a collection of simple text markup commands that is parsed to generate hypertexts.
Figure B.1: The reading and authoring environment CoPE.
Patterns are represented as ovals. In most diagrams, the hyperlinks between patterns are not detailed. This means that the relation between the patterns will be shown using a simple arrow without a label. CoPE allows the author to change the type of an association in order to express specialization (extension) of a pattern, or variants where two patterns solve the same problem differently.

If needed, pattern diagrams can be extended with information on known uses and participants. Known uses are shown as UML components (a rectangle with two little rectangles decorating the top-left corner). A solid line between the pattern and the known use shows that the pattern was applied in the known use.

Participants are represented as rectangles (following the notation for objects in UML diagrams). They can be connected with directed associations. Each participant can link to a class. This will be shown as a UML class symbol (with collapsed attributes and methods). It is not intended to include full UML object diagrams as part of the pattern diagram. So, the number of potential associations between objects is limited to only one generic type.

Patterns can be grouped in families. A pattern family is visualized as an oval with two bottom lines. Families can also be shown using colored areas around the patterns.

Note that version 1.5 of the UML specification (Object Management Group, 2003) represents patterns as dashed ellipses. This notation is valid if the pattern is a secondary information in a class or an interaction diagram. But in pattern diagrams the patterns should be in the main focus. One can assume that many patterns are shown in one diagram. Using ellipses (like in use case diagrams) would consume too much screen space and using dashed lines would make the elements secondary compared to other elements like participants.
CoPE allows the reader to dynamically explore the relations between patterns. Nodes can be expanded or collapsed and thereby show more or less of the pattern’s context. An automated layout algorithm ensures that the layout of the pattern map stays consistent and clear.

Users can apply filters to the diagrams. One example is to filter all relations except the sequence relation, so that a reader can arrange the patterns in a linear sequence that can be read like traditional text. Filtering all but the uses relations will result in an acyclic graph that can be visualized in a layered way. We refer to this combination of the pattern graph with the filtering algorithms as interactive pattern map.

All nodes in the interactive pattern map link to the textual description of the pattern, the participant, or the known use. Clicking on the node shows the corresponding text in the right part of CoPE’s window.

**B.2 Example Diagram**

Figure B.3 shows the visualization of the Local Awareness pattern and its context.

![Local Awareness Diagram](image_url)

**Figure B.3:** The detailed context of the Local Awareness pattern.

The Local Awareness pattern is the centered colored node. In the lower part of the diagram, related patterns are shown. Thick arrows between the patterns indicate that a larger pattern uses a smaller pattern. The larger pattern is positioned on a higher level on the abstraction scale than the smaller pattern. Note that the diagram only shows some of the relations of the pattern language. The reason for
this is that the inclusion of all relations would make the diagram hard to read in a static version. In a dynamic version, the user can explore the relations by expanding or hiding related patterns.

The upper left part of the diagram shows known uses in which the pattern was applied. The upper right part of the diagram shows the different participants of the LOCAL AWARENESS pattern and how these participants interact. Note that other patterns also relate to these participants. By examining the use of the same participant in different patterns, the user can gradually understand the different design aspects for specific participants in groupware applications.
Appendix C

Tailoring Mechanisms in CURE

In order to provide the users with opportunities for tailoring the CURE environment, the environment has to provide tailoring access points. CURE provides tailoring at two levels (Bourimi et al., 2003).

At the structural level, users can reshape the environment (the room structure) in order to support different group structures and collaboration spaces. Users could for instance create a room for supporting a virtual seminar that has several sub-rooms:

- one room for each subgroup in which they can prepare their presentations,
- a cafeteria in which the participants socialize,
- a literature room in which all groups collect and share their literature,
- a speaker’s room in which the participants can rehearse their talk, and
- a presentation room for giving and filing presentations.

Users can prepare full room structures and share them with other users. For a concrete seminar, the users copy the template room structure and fill it with appropriate (seminar specific) content.

In a lower tailoring level, users can define how the pages in a room should look and behave. They can define fields that have to appear on each page using a template description language that is an extension of the WIKI syntax used to edit page content. The template description language provides elements for defining input fields and displaying entered values. The most important elements are:

**wikiTextInput** is used to define an input field. The template author provides an id and optionally the size of the field. Note that the template author can invent new ids. This empowers him to describe any domain specific objects with a template. The wikiTextInput fields can also be used to connect different templates. Imagine that a tailorer wants to provide a room in which patterns can be collected. He defines a pattern template with the needed input fields for describing a pattern in a PLML-like style. Then he decides that users
should be able to connect patterns with descriptions of known uses. Instead
of including the known uses on the pattern pages, the template author creates
a second template (the knownUses template) and decides that pages created
from the known uses field of the pattern template should automatically link
to new instances of knownUses pages.

The element would look like this:

```xml
<wikiTextInput id="knownUses" template="knownUses"/>
```

**renderedText** is used to visualize the content of a field. It translates the plain
text entered by the user to a html representation. Imagine that a user has
provided a field with the id *scenario* in his edit template. He can then use
the element

```xml
<renderedText id="scenario"/>
```

in his display template to show a formatted version of the text entered by the
user.

**unrenderedText** is comparable to the renderedText element with the difference
that no formatting is applied before displaying the content.

**wikiSelectUser** can be used to select a user from a list of users who can enter the
room. It can be used, e.g., on a page that describes user stories. The author
selects his name from the list of users in order to say that the story comes
from his personal context.

**wikiShowUser** takes the input from a wikiSelectUser field and transforms it an
**Interactive User Info**.

**wikiAppendText** can be used in the display template in order to provide fields
that can be appended by the users. In the case of the virtual seminar, one could
think of a positionStatement template that is used to collect the participants’
positions regarding the seminar’s topic. In a later phase, the students are asked
to comment each position. One way to support this process is to provide an
appendable field in the display template of the positionStatement. When users
see the page, they can comment it and the comments will be appended in the
field specified in the wikiAppendText element.

The example in chapter 6.5 used the following templates:

**Display Template**

```xml
<pageToolBar />
<box>
  <boxTitle>Details for the paper "<pageTitle />"</boxTitle>
  <box>
    <boxTitle>Cite as:</boxTitle>
    <box style="border-width:0px; background-color:#fffee;">
      __<unrenderedText id="author" /> (<unrenderedText id="year" />).
      ~~<pageTitle />.~~ In: <unrenderedText id="volume" />.
    </box>
  </box>
</box>
```

In the above example, the tailoror provided fields for the paper’s bibliographic attributes (author, title, date, ...), a link to the paper's source (the pdf file), and a student summary containing addressed research questions, used methods and results. The student can fill these fields when pressing the edit button of a literature page (cf. fig. C.1).

In the display view (fig. C.2), the fields of the template are automatically arranged. For instance, the bibliographic data is brought into the form required by
Figure C.1: Edit view of a literature summary.

diploma theses at the specific university department. In addition, the display template contains a field for commenting the page. Teachers as well as students can use this field to add a public assessment.
Figure C.2: Display view of a literature summary.
Bibliography


http://csis.pace.edu/~bergin/patterns/DoctoralPatterns.html


http://www.it.bton.ac.uk/staff/rng/CHI2K\_PLworkshop/PositionPapers/Borchers.pdf


[http://www.cutter.com](http://www.cutter.com)

[http://easycomp.info.uni-karlsruhe.de/~jcoplien/HarrisonCoplien.pdf](http://easycomp.info.uni-karlsruhe.de/~jcoplien/HarrisonCoplien.pdf)


[http://www.pi6.fernuni-hagen.de/publ/expertenfindung.pdf](http://www.pi6.fernuni-hagen.de/publ/expertenfindung.pdf)


gent Agents and Multi-Agent Technology. London, UK.


http://www.wupatterns.com/


http://www.bvu.edu/ctown/CHE.html


BIBLIOGRAPHY


http://www.ietf.org/rfc/rfc1094.txt


http://www.groupware-patterns.org/cscw2002


World Wide Web Conference (WWW7). Brisbane, Australia.


http://darkwing.uoregon.edu/~uplan/FAQPatternLanguage.html


http://www.welie.com/


Index

Abstraction, 7
ACTIVE NEIGHBORS, 126, 130, 135, 212, 216, 225, 241
Activity, 114, 123, 128, 151
ACTIVITY COUNTER, 216, 225
Activity flower, 190
Activity Log, 151
ACTIVITY STATUS BAR, 225
Ad-hoc customer, 81
Adaptive Software Development, 50
ADHOC MEETING, 126
Agile software development, 49
Alexander, 11, 17
APPLICATION SHARING, 226
ARE YOU ALIVE, 144
Artifact, 7, 114, 128
ASK THE AUTHOR, 226
ATTENTION SCREEN, 161, 195, 216
AUTHORITATIVE INSTANCE, 176, 226
AWARENESS PROXY, 212, 226

Backlog, 56, 179
in OSDP, 99
BELIEVE IN YOUR GROUP, 216, 227
BELL, 194, 216, 227
BIG JOLT, 167
BIRDS OF A FEATHER, 176, 199, 210, 216, 236
Bottom-up development, 41
Breadcrums, 68
Break, 10, 15, 101
in OSDP, 97
Breakdown, see break
BCSW, 155, 160
BUDDY LIST, 98, 144, 216, 227

Center, 26, 28

CENTRALIZED OBJECTS, 176, 213, 216, 228
CHANGE LOG, 156
CHANGE WARNING, 123, 133, 134, 140, 154, 161, 177, 188, 199, 213, 216, 228, 231
ChiPlace, 166
CLUSTR, 12
CMM, 50
COAST, 142
CoCoBrowse, 132
Collaboration scenarios, 82
COLLABORATIVE FILTERING, 228
Collaborative Learning, 175
COLLABORATIVE SESSION, 126, 141, 144, 216, 228
Collaborative Web research, 211
COLOR-CODED DISTANCES, 140, 228, 243
COMMAND, 156
Community, 1
of circumstances, 2
of interest, 2
of practice, 2
of purpose, 2
Conceptual iterations, 95, 175
Confocal user, 129, 137, 187
Context, 7, 10, 26
Contextual Awareness, 127
CONTROL YOUR PRIVACY, 210, 229
Coordination, 21
Crystal Methods, 50
CSCL patterns, 206
CSCW3, 132
CURE, 174
Expert finder, 210
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS, 124, 129, 154</td>
<td>279</td>
</tr>
<tr>
<td>Daily Report, 119, <strong>158</strong>, 188</td>
<td></td>
</tr>
<tr>
<td>Decomposition, 17, 41, 78, 98</td>
<td></td>
</tr>
<tr>
<td>Semilattice, 12, 13</td>
<td></td>
</tr>
<tr>
<td>Tree, 12</td>
<td></td>
</tr>
<tr>
<td>Democracy, 24</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
</tr>
<tr>
<td>Conservative account, 8</td>
<td></td>
</tr>
<tr>
<td>Pragmatic account, 8</td>
<td></td>
</tr>
<tr>
<td>Romantic account, 8</td>
<td></td>
</tr>
<tr>
<td>Design customer, 81</td>
<td></td>
</tr>
<tr>
<td><strong>DETECT A CONFLICTING CHANGE</strong>, 216, 227, <strong>229</strong></td>
<td></td>
</tr>
<tr>
<td>Development iterations, 98, 179</td>
<td></td>
</tr>
<tr>
<td><strong>DISTINCT AWARENESS INFO</strong>, 139, 140, 212, <strong>229</strong></td>
<td></td>
</tr>
<tr>
<td>Document Awareness, 127</td>
<td></td>
</tr>
<tr>
<td>Domain-language, 64</td>
<td></td>
</tr>
<tr>
<td><strong>DON'T DISTURB</strong>, 216, <strong>229</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DON'T TRUST YOUR FRIENDS</strong>, 216, <strong>230</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DOOR</strong>, 183, 192, 193, 216</td>
<td></td>
</tr>
<tr>
<td>Dynamic Systems Development Method, 50</td>
<td></td>
</tr>
<tr>
<td>Eclipse, 212</td>
<td></td>
</tr>
<tr>
<td>Edition, <strong>156</strong></td>
<td></td>
</tr>
<tr>
<td>Emoticon, <strong>230</strong></td>
<td></td>
</tr>
<tr>
<td>End-user development, 63, 105</td>
<td></td>
</tr>
<tr>
<td>End-user involvement, 24</td>
<td></td>
</tr>
<tr>
<td>Enterprize Unified Process, 47</td>
<td></td>
</tr>
<tr>
<td>Envy, 124</td>
<td></td>
</tr>
<tr>
<td>Ethnographic methods, 61</td>
<td></td>
</tr>
<tr>
<td>Ethnography, 85, 105</td>
<td></td>
</tr>
<tr>
<td>Ethnomethodology, 19</td>
<td></td>
</tr>
<tr>
<td>EuroPLoP, 214</td>
<td></td>
</tr>
<tr>
<td>Evaluation method, 171</td>
<td></td>
</tr>
<tr>
<td>Evaluation user</td>
<td></td>
</tr>
<tr>
<td>in OSDP, 101</td>
<td></td>
</tr>
<tr>
<td>Evolutionary growth, 84</td>
<td></td>
</tr>
<tr>
<td>Expert assessment, 174</td>
<td></td>
</tr>
<tr>
<td>Expertise tension, 53</td>
<td></td>
</tr>
<tr>
<td><strong>EXTERNAL AWARENESS VIEW</strong>, 188, 213, <strong>231</strong></td>
<td></td>
</tr>
<tr>
<td>eXtreme Programming, 51, 105</td>
<td></td>
</tr>
<tr>
<td>for Groupware development, 81</td>
<td></td>
</tr>
<tr>
<td>Feature-Driven Development, 58</td>
<td></td>
</tr>
<tr>
<td>Field experiment, 173</td>
<td></td>
</tr>
<tr>
<td>Field study, 173</td>
<td></td>
</tr>
<tr>
<td><strong>FIND THE GURU</strong>, 216, <strong>231</strong></td>
<td></td>
</tr>
<tr>
<td>Fitness, 11</td>
<td></td>
</tr>
<tr>
<td>Flashmob, 1</td>
<td></td>
</tr>
<tr>
<td><strong>FLOOR CONTROL</strong>, 177, 178, <strong>231</strong></td>
<td></td>
</tr>
<tr>
<td>Fly on the wall, 215</td>
<td></td>
</tr>
<tr>
<td>Forces, 11, 16, 101</td>
<td></td>
</tr>
<tr>
<td>Field of, 26</td>
<td></td>
</tr>
<tr>
<td>From Shared Data to Shared Work, 118, <strong>120</strong>, 127, 129, 176, 187, 212, 213</td>
<td></td>
</tr>
<tr>
<td>Fundamental properties, 27</td>
<td></td>
</tr>
<tr>
<td><strong>BOUNDARIES</strong>, 28, 28, 192, 193, 195, 196, 199–201</td>
<td></td>
</tr>
<tr>
<td><strong>CONTRAST</strong>, 28, 29, 192–194</td>
<td></td>
</tr>
<tr>
<td><strong>DEEP INTERLOCK AND AMBIGUITY</strong>, 28, 29, 194, 195, 198, 200</td>
<td></td>
</tr>
<tr>
<td><strong>GRADIENTS</strong>, 28, 29, 193, 195, 200</td>
<td></td>
</tr>
<tr>
<td><strong>LEVELS OF SCALE</strong>, 27, 28, 192, 193, 196</td>
<td></td>
</tr>
<tr>
<td><strong>LOCAL SYMMETRIES</strong>, 196</td>
<td></td>
</tr>
<tr>
<td><strong>NOT-Separateness</strong>, 28, 29, 193, 195, 198–200</td>
<td></td>
</tr>
<tr>
<td><strong>Simplicity and inner calm</strong>, 28, 29</td>
<td></td>
</tr>
<tr>
<td><strong>Strong centers</strong>, 28, 28, 193, 197, 199</td>
<td></td>
</tr>
<tr>
<td>GAMA Mall, 126, 139, 148</td>
<td></td>
</tr>
<tr>
<td>Gang of Four (GoF), 66</td>
<td></td>
</tr>
<tr>
<td>Gardener, 81</td>
<td></td>
</tr>
<tr>
<td>in OSDP, 100, 180</td>
<td></td>
</tr>
<tr>
<td><strong>GAZE OVER THE SHOULDER</strong>, 119, 123, 126, 129, 130, 133, <strong>145</strong>, 154, 155, 169, 187, 188, 210, 212, 213, 216, 226, 245</td>
<td></td>
</tr>
<tr>
<td><strong>GROUP AWARD</strong>, 216, <strong>232</strong></td>
<td></td>
</tr>
<tr>
<td>Group awareness, 114</td>
<td></td>
</tr>
<tr>
<td>Group formation, 113, 118, 120, 197</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Groupware, 3
Guess What I Need, 216, 232
Hall of Fame, 166–168, 210, 216, 232
HCHI, 3
Heidegger, 10
Hello Hello, 119, 163, 169, 176, 191, 205, 206, 210, 216
Holism, 26
I Am Alive, 144
I2I, 126
Immutable Versions, 233
In-Place Awareness View, 135, 212, 233, 237
Interaction, 115
Interactive User Info, 140, 199, 254
Interviews, 62
Introduction Session, 168
Invitation, 216, 233
Iterative development, 17
Iterative processes
  Piecemeal growth, 22
Iterative software development, 45
Knowing-in-action, 15
Laboratory experiment, 173
Letter of Recommendation, 176, 177, 210, 212, 216, 228, 233
Life, 12
Life Indicator, 190, 204
Lights off, 177, 216, 234
Lingua Franca for design, 68
Listener, 146
Literature Pool, 208
Local Awareness, 119, 122, 123, 126, 127, 135, 144, 146, 156, 187, 212, 213, 216, 225, 231, 232, 238, 242, 251
Lock and Key, 193, 194, 216
Locomotion, 216, 234
Login, 177, 181, 192, 193, 201
Lovely Bags, 216, 235
Mailing list, 160
Map, 184, 216
Marker, 235
Masquerade, 130, 148, 165, 167, 196, 216
Mediated Updates, 213, 216, 236
Meme, 30
Mentor, 167, 236
Metaphor, 53
Misfit, 11
Mode of collaboration, 120
Model-View-Controller, 156
Moderator, 177, 236
More than 1000 Words, 236
MSN Messenger, 142
NATO conference, 40
Navigation Observer, 149
NESSIE, 148, 154
Odigo, 132
Opposites Attract, 176, 236
Oregon experiment, 21
Organic order, 22
OSDP, 95, 171
  Conceptual iterations, 95, 175
  Development iterations, 98, 179
  Tailoring iterations, 101, 206
OTD, 82, 106
Participation, 21
  in XP, 53
  informed participation, 85
Inter-organizational level, 25
Social or working life level, 25
Work situation level, 25
Workplace or organizational level, 25
Participatory design, 21, 24, 60, 105
Pattern scout, 101, 209
Patterns
  Abstraction scale, 112
  Alexandrian form, 18
  Analysis patterns, 68
  Application, 22, 73
  as Memes, 30
  Behavioral, 21
  CSCL, 89, 206
  Ethnography, 85
INDEX

Evaluation, 23
Evolution, 23
Generic centers, 30
Groupware patterns, 106
HCI patterns, 68, 106
Level of scale, 19
Low-level, 89
Map, 17
Mathematical description, 17
Means for communication, 22
Morphological law, 17
Novelty, 70
Participatory design, 75
Pedagogical patterns, 206
Socio-technical, 87, 106
Software development, 65
Structure preserving transformation, 30
Usability engineering life cycle, 74
Pay Back, 210, 216, 236
PDSA, see Plan-Do-Study-Act
Personal Annotation, 237
Personalized Attributes, 237
Plan-Do-Study-Act, 16
Planning, 16
Planning game, 53, 181
in OSDP, 98
Platform 2000, 174
PLML, 71, 106
PLoP conference series, 89
PoliAwaC, 139
PoliTeam, 139
Presence Indicator, 119, 126, 133, 135, 154, 188, 210, 212, 216, 231
Privacy, 193
Prototyping, 62
Low-fidelity, 98
Proxy, 148
Proxy Object, 147, 155, 213, 237
Proxy-Logger, 148
Publisher-Subscriber, 134, 144
Quality without a name, 27
Radar View, 136, 238
Rational Unified Process, 45, 106
Read, Read, Read, 206, 206
Ready-to-hand, 10
Reciprocity, 110, 111, 148, 196, 216, 229, 235, 238
Refactoring, 52, 99
Reflection, 16, 55
Diagnosis, 22
Reflection-in-action, 16
Rehabilitation, 238
Remember to Forget, 212, 238
Remote Selection, 216, 239
Remote Subscription, 239
Remote Viewport, 216, 239
Replicate for Freedom, 216, 239
Replicate for Speed, 216, 239
Requirements
Education, see R10
End-User Development, see R13
End-User Involvement, see R9
Explicit Consequences, see R11
Explicit Forces, see R8
Field of Forces, see R7
Integration, see R12
Iterative Design, see R5
One Problem at a Time, see R6
Process Awareness, see R4
Quality Review, see R15
R11, 82, 90, 91, 104, 173, 179, 204, 209
R14, 85, 90, 91, 104, 178, 204, 209, 220
R10, 48, 55, 58, 63, 81, 84, 90, 91, 104, 173, 179, 204, 209, 213, 220
R9, 39, 42–44, 48, 53, 58, 81, 84, 90, 91, 104, 179, 204, 220
R13, 39, 48, 55, 63, 84, 90, 91, 104, 178, 204, 209
R8, 82, 84, 90, 91, 104, 179, 204, 209
R7, 82, 84, 90, 91, 104, 179, 204, 209, 213
R12, 63, 84, 104, 179, 204, 213
R5, 35, 39, 41–44, 48, 84, 103, 179, 204, 220
R6, 48, 55, 59, 84, 90, 91, 103, 179, 204, 213
**INDEX**

**R4**, 42, 48, 54, 80, 81, 84, 103, 178, 203, 209, 220

**R3**, 35, 42, 43, 48, 51, 54, 63, 80, 84, 85, 90, 91, 103, 178, 203, 209, 220

**R15**, 85, 90, 91, 104, 178, 204

**R2**, 52, 58, 84, 90, 91, 103, 178, 203

**R1**, 34, 84, 103, 178, 203

Reflection, *see* **R3**

Sharing of Best Practices, *see* **R14**

Structure-Preserving Transformations, *see* **R2**

Wholeness, *see* **R1**

**Requirements engineering**, 8

**Role**, 177, **240**

**Room**, 123, 144, **182**, 192, 194, 201, 216, 231

Room directory, 185

Room map, 185

Rugby approach, 32

Scenario interest group (SIG), 97

Scenario-based design, 61

**SCOPE**, 212

Scrum, 55, 105

Scrum Master, 56

**Semantic Distance**, 212, 225, 228, 240, **240**

**Semantic Net**, 212, 216, 231, 240, **240**, 241

**SER**, 84, 106

**Session Timeout**, **144**

Shared Annotation, 176, 177, **241**

Shared Folder, 176, 177, **182**

Shared Repository, **157**

Shepherd, 214

Show the Expert, 210, **241**

SIG, 175

Situatedness, 10, 28, 31, 113

**Spatial Domain Model**, **241**, 243

Spiral development, 45

**STEPS**, 64, 106

Stories in OSDP, 97

**Strong Center**, 199

Structure preserving transformation, 29

Swarm and Collect, 212, **241**

Tacit knowledge, 10

Tailoring, 63, 101

Collaborative tailoring, 82

Tailoring iterations, 101, 206

Talk First, 123, 126, 199, 231, 233, **242**

Telecursor, 216, **242**

Tell Me a Story, **242**

Test-first, 51

Testing

Diagnosis and repair, 22

Theoretical evaluation, 174

Threaded Discussions, 177, **242**

Time boxing, 54

Time Compressor, 132, 133, 212, 238, **242**

Timeout, 225, **243**

Top-down development, 41

Trade unions, 24

Travel Together, 177, 216, **243**

TUKAN, 125, 139, 155

**Unit of Work**, 225, **243**

Update Your Friends, 216, 240, **243**

User Gallery, 110, 111, 144, 166, 168, 191, **197**, 216

User List, 119, 134, **141**, 177, 187, 188, 191, 216

User story, 99

User-Centered Push (UCP), 148

User-evaluation customer, 81

V-Model-XT, 44

Virtual Me, 142, 144, 176, 177, 187, 197, 205, 206, 244

virtual project backlog, 180

Virtual Seminar, 175

Virtual seminar, 176

Visa, **200**, 216

VisualWorks Snalltalk, 154

Vote, 238, **244**

Waterfall method, 41

for Groupware development, 78

Welcome area, 164

**WELL**, 1

Who am I, 142, **191**
INDEX

WHO’S LISTENING, 210, 216, 245
WHO’S THAT GIRL, 147, 191, 245, 245
Wholeness, 26, 201
  Mathematical definition, 27
Wicked problem, 14, 54
Writer’s workshop, 89, 215

YOU ARE HERE, 245