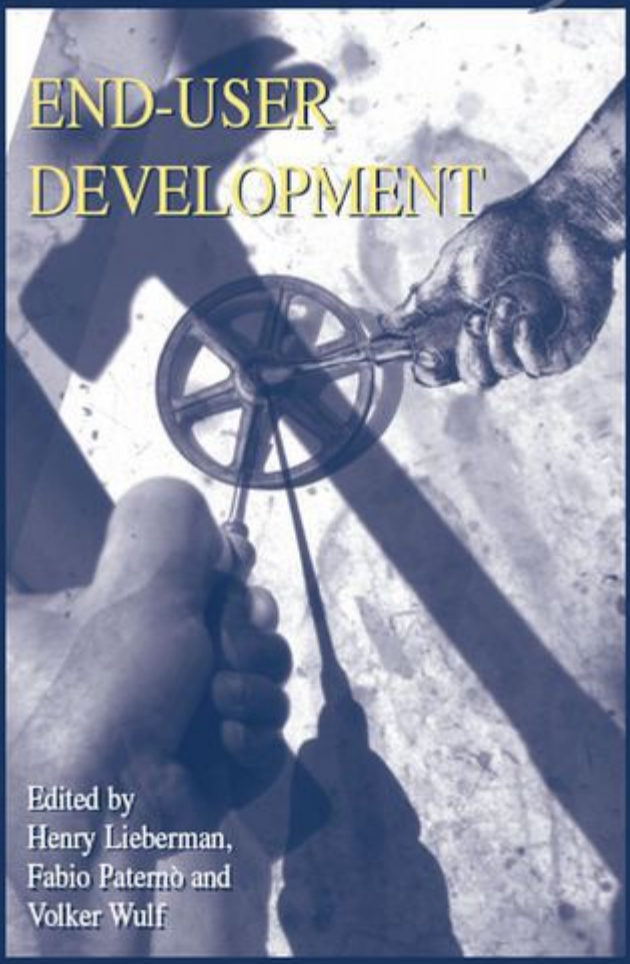
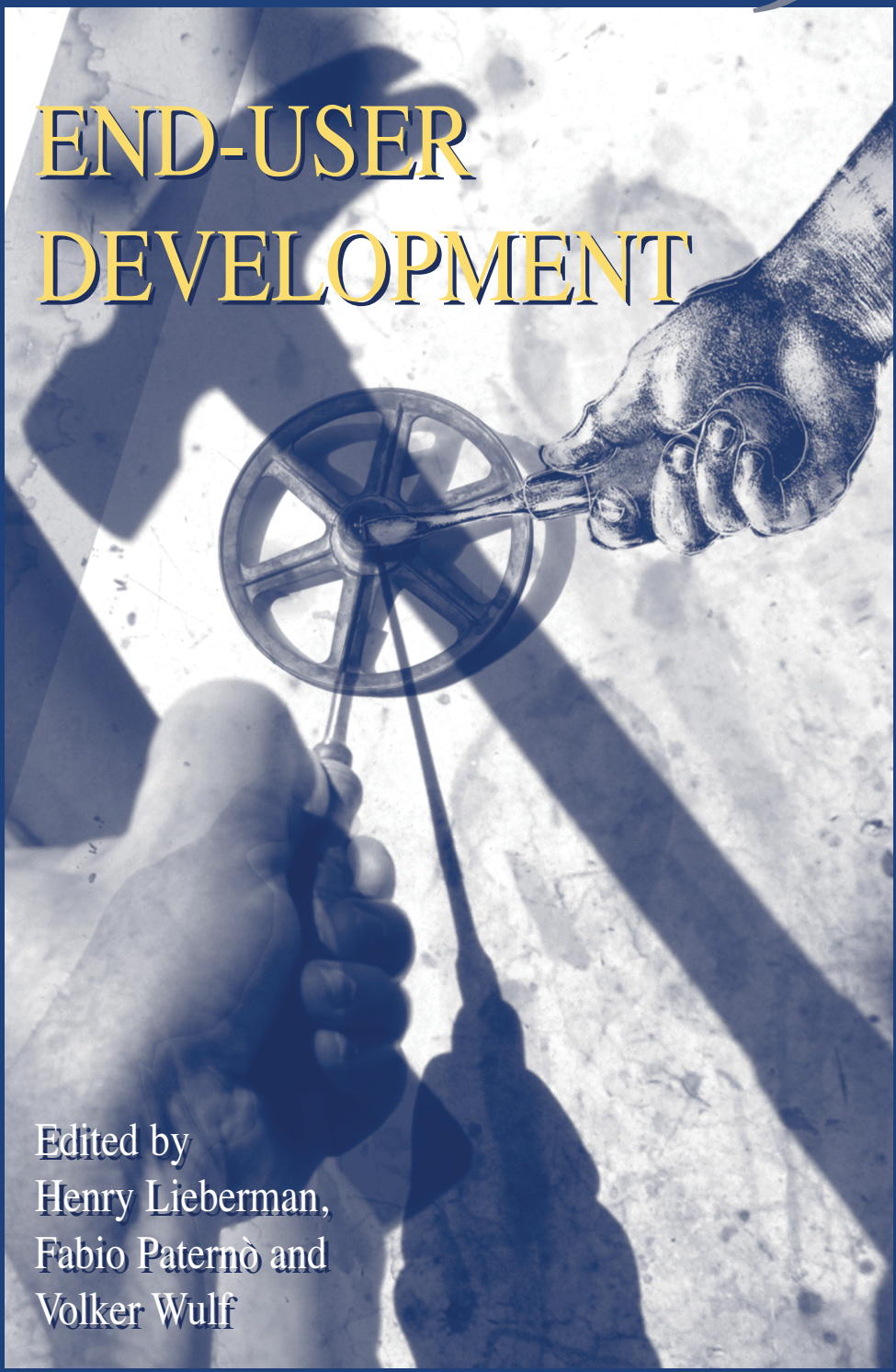
HUMAN-COMPUTER INTERACTION SERIES 9



Springer

HUMAN-COMPUTER INTERACTION SERIES 9



END-USER DEVELOPMENT

Edited by
Henry Lieberman,
Fabio Paternò and
Volker Wulf

Springer

END USER DEVELOPMENT

HUMAN-COMPUTER INTERACTION SERIES

VOLUME 9

End User Development

Edited by

Henry Lieberman

Fabio Paternò

and

Volker Wulf

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-5309-6 (PB)
ISBN-13 978-1-4020-5309-2 (PB)
ISBN-10 1-4020-4220-5 (HB)
ISBN-13 978-1-4020-4220-1 (HB)
ISBN-10 1-4020-4221-3 (e-book)
ISBN-13 978-1-4020-4221-8 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

First edition 2006. Secont Printing

Contents

Preface

Imagine, for a moment, that you hired a new assistant to work for you. He came highly recommended, and had a long and extensive resume of experience in the general area of what you wanted him to work on. The first day on the job, however, you find out that he is really set in his ways. If you want him to do something, you have to explain it precisely in terms of what he did on his previous jobs. He does every task exactly the same way he did in all his former positions. He doesn't have any interest in learning anything new. If he hadn't performed exactly the same task before, he is simply unable to do it. He can't accept any criticism or feedback on his performance. How useful would such an assistant be? I don't know about you, but I certainly wouldn't be happy about his job performance.

It's that way with almost all of today's software. So-called "applications" software for end-users comes with an impressive array of capabilities and features. But it is up to you, the user, to figure out how to use each operation of the software to meet your actual needs. You have to figure out how to cast what you want to do into the capabilities that the software provides. You have to translate what you want to do into a sequence of steps that the software already knows how to perform, if indeed that is at all possible. Then, you have to perform these steps, one by one.

And what's worse is, even if you succeed in doing this for a particular problem, it doesn't help you very much the next time you have a similar problem. You'll have to go through a similar sequence of steps again. Even if the system could combine or modify capabilities it already has to perform a new task, you can't do it. The best you can hope for is that enough other people will have the same problem so that the software company will include something to solve that problem in the next release. Why can't you extend the computer's repertoire of capabilities yourself?

What is sad about the whole thing is that we could do much better. While end-users often feel powerless in the face of inflexible and unforgiving software, that small minority of people who learn software development techniques perceive the capabilities of computers quite differently. To them, a computer is like the kind of helpful and eager assistant who is always willing to take on new challenges. If you teach the computer carefully, you can get it to put together things it already knows to be able to solve new problems. If you don't like what it does, you can always change it. If you want it to do something slightly differently, a few changes to the program—its "job description"—can get it done. For people who are knowledgeable enough, the process of software development gives them the empowering feeling that they can do practically anything. That's what's so seductive about computers to programmers. Problem is, the price of

entry—knowing the complex and arcane world of programming languages and tools—is, today, very high. But we can bring that price down. That's what this book is about.

Even if you don't believe in the dream of Artificial Intelligence enough to think that a computer could ever provide as much helpful assistance as a human assistant would, the argument still stands. No prepackaged commercial software can ever fully meet everyone's needs. Even when computers help humans with very simple and boring tasks, flexibility is still needed to deal with changing contexts. Details change from one job to the next, managers or customers are always asking for small changes in specifications, unforeseen situations occur. What we would like to achieve is the ability for end-users to construct or modify existing software themselves without "waiting for the next release".

AI holds out the promise of the ability to do some learning, adaptation and advice-taking at run time. Those capabilities would certainly be useful in enabling end-users to develop software, if indeed they are possible. But even if the end-users have to specify everything themselves without explicit help from the system, we hope to convince you that even the most simple capability for application programs to modify and extend their own behaviour from interaction with end-users could have an enormous impact on the usefulness of computers.

The vanguard target audience for End-User Development consists of two distinct communities of users. Interestingly, they fall at opposite ends of the spectrum of sophistication of computer use.

The first are beginning users. Today, beginning users start by learning how to use application software such as text editors, spreadsheets, browsers, etc. But if they want to do something even slightly different than what the software provides, they're stuck. Because their needs are casual, and they can't spend much money, companies are not motivated to provide specialized application software to do particular things that a beginning user might want to do. So they would well appreciate easy-to-use systems that allowed them to make or modify there own software. Some ideas that would make it possible for beginning users to write software without learning conventional programming languages, including visual programming languages, scripting languages and Programming by Example.

Some of these are discussed in this book. Some beginners will also eventually want to learn how to program as professional programmers do. But it is generally too difficult to begin to learn a conventional programming language such as C++ or Java, directly. So, ideas like visual programming languages or Programming by Example can be used as teaching tools. The beginner can first come to understand the fundamental concepts of programming, such as variables, functions, loops and conditionals and only later (if ever) deal with the messy details of programming languages and compilers.

The second group that is a major constituency for End-User development is professionals in diverse areas outside of computer science, such as engineering, medicine, graphic design, business and more, who are not professional programmers. These people need to get specific jobs done in their fields that might benefit by computer automation, but that are not common enough to warrant commercial development of applications

just for that purpose. An accountant needs to specialize a bookkeeping program to the idiosyncracies of a particular business. A physicist needs to make a specialized data collection program for a particular experiment. These users are very sophisticated in the fields of their expertise, but have little time or interest in learning a programming language or software engineering methodology.

The papers in this book span a wide variety of conceptual issues, technical topics, applications areas and experience. First, we begin with some introductory papers that survey the area, provide background and a taste of what is to come. Next, there are some contributions which draw on inspiration from the field of Software Engineering, which has long studied issues relating to the software life-cycle. These chapters try to present novel methods for EUD exploiting and enriching some concepts from Software Engineering. The following section shows some systems where End-User Development has been specialized to particular application areas or reports on some industrial case studies. The diverse application areas give an indication of the broad usefulness of End-User Development, and show the importance of user context.

To start off the introductory section, Alan Blackwell gives us the cognitive science, psychological and philosophical perspective in his "Psychological Issues in End-User Programming". His paper gives us insight into what is known about how people approach the cognitive skill of programming. He reviews the psychological impact of several popular End-User Development systems. And he provides us with a window onto the rich interdisciplinary literature relevant to this topic.

John Pane and Brad Myers, in "More Natural Programming Languages and Environments", continue on the theme of using studies of people to inspire End-User Development systems. Their approach of Natural Programming begins by studying how non-programming users describe a programming task, and analysing the results for what kinds of conceptual constructs are used. Only then do they design an End-User Development environment, HANDS, that embodies some of the principles discovered in the user studies.

Alexander Repenning and Andri Ioannidou, in "What Makes End-User Development Tick", deliver a set of guidelines for End-User Development environments born out of their vast experience with the AgentSheets environment. The article balances conceptual guidelines with concrete illustrations of applications built by users with this system, illustrating the principles. This report from the trenches of End-User Development gives a candid look at the promise and pitfalls of the field.

The next set of articles describes Software Engineering-based approaches and methods for EUD. Margaret Burnett, Gregg Rothermel and Curtis Cook's "An Integrated Software Engineering Approach for End-User Programmers" show us why End-User Development is about more than just programming. Their work takes place in that most common of End-User Development environments, the spreadsheet. Spreadsheets' success in giving end-users the ability to do programming with cell formulas shows that they must be doing something right, and Burnett's group gives us a full-blown End-User Programming environment based on the spreadsheet paradigm. They focus on providing testing and debugging facilities that draw users' attention to likely problems.

Their innovative "Help Me Test" feature provides mixed-initiative heuristic help from the system in an unobtrusive way.

In "Component-based Approaches to Tailorable Systems" by Markus Won, Oliver Stiemerling and Volker Wulf, they use the idea of plug-and-play "component" software modules to achieve End-User Development. The FreEvolve platform allows to (re-)assemble components at runtime. A graphical component diagram editor lets end-users visually connect components, allowing users to customize and develop new applications without going directly to code. The paper gives an account on a long term research effort presenting experiences with different types of graphical editors as well as features which support users in connecting components appropriately.

Silvia Berti, Fabio Paterno and Carmen Santoro, in "Using Conceptual Descriptions to Achieve Natural Development of Nomadic Applications" show an End-User Development environment oriented to the currently hot topic of "nomadic" applications, those that are accessible through a variety of devices, including wireless devices, that are distributed geographically. Often, the problem in developing such applications is to make it so that they will work under a wide variety of user contexts. Applications may run on different platforms, with different interfaces, possibly restricted by small screens and different input methods.

In "End User Development of Web Applications", Jochen Rode, Mary Beth Rosson and Manuel A. Pérez Quiñones provide us with a reality check on the activities and needs of present-day Web developers. Since the Web is such an important platform, it is instructive to see such a careful survey of what today's Webmasters actually do and how End-User Development might fit into today's Web engineering environments. Beyond particular Web technologies, there is focus here on the mental models adopted by both professional developers and non-professional users for Web applications, and understanding how End-User Development might support and extend those models.

Costabile, Fogli, Mussio and Piccinno present an environment that allows domain-experts to modify their applications for their needs in "End-User Development: The Software Shaping Workshop Approach", by analogy to the workshops used by artisans and craftsmen. They illustrate their approach by an analysis of problem solving in a medical domain, looking at the communication between a radiologist and a pneumologist (lung specialist) cooperating in a diagnostic task.

While there are a number of projects that aim at general-purpose End-User Development, sometimes one way to make a more effective development tool is to specialize the environment to a particular application area or category of users. In the next section, we explore some End-User Development environments that have been constructed with specific application users in mind, and provide facilities that have been thoughtfully customized to the needs of that class of users. We also report on some interesting industrial case studies.

Catherine Letondal, in "Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users", uses a participatory design philosophy to understand the needs of biologists. She presents a discussion of the issue of when programmability is needed and what kind of programmability is best for professional scientists in

fields other than computer science. She then proposes the Biok environment, which provides some End-User Programming in a gene sequencing application. This application is basically a pattern-matching task that requires programmability for semi-repetitive tasks. Her biologist users can be considered typical of a wide range of scientific users in areas other than computer science. She shows how providing a vocabulary and operations well-suited to the users' tasks facilitates their problem-solving ability.

The computer revolution is now filtering down from personal computers to consumer electronics and appliances, and Boris de Ruyter and Richard van de Sluis give us, in "Challenges for End-User Development in Intelligent Environments", some preliminary thoughts on how End-User Development might impact the world of consumer electronics. It holds the promise of liberating us from an unruly tangle of buttons, switches, modes, flashing lights and complex remote controls as consumer electronics increases in sophistication.

Yasunori Harada and Richard Potter offer us a particularly innovative EUD approach to interactive graphic applications such as games, based on "Fuzzy Rewriting". Systems like Alex Repenning's Agentsheets and Allen Cypher and David C. Smith's Stagecast Creator have showed that even young children can effectively use a Programming by Example system based on rewriting rules. But insisting on exact matching of rule conditions puts a damper on the generality of such systems. Harada and Potter show how relaxed matching conditions can get these kinds of systems "out of the box" and dramatically extend the scope of applications possible with them.

Stevens, Quaisser and Klann apply the component-based approach in an industrial case study. While realizing a highly tailorable access control module by means of the FreEvolve platform, the authors had to break the application down into components which could be understood and manipulated by end-users. The paper demonstrates, how such a modularization can be obtained by using ethnographic methods and design metaphors. The ethnographic study helps to capture tailoring needs within the application context while the selection of the design metaphor helps to define components which are meaningful for ordinary users.

Yvonne Dittrich, Lars Lundberg and Olle Lindeberg's article, "End-User Development as Adaptive Maintenance", reflects the reality that what seems to be small, routine maintenance changes sometimes escalate to the point that they really become development of a new application. Rather than bemoan the lack of clear separation between maintenance and tailoring of applications, they look for new ways to take advantage of it, including an innovative use of the meta-object protocol in object-oriented languages.

End-User Development at the workplace has its particular organizational and social aspects. The activity to "tailor" an application to fit the diverse and changing use situations has been addressed in its effects on software architecture as well as application interfaces. Volkmar Pipek and Helge Kahler turn toward the collaborative aspects that can be encountered in practice, and give an overview on different approaches for "Supporting Collaborative Tailoring". Two studies with prototypes supporting collaboration in End-User Development activities at the workplace are described in more detail, and open-up a perspective on "appropriation support" as a category of functionality that aims at visualizing and sharing use cultures among end-users.

Stefania Bandini and Carla Simone deal with collaborative EUD from the angle of component-based software development. In an empirical case study the authors explore the cooperation in a company which develops software by means of component technology. Different types of artifacts are used as boundary objects to represent organizational knowledge about software components. These artifacts help actors who do not have experience in programming to understand the qualities of given sets of components. Based on the findings of the case study, Bandini and Simone discuss how similar artifacts could help cooperative EUD which is understood here as a creative discovery and integration of off-the-shelf components.

The perception of End-User Development in organizations today is also the subject of Darren Lee, Nikolay Mehandijev and Alistair Sutcliffe's article, "Organisational View of End-User Development". They present a detailed survey of management viewpoints on the issue. Though as End-User Development systems are evolving, these perceptions are likely to change rapidly, their paper gives a here-and-now look at what private and public organizations are thinking. They treat the issues of motivation to adopt it, control, and risk issues. The article is likely to be useful for managers considering adopting End-User Development, as well as for innovators seeking to understand the adoption path for new technology.

The final section takes us to a set of more reflective and speculative articles, pointing the way to future directions in the field. Sometimes, interdisciplinary progress can come from synergy with another academic field that, at first, is seemingly unrelated. Clarisse Sieckenius de Souza and Simone Barbosa, in "A Semiotic Framing of End-User Development", take us on a journey to the field of semiotics, the study of the meaning of symbols.

Gerhard Fischer and Elisa Giaccardi present their manifesto, "Meta-Design: A Framework for the Future of End User Development". They see End-User Development as another species of design, where the artifacts being designed are themselves interfaces for designing—hence, meta-design. They urge us to apply many known principles of good design, both in the human and machine ends of the equation.

Henry Lieberman and Hugo Liu, in "Feasibility Studies for Programming in Natural Language", chase the Holy Grail of using natural language itself as a programming interface, reducing dependence on error-prone formal programming languages as a medium for human–machine interaction. While they don't claim to have reached the point where we can simply talk to a computer, they do present some feasibility studies based on John Pane and Brad Myers' Natural Programming experiments, where they asked non-programming users to describe programming tasks. Lieberman and Liu aim to show that, perhaps, this dream might not be so crazy, after all.

And to conclude, Markus Klann, Fabio Paterno and Volker Wulf in "Future Perspectives in End-User Development", outline some of the most promising areas for future research. They develop a roadmap on how to proceed.

By presenting overviews, specific applications areas, implemented systems, industrial experience, conceptual frameworks and exciting new directions, we hope to convince you, the reader, that End-User Development is an idea whose time has come. We

hope to see the day where a computer isn't just a set of pre-packaged applications, but a set of capabilities, to be shaped according to the users' own needs and desires.

One of the major contributions of this book is bringing together people interested in End-User Development from Artificial Intelligence, Software Engineering and other perspectives. The field of Software Engineering has traditionally been hostile to working on systems that make programming easy to use for beginners and non-programming professionals. The focus of traditional software engineering is industrial software projects involving large teams of programmers and analysts where the primary concerns are reliability and efficiency. In those systems, you don't want make it too easy for individuals to introduce risky changes, so they mandate precise software development processes involving many reviews and verifications. But this is beginning to change, as the need for more flexibility in software debugging and software evolution is felt, even in traditional industrial settings. Conversely, even beginners can benefit by using some more structured software design, modification and testing approaches that are inspired by software engineering.

The word "developer" is a strange word to use for someone who writes software. Dividing people arbitrarily into "users" and "developers" is an artificial distinction that should be eliminated. We all develop, mentally and spiritually, each in our own way, every day. Let's get our computers to help us do it.