

End Users at the Bazaar: Designing Next-Generation Enterprise- Resource-Planning Systems

Christian Dörner, Sebastian Draxler, Volkmar Pipek and Volker Wulf,
University of Siegen

Studying changing software architectures from an end-user development perspective inspires an ERP (enterprise resource planning) architecture that lets end users create their own solutions.

The “bazaar,” a notion Eric Raymond coined, is a well-known metaphor for a software engineering model that developers often use in open source projects.¹ In this model, software development is in view of the public, which means that anyone has access to the source code and can change, improve, test, and use it. Open source projects such as Firefox or the ADempiere ERP Business Suite have produced high-quality results, indicating that bazaar-like software engineering models are effective.

The bazaar has three major advantages over traditional approaches, which Raymond called the “cathedral.” First, anyone can see what happens by viewing the source code, feature requests, and bugs. Second, anyone can join the bazaar by requesting new features, reporting or fixing bugs, or contributing source code. It’s even possible to reuse existing code and fork a new branch of the project. Third, anyone can participate in negotiating conditions, such as which features to integrate, which bugs to fix first, and which technologies to use.

Despite the bazaar’s proposed advantages, being at one isn’t always a blessing. Real-life bazaars are complex, with narrow, winding streets making it difficult to get an overview and find directions. This complexity is why customers sometimes can’t find the products they desire.

This complexity also applies to software. Consider Julia, who’s responsible for ordering raw materials in a production company. She regularly must calculate how much material to order. Her ERP system provides a variety of functions to help her achieve her goal. However, she can’t understand

the available functions or connect them to create a solution fitting her needs. Her problems arise owing to a lack of specific design tools for end users that would not only support the search for functions but also help users understand and connect them. Here, we address the challenge of building a tool that lets Julia create a solution.

End-User Development and Prosumers

superscript

End-user development (EUD) aims to enable end users “at some point to create, modify, or extend a software artifact.”² Enabling end users to program has a long research tradition, so the ideas behind EUD are well established.³ Because the estimated number of end-user programmers is steadily increasing, supporting end-user programming will become increasingly important. According to Christopher Scaffidi and his colleagues, by 2012 “over 13 million [end users in American workplaces] will describe themselves as programmers.”⁴

This development means a new actor must join

the bazaar: the *prosumer*. A prosumer serves as producer and consumer. If appropriate, the prosumer produces the goods he or she and, in some cases, others consume.⁵ Prosumers have in-depth domain knowledge that's invaluable and usually inaccessible for producers. This knowledge gap impedes producers from designing products that fit market niches. Prosumers possess a key feature that Raymond cited as making the bazaar of professional developers successful: "Every good work of software starts by scratching a developer's personal itch." However, the prosumer is nonprofessional, having neither the knowledge nor the tools and skills to create software the way professionals do. Nevertheless, the right EUD tools can unleash this potential, giving prosumers the opportunity to scratch their itches.

ERP Systems and the Bazaar: Breaking Up the Monolith

ERP systems have evolved from manufacturing resource planning (MRP II) systems to some of the most complex applications, highly generalized to meet requirements in many contexts.⁶ Technically, ERP architectures are advancing from monoliths to the bazaar, because the big players, SAP and Oracle, switched to service-oriented architectures (SOA), making it easier for third-party vendors to program add-ons. However, because ERP providers still control technology, it appears as if the bazaar construction is taking place inside a cathedral rather than out in the open. ERP providers offer reliable platforms, such as SAP NetWeaver, and other service providers can offer complementary solutions on top of these platforms for market niches.⁷

SOAs shorten software production cycles because of an increased reuse of services and decrease costs for individuals because of a higher flexibility. Software vendors now can provide goods for larger market niches. Smaller market niches still remain, however, because they require a specific domain knowledge not easily acquired. Consequently, SOA isn't sufficient for achieving our goal because it provides a space and booth at the bazaar but not the necessary development tools for prosumers. Currently, composition tools are available that let only professional developers reconfigure SOAs comparatively easily by manipulating service compositions.

Various accepted design principles aid the design of development tools for end users. Considering EUD research, we developed a business-process design environment that empowers end users to adapt and develop ERP systems to fulfill their needs.

Empowering End Users to Become Prosumers

In developing our design environment, we applied a user-centered approach that case-based prototyping inspired.⁸ Development comprised three steps:

1. understanding the cathedral's architectural problems (theoretically and empirically),
2. using the results from step 1 to design and develop suitable tools for potential prosumers, and
3. evaluating the tools to determine their effectiveness and usefulness.

The approach closely involved end users of small and medium enterprises (SMEs)—that is, enterprises with fewer than 250 employees and an annual turnover not exceeding 50 million euros.

Problems of the Bazaar in the Cathedral

First, to understand and identify existing problems as well as to experience them empirically, we went to the cathedral (a monolithic ERP system). In our interview study with employees from five SMEs, we set out to explore ERP users' application domain and understand their software-related problems. To develop a rich view, we selected 18 participants from the software, automotive, luggage, and craft industries, including, for instance, managers and employees from financial accounting, purchasing, production planning, and human resources.

We conducted an exploratory interview study with semistructured interviews that let us grasp the interviewees' subjective views. Two of our team members carried out the interviews, taking from 45 to 60 minutes per interview and recording them with a digital voice recorder. To keep users in their familiar work context, we conducted all interviews at the company sites. We then transcribed all the interviews in an abbreviated form so that we could create answer categories.⁹

In this article, we're particularly interested in functional problems, such as interoperability problems and missing or inappropriate functionality. The following case exemplifies this problem class. The user couldn't create a production plan in the SAP system (SAP R/3, 4.6c), because this functionality wasn't available and customization demanded too much knowledge. So, the user started to create her report by collecting the data in the SAP system and exporting it to a spreadsheet to do the necessary calculations for creating the plan:

Pullquote goes here. About 14 words.

Pullquote: Prosumers have in-depth knowledge that's invaluable and usually inaccessible for producers.

INSERT PULL QUOTE HERE.

Pullquote: The need arises for user-centric approaches that let end users model or adapt business processes.

We like to have a production list directly out of SAP, to avoid exporting to Excel. Our current solution is quite complicated. I have to create a template in SAP, containing only the information I need from the list, to keep the file small. Then I have to export it [to Excel]. Then I have to sort the data and insert them into a pivot table. This pivot table has to be edited to make the user interface understandable for people working in the production. ... We have to work together with a consultant to solve this issue in SAP. ... It would be great if we could make it by ourselves [without the consultant]. ... I have to create such a list it twice a week. ... Each time I need approximately one hour to create it [with Excel]. ... [A] small thing would help us, but SAP doesn't support it. ... It has to be programmed.

The analysis of this example, which we translated from German to English, reveals three major problems. First, applications inside (SAP) and outside (Excel) the cathedral lack interoperability. Second, end-user developers lack process support tools that let them automate creation of such reports. Third, experts or consultants are often the only people who can perform software adaptations in cathedral-like development.

The SMEs we studied still used the classic cathedral-like ERP systems and were just beginning to upgrade their systems to the new SAP versions (based on SAP NetWeaver), which we characterized previously as a “bazaar within the cathedral.” With this upgrade, the lack of interoperability will become obsolete because an SOA environment eliminates the technical heterogeneities. However, the second and third problems remain, so we must analyze SOA theoretically to design an appropriate solution based on this technology.

SOA's creators designed the concept to be machine processable. Applications should select and integrate services automatically during runtime on the basis of constraints. We introduced the end-user perspective on this architecture—a perspective that requires fundamental changes. Implementing SOA-based process-support functions is problematic even for professional developers because the service interfaces are only technical and don't provide nonfunctional information, except for programmer documentation. So, the service interfaces pose usability challenges for programmers, such as understanding long identifier names.¹⁰ Additionally, appropri-

ate mechanisms to discover services and understand their functionality are lacking. These difficulties are more significant for end users because they're nonprofessionals with fewer technical skills than professional developers.

Visual process-modeling solutions, such as those IBM and SAP offer, demand both domain expertise and advanced computer skills. So, process modeling remains inaccessible for end users having domain expertise but only limited computer skills. The need arises for user-centric approaches that let end users model or adapt business processes. In contrast, many popular mashup tools are available on the Web and let nonprofessionals model processes. That is, the tools, such as Yahoo Pipes and Microsoft Popfly, let them combine data and services to create new information-retrieval applications. In the ERP context, composition tools such as IBM WebSphere Business Modeler and SAP Visual Composer seem outdated compared to recent innovations such as mashup tools.

Designing and Developing Prosumer Tools

From the previous analysis, we developed SiSO (*Simple Service Orchestration*), a business-process design environment for end users. Development followed the *participatory design* approach,¹¹ involving three users who had participated in our empirical study. Figure 1 shows the mashup-inspired SiSO GUI. The logic and data tiers are different, because mashup tools often extract the data from Web sites and don't use SOAP for message exchange. Furthermore, mashup tools aren't designed to output Business Process Execution Language (BPEL) code.

SiSO's functionality covers three primary aspects: search for services, their community-oriented annotation and documentation, and their orchestration. The main idea is to open SOA for end users, in effect turning abstract representations into application units.¹² We achieve this by enabling users to understand the functions a service offers and by empowering them to change the documentation of service descriptions stored in UDDI and the Web Services Description Language (WSDL). This is particularly novel because the documentation is seamlessly integrated in the modeling environment, and business domain users can easily contribute to it, similar to Web 2.0 applications such as wikis. Because this documentation is tied to a specific service, context-based help systems such as CHiC (Community Help in Context)¹³ are needed to provide higher-level conceptual documentation.

The GUI has three parts: *search*, *information*, and *modeling*. The search part provides an easy interface with which users can search for services and functions by name, keyword, or description. A service's functions are of particular interest because they provide the specific functionality. In contrast, a service is only a structural element that bundles together several functions. By providing further details about the query results, the search returns a list of services and functions that resemble Web search engine results. Users can select functions from the list and add them to the toolbox at the bottom. The search system has three advantages over existing solutions. First, it considers additional nontechnical descriptions. Second, it returns not only the services but also their functions. Third, it processes user feedback, which influences the result list's order. The system presents the results' scores separated by both user rating and computed rating on the basis of the entered keywords.

The information part provides more comprehensive information about the services and their functions. Taken from the service interface descriptions (WSDL), this part includes explanations about a selected service, its ports, thereby aiding user understanding. The data comprises nontechnical information, such as service descriptions and the service's functions or keywords, as well as technical information, such as the parameter names and their data types. The information part lets users read and directly edit this information. Because the search algorithm uses this information, user-created documentation of services and their functions can improve the search results.

The modeling part lets users connect the palette functions to create a process. Functions are shaped as boxes, a metaphor researchers have already evaluated in component-based approaches to EUD.¹⁴ To ease function wiring, icons depict the input and output ports according to the port's data type. Colors further distinguish the ports: yellow for input and blue for output. Mouse-over popups provide additional information about the ports from the data the GUI displays in the information part. The system also uses a simple syntax check, preventing users connect an output port with an output port, connect an input port with an input port, or connect ports with different data types. The implementation's power is limited compared with business-process-modeling tools for professionals, but its strength is its low cognitive demands that let end users easily use the system.

SiSO's architecture is on top of an SOA, which

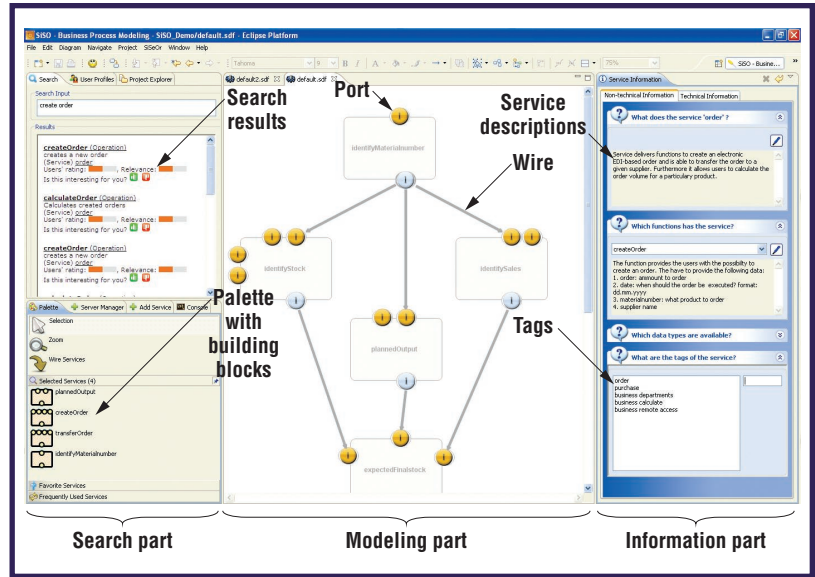


Figure 1. The SiSO (Simple Service Orchestration) GUI. The mashup-inspired GUI has three parts: search, information, and modeling.

uses Web services (WSDL, SOAP, and UDDI) for its implementation. Any ERP system, which is accessible over TCP/IP-based networks, could provide the services. SiSO has two major components: server and client. The server provides basic abstraction mechanisms from the Web service technology and enhances it by creating the possibility to store additional, nontechnical metadata in the WSDL files and the UDDI. This feature is conceptually new to SOA in that it supports service descriptions by those other than the service providers. The client component has three sub-components: business process management (BPM), search, and GUI. The BPM component is based on a universal component and container model, which enables BPEL process creation. The search component implements a three-level search algorithm, starting with a keyword-based query of the Web service descriptions. In the second step, it computes which services are similar to those in the first step. In the last step, the search component takes user profiles and feedback information into account to personalize the results. The GUI is based on the Eclipse Rich Client Platform, using an Eclipse Modeling Framework model and visualized through the Graphical Editing Framework.

Tool Evaluation

We evaluated the tools through a case-based lab study, addressing the work of the practitioners who participated in our prestudy. We sought to answer these questions:

- How effectively do the users perform tasks in SiSO?
- How do users work with SiSO? What are its

**INSERT Pull
Quote here.
About 14 words
long. Thanks!**

**Pullquote: Users
said the system
could add value
in visualizing
business
processes.**

strengths and weaknesses?

- What do users think about SiSO's usefulness? Does it add value to their work?

The evaluation took place in a computer laboratory at our university. We simulated a service-oriented ERP system by deploying self-created Web services (approximately 10, each having up to three functions) to an application server. We then registered these services in SiSO, representing the necessary functionality for the scenario. We annotated all the services with additional, nonfunctional metadata—specifically, short descriptions of services, functions, and ports. Figure 1 shows a representative example.

We combined the thinking-aloud method and participatory observations and selected six employees from the SMEs that participated in the prestudy. We asked them to use SiSO to accomplish two tasks in a work scenario:

- model a purchasing process, which was similar to a scenario we discovered as part of their work practice in the prestudy, and
- annotate any service.

While performing these tasks, users had to think aloud as a researcher asked open-ended questions concerning information that was incomprehensible and what they liked and disliked. After they accomplished the tasks, we conducted short, semistructured interviews with each participant to address the usefulness beyond technical and graphical design. On the basis of the users' statements, we identified critical-use situations, overall design issues, the necessary knowledge and organizational issues for system deployment, and improvement recommendations.

To varying degrees depending on their prior experience, all users effectively performed the tasks. All four users who had at least a little experience with business processes solved both tasks in approximately 40 minutes. The two other users had some problems with the tasks, showing that a basic understanding of business processes is an important precondition for using the system.

To find out how they could locate services and model a process, all users had to explore the system for some time. They perceived the system as "well arranged" and easy to understand but struggled with technical names and icon representations of services, functions, and ports. Users demanded validity checks and access to live data to see whether the process worked like they expected. Finding the right service was also difficult because neither the

services' purpose nor the kind of data the service delivers was always clear.

Users had different perceptions of the system's usefulness. Some said they could use the system for organizing complex calculations, such as determining order quantities and data aggregation. Others perceived the system as useful in any context in which processes recur, such as production. Users said the system could add value in visualizing business processes, as well as automating and modeling processes containing data from different systems. They said the system would let them do more tasks on their own, without needing a consultant. Users further mentioned prerequisites for deploying the system in their organizations: improved service descriptions, the ability to use services from existing systems (SAP, Excel, and so on), available data filters, and reliable data from the services.

ERP Systems: From the Cathedral to End Users at the Bazaar

Our work implements aspects of Joerg Beringer's vision of ERP systems consisting of high-level building blocks that enable end users to create solutions by combining existing entities.¹⁵ Studies in real-world settings are necessary to further evaluate our system in a long-term study—for example, to see if and how users will create domain-specific annotations of services. These annotations are also necessary to guarantee a high quality of the search results.

Like any other user-created software artifact, compositions of services can be error prone.³ However, an abundance of research already focuses on designing quality support tools, such as debugging and testing tools for end users.¹⁶ We'll have to integrate some of these tools into SiSO to ensure high-quality designed processes and provide end users the testing functionalities they demanded in the evaluation. Interesting approaches include Whyline,¹⁷ which lets users ask "why did" and "why didn't" questions, or the WYSIWYT (what you see is what you test) methodology¹⁶ and exploration environments,¹⁴ which help users understand their adapted applications' dynamic aspects.

Current ERP architectures advance to the bazaar having begun to build a bazaar in the cathedrals. Nevertheless, these approaches haven't yet sufficiently supported end users' "presumption." So, we envision a new kind of ERP architecture (see Figure 2) that will overcome those problems and increase ERP systems' adaptation potential, letting end users create innovative solutions. In the

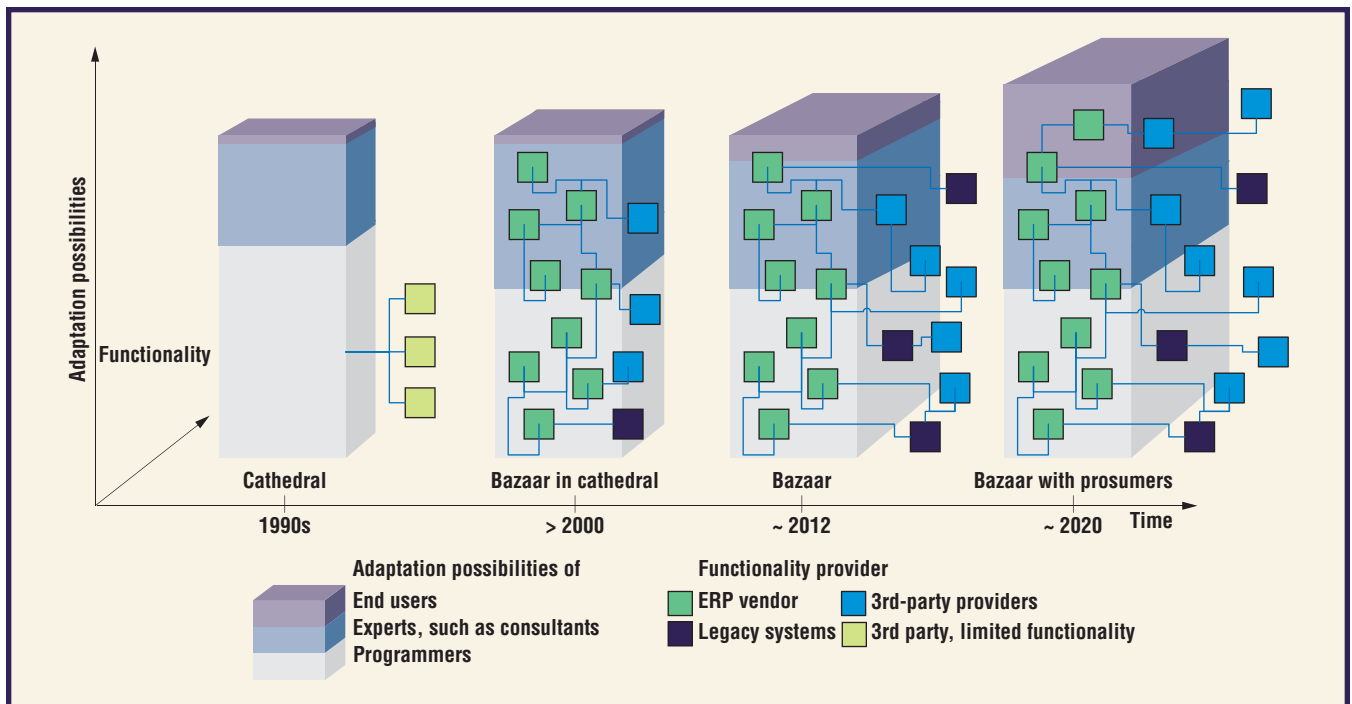



Figure 2. ERP system architecture challenges. A new kind of ERP architecture will address end-user “prosumption” and increase ERP systems’ adaptation potential.

1990s, ERP architectures were cathedrals (monoliths), offering few opportunities to change systems or extend their functionality—for example, by charging consultants. Often, they offered only export functions, which allowed data processing in other applications. After 2000, the architectures became service-oriented, moving the bazaar into the cathedral, or software ecosystem. This step increased systems’ adaptation possibilities for experts and let third-party vendors more easily extend functionality and access previously hidden internal ERP functionality. By the year 2012, ERP systems will become bazaars, as software-as-a-service and cloud-computing approaches—which use the Internet as a common and open technology platform—become more popular.

Programming prosumers will always find individual approaches to the level of professionalization they try to achieve. The SMEs participating in our research revealed numerous delegation patterns among prosumers, patterns that resulted in collaborative approaches to tackle everyday problems with ERP systems. Taking the prosumer seriously—as a sustainably nonprofessional programmer—might also require more profound changes to software architectures. Given the SiSO example, comments, explanations, and representations that end users might need to produce and consume are a new type of metadata stored in a service’s WSDL description. Scenarios involving larger organizations and correspondingly larger amounts of metadata might force us to find differ-

ent server structures that can deal with this information without losing bandwidth for the services themselves.

The evaluation showed that the distinction between application logic and interface, which is deeply embedded in current programming infrastructures and has been criticized before,¹⁸ isn’t always appropriate for prosumers. Visible and invisible software structures must be more congruent for end-user programmers: the architecture must handle the service description and service user interface as a single unit.

With appropriate prosumer programming paradigms and flexible service architectures supporting prosumer collaboration and the associated metadata traffic, ERP architectures by 2020 will create a level playing field for software vendors and prosumers in organizations. The resulting new division of work also must be reflected in new approaches that mirror the nature of large software systems as infrastructure.¹⁹ With the bazaar extending into the workshops and homes of people who formerly had to visit it, software confirms its role as a unique matter that allows a new division of labor in the knowledge society. 

Acknowledgments

The German Federal Ministry of Education and Research (BMBF) funded this research under project

About the Authors



Christian Dörner is a PhD student at the University of Siegen. His research interests include end-user development, service-oriented architecture, and human-computer interaction; his PhD thesis covered business-process-modeling tools for end users. Dörner has a Diploma in information systems from the University of Siegen. Contact him at christian.doerner@uni-siegen.de.

Sebastian Draxler is a PhD student at the University of Siegen. His research interests include the appropriation of the Eclipse ecosystem, human-computer interaction, computer-supported cooperative work, participatory design, and agile software development. Draxler has a Diploma in information systems from the University of Siegen. Contact him at sebastian.draxler@uni-siegen.de.



Volkmar Pipek is an assistant professor of collaborative systems at the University of Siegen's Institute for Information Systems. His research interests include end-user development (in service-oriented architectures and ubiquitous computing) and complex software infrastructures, such as enterprise resource planning and emergency-response systems. Pipek has a PhD in information processing sciences from the University of Oulu. Contact him at volkmar.pipek@uni-siegen.de.

Volker Wulf is a professor of information systems at the University of Siegen, and head of the User-Oriented Software Engineering research group at the Fraunhofer Institute for Applied Information Technology. His research interests include human-computer interaction, end-user development, computer-supported cooperative work, participatory design, and organizational computing. Wulf has a Habilitation degree in computer science from the University of Hamburg. Contact him at volker.wulf@uni-siegen.de.



EUDISMES (End User Development in Small and Medium Enterprise Software Systems), grant 01 IS E03 C.

References

1. E.S. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly, 2001.
2. H. Lieberman, F. Paternò, and V. Wulf, eds., *End-User Development*, Springer, 2006.
3. M. Burnett, "What Is End-User Software Engineering and Why Does It Matter?" *Proc. 2nd Int'l Symp. End-User Development (IS-EUD 09)*, LNCS 5435, Springer, 2009, pp. 15–28.
4. C. Scaffidi, M. Shaw, and B. Myers, "Estimating the Numbers of End Users and End-User Programmers," *Proc. 2005 IEEE Symp. Visual Languages and Human-Centric Computing (VLHCC)*, IEEE Press, 2005, pp. 207–214.
5. E. von Hippel, *Democratizing Innovation*, MIT Press, 2005.
6. K. Kumar and J. van Hillebergersberg, "ERP Experiences and Evolution," *Comm. ACM*, vol. 43, no. 4, 2000, pp. 23–26.
7. R. Kilian-Kehr, O. Terzidis, and D. Voelz, "Industrialisation of the Software Sector," *Wirtschaftsinformatik*, vol. 49, no. 1, 2007, pp. 63–71.
8. J. Blomberg, L. Suchman, and R.H. Trigg, "Reflections on a Work-Oriented Design Project," *Human-Computer Interaction*, vol. 11, no. 3, 1996, pp. 237–265.
9. C. Dörner, J. Heß, and V. Pipek, "Improving Information Systems by End-User Development: A Case Study," *Proc. European Conf. Information Systems (ECIS 07)*, University of St. Gallen, 2007, pp. 783–794; <http://is2.lse.ac.uk/asp/asppecis/20070043.pdf>.
10. J. Beaton et al., "Usability Challenges for Enterprise Service-Oriented Architecture APIs," *Proc. 2008 IEEE Symp. Visual Languages and Human-Centric Computing (VLHCC 08)*, IEEE Press, 2008, pp. 193–196.
11. P. Ehn, *Work-Oriented Design of Computer Artifacts*, Almqvist & Wiksell, 1988.
12. A. Mørch and N.D. Mehandjiev, "Tailoring as Collaboration: The Mediating Role of Multiple Representations and Application Units," *Computer Supported Cooperative Work*, vol. 9, no. 1, 2000, pp. 75–100.
13. G. Stevens and T. Wiedenhofer, "CHiC: A Pluggable Solution for Community Help in Context," *Proc. 4th Nordic Conf. Human-Computer Interaction*, ACM Press, 2006, pp. 212–221.
14. V. Wulf, V. Pipek, and M. Won, "Component-Based Tailorability: Enabling Highly Flexible Software Applications," *Int'l J. Human-Computer Studies*, vol. 66, no. 1, 2008, pp. 1–22.
15. J. Beringer, "Reducing Expertise Tension," *Comm. ACM*, vol. 47, no. 4, 2004, pp. 39–40.
16. M. Burnett, C. Cook, and G. Rothermel, "End-User Software Engineering," *Comm. ACM*, vol. 47, no. 9, 2004, pp. 53–58.
17. B.A. Myers, J.F. Pane, and A. Ko, "Natural Programming Languages and Environments," *Comm. ACM*, vol. 47, no. 9, 2004, pp. 47–52.
18. K. Kuutti and L. Bannon, "Searching for Unity among Diversity: Exploring the "Interface" Concept," *Proc. Interact '93 and CHI '93 Conf. Human Factors in Computing Systems*, ACM Press, 1993, pp. 263–268.
19. V. Pipek and V. Wulf, "Infrastructuring: Towards an Integrated Perspective on the Design and Use of Information Technology," *J. Assoc. for Information Systems*, vol. 10, no. 4, 2009, article 1; <http://laisel.aisnet.org/jais/vol10/iss5/1>.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.