In This Issue

Here we present articles, an interview, and a Point-Counterpoint discussion that begin to answer this question, as part of an area called end-user software engineering. This content comes from several perspectives. For example, in “Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover,” Joel Brandt and his colleagues show that when code is a means to an end (rather than
a deliverable), opportunism is often quite rational. They highlight the unique challenges that arise in debugging, reuse, and version control for populations whose view of programming is often based on a short-term, opportunistic mindset.

Two articles represent contrasting responses to this opportunism. In “Software Engineering for Spreadsheets,” Martin Erwig embraces the opportunism, describing a pair of type-checking and debugging systems for spreadsheets that work with users’ opportunistic habits, instead of against them. The systems he describes exploit patterns in the underlying structure of spreadsheets and spreadsheet formulas to automatically detect type errors and recommend changes.

In contrast, in “Test-Driven Development for Spreadsheet Risk Management,” Kevin McDaid and Alan Rust describe how to train users in test-driven development. They find that asking users to learn even a small amount of software engineering discipline goes a long way in improving dependability and software quality. In the Point-Counterpoint department, Janice Singer and Mark Vigder debate this issue with Judith Segal and Steven Clarke, exploring whether tools should be adapted to users or users should learn more-rigorous principles.

The other articles step back from these issues, considering the larger problem of empowering the masses to create real, robust software solutions. For example, in “Metadesign: Guidelines for Supporting Domain Experts in Software Development,” Gerhard Fischer and his colleagues argue that what makes software design difficult these days is the scarcity of domain expertise. They claim that the only way to truly design software for the myriad of domains is to empower domain experts to create their own solutions. Their viewpoint has several implications for the software that professional developers create. For example, they propose that systems must be “underdesigned” to support “hackability” and “remixability,” making it easier for domain experts to appropriate and adapt software for their needs.

Christian Dörner and his colleagues provide an example of this approach for the domain of enterprise resource planning (ERP) systems. In “End Users at the Bazaar: Designing Next-Generation Enterprise Resource Planning Systems,” they analyze the limitations of monolithic ERP systems supported by SAP and Oracle’s service-oriented architectures. They describe a new tool that lets users stitch together services to support their unique business needs.

We also offer an interview with Tessa Lau, who discusses CoScripter, which empowers Web

---

**About the Authors**

Andrew J. Ko is an assistant professor at the University of Washington’s Information School. His research interests include human and cooperative aspects of software engineering, end-user software engineering, and program visualization. He has a PhD in human-computer interaction from Carnegie Mellon University’s Human-Computer Interaction Institute. Contact him at ko@cs.washington.edu.

Robin Abraham is a program manager at Microsoft’s WinSE (Windows Serviceability) group. His research interests include software engineering, end-user software engineering, and program visualization. He has a PhD in computer science from Oregon State University. Contact him at robin.abraham@gmail.com.

Margaret M. Burnett is a professor in Oregon State University’s School of Electrical Engineering and Computer Science. Her current research focuses on end-user programming, end-user software engineering, information-foraging theory as applied to programming, and gender issues in those contexts. Burnett has a PhD in computer science from the University of Kansas. Contact her at burnett@eecs.oregonstate.edu.

Brad A. Myers is a professor in the Human-Computer Interaction Institute in Carnegie Mellon University’s School of Computer Science. His research interests include software development, end-user software engineering, natural programming, programming environments, user interface development systems, handheld computers, and programming by example. Myers has a PhD in computer science from the University of Toronto and is an ACM Fellow, a member of the CHI Academy, and a Senior Member of the IEEE. He also belongs to the IEEE Computer Society, SIGCHI, the ACM, and Computer Professionals for Social Responsibility. Contact him at bam@cs.cmu.edu.

---

**Reference**


---

**This diverse collection of articles reveals not only that the masses need more helpful, lightweight tools to catch bugs but also that the software industry itself must adapt to these shifting demands to truly serve user needs. If any of these topics pique your interest, we encourage you to dive deeper into the 10 years of R&D during which the end-user software engineering field has emerged (a useful collection of resources is at http://eusesconsortium.org), and stay tuned for its future.”**