Designing Software for Unfamiliar Domains

Parmit K. Chilana, Andrew J. Ko and Jacob O. Wobbrock
The Information School
DUB Group
University of Washington
{pchilana, ajko, wobbrock}@u.washington.edu

In recent years, software has become indispensable in complex domains such as science, engineering, biomedicine, and finance. Unfortunately, software developers and user researchers, who are usually experts in programming and Human-Computer Interaction (HCI) methods, respectively, often find that the insight needed to design for complex domains only comes with years of domain experience. How can everyone on a software design team acquire just enough knowledge to design effective software, especially user interfaces, without having to become domain experts? We are performing a series of studies to investigate this question, with the ultimate goal of designing tools to help software teams better capture, manage and explore domain knowledge.

We have already completed two preliminary studies. In the first study, we investigated developers designing bioinformatics tools, comparing those with biology backgrounds and those without (i.e., only computer science). The results showed that developers with no domain knowledge relied extensively on biologists to understand the biological problem and interpret the output rather than using other information sources. The developers preferred an informal exchange with biologists to acquire *just enough* knowledge, but the process of doing this was inefficient.

In another study, we investigated the work of usability professionals in complex domains to understand if and how they adapted traditional evaluation techniques. Our 21 informants worked in industry and research, with an average experience of 10 years designing for domains such as medical imaging, software development, network security, aviation, genomic analysis, healthcare, financial derivatives, and business-process support. A main finding from our study was that usability professionals regarded domain experts as the best resource for understanding domain-specific nuances, but access to domain experts was limited. Our results indicate the need for more *lightweight* methods that can take into account the limited availability of domain experts and still allow usability professionals to capture the necessary domain-related details.

Of course, these two perspectives are only part of the story. We are planning other studies of how software developers, testers, managers, and requirements analysts cope with a lack of domain expertise. For example, in an ongoing study, we are investigating design arguments that occur in open source bug reports, analyzing how developers form and defend design decisions for unfamiliar domains. One preliminary finding is that developers frequently invoke the "elastic user:" they describe users in whatever terms that support their opinion for the time being, even if it means being inconsistent with prior ascriptions.

We are also practicing user-centered design in complex domains ourselves. The first author, for example, is devising user studies with clinical researchers to guide the design and evaluation of large-scale biomedical systems at the University of Washington. Through this firsthand experience, we are gaining insight into the intricacies of this complex domain and the challenges of applying conventional design and evaluation approaches.

Based on the results from our studies, we will prototype ways of facilitating communication between members of a software design team, as well as communication with domain experts who help inform design decisions. Some tools have attempted to address this issue, but they have several limitations. For example, artificial intelligence systems for capturing domain expertise focus explicitly on creating formal structured representations of a domain, but not for the purpose of posing questions about design. Requirements engineering tools have the same formality, leading to low-level formalisms and diagrams that facilitate the creation of software architectures, specifications, and components. Tools designed for user testing limit data to user profiles and scenarios. None of these tools take a team-oriented or software lifecycle view of domain knowledge. Our proposed work will support multiple roles and phases of software design, including early sketches and implemented systems.

The benefit of this research lies not only in understanding how software teams currently capture, manage, and explore domain knowledge, but also in providing new ways of improving this process. This will lead to a better match between software and the activities it supports, and improved usability for millions of software users.