Designing tools for end users takes an understanding—and appreciation—of various levels of knowledge (or lack thereof).

REDUCING EXPERTISE TENSION

By Joerg Beringer

As an explicit design topic, end-user development (EUD) is rather new to human-computer interaction (HCI), although it is implicitly embedded in many design projects. What makes EUD different from other HCI topics is that in traditional HCI terms, users are experts in their tasks, and good tools should match these tasks. Conversely, end-user developers are trying to complete development tasks in which, by definition, they are not experts. Therefore, the dominating design goal of EUD tools is to compensate for a discrepancy between the user's expertise and the development task to be performed.

Taking this view helps to clarify differences in the assumptions that underpin the major types of EUD tools. It also helps transcend the simplistic assumption that end-user developers are excellent domain experts who start from scratch in terms of development knowledge and skills. In fact, as the accompanying figure illustrates, expertise tension exists in a two-dimensional continuum of job-related domain knowledge and system-related development knowledge. Different EUD practices and tools apply depending on the direction and level of (lacking) expertise.

Lacking systems knowledge. If we want to enable domain experts to modify or extend software without having a deep understanding of a computer system or coding skills, then expertise tension is in the direction of system knowledge.

The arrow in the figure labeled "Easy DevTools" stands for enabling users to act as IT experts by offering simplified software development environments. For example, Web-page builder, ad hoc workflow systems, or simulation packages are initially generic development tools. To enable end users to use that tool, the tools must be simplified by either optimizing the usability or by offering higher-level modeling primitives with limited options still not specialized to any domain, but encapsulating low-level technical aspects of the underlying technology.

To further support the domain expert, such generic development environments can also be specialized to a specific

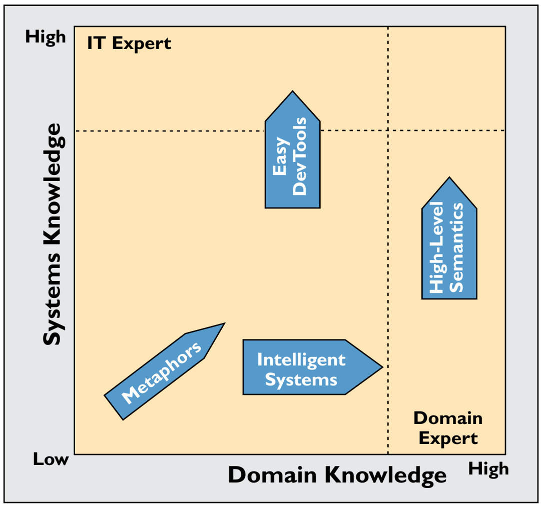
The dominating design goal of EUD tools is to compensate for a discrepancy between the user's expertise and the development task to be performed.

task domain, for example, modeling factory production. Here, the design challenge is both to hide any system-related details or technical constraints and to offer a high-level semantic language in which constructs directly reflect the entities and actions within the target domain; for example, factory production allows users to modify or extend the system by composing semantic entities they are familiar with. This is the idea behind domain-oriented design environments (DODEs) [1]. The "High-Level Semantics" arrow illustrates how domain experts are shifted closer to the direction of IT experts.

Lacking job domain knowledge. If we want to enable a user to complete a development task for which he or she does not have appropriate domain knowledge—for example, a manager setting up a collaboration room or a purchaser setting up a supply chain—then the development tool must offer more guidance. In addition to high-level semantic entities, such development tools must provide "best practice" advice and domain know-how in order to help the user understand key concepts and offer executional guidance.

In the collaboration room example, the configuration environment provides several ready-to-use templates for different work group types with different collaboration requirements. In a similar manner, the system can help the purchaser understand the basics about supply chain management and provides best practice guidance.

Overcoming the expertise tension on this dimension focuses on compensating for a lack of specific domain expertise. The design response to this tension is a well-designed configuration environment offering best practice templates and high-level semantic building blocks, as well as encapsulating domain knowledge by means of rules or ontologies. If appropriate, the entire development tool could be based on metaphors building on knowledge out of another well-known domain.



The continuum of expertise tension.

Reducing Expertise Tension

Depending on the dimension and level of expertise, some approaches seem to crystallize as principles to deal with expertise tension:

Intelligent systems have a built-in domain or system knowledge to guide users and cooperatively help the user to act.

High-level semantic building blocks enable users to create new things by composing familiar domain specific entities. Such applications usually do not generalize across different domains but are specialized solutions for a well-defined task domain.

Easy development tools provide end-user design times that come with a simplified function set, higher-level primitives, direct manipulation, WYSIWYG, or logical user interfaces that encapsulate system-oriented low-level details.

Metaphors use a replacement domain model to enable the user to act without having any domain knowledge.

Wizards have built-in rules and sequences to guide the user's task or interaction flow. They compensate for insufficient knowledge to plan actions or understand all the dependencies of options. This may be on the dimension of IT knowledge (configuring a firewall) or on the dimension of a task domain (booking a flight).

With this 2D-model of expertise tension in mind and the various design options identified, future research must further investigate which approach best applies to what type of tension. C

Reference

1. Fischer, G. Domain-oriented design environments. *Automated Software Engineering—The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering 1*, 2 (June 1994), 177–203.

JOERG BERINGER (joerg.beringer@sap.com) is a managing product designer at SAP–AG, Walldorf, Germany, specializing in xApps and composite application framework.