Available online at www.sciencedirect.com

ScienceDirect

Interacting with Computers 19 (2007) 597–613

Interacting with Computers

www.elsevier.com/locate/intcom

Towards useful and usable interaction design tools: CanonSketch

Pedro Campos [*], Nuno Nunes

LabUSE – Laboratory for Usage-centered Software Engineering, Department of Mathematics and Engineering, University of Madeira, Campus Universitario da Penteada, 9000-390 Funchal, Madeira, Portugal

Received 21 April 2006; received in revised form 19 January 2007; accepted 29 May 2007
Available online 27 June 2007

Abstract

Despite all the effort dedicated to bringing better User-Centered Design (UCD) tools to market, current studies show that the industry is still dominated by tools that do not support the activities and workstyles of designers. Also, there is a growing need for interaction design tools aimed at software engineers, a problem related to bringing usability into the software engineering processes.

We propose a new workstyle model that can be effectively used to envision, design and evaluate a new generation of innovative interaction and software design tools, aimed at integrating usability and software engineering.

We illustrate the effectiveness of our model by describing a new tool, called CanonSketch, that was built in order to support UCD in terms of the dimensions in our workstyle model. We also describe an evaluation study aimed at contrasting paper prototyping with our tool as well as the level of workstyle support.
© 2007 Elsevier B.V. All rights reserved.

Keywords: User interface development environments; User-centered design; Graphical user interfaces; Human–computer interaction; OO&HCI

1. Introduction

User-Centered Design (UCD) is a process that fosters an early and continuous focus on users in designing and evaluating a system, in order to obtain products that are better suited to the users' expectations (Gould and Lewis, 1985). However, and after almost two decades of research, the adoption and usage of UCD tools and techniques still remains limited to large organizations and a limited number of practitioners who recognize its value. Despite all the research efforts dedicated to bringing better tools to the industry, designers have frequently considered that tools do not meet their needs (Myers and Rosson, 1992; Iivari, 1996; Myers et al., 2000).

In this paper, we analyze the problems behind the weak adoption and usage levels of modeling and design tools, and propose a new workstyle model to support a new generation of usable tools. Some of the requirements for such tools were discussed in a recent workshop about usability of model-based tools (Campos and Nunes, 2004). Among other issues, the participants at the workshop highlighted the following requirements as paramount to promote usability in tools: traceability (switching back and forth between models, knowing which parts can be affected by changes), support for partial designs, knowledge management and smooth progression from abstract to concrete models and back.

It is a generally accepted notion that software engineers need better interaction design tools: tools that will help them create higher quality user interfaces. Besides the fact that software development companies usually do not have the budget for having dedicated interaction designers, it is also a fact of life that software engineers – who are not experts at interaction design – *will* create UIs.

What differentiates our design strategy from others – besides this line of argumentation focused on a software engineering perspective – is the fact that our approach is specifically aimed at designing in order to support *workstyle transitions*.

Workstyle modeling (Wu and Graham, 2004) has been proposed as a technique to record the interaction style of

[*] Corresponding author. Tel.: +351 96 2305353; fax: +351 291 705199.
E-mail address: pcampos@uma.pt (P. Campos).

a group of collaborators during any software development activity. In our own research, we observed that the UI aspects of current software development could be relieved if both designers and developers used tools that could transparently adapt to any given *workstyle*: a workstyle is a description of the most important aspects of the *way* users *work* in order to achieve their tasks. We have found – through empirical observation in small software development settings (Nunes, 2003), informal and formal usability studies (Campos and Nunes, 2005a) and data obtained from surveys and questionnaires (Campos and Nunes, 2007), as well as from current research literature – that both designers and software engineers often engage into different workstyles and move between workstyles very frequently (what we call a *workstyle transition*).

UCD is an iterative process, in which designers often engage in different workstyles as they iterate towards the final design. Thus we claim that modeling the workstyles can be particularly useful in UCD.

In related research, Constantine and Lockwood (1999) described their ideas of "galactic dimensions" as a metaphor change towards fully interconnected and synchronized visual development tools. Traditional CASE tools were based on a metaphor referred to as the "glass drawing board", since they merely represented two-dimensional paper models on the glass surface of a monitor. The "glass galaxy" was then proposed as a multidimensional problem-solving space in which developers could drill down into objects in one-dimension, and be taken via software "worm holes" to another. Clicking on a use case could take the developer to its definition in a glossary. Selecting that use case could also show the abstract components that support it, or the concrete widgets for a given realization of that model. Even entries in help files could be linked to the user roles they support, or to the actual code and visual UI controls.

We take this idea further and argue for UCD tools supporting "galactic" dimensions: tools that not only support accelerated development through traceability and integration, but are also able to rapidly adjust to any given workstyle in a transparent way. We propose a new workstyle model comprised of eight "galactic" dimensions that we consider as fundamental to supporting the UCD process. Our proposal is new because it is the first time workstyle modeling is applied to UCD and our model can be used to estimate the stage/effort of development in a graphically intuitive way. Another contribution of our approach is that it illustrates how effectively workstyle modeling can drive the development of a new generation of UCD tools, including our CanonSketch tool. CanonSketch was designed in order to support some of the most frequent and difficult workstyle transitions in UCD, and it allows the modeling, design and prototyping of a full application.

This paper is organized as follows: in the next section, we describe the current state of the art on tools that can be used in interaction design. Section 3 describes our new workstyle model for UCD and illustrates how it graphically conveys information regarding the stage of development of the UCD process. We also apply our model to evaluate several representative tools used in UCD. Section 4 describes our CanonSketch tool and how it supports regions in our model. We also present the results from two usability studies that partially corroborate our tool's usability. Finally, in Section 5, we draw some conclusions and outline work that might bring benefits to both software development and UCD.

2. State of the art in UI tools

In this section, we will review the state of the art in interaction design tools. We focus specifically on reviewing interaction design tools that are primarily aimed at software engineers, since that is how we position the CanonSketch tool.

In UI design tools, there has always been a dichotomy between the sophistication of what can be created (the *usefulness* of a tool) against the ease of use (the *usability*). This is related to the *threshold* and *ceiling* of tools (Myers et al., 2000). Threshold is how difficult it is to learn how to use the tool, and the ceiling is how much can be done with it (Myers et al., 2000). Building tools that can provide low-threshold and high ceiling at the same time has been identified as an important challenge (Myers and Rosson, 1992). Fig. 1 depicts a framework that illustrates the dichotomy between usefulness and usability. These two issues often collide: useful tools tend to suffer from lack of usability, whereas usable tools are not very useful (only allow the creation of simple artifacts, such as sketches, or semantic-less models).

Under this framework of usability and usefulness, and also using our workstyle model, we have surveyed and classified four classes of tools that are used in UCD: analysis, modeling and design (AMD) tools; Model-Based User Interface Design (MB-UID) tools; informal tools, and collaborative design tools.

Any tool that supports software development can be generally characterized as a CASE tool (Computer-Aided Software Engineering). However, the software development tradition relates CASE tools with high-threshold,

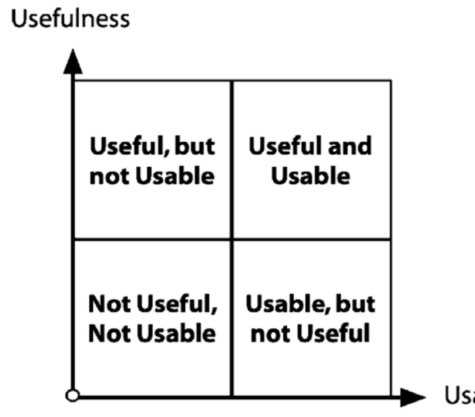


Fig. 1. Two issues that often collide: useful tools are not usable, usable tools are not sufficiently useful.

high-ceiling tools that are, to some extent, capable of generating code from high-level models (for instance, from UML class diagrams). CASE tools can be classified in many perspectives: for instance in terms of styles of development (e.g. RAD – Rapid Application Development), or degree of integration (e.g. IDEs – Integrated Development Environments). It is not our goal here to discuss the classification of CASE tools; instead we focus on the previously mentioned classes of tools, because they more clearly reflect the development tasks involved in any UCD process.

2.1. Analysis, modeling and design tools

Analysis, modeling and design tools are a class of software development tools characterized by their support of the earlier phases of software development (inception and elaboration) (Robbins, 1999; Nunes, 2003). Popular examples of AMD tools include Rational Rose and, in general, UML-based tools.

Rational Rose (IBM, 2004) is a set of tools that try to leverage the whole Rational Unified Process. Rose provides support for version control, IDE integration, modeling with design patterns, test script generation and collaborative modeling environment. However, the complexity of Rose is very high: although Rose follows MS Windows UI Guidelines, its dialogs are overly complex, diagram editing has to be done through many-tabbed dialogs, and other usability defects abound (Robbins, 1999).

Enterprise Architect is also a powerful means by which to specify, document and build software engineering projects (Sparks, 2004). It is based on the UML 2.0 notation and semantics and can also generate and reverse-engineer source code in a variety of languages, import database designs from standard data sources, and import and export models using the XMI industry standard.

The open-source community has not been able to compete with the fast-paced industry of modeling tools. ArgoUML (Robbins, 1999) is one of the very few open-source tools for analysis, modeling and design of software systems. The features included in ArgoUML are all based on Cognitive Theory: knowledge support via critics and checklists, process support via "to-do" lists, and visualization support via navigational perspectives. However, the zero-cost factor of this tool contributed to its dissemination more than the cognitive support features (Robbins, 1999). Like the majority of AMD tools, there is no explicit support for UI modeling.

2.2. Model-based user interface design tools

Model-based UI design tools are characterized by being able to allow the designer to specify interfaces at a very high-level, with the details of the implementation to be provided by the system (Myers et al., 2000). However, and contrary to AMD tools, model-based UI design tools employ automatic generation techniques which are expected to give programmers with no UI design experience the ability to create high-quality interfaces (Myers et al., 2000).

Puerta and Eisenstein (1999) proposed a general computational framework for model-based systems, and built an interface development environment, called the MOBI-D (MOdel-Based Interface Designer). MOBI-D allows developers to view and manipulate mapping of abstract task models into concrete interface designs. While using this tool in an exploratory fashion, the developer is expected to detect patterns of usage that are good candidates for automation. Although very useful, this tool suffers from problems that make it difficult to achieve a larger market acceptance, i.e., the connection between the models and the concrete UI is difficult to control and understand (Myers et al., 2000). UI Pilot (Puerta et al., 2005) is a successor tool which acts as a guide (a pilot) through the creation and manipulation of wireframes using the concepts of *user tasks, user types*, and *data objects*. The main difference between this tool and MOBI-D is that UI Pilot does not impose a rigid model-driven methodology, and also functions well within common software engineering development processes (Puerta et al., 2005). This flexibility suggests the importance of our claim for more flexible, adjustable tools.

ConcurTaskTrees (Paternò, 2000) is a widely accepted notation for specifying Task Models. Fabio Paternò and his team developed a set of tools that support the CTT notation, enabling editing, simulation and automatic generation of code from task models. Under the same line of research, several tools have been developed in the UsiXML consortium. UsiXML (which stands for USer Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as character user interfaces, graphical user interfaces, auditory user interfaces, and multimodal user interfaces. GrafiXML (Limbourg et al., 2004) is a graphical and textual UsiXML editor. Another UsiXML-based tool is KnowiXML (Furtado et al., 2004), which uses a knowledge-based system to facilitate the allocation of appropriate visual elements during the generation process. ReversiXML (Bouillon et al., 2004) is a tool for reverse-engineering the presentation model of an HTML Web interface back to UsiXML code, thus allowing re-incorporation of an existing UI into the development process. UsiXML tools were developed along a framework called the Chameleon Reference Framework (Limbourg et al., 2004). The workflow and level-of-abstraction transitions in the UsiXML environment have certainly parallels to our CanonSketch proposal. The Cameleon Reference Framework defines a series of steps towards the development of multi-context interactive systems. This framework exhibits three types of basic transformation types: abstraction (respectively, Reification) is a process of elicitation of artifacts that are more abstract (respectively, concrete) than the artifacts that serve as input to this process. Translation is a process that elicits artifacts intended for a particular context of use from artifacts of a similar development step but aimed at a different context of use. With respect to this framework, multi-path UI development (Limbourg et al., 2004) refers to a UI engineering method and tool that enables a designer to start a development activity from any entry point of the reference framework, get considerable support in the performance of all transformation types and their possible combinations.

SUPPLE (Gajos and Weld, 2004) is an application and device-independent system, that automatically generates user interfaces for a wide variety of display devices. SUPPLE differs from other approaches because it uses decision-theoretic optimization to render an interface from an abstract functional specification and an interchangeable device model. In other words, SUPPLE treats interface generation as an optimization problem. SUPPLE also provides mechanisms to adapt and customize the interface elements by changing the appearance, organization and navigational structure of the interface. When asked to render an interface with specified functionally on a specific device and for a specific user, SUPPLE searches for the rendering that meets the device's constraints and minimizes the estimated cost (user effort) of the person's activity. This multi-device fast-rendering capabilities make SUPPLE a useful tool. Its customization capabilities also add to its usability. However, there is still a lack of support for different workstyles assumed by interaction designers: work is done only at concrete detail level, without support for collaboration, informal/partial designs or other aspect of UI modeling.

2.3. Informal tools

These tools are mainly used for early, creative development stages, when we expect to take advantage from informal input modalities – such as sketching – in order to foster exploration of designs, fast communication of ideas and, in general, creativity. In this sense, informal tools are particularly important to support UCD.

Prototyping interfaces with electronic sketching tools has proven successful in systems such as SILK (Landay and Myers, 2001) or DENIM (Newman et al., 2003). Sketching is believed to be important during the early stages of prototyping, because it helps the designers' creative process: the ambiguity of sketches with uncertain types or sizes encourages the exploration of new designs without getting lost in the details, thus forcing designers to focus on important issues at this stage, such as the overall structure and flow of the interaction (Landay and Myers, 2001). However, widget recognition is hard for these systems, since any widget recognition algorithm might be too error prone. Also in this particular tool, usability tests reported that some users had trouble manipulating and entering text, and understanding how to select, group and move objects (Landay and Myers, 2001).

Damm et al. (2000) describe a tool, called Knight, which is based on a direct, whiteboard-like interaction achieved using gesture input on a large electronic whiteboard. Damm et al. argue that the conflicting advantages and disadvantages of whiteboards and modeling tools can lead to frustrating and time consuming switches between the two technologies, and therefore aim at offering a tool capable of offering the best of both worlds. The Knight tool uses an electronic board that naturally supports collaborative work, since several persons can work around it. Knight uses gestures as the main input mechanism. These are used for text input as well as for creating and modifying other diagram elements. By sketching boxes and lines on the screen, elements of UML class diagrams can quickly be created and manipulated, through an interaction similar to that of a traditional whiteboard. The idea of using gestures in modeling is also useful with other input devices, such as graphic tablets and PDAs.

2.4. Collaborative design tools

Software design (which includes interaction design) is often a team activity and most projects involve stakeholders with different backgrounds that must cooperate in many different and interrelated activities. Although any collaborative design activity involves communication and coordination, software design has an extra complicating factor: the product being designed is an incomplete description, not a concrete object. Software designers can envision user interfaces by writing prototypes and storyboards. But the end product is inevitably more complex and difficult to describe or portray (Potts and Catledge, 1996).

This has motivated the development of tools that support cooperation in software design. Recognizing that designers work together in a variety of styles and move frequently between these styles throughout the course of their work, Wu and Graham (2004) propose a tool called the Software Design Board (SDB), a collaborative design tool that supports a variety of styles of collaboration and facilitates transitions between them. The SDB tool supports the freehand creation of syntactically correct UML diagrams by using stylus-based input or enhanced whiteboards (also known as smartboards).

The tool also supports transitions between asynchronous and synchronous styles of collaboration, and between co-located and distributed styles of collaboration. The whiteboard space can be divided into any number of *segments*. These segments allow data to be shared in different ways. A *segment* is an area in the board containing contextually related data. As with a regular whiteboard, a user explicitly specifies the segmentation of data in the board through delineating strokes, e.g. a surrounding box or circle. Segments can be shared with others to allow users of other SDB clients to connect and synchronously interact with each other and share data. To share segments asynchronously, another client connects and copies the content of the segment to his/her local client. This data can then be manipulated without affecting the data in the original segment. Diverging copies of segments may be manually or automatically reconciled.

The Knight tool (Damm et al., 2000), also discussed in the Section 2.3, includes a distributed version that was developed to support collaborative design. Awareness mechanisms are implemented in Knight using visual cues on the drawing surface (a technique known as workspace awareness). Since there may be more than two users in the same session, each user has his or her own color that is used for all visual cues. For instance, to see where other participants are looking at (in the diagram), this tool uses a radar view for showing the current user's viewport as well as other participants' viewports.

All these examples show the importance of explicitly supporting the activities and styles of work adopted by interaction designers. In the following section, we describe a model for representing those styles of work.

3. A model of the designers' styles of work

By studying and modeling the styles of work adopted by interaction designers and developers, we aim at providing a useful discussion tool capable of driving the development of a new generation of tools. This new generation of tools should clearly support UI specific activities and styles of work as well as transitions between them, in the same way Wu and Graham (2004) have shown for general software development activities. We argue that three main categories should be addressed and supported: the **notation** (or language) used, the **tool-usage** style (*how* is the tool being used) and the type of **collaboration**. These categories are each described by the following dimensions:

1. **Notation** style dimensions: *Perspective*, *Formality* and *Detail*.
2. **Tool-usage** style dimensions: *Traceability*, *Functionality* and *Stability*.
3. **Collaboration**-style dimensions: *Asynchrony* and *Distribution*.

The proposed model ([SELF-REFERENCE]) is shown in Fig. 2. In this section, we will describe these dimensions and show how this model can be used to evaluate a tool's support for styles of work as well as how it relates to a UCD project's effort.

3.1. Dimensions of the model

Each of the model's dimensions describes a component in the style of work adopted by a team of interaction designers. Along with the informal description of the dimension's meaning, we also provide a set of questions that can act as guidelines for plotting a value in our model. These questions help place a tool or method against the corresponding axis. Questions about the tool-usage style dimensions are tool-specific. All the other questions are tool-independent.

3.1.1. Notation style dimensions

- **Perspective.** This axis plots the perspective of the artifact being developed. On the lower extreme of this axis (labeled "Problem/Requirements" perspective) one plots use cases, business logic and goal negotiation and prioritization. As the project moves forward, the designers transition towards the other extreme of this axis. Analysis perspective comes in-between, then issues such as modeling, simulation and validation, and finally the finished product. The axis' extreme is labeled "Solution/Design" perspective. ***Questions:*** Is the notation capable of expressing business goals? Or non-functional requirements such as customer experience requirements? Does it help define the purpose of the system? Does it describe interaction aspects of the system? How close is it to the final product?
- **Formality.** This axis classifies the workstyle of a designer creating artifacts in a formal vs. informal way. In the early stage of the process, designers use rough, ambiguous sketches to freely express ideas quickly (Landay and Myers, 2001). This workstyle also fosters comparison of design alternatives and creativity, since the uncertainty of sketches encourages the exploration of design ideas. As design progresses, a more formal style of work is incrementally adopted, as designers need to focus on the precise meaning of their models. An example of this shift is moving from a whiteboard to a CASE tool. ***Questions:*** How easy is it to define rough ideas and make the transition to more formal ideas? Does the notation force you to use a rigid syntax?
- **Detail.** This axis plots the level of detail (or abstraction) the designer is working at. High-level, abstract models facilitate problem solving in organization, navigation and overall structure of the UI, leaving aside the details. On the other hand, realistic (or figurative) prototypes address high-detail design issues. Disciplined designers tend to assume a workstyle that goes from higher-level abstract representations towards more realistic and detailed representations as the process evolves (Constantine and Lockwood, 1999). ***Questions:*** Can you avoid irrelevant details using the notation? Can you think about navigation and structure of the overall interaction using the notation? Can you incrementally add just enough detail?

3.1.2. Tool-usage style dimensions

- **Stability.** This dimension describes how difficult it is to modify any aspect of the artifact(s) being developed. High values in this axis indicate difficulty in performing changes. A content inventory of the UI modeled in a UML tool is highly modifiable because it is easy to change names, positioning, size and other aspects of the elements. This is opposed to drawing a model of the UI with pen and paper, since changes are harder

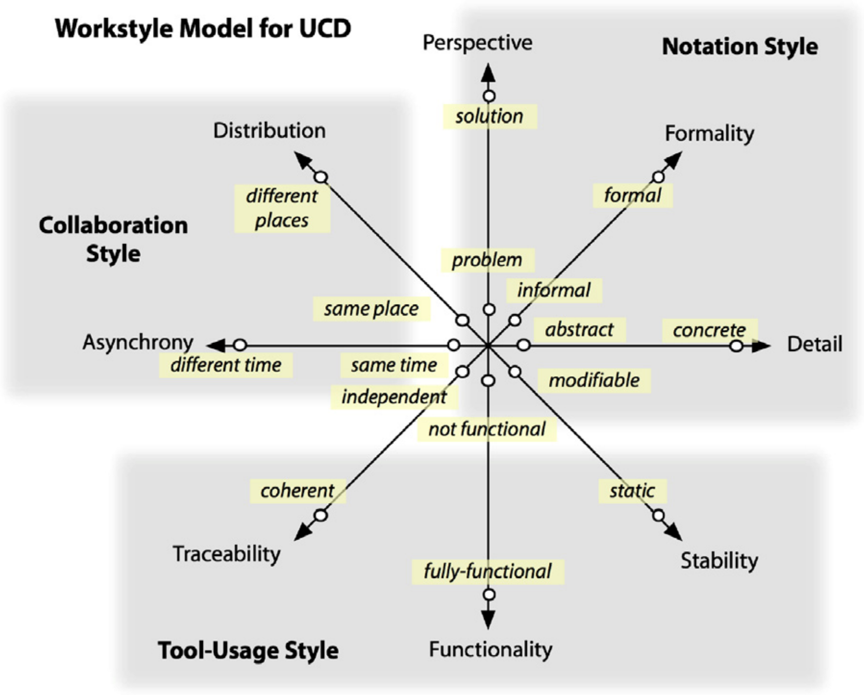


Fig. 2. A unifying model of the workstyles in User-Centered Design.

to accomplish. Brainstorming, for instance, is a very unstable workstyle because changes are very frequent. ***Questions:*** How easy is it to modify previously created artifacts using the tool? How comfortably do you make those changes? Are there particular changes that are difficult to accomplish with the tool?

• **Traceability.** This dimension describes if the elements of the artifact being developed are consistent and interconnected (thus being highly traceable) or if they are completely unrelated and independent. As an example, developers might adopt a workstyle in which they choose to keep links from task cases steps and the concrete UI widgets that implement those task steps. In this case, it is possible to trace a task step to the concrete widget and to trace a widget to the task step it implements. As the number of artifacts produced during a project increases, traceability becomes more important. ***Questions:*** Are you using the tool to maintain interconnections between model elements? How important is it to navigate through your model? Does the tool maintain several different views in a synchronized way (e.g. design view and code view)? Can you check or change other elements if a traced element changes?

• **Functionality.** For tools which are used for creating prototypes, this dimension represents how much functionality is being addressed (by using the tool to build a prototype). There is a barrier between software engineers and usability professionals regarding this matter: software engineers are engaged into building reliable, functional systems, leaving user-friendliness to the usability specialists. Usability and interaction designers, on the other hand, first design and test the interface with end-users, leaving implementation to software engineers, regarded as functionality builders. Those two processes should not be separated (Seffah and Metzker, 2004) and providing a high degree of prototype functionality in tools will help overcome this barrier. This dimension is also important because designers combine interaction design (behavior issues) with visual design (presentation issues). ***Questions:*** How much functionality, behavior and dynamics can you add to your prototypes using the tool? How easy is it to test the interaction by using the tool?

3.1.3. Collaboration-style dimensions

• **Asynchrony.** This axis refers to the collaboration style that designers assume: they can make changes to the work being developed at the same time (engaging in a synchronous workstyle) or they can work at different times (engaging in an asynchronous workstyle) (Wu and Graham, 2004). The higher the value in this axis, the more asynchronous the workstyle. ***Questions:*** Do the team members change artifacts at the same time? Or do they make changes at different times?

• **Distribution.** This dimension describes whether work is being conducted at the same physical location or at geographically distant locations. ***Questions:*** How far apart are the collaborating team members? Are they in the same building? Or are they in a different continent, or scattered through a country?

These dimensions can be effectively used to assess a given workstyle adopted by an interaction designer or a team of designers. A single workstyle is plotted as a line (a point in the eight-dimensional space) whereas regions (or planes) represent sets of workstyles.

3.2. Evaluating tools using the model

Our workstyle model described in the previous section enables discussion of existing and future tools that effectively support UCD. An example is depicted in Fig. 3 that shows the Knight tool described in Section 2.4 plotted in our model. It covers a large region, which suggests its adequacy to several styles of work during UCD. Knight only supports the syntax of the UML. It covers part of the perspective, abstraction and modifiability axis. Traceability exists to some extent, since some views are interconnected automatically and there is something like a model navigator. More aspects arise in the collaboration-style dimensions: Knight employs a sketch recognition language and can be effectively used in electronic whiteboards. There is also a distributed version with built in awareness mechanisms.

Fig. 3 also plots the CanonSketch tool in the workstyle model. CanonSketch (Campos and Nunes, 2004, 2005a,b) is an UML-based tool that supports multiple levels of detail by providing the designer with three views: UML view of the UI and domain models, canonical abstract prototype (Constantine, 2003) and HTML concrete prototype (as the right side of Fig. 3 exemplifies). The first two views are synchronized and the UML semantic model is used to support traceability. The tool seamlessly supports designers while switching from high-level abstract views of the UI and low-level concrete realizations. There is also a collaborative version of this tool in which designers can work at the same time on the model and at different places. However, support for distribution is still limited (for instance, there are no awareness mechanisms). The region of the model supported by CanonSketch is shown in Fig. 3. By contrast, a visual interface builder only supports a line in the workstyle model (the dashed line in Fig. 3).

Fig. 4 summarizes the results of analyzing and classifying some of the surveyed tools in Section 2 using a four-dimensional plot, where the x and y axes represent the levels of usability vs. usefulness found for a given tool, and where shading indicates the type of tool, the size of the shape indicates the level of tool support in our workstyle space. The x values for usability were obtained by heuristic evaluation (Nielsen, 1993) of the plotted tools: those which showed heuristic violations had lower values. The y values for usefulness were obtained by using the sophistication of the artifacts that could be created by using the tool: for instance, a whiteboard has a very low value as opposed to a tool that can be used to sketch models, validate them and generate code. ReversiXML can do this and also reverse-engineer a model, which explains its high value of usefulness. Each tool was also subject of a workstyle support study following the questions described in Section 3.1. Tools which revealed positive answers to the workstyle support questions were plotted with larger circles, those which revealed negative answers were plotted with smaller circles.

Typically, sophisticated AMD tools (such as Rational Rose) tend to present much functionality at the cost of poor workstyle support and a high learning curve that makes them very hard to use tools (there are even companies dedicated to provide training in using these tools).

On the other extreme of this classification framework, low-tech tools such as paper and pencil or whiteboards are very usable but they are unable to produce sophisticated artifacts. Informal tools such as MS Visio, DENIM (Landay and Myers, 2001) or Knight (Damm et al.,

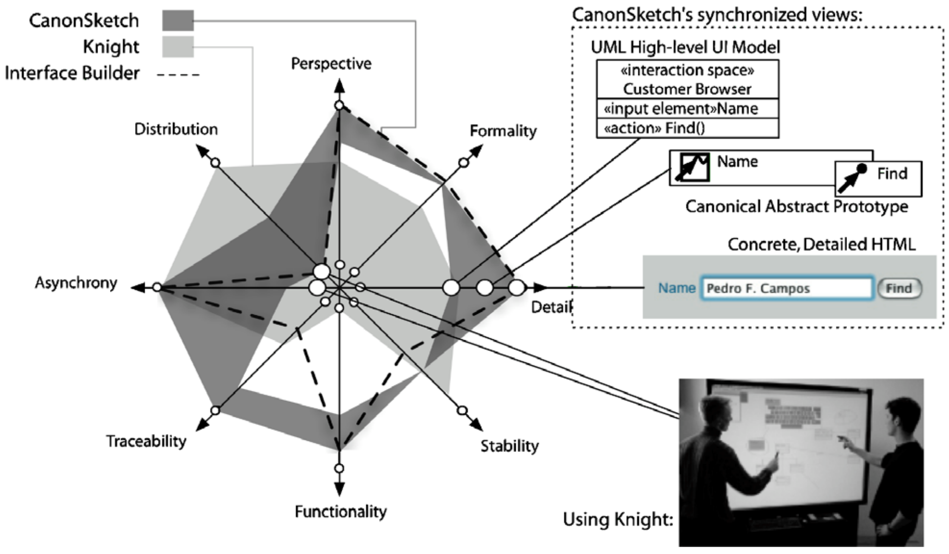


Fig. 3. CanonSketch, Knight and a visual interface builder plotted in our model.

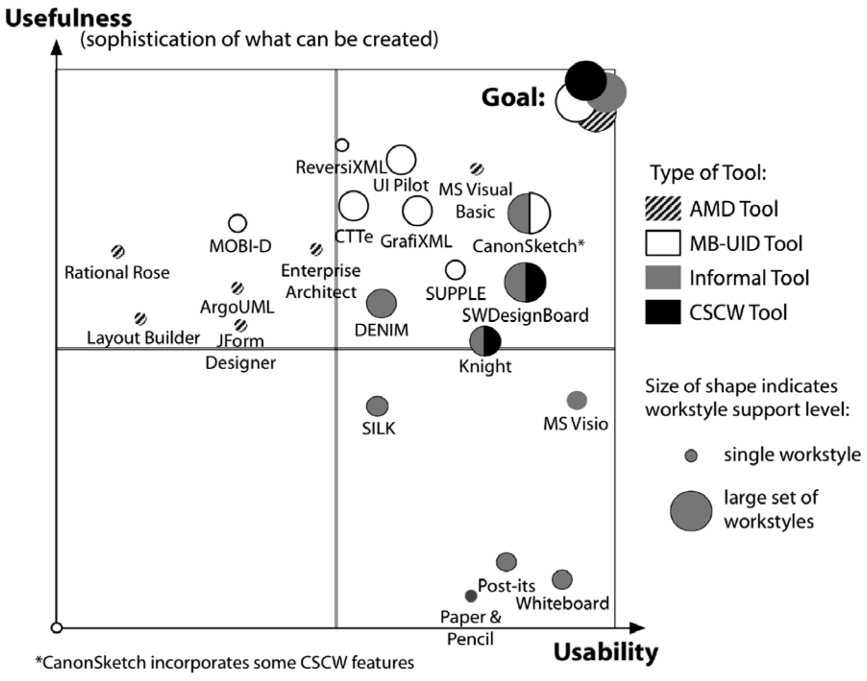


Fig. 4. A four-dimensions analysis of current UCD tools.

2000) although very usable and supporting a considerable surface in the workstyle space, do not produce full semantic models or support tasks such as formal scenarios execution and simulation.

The goal is, therefore, to achieve highly useful and highly usable *killer apps* for any class of AMD, MB-UID, informal or CSCW tools, while maintaining support for many styles of work as well as transitions between those styles.

3.3. Relating UCD projects with the workstyle model

In general terms, as the UCD process evolves, designers tend to assume a workstyle that spreads them away from the center of our model. We will briefly describe the typical changes that happen according to the following model's axes:

- *Perspective:* It is clear that as time goes by, developers move from the domain/problem level towards the solution/design space.
- *Formality:* Designers start out with informal, ambiguous sketches and move to formal languages later on, when coding increases and implementing functionalities becomes more important.
- *Detail:* As we have already seen, skilled designers tend to go from higher-level abstract representations towards more realistic and detailed representations as the process evolves.
- *Functionality:* In an initial phase, designers do not spend much effort on implementing functionality: it is more important to rapidly compare design alternatives. As the design process evolves, prototype functionality and behavior is added (and is needed for user testing).
- *Stability:* In addition, as time goes by, ideas start to solidify and changes become more incremental, rather than extensive (thus increasing stability).
- *Traceability:* As the number of artifacts increases, so does the number of interconnections between them, which motivates the need for increased traceability. This also happens because developers are not interested in throwing away the models (as in brainstorming) but rather in keeping all models created so far in a coherent state.

As we can see in Fig. 5, one of the advantages of our model is the fact that it graphically conveys implicit information regarding the temporal stage of development. Fig. 5 shows the workstyle model plotted for several phases of a UCD process (in this case the Wisdom process (Nunes, 2003)). The *thickness* of the plotted circles represents the number of iterations, whereas the *diameter* represents the typical workstyle. This is a rough modeling of the workstyles and how they vary according to the activity being performed. In the *inception* phase, all workstyle dimension values are low: developers think informally in terms of requirements, at the same time and place, without functionality issues in mind. As they move to *elaboration* and *construction*, they start to adopt a workstyle with higher values in all dimensions (although some more than others). Iterations begin and as these increase, the circles become thicker.

Fig. 5 also depicts in bold the axes (in the bottom part of the figure) that most contribute to each of the increasing

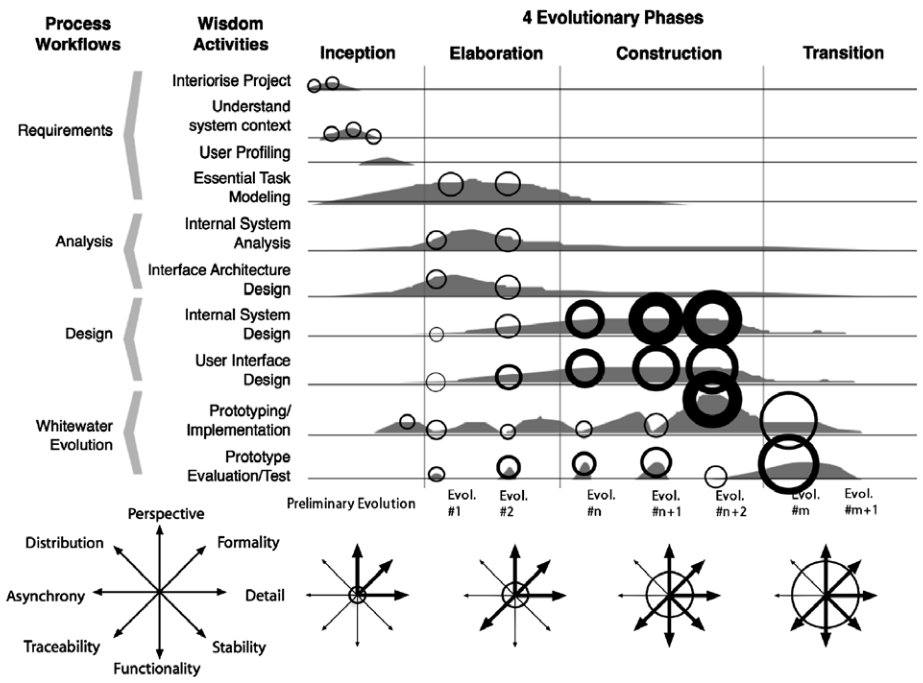


Fig. 5. Plotting our model along a UCD project. Below, axes in bold are the ones which contribute to the increasing circle of plotted workstyles.

workstyles. For instance, during the inception phase, there is no concern about traceability, whereas during construction, all axes except collaboration-style contribute to an increase of workstyle transitions.

Of course, the evolution in activities over time would never be depicted as a perfect circle, but we use that convention to convey the flavor of changes in activities. Also, no two projects would have the same goals and needs. For instance, the larger the organization, the greater the formality and location dispersion.

Based on the application of this model to current UCD tools, we identified several areas of improvement that should be addressed in order to start building a new generation of user-centered tools for UCD. As an example of this new generation of tools, we have built and evaluated CanonSketch, a prototype tool that addresses some of the currently unsupported UCD workstyles.

4. CanonSketch: designing for workstyle transitions

To illustrate how our workstyle model can be used to envision innovative interaction and software design tools, in this section we present the CanonSketch prototype tool that supports interaction design at multiple levels of abstraction (or detail). Since the advent of OO&HCI (or oohci) methods (van Harmelen, 2001) several approaches have been proposed to support interaction design using OO notations like the UML. In the Wisdom method (Nunes, 2003), we propose to use stereotyped class diagrams to depict the presentation aspects of interactive systems (through the concept of interaction space). Although the Wisdom notation could be supported in any UML modeling tool, our experience showed that these tools were not particularly effective in supporting UI design. As a result, developers adopting the Wisdom method usually shifted from traditional modeling tools to more informal tools while seeking the design freedom associated with interaction design.

The original idea for the CanonSketch tool (Campos and Nunes, 2004) was to support the Wisdom notation at the presentation level and also to combine UML class stereotypes with the canonical abstract prototype notation. Canonical abstract prototypes were developed by Constantine and Lockwood (1999) after a growing awareness among designers regarding the gap between task models and realistic prototypes. They provide a common vocabulary for expressing visual and interaction designs without concern for details of behavior and appearance. Moreover, they fill an important gap between existing higher-level techniques, such as the Wisdom UML-based interaction spaces and lower-level techniques, such as concrete prototypes. For this reason we chose canonical abstract prototype as the intermediate notation to support interaction design in CanonSketch. In the next sections, we will describe the Wisdom UML profile, canonical abstract prototypes and the linkage between these that is the basis for the synchronization between views in CanonSketch.

4.1. The Wisdom UML profile for UI design

The Wisdom notation is a UML extension aimed at supporting interactive systems modeling. In the CanonSketch project we refined the Wisdom notation to achieve full semantic support for canonical abstract prototypes. We also worked towards automatically maintaining synchronized UML and canonical views, by means of a common semantic model. The specification of such a common model allowed the tool to support the design process at different levels of detail.

To support the modeling of presentation aspects of the UI, the Wisdom method proposes the following extensions to the UML (Nunes, 2003):

- ≪Interaction space≫, a class stereotype that represents the space within the UI where the user interacts with the tools and containers during the course of a task or set of interrelated tasks;
- ≪navigates≫, an association stereotype between two interaction space classes denoting a user moving from one interaction space to another;
- ≪contains≫, an association stereotype between two interaction space classes denoting that the source class (container) contains the target class (contained); the ≪contains≫ association can only be used between interaction space classes and is unidirectional;
- ≪input element≫, an attribute stereotype denoting information received from the user, i.e., information the user has input;
- ≪output element≫, an attribute stereotype denoting information displayed to the user, i.e., information the user can perceive but not manipulate;
- ≪action≫, an operation stereotype denoting something the user can do in the concrete UI that causes a significant change in the internal state of the system.

4.2. Canonical abstract prototypes

Constantine (2003) proposes a stable collection of abstract components, each specifying an interactive function, such as data input and display notification. These components can be selected from a palette in order to build abstract prototypes, thus fostering flexibility.

The symbolic notation underlying canonical abstract prototypes is built from three generic, extensible[1] symbols or glyphs (see Fig. 6):

- A generic *tool* or *action*, represented by an arrow. Tools represent operators, mechanisms or controls that can be used to manipulate or transform materials (Constantine, 2003).
- A generic *material* or *container*, represented by a square box. Materials represent content, information, data or other UI objects manipulated or presented to the user during the course of a task.
- A *hybrid* or *active material*, which represents any component with characteristics of both composing elements, such as a text entry box (a UI element presenting information that can also be edited or entered).

[1] Meaning all other components can be derived, or specialized, from these classes.

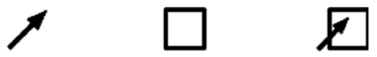


Fig. 6. The three basic symbols underlying the symbolic notation of canonical abstract prototypes (from left to right): a generic abstract tool, a generic abstract material and a generic abstract hybrid, or active material (taken from Constantine, 2003).

Although canonical abstract prototypes lack a precise formalism and semantics required to provide tool support and automatic generation of UI, we found the notation expressive enough to generate concrete user interfaces from abstract prototypes. Our tool presents a proof of concept, since we generate HTML pages from sketches of canonical abstract prototypes. We also found the notation very useful for expressing design patterns in an abstract, platform-independent way.

In CanonSketch, the designer is able to choose between the UML model view and the canonical abstract view, and switch back and forth while maintaining coherence between the models: changes made in each view are automatically reflected in others according to the mapping model. This informal mechanism adds some support for the traceability dimension in our workstyle model.

4.3. Linking Wisdom UML to canonical abstract prototypes

In Fig. 7, we show the specification of a mapping between the Wisdom presentation model and canonical abstract prototypes.

- An interaction space in Wisdom is mapped to an interaction context in a canonical prototype;
- an ≪input element≫ attribute stereotype is mapped to a generic active material, unless typified. Input elements specify information the user can manipulate in order to achieve a task;
- the ≪contains≫ association stereotype is mapped to a container;
- an ≪output element≫ attribute stereotype maps to an element;
- an ≪action≫ operation stereotype maps to an action/operation canonical component.

The ≪navigates≫ association, although not present in Fig. 7 because there is no corresponding element in canonical glyph notation, can be unidirectional or bidirectional; the latter usually meaning there is an implied return in the navigation. This essentially has the same meaning Constantine defines when describing the canonical contexts' navigation map (Constantine, 2003).

We can also see from Fig. 7 that our extension to the Wisdom presentation model notation fully supporting

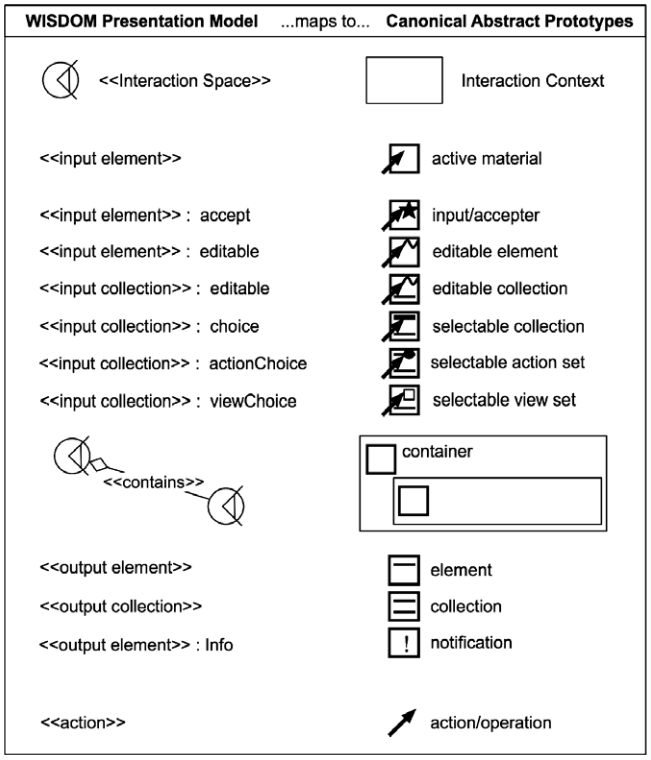


Fig. 7. Extending the Wisdom profile to support canonical abstract prototypes: this figure shows the correspondence between Wisdom UML stereotypes and canonical components.

canonical abstract prototypes consists of adding two more attribute stereotypes:

- ≪input collection≫, an attribute stereotype denoting a set of related information elements received from the user, i.e., a set of input elements; an ≪input collection≫ can be used to select from several values in a drop-down list, or choosing one element from a table to perform any given operation;
- ≪output collection≫, an attribute stereotype, denoting a set of related information elements displayed to the user, i.e., a set of output elements. Typically, an ≪output collection≫ conveys information to the user about a set of elements of the same kind, for instance a search results list or the results display from a query to a database.

By typifying these attribute stereotypes, one can map a Wisdom presentation model to all canonical components that belong to the classes of materials or hybrids. For instance, an input collection typified as choice can be mapped to a selectable collection.

As Fig. 8 illustrates, CanonSketch shows designers four synchronized views of the UI at different levels of detail:

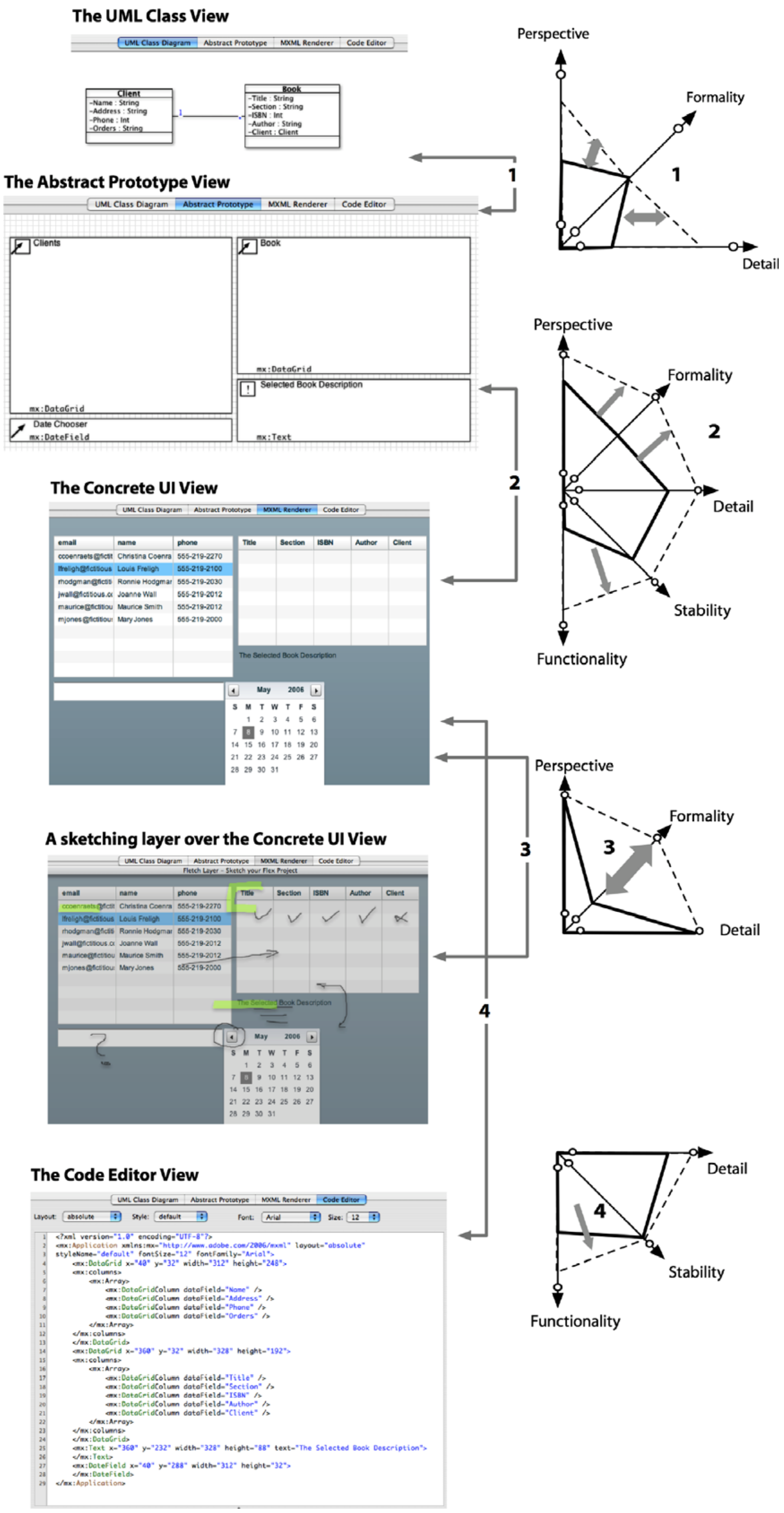
- The **UML view**, primarily used for domain modeling, but which also shows the abstract contents of the UI as well as navigation and containment relationships between them, using the Wisdom UML profile.
- The canonical **abstract prototype view**, which shows the spatial layout and interactive function of the abstract UI elements.
- The **concrete view**, which shows a possible concrete rendering of the UI.
- The **code editor view**, where the user can edit the textual description of the UI.

Through the mapping model presented in Fig. 7 we achieve round-trip engineering between the UML view and the canonical abstract prototype view. Thus, if we change e.g. the attribute stereotype in a given class from ≪input element≫ to e.g. ≪input collection≫ then changes are propagated (through an "observer" class in the tool's code) to the canonical prototype view corresponding element (which in this example would automatically change from "input/accepter" into "selectable collection"). The same happens for other kinds of model elements and other attributes (like changing names). Note however that if one changes the size or position in the abstract view, the only update made is to the concrete view, since the UML view does not model layout issues.

The round-trip problem of propagation of changes made in the concrete code view back to the more abstract views was not solved/implemented. Instead our focus was on helping smooth the transitions between the most employed workstyles, in interaction design. This also helps to narrow our problem in order to better test our research hypothesis (namely whether or not the proposed features allow a good support of workstyle transitions).

Fig. 8 illustrates the five views provided by CanonSketch, as well as the workstyle transitions supported. Consider the transition illustrated by the grey arrows labeled "1". When the designer is modeling the domain concepts using the UML view and then switches to the abstract prototype view for laying out the abstract UI elements, the workstyle transition dimensions that change values are perspective and detail, as Fig. 8 illustrates. The transition supported by this easy, synchronized switching of views is a transition from a medium value of perspective into a higher value; and from a low-level of detail into a higher value of detail. Fig. 8 shows the initial workstyle plotted as a thick, solid line, and the final workstyle as a dashed, thinner line.

The workstyle transition supported by CanonSketch's switching between abstract, non-functional prototype view and the concrete, fully functional view is labeled "2". The initial workstyle is characterized as having intermediate values of perspective, formality, detail, stability and, most importantly, functionality. Traceability, and the collaboration-style dimensions asynchrony and distribution are irrelevant in describing this transition. According to the results of a survey we performed (Campos and Nunes, 2007), this is one of the workstyle transitions considered by professionals as being the most difficult. Therefore, special care should be given to designing support features for this particular workstyle transition.



email	name	phone
ccoenraets@fictit	Christina Coenra	555-219-2270
lfreligh@fictitious	Louis Freligh	555-219-2100
rhodgman@fictit	Ronnie Hodgmar	555-219-2030
jwall@fictitous.cc	Joanne Wall	555-219-2012
maurice@fictitiou	Maurice Smith	555-219-2012
mjones@fictitiou	Mary Jones	555-219-2000

Title	Section	ISBN	Author	Client

The Selected Book Description

May 2006

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

```
<?xml version="1.0" encoding="UTF-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
styleName="default" fontSize="12" fontFamily="Arial">
    <mx:DataGrid x="40" y="32" width="312" height="248">
    <mx:columns>
        <mx:Array>
            <mx:DataGridColumn dataField="Name" />
            <mx:DataGridColumn dataField="Address" />
            <mx:DataGridColumn dataField="Phone" />
            <mx:DataGridColumn dataField="Orders" />
        </mx:Array>
    </mx:columns>
    </mx:DataGrid>
    <mx:DataGrid x="360" y="32" width="328" height="192">
    <mx:columns>
        <mx:Array>
            <mx:DataGridColumn dataField="Title" />
            <mx:DataGridColumn dataField="Section" />
            <mx:DataGridColumn dataField="ISBN" />
            <mx:DataGridColumn dataField="Author" />
            <mx:DataGridColumn dataField="Client" />
        </mx:Array>
    </mx:columns>
    </mx:DataGrid>
    <mx:Text x="360" y="232" width="328" height="88" text="The Selected Book Description">
    </mx:Text>
    <mx:DateField x="40" y="288" width="312" height="32">
    </mx:DateField>
</mx:Application>
```

Fig. 8. The views in CanonSketch, from top to bottom according to the process of abstract to concrete model-driven design. The labeled arrows correspond to the workstyle transitions supported.

Functionality is added in the code editor view, and can be tested using the concrete view. There is the option of automatically generating MXML code from the canonical abstract prototype, which is also a way of smoothing the transition from problem to solution, and from non-functional to functional. MXML (ADOBE, 2006) is an XML markup language for user-interface components layout. One can also uses MXML to declaratively define non-visual aspects of an application, such as access to data sources on the server and data bindings between user-interface components and data sources on the server.

A related workstyle transition, also related with moving from non-functional to fully functional prototypes, is illustrated by the two arrows labeled "4" in Fig. 8. In CanonSketch, the MXML markup code is automatically generated from the abstract prototype, and using the code editor view, methods, variables and services are added smoothly to the code, thus describing both the visual and the behavior interaction details of the system being designed.

CanonSketch supports both informal and formal workstyles. It is possible to use the tool in conjunction with a Tablet or a smartboard, and apply sketch recognition to informally drawn sketches (thus transitioning from a semantic-less workstyle to a semantically sound, UML-based model). There is also a semi-transparent layer (shown in Fig. 8), where the user can freely annotate any of its models or views. The corresponding transition is also shown by the arrows labeled "3", also in Fig. 8.

4.4. Expressing UI patterns in CanonSketch

Our experience using the Wisdom notation in different settings enabled us to identify some problems with using stereotyped class diagrams to capture the presentation aspects of interactive systems. In particular, spatial information and other lower-level aspects are important for some common UI patterns. As we discuss in (Campos and Nunes, 2004) when applying the Wisdom approach to UI patterns, some problems derive from detailed presentation aspects, such as size, position, or use of color. Specifying a linkage between canonical abstract prototypes and the Wisdom presentation model can help solve some of these problems, while also adding the necessary formalism to the canonical notation.

The capability of identifying UI patterns and expressing the solution in an abstract way independent of any particular platform or implementation is becoming more and more important, with the increase in the number of information appliances. The Wisdom notation enables an abstract definition of UI patterns (Nunes, 2003), and also complies with the UML standard. However, some problems remain for patterns expressing more concrete presentation aspects, such as size or positioning. Having a tool that provides a common semantic model linking canonical components to Wisdom elements can help solve some of these problems.

The designer starts by specifying the general structure of the UI using the Wisdom UML extension (see the top part of Fig. 8). That specification is mapped to one or more canonical interaction contexts, where the designer expands and details the model in terms of size, position and interactive functions (the second view, from top to bottom, shown in Fig. 8). This mapping clearly shows the role of Wisdom interaction spaces realizing the interface architecture, and how they can be combined with the canonical notation to help bridge the gap between abstract and concrete models of the user interface. Switching to the code editor and the concrete UI view (bottom and middle views in Fig. 8, respectively) allows the user to express a possible realization of the pattern's solution.

As an example, consider the "GUI Preview" pattern from the Amsterdam collection of UI patterns (van Welie and Traeetteberg, 2000). This pattern is particularly helpful when the items' content nature does not match its index (e.g. a set of images or audio files are indexed by a textual label). The problem this pattern tries to solve occurs when the user is looking for an item in a small set and tries to find the item by browsing the set. The solution is to provide the user with a preview of the currently selected item from the set being browsed (van Welie and Traeetteberg, 2000). In Fig. 9, we present the Wisdom and canonical representations for this pattern. We also present a concrete realization of this pattern (a dialog from MS PowerPoint).

On the one hand, the Wisdom representation (on the left of Fig. 9), is much more compact, because it is based on the UML. It is evident that the canonical notation is much closer to the concrete representation of this pattern. But the canonical representation has the advantage of clearly stating that the browsable list of items is placed to the left of the item preview, which conforms to the western way of reading and therefore adjusts to the task being performed: the user first selects an item, and only then does she focus on the preview.

4.5. Evaluating CanonSketch

Our workstyle model enabled us to discuss and create a new class of design tools that support the increasingly complex tasks involved in modern software development. From our workstyle model we created a new tool to support interaction design at multiple levels of detail using different notations that are particularly effective in different stages of development. The goal of the CanonSketch tool was to support abstract UI design, using the UML semantics, while also leveraging multiple interfaces. The existing version supports rendering for HTML and MXML but our long term goal is to support a device-independent markup language such as UsiXML (Vanderdonckt et al., 2004).

After iterative user interface improvement via discount usability techniques (Nielsen, 1993), we evaluated CanonSketch in two ways: we extracted basic usability satisfaction measures and we studied the users' behavior according to our workstyle model. To determine if designers would find our tool useful and usable, we conducted two usability tests, aimed at answering questions such as:

1. Can designers use the tool to model the UI at different levels of detail in a usable and useful way?

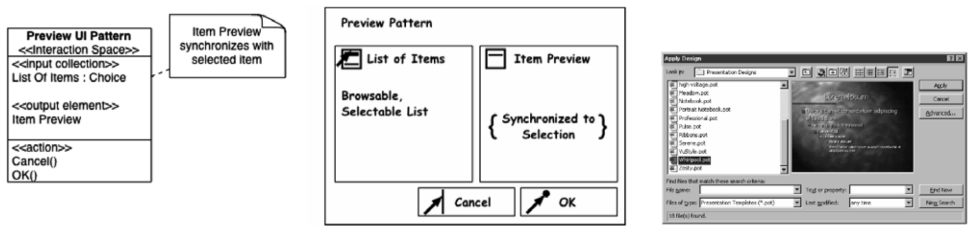


Fig. 9. (From left to right). A Wisdom UML model and the corresponding canonical prototype, both applied to the preview pattern. A concrete example is also shown: a dialog from MS PowerPoint.

2. Does CanonSketch provide the design teams with a standard, non-ambiguous and easy way to communicate, thus fostering collaboration and sharing of ideas?
3. Does CanonSketch enable fast comparison of designs, and does it leverage the creativity of designers, thus leading to more innovative prototypes?

In this section, we describe our study design, present the usability results and briefly describe some design findings.

4.5.1. Study design

We performed the design study in a laboratory setting with 18 computers running the collaborative version of CanonSketch. Our subjects were 32 undergraduate students with UML modeling knowledge, OO-programming concepts and practice, and basic computer engineering principles. It seems, at first sight that having used students as the test subjects only weakens the value of the experimental results. But there is a positive benefit; the experiment confirms the usability of CanonSketch to students in an educational context. Therefore CanonSketch is a useful teaching aid. As UI designers, the authors find the system useful, but because of the sampling the reported results are not strictly applicable to UI design professionals. This is worth investigating to either confirm CanonSketch's applicability/usability, or to formatively improve CanonSketch's applicability/usability. Consequently, the authors are working on a similar study in industry.

The design tasks were similar for both studies:

- **Study A:** In this study, we asked users to design a UML model and an abstract prototype for an interface to a weather system for travelers, as (Landay and Myers, 2001) did. This problem is described in *Usability Engineering* (Nielsen, 1993) and provides a solid benchmark, since possible design errors are well-documented (Nielsen, 1993).
- **Study B:** In this study, we gave users the task of designing and building creative solutions for a book-selling website UI. We asked them to focus their efforts on designing creative solutions, i.e., to compare and iterate designs towards an inventive prototype.

The evaluation for Study A consisted on a between subjects design, where each user acted first as a designer and afterwards as a client (or the opposite). In the role of designer, the user had to define/iterate design ideas and communicate/discuss them with the client. In the role of client, he had to understand/discuss the designer's prototype. We randomized the attribution of all combinations of designer/client. This study involved 32 users. Half of them used CanonSketch, the other half used pencil, paper and Post-it notes.

The evaluation for Study B involved solving the task in groups of two, using CanonSketch only.

4.5.2. Usability results: Study A

Based on observing users interact with CanonSketch, answering questions during the tasks, and user discussion after the evaluation, we collected several usability results.

In a post-evaluation questionnaire, we asked users to rate the ease of change, satisfaction, ease of use, exploration of designs, comparison of designs, ease of communication and expressive power of their tool on a 7-point Likert scale, with (7) being a highly positive rating and (1) a highly negative rating.

Fig. 10 plots the mean values for the obtained results. We can see that CanonSketch easily outperforms the low-tech tools of paper, pencil and Post-it notes in all aspects, except for the ease of comparison. Table 1 summarizes the statistical results. At a 5 percent significance level, we can see that only comparison and expressiveness are not statistically different. In the ease of change-related

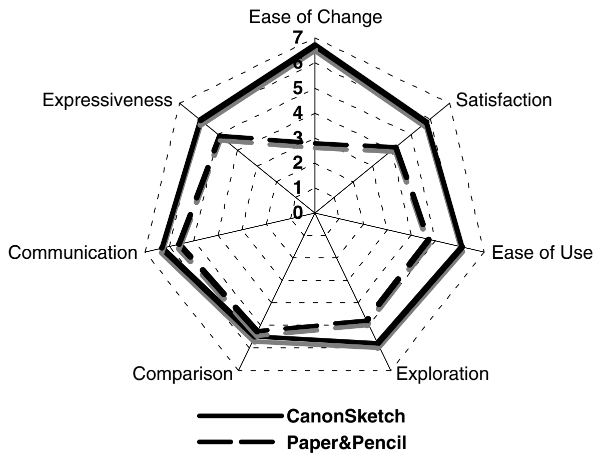


Fig. 10. Mean values for CanonSketch and paper and pencil.

questions, participants using CanonSketch accomplished changes easier (mean = 6.1, var = 1.5) than those using pencil and paper (mean = 3.6, var = 1.9), $t(28) = 5.1$, $p < .05$. They also felt more satisfied using CanonSketch (mean = 5.8, var = 0.7) than using pencil and paper (mean = 4.2, var = 3.3), $t(28) = 3.07$, $p < .05$.

We can see that in all dimensions CanonSketch fares well, with users rating the tool with highly positive values. Compared to using low-tech tools (paper, pencil and Post-it notes), CanonSketch is significantly better, except for the ease of comparison of designs. This makes sense, since using several sheets of paper and placing them over a table allows observing the "bigger picture" in a natural way. The greatest statistical difference was shown in the ease of change questions: users found CanonSketch to be significantly useful for rapidly changing design elements. This was particularly evident when "clients" added new requirements/suggestions and designers were forced to change their prototypes.

However, these results only apply to OO&HCI students. Although they had knowledge of UML modeling and OO design from previous courses, they do not have enough experience or skills at these techniques. This accounts to explaining how appealing CanonSketch was to them. In fact, many universities teaching similar subjects (such as OO&HCI techniques) are already using the CanonSketch tool.

4.5.3. Usability results: Study B

During Study B, participants were asked to rate questions about how well did CanonSketch support their working styles. For this study, we made a similar rating questionnaire as in Study A, but focused our questions on how well the tool supported our workstyle model axes as well as transitions in those axes. Fig. 11 graphically plots the results and Table 2 the values. The questions about asynchrony and distribution were made in the context of students having used CanonSketch throughout the whole semester (including during the elaboration of a semester-long HCI project). Besides working at the same time and place, students also worked *asynchronously* because they typically divide sections of a project, and *distributed* because sometimes they work at home, or in different laboratories.

Table 1
Summary of the results for Study A

Topic	Mean (variance)		p-value
	CanonSketch	Paper and pencil	
Ease of change	6.1 (1.5)	3.7 (1.9)	0.0002
Satisfaction	5.8 (0.7)	4.2 (3.3)	0.004
Ease of use	6.1 (0.7)	4.7 (3.7)	0.01
Exploration	5.9 (0.6)	4.8 (2.8)	0.03
Comparison	5.5 (1.1)	5.3 (3.1)	0.75
Communication	6.3 (0.7)	5.6 (1.3)	0.01
Expressiveness	5.9 (0.8)	4.9 (3.2)	0.07

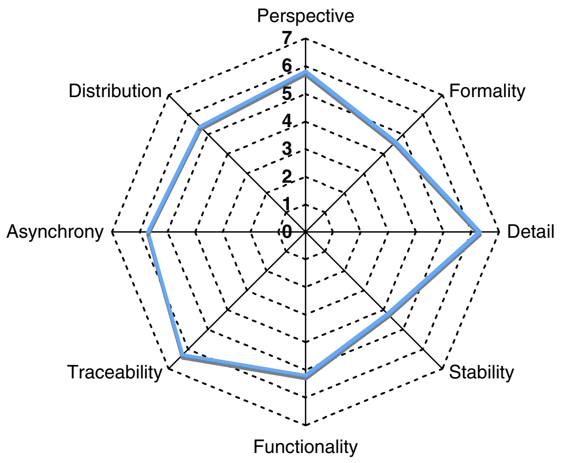


Fig. 11. The level of workstyle support as evaluated by CanonSketch's users, according to the galactic dimensions.

Table 2
Summary of the results for Study B

"*Galactic*" dimension	Mean (variance) CanonSketch
Perspective	5.8 (1.2)
Formality	4.6 (3.2)
Detail	6.3 (0.6)
Stability	4.2 (2.1)
Functionality	5.2 (1.1)
Traceability	6.3 (0.7)
Asynchrony	5.7 (1.1)
Distribution	5.4 (2.2)

From the results, we can set out a set of future directions for the CanonSketch tool. The axes that obtained the lowest ratings were Formality (mean = 4.6) and Stability (mean = 4.2). We can infer that designers had trouble crafting informal/formal artifacts and also were not fully satisfied with the capability to support fast (or incremental) changes to models.

Besides these formal studies, we performed extensive observation of the subjects during a semester-long HCI course.

4.5.4. Design findings

Our extensive, one-semester long study resulted in a series of observations about the users' behavior, which we summarize here in terms of our model's notational, tool-usage and collaboration styles.

Notational style issues. One of the interesting observations we made along this study was that users reverted to natural language to describe certain dynamic aspects of the interfaces being prototyped, such as specifying that selecting an item from a list will update the image of an UI element, or that choosing a given item will show different content in a subsequent window. In addition, some users found it difficult to design a good spatial layout for the abstract user interface. This suggests that UI tools should provide guidance for design activities, without establishing constraints on creativity.

Another observation is related to the naming applied by users to their model elements. The act of giving/finding a name for a class, task or screen was sometimes the cause of team misunderstandings and could even lead to bad design choices. Since the process is guided by tasks and essential use cases, users showed a tendency to create too many interaction spaces by not aggregating the ones without sufficient elements to fill up a whole window.

In spite of a lack of expressiveness for some dynamic aspects we referred above, use of Constantine's abstract prototype notation revealed a much lower learning curve than we expected at the beginning of the study. After only a brief 30 min introduction and examples, users were not confused about the notation and successfully used it to prototype several design solutions. Clear, simple tool tips over the palettes of abstract components may have helped accomplish this.

Tool-usage style issues. We observed that the tools' usability was often appreciated by the subjects. We also observed that the tools' interaction idioms were able to keep users engaged in the development process. Sometimes the users were interested in fully maintaining traceability between all elements in all views and at other times they were not interested at all in maintaining that consistency (they used a single view). This shows the adequacy of the traceability axis in our model. On the negative side, the beta version of the tool (used in this study) still revealed some implementation bugs which were caught during the study itself.

Collaboration style issues. The collaborative version catalyzed the development process and group discussion. This was surprising, since most of the empirical results on technology-mediated collaboration report degradation in task performance and information sharing (Dimicco et al., 2004; Kiesler and Sproull, 1992; Hollingshead, 1996). We believe much of this is related to the usability of the collaborative version (by using a Messenger-like interface and drag and drop interaction). Users were engaged in both asynchronous/distributed and synchronous/co-located styles of work.

5. Conclusions

The research described here has aimed at building better models of the developers' behavior and at a better understanding of UCD tools-usage and UI development activities. More importantly, the development of several innovative prototype tools such as CanonSketch or TaskSketch (Campos and Nunes, 2005a) has already generated interest from both industry and academia and the approach looks promising.

We have shown how useful our workstyle modeling can be when applied to UCD and to the development of new UCD tools that can better support the activities of their users. Current models are too specific and are not expressive enough to be applied more generally to UCD. Our framework is the first workstyle model tailored to UCD. Our model can be effectively used to (a) choose adequate tool support for a given phase of a project and (b) drive the development of new UCD tools, as we have shown with CanonSketch. The model can also help to spot mismatches between the capabilities of current tools and the actual needs of designers, as well as to identify those areas that lack proper tool support.

One limitation of our approach is the fact that there is not a simple or clearly defined process of using the canonical notation to specify interfaces for multiple devices. Although CanonSketch can clearly allow multi-platform development (Win, Mac, Palm, Web, etc.), multimodal interfaces are not supported by this tool.

Nevertheless, a tool like CanonSketch has significant value in specifying the architecture of complex interactive systems. Being able to generate HTML and MXML also means the notation is expressive enough to support automatic generation techniques and that it is possible to generate UI's for any platform based on GUI's and Forms like JavaSwing, Palm, Windows or MacOS.

Our usability results can be validated through laboratory evaluations such as the usability studies presented here, which corroborate our tool's usability and usefulness. However, there are some limitations to these tests, the most important being that the experiences took place in a controlled laboratory setting which does not reflect the turbulent environment of small software development companies. The results are only applicable to undergraduate students; a different population to that of professional designers. There is a lack of research in the area of UI-issues tool support in real-world industrial settings. In mitigation of this, we are planning a usability study at an international software design company in Portugal. We believe that a "glass galaxy" tool supporting these "galactic" workstyle dimensions will cause a positive impact on the practitioner's productivity and creativity.

Acknowledgements

The authors wish to thank all the users involved in the CanonSketch usability studies, for showing a deep interest and a sharp, critical look at the tool and notations involved. Also to Larry Constantine and Leonel Nóbrega, for feedback on CanonSketch and the workstyle model, and to the anonymous reviewers for helping us improve this paper in a generous, constructive way.

References

ADOBE, 2006. Flex Product white papers. Available from: http://www.adobe.com/flex.

Bouillon, L., Vanderdonckt, J., Chow, K.C., 2004. Flexible re-engineering of web sites. In: Proceedings of 8th ACM International Conference on Intelligent User Interfaces IUI'2004, Funchal, January 13–16, 2004. ACM Press, New York, pp. 132–139.

Campos, P. Nunes, N., 2004. CanonSketch: a user-centered tool for canonical abstract prototyping. In: Bastide, R., Palanque, P., Roth, J. (Eds.), Joint EHCI-DSVIS 2004 Conference Proceedings. Lecture Notes in Computer Science, vol. 3425. Springer-Verlag, pp. 108–125.

Campos, P. Nunes, N., 2005. Galactic dimensions: a unifying workstyle model for user-centered design. In: Maria Francesca Costabile, Fabio Paternò (Eds.), Proceedings of the Human–Computer Interaction-INTERACT 2005: Tenth IFIP TC13 International Conference on Human–Computer Interaction, Rome, Italy, September 12–16, 2005. LNCS, vol. 3585/2005, pp.150.

Campos, P. Nunes, N., 2005. A UML-based tool for designing user interfaces. In: UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3297, Springer-Verlag.

Campos, P., Nunes, N., 2007. Practitioner's tools and workstyles for user-interface design. IEEE Software 24 (1), 73–80.

Constantine, L., 2003. Canonical abstract prototypes for abstract visual and interaction design. In: Jorge, J., Nunes, N., Falcão e Cunha, J. (Eds.), Proceedings of DSV-IS'2003 – 10th International Workshop on Design, Specification and Verification of Interactive Systems. Lecture Notes in Computer Science. Springer-Verlag.

Constantine, L., Lockwood, L., 1999. Software for Use. A Practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley, Reading, MA.

Damm, C.H., Hansen, K.M., Thomsen, M., Tyrsted, M., 2000. Supporting several levels of restriction in the UML. In: Proceedings of UML'2000, York, United Kingdom, October 2–6, pp. 396–409.

Dimicco, J.M., Pandolfo, A. Bender, W., 2004. Influencing group participation with a shared display. In: Proceedings of CSCW'04, Chicago, USA.

Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q., 2004. KnowiXML: a knowledge-based system generating multiple abstract user interfaces in UsiXML. In: Palanque, Ph., Slavik, P., Winckler, M. (Eds.), Proceedings of 3rd International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2004, Prague, November 15–16, 2004. ACM Press, New York, 2004, pp. 121–128.

Gajos, K., Weld, D.S., 2004. Supple: automatically generating user interfaces. In: Proceedings of Intelligent User Interfaces (IUI'04), Island of Madeira, Portugal, 2004. ACM SIGCHI, ACM Press, New York.

Gould, J.D., Lewis, C., 1985. Designing for usability: key principles and what designers think. Communications of the ACM 28 (3), 300–311.

Hollingshead, A.B., 1996. Information suppression and status persistence in group decision making – the effects of communication media. Human Computer Research 23 (2), 193–219.

Iivari, J., 1996. Why are CASE tools not used? Communications of the ACM 39, 94–103.

IBM corp., 2004. Rational Rose Technical Developer.

Kiesler, S., Sproull, L., 1992. Group decision making and communication technology. Organizational Behavior and Human Decision Processes 52, 96–123.

Landay, J., Myers, B., 2001. Sketching interfaces: toward more human interface design. IEEE Computer 34 (3), 56–64.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-jaquero, V., 2004. UsiXML: a language supporting multi-path development of user interfaces. In: Proceedings of 9th IFIP Working Conference on Engineering for Human–Computer Interaction Jointly With 11th International Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004. Springer-Verlag.

Myers, B., Hudson, S., Pausch, R., 2000. Past, present and future of user interface software tools. ACM Transactions on Computer–Human Interaction 7, 3–28.

Myers, B., Rosson, M., 1992. Survey on user interface programming. In striking a balance. In: Proceedings of CHI'92. ACM Press, Monterey, pp. 195–202.

Newman, M.W., Lin, J., Hong, J.I., Landay, J., 2003. DENIM: an informal web site design tool inspired by observations of practice. Human–Computer Interaction 18, 259–324.

Nunes, N., 2003. Representing user-interface patterns in UML. In: Proceedings of OOIS'03 – 9th European Conference on Object-Oriented Information Systems, Geneva, Switzerland, pp. 142–163.

Nielsen, J., 1993. Usability Engineering. Morgan Kaufman, San Francisco, CA.

Paternò, F., 2000. Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, Berlin.

Potts, C., Catledge, L., 1996. Collaborative conceptual design: a large software project case study. Computer-supported cooperative work. The Journal of Collaborative Computing 5, 414–445.

Puerta, A., Eisenstein, J., 1999. Towards a general computational framework for model-based interface development systems. In: Proceedings of Intelligent User Interfaces (IUI'99), 1999. ACM SIGCHI. ACM Press, New York.

Puerta, A., Micheletti, M., Mak, A., 2005. The UI pilot: a model-based tool to guide early interface design. In Proceedings of the 10th International Conference on Intelligent User Interfaces, San Diego, California, USA, January 10–13, 2005. IUI '05. ACM Press, New York, NY, pp. 215–222.

Robbins, J., 1999. Cognitive support features for software development tools. PhD Thesis. University of California, Irvine, USA.

Seffah, A., Metzker, E., 2004. The obstacles and myths of usability and software engineering. Communications of the ACM 47 (12), 71–76.

Sparks, G., 2004. Enterprise Architect 4.1 User Guide.

Wu, J., Graham, T.C.N., 2004. The software design board: a tool supporting workstyle transitions in collaborative software design. In: Bastide, R., Palanque, P., Roth, J. (Eds), Joint EHCI-DSVIS 2004 Conference Proceedings. Lecture Notes in Computer Science, vol. 3425. Springer-Verlag, pp. 5–23.

van Harmelen, M., 2001. Object Modeling and User Interface Design. Addison-Wesley.

van Welie, M., Traeetteberg, H., 2000. Interaction Patterns in User Interface. PLoP, 2000.

Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., 2004. UsiXML: a user interface description language for specifying multimodal user interfaces. In: Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, 19–20 July, 2004.