

Can inspection methods generate valid new knowledge in HCI? The case of semiotic inspection

Clarisse Sieckenius de Souza^a, Carla Faria Leitão^{a,*}, Raquel Oliveira Prates^b,
Sílvia Amélia Bim^{a,c}, Elton José da Silva^d

^aDepartamento de Informática, PUC Rio, R. Marquês de S. Vicente 225, 22451-900 Rio de Janeiro, RJ, Brazil

^bDepartamento de Ciência da Computação, UFMG, Av. Antonio Carlos 6627, 31270-010 Belo Horizonte, MG, Brazil

^cDepartamento de Ciência da Computação, UNICENTRO, R. Presidente Zacarias 875, 85015-430 Guarapuava, PR, Brazil

^dDepartamento de Computação, UFOP, Campus Morro do Cruzeiro, 35400-000 Ouro Preto, MG, Brazil

Received 29 September 2008; received in revised form 8 May 2009; accepted 21 August 2009

Communicated by Prof. E. Motta

Available online 28 August 2009

Abstract

HCI evaluation methods tend to be proposed and used to verify the interactive qualities of specific systems and design strategies. A discussion about the scientific merits of such methods to advance knowledge in HCI as a field is very rare, although much needed. This paper shows that, under certain conditions, inspection methods *can* be safely used in scientific research in HCI and extend their advantages beyond the territory of professional practice. Taking the Semiotic Inspection Method (SIM) as an example, we argue that its interpretive results are objective, can be validated, and produce scientific knowledge comparable to that generated by more widely accepted methods.

© 2009 Elsevier Ltd. All rights reserved.

Keywords: Semiotic engineering; Evaluation method; Semiotic Inspection Method; Scientific application; Technical application; Communicability

1. Introduction

Interface inspections are widely used in HCI professional practice. When it is difficult to recruit users or deadlines are approaching rapidly, experts are often called in to evaluate the quality of interaction supported by a system's interface. Heuristic evaluation (Nielsen and Molich, 1990; Nielsen, 1994) and cognitive walkthroughs (Lewis et al., 1990; Spencer, 2000; Blackmon et al., 2002) are well known inspection methods. Although valuable in the industry, the results of inspection methods are generally considered and explicitly referred to as *expert opinions*. This has probably contributed to the perception that inspection methods are questionable when used to produce and validate *scientific*

knowledge. Empirical experiments and analytical evaluations based on predictive models have been much more widely accepted in this context. Even when backed by theory, most inspection methods are frequently disregarded because they can produce *interpretations* of reality, knowledge that *cannot be generalized* to predict how users interact with computer-based systems.

This paper shows that, under certain conditions, inspection methods *can* be safely used in scientific research in HCI and extend their advantages beyond the territory of professional practice. Taking the Semiotic Inspection Method (SIM) as a case, we argue that its interpretive results are objective, can be validated, and produce scientific knowledge comparable to that generated by more widely accepted methods.

SIM is a semiotic engineering method (de Souza et al., 2006; de Souza and Leitão, 2009) with which inspectors can analyze the communicability of interactive computer-based artifacts. It can be used *technically*, with the purpose of

*Corresponding author. Tel.: +55 21 3527 1618; fax: +55 21 3527 1132.

E-mail addresses: clarisse@inf.puc-rio.br (C.S. de Souza),
cfaria@inf.puc-rio.br (C.F. Leitão), rprates@dcc.ufmg.br (R.O. Prates),
sabim@inf.puc-rio.br (S. Amélia Bim), elton@iceb.ufop.br (E.J. da Silva).



improving a particular product's interface, and *scientifically*, with the purpose of discovering new knowledge to advance the state of the art in HCI.

We begin, in Section 2, with an overview of semiotic engineering and the definition of key concepts for understanding the method. Then, in Section 3, we describe SIM steps and procedures. In Section 4 we present a detailed illustration of how SIM is used to analyze interaction with a Cascading Style Sheets editor and to generate knowledge that transcends the limits of this particular application. In Section 5 we discuss SIM results with an emphasis on their validity. Finally, in Section 6 we present our conclusions.

2. Semiotic engineering and communicability

Semiotic engineering is a semiotic theory of HCI (de Souza, 2005). It views human–computer interaction as a special case of computer-mediated human communication. A system's interface is a *message* sent from designers to users about the designers' vision, how their product meets the users' needs, and what benefits and value it can ultimately bring to the users' lives. A special characteristic of the designers' message is that it is *performative*—it enacts and unfolds its content as users interact with it. Through interaction, the performative message reveals to users even the communicative principles that must be used to communicate back with the system in order to achieve various goals and effects. For example, EclipsePalette[®] is “a color palette and screen color capture program designed not only for web designers, but Visual Basic, C++, C#, and Java developers as well.” (<http://www.greeneclipse.com/eclipsepalette.html>). In Fig. 1 we see its full interface, a small 282 × 294 window with no menus, a few push buttons and tabs (e.g. the ‘HTML’ tab, the ‘RGB()’ tab, and the ‘VB’ tab at the bottom right corner of the window), an input/output text box (showing the HTML code of the dark blue color on the large square form on top of the window), manual controls for RGB values input, and numerous ‘palette swatches’, as the designers call them, for selection and drag-and-drop operations.

As it stands statically in Fig. 1, EclipsePalette's interface communicates some of the designers' intent, but not all of it. In its designers' own words, EclipsePalette “features a drag-and-drop interface, as well a color mixer, and manual

color adjustment tools. You can select colors from a pre-built palette, or build your own custom palette. Also, EclipsePalette stays next to your clock, saving space and reducing clutter.” (<http://www.greeneclipse.com/eclipsepalette.html>) The entirety of the designers' message can only be grasped if the user *interacts* with the program, exploring the behavior and affordances of interface controls. Interaction will eventually *tell* users the meaning of  and , and help them see the value of the program in practice.

Because most of the designers' communication through the interface is meant to tell users about how and for what purposes to communicate with the system, in semiotic engineering interactive systems are defined as *metacommunication* artifacts. A central concept for the theory is *communicability*, “the distinctive quality of interactive computer-based systems that communicate efficiently and effectively to users their underlying design intent and interactive principles” (Prates et al., 2000, p. 32). In our terms, efficient and effective communication is simply communication that is organized and resourceful (*efficient*), and achieves the desired result (*effective*).

Communication is achieved through *signs*. A sign has been defined as anything that someone can take to stand for something else in some respect or situation (Peirce, 1992, 1998). The definition underlines the fundamental role of the *interpreter* in establishing what constitutes a sign. On the one hand, going back to Fig. 1, the unlabeled grey sections at the bottom of the EclipsePalette window may be taken as signs by one user but not by another. For example, one user may take them to stand for drop-down areas, just like other grey rectangles in the interface (see the different palette swatches in the figure), whereas another user may not see any signification in the unlabeled grey sections (which, in fact, are no more than decoration).

On the other hand, the role of the *interpreter* in Peirce's definition of a sign also points to the openness of what signs can mean. For example, when the user hovers the mouse over the ‘Copy color’ button, she gets a tool tip saying: “Copy the color listed at the right to the clipboard.” This sentence is a sign, which users may interpret in different ways (i.e. it may stand for different things). One user may think that the color itself, say dark blue, will be *copied* to the clipboard and then be *pasted* onto a polygon, for instance, as a fill color. Another user may think that the color code, say “#0A246A”, will be copied and then possibly pasted into a color specification slot for further interpretation. Unless the users put their interpretations to test, they will not really know what this particular tool tip means. Consequently, when asked how EclipsePalette works in this respect, they will say what they *anticipate*. Anticipations may not only differ from one user to the other, but they may also differ from the designers' intended meaning, of course. Sometimes users will find out very quickly that their anticipated meanings differ from the designers'; they will try to apply them while interacting with the system and the system will not behave as expected. However, it is also possible that anticipated meanings that



Fig. 1. EclipsePalette interface.

are indeed different from the intended ones survive long periods of interaction—the difference may even go undetected forever. For example, because when users click the ‘☒’ box on top of the interface window they typically exit programs, they may infer that the way to exit programs in general is to click on ‘☒’. Some programs will capture the event, show a proper exiting dialog, and terminate normally. Other programs, however, will terminate in exceptional conditions. They may, for instance, leave intermediary object representations in the computer memory, occupying a considerable amount of space. In this case, the user’s interpretation of ‘☒’ as ‘exit the program’, although pragmatically fit for the user’s purpose, is different from the designers’ intent and potentially harmful for the operating system and the program. The user, however, may never realize the difference.

Interpretation, in Peircean Semiotics, is described as a process where interpreters assign meanings to signs by virtue of habitual associations or hypothetical reasoning, called abduction (Peirce, 1992, 1998). The role of interpretation in communication is further explored by Eco (1976), whose theory sheds light on critical aspects of HCI. According to Eco, communication is a process in which humans explore a variety of signification systems to produce signs (which may belong to such systems or not) and achieve an unlimited range of goals and effects. A signification system is the result of regular and culturally-established associations between expressions and contents. So, for example, the designers of EclipsePalette use different signs to communicate their intent that users will pick colors of objects displayed on the entire screen (and not just in the program’s interface window). On their website, they say explicitly that EclipsePalette is a “screen color capture program”. Through the program’s interface, they use the eyedropper icon as a metaphor for making the user indicate the color she wants to use (she picks a drop of the desired color). The small size of EclipsePalette’s interface plays an important role in communicating the fact that the user can (and should) work with colors *outside* the interface, on the rest of the screen. As shown in Fig. 2, as soon as the eyedropper tool is activated (by pressing the button with the eyedropper icon on it), the cursor takes the form of an eyedropper and

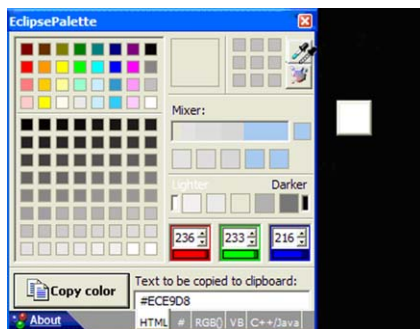


Fig. 2. Active eyedropper tool in EclipsePalette’s interface.

a colored square box appears *outside* the limits of the program interface. The color of the square box is the same as that of the area under the tip of the eyedropper. As the user moves the cursor, the square box moves too, changing its color according to the color of the region under the tip of the eyedropper. If the user places the tip outside the limits of the EclipsePalette window (even if accidentally), the color of that screen region will be *picked* as well. So, by ‘prompting’ the user to explore the screen with the cursor, and giving instant feedback on color selection and color coding, the designers of this program are communicating their interactive intent through interaction itself.

The signs used by EclipsePalette designers are drawn from different signification systems. On their website, they use English text. On the interface they also use English text on labels and tool tips (e.g. the tip for the ‘Copy color’ button discussed above), but their main signification sources are visual signs. They also combine visual signs from different systems. The tool for selecting colors is represented by the image of an eyedropper, which is itself a metaphor of the intended meaning of the tool. And the possibility of picking colors anywhere on the screen is represented by a combination of the behavior of the eyedropper and square box compound and the small size of the program’s interface window. Notice that whereas the signification of color selection is already a convention in graphic editor interfaces, the signification of screen color capture possibilities contains novel ingredients. And the interpretation of novelty always involves abductive reasoning processes.

An important aspect of abduction is that unlike deduction and induction it contains (and relies on) self-correction procedures. Abductive inference starts with a fact, for example: when the user clicks on ‘☒’ in the EclipsePalette window, the program interface is closed. The user may then take this fact to be the result of a hypothetical rule: if one clicks on ‘☒’, then one exits the program. This hypothetical rule is tested against immediately available evidence, for example: after clicking ‘☒’, not only is the program interface closed, but it also does not figure on the list of active program windows that show up when one cycles through active programs pressing Alt + Tab in the Windows operating system interface. The success of few tests may lead the user to conclude that the hypothetical rule is true, and that a click on ‘☒’ causes EclipsePalette to terminate.

The rule will typically persist until counterfactual evidence is encountered. For example, the user may realize that after clicking on ‘☒’, the EclipsePalette logo still shows on the system tray “next to the clock, saving space and reducing clutter.” This self-correcting step in the user’s abductive chain will not only bring the user’s meanings closer to the designer’s intent, but it will possibly lead to further meaningful hypotheses about how EclipsePalette works and how it can be used (e.g. EclipsePalette can start automatically when Windows is started, to have it always at hand).

In this semiotic perspective, meaning, as we see, is a process rather than an abstract concept. It is impossible to predict the exact path that each individual will follow to achieve a satisfactory (but not necessarily *correct*) interpretation of a given sign. The interpretive process is halted for pragmatic reasons (e.g. lack of resources, lack of need, and lack of interest) and resumed if and when needed. Hence, using predictive methods to investigate the *communicability* of interactive artifacts, or aiming to find predictive principles as a result of research, contradicts the essence of the phenomenon under investigation. In other words, empirical evidence of how designers express what they mean through interface signs and of how users interpret such signs in various contexts cannot be generalized and transformed into interpretive *laws*. Although there are interpretive *trends* stemming from the very cultural conventions at the origin of signification systems (Eco, 1976), communication processes produce signs that are especially expressive precisely because they contradict aspects of the signification system where they originate (e.g. jokes and irony can communicate the opposite of what is being said), or because they are completely innovative signs (i.e. they do not belong to any established signification system, and thus signify something perceived as different from everything else). Cultural conventions therefore indicate expressive and interpretive *probability* in communication, but they do not constrain communicative *possibilities*.

The aim of semiotic engineering inspections is to investigate communicative possibilities with reference to the theoretical foundations presented above. Our goal is to produce a detailed account of how systems designers communicate their design to systems users through interface and interaction signs. Given this account, researchers may decide which related HCI phenomena can be consistently investigated with predictive methods and models associated with other scientific theories and paradigms.

Before we proceed to present the Semiotic Inspection Method, it is important to recall very briefly the original definition of sign types often invoked by HCI researchers and professionals: icons, indices, and symbols. These types, or sign classifications, were established in view of three universal categories in Peircean phenomenology (Peirce, 1992, 1998, pp. 145–159). Phenomena of the first category, plainly called *firstness*, are perceived in and of themselves as a **unary quality** independent of anything else. The experience of *firstness* is usually identified with our senses and feelings. Phenomena of the second category are perceived as **binary** relations. The experience of *secondness* is usually identified with sensorial or mental associations (e.g. spatial and temporal co-occurrence, affective association, cause and effect, etc.). Finally, phenomena of the third category are perceived as **mediated** relations, forming a triple $\{\alpha, \beta, \chi\}$ where α is related to β by means of χ . The experience of *thirdness* is usually identified with mental operations such as learning and interpretation (e.g. the

English word ‘cat’ is related to ‘a particular class of felines’ by mediation of linguistic rules known to English speakers). Simplified definitions of these categories often state that: *firstness* is the category of sensorial experience; *secondness* is the category of associative experience; and *thirdness* is the category of reasoning and conventions.

Icons, indices and symbols are different types of signs defined in terms of which phenomenological category associated with the referred object is evoked by the representation being used. *Icons* are signs that evoke the *firstness* of the referent object, some quality that impacts our senses and signifies the object by virtue of perceptual properties. In HCI icons are usually said to ‘resemble’ their referent object. This definition, however, excludes many other sign instances that Peirce defines as icons. *Indices*, in their turn, are signs that evoke the *secondness* of the referent object and signify the object by virtue of immediate associations. For example, the representation may take the form of the effect caused by the referent object (consider using a slender body silhouette to represent a weight-control diet), the location of the referent object (consider using a church to represent a religious service), etc. Finally, *symbols* are signs that evoke the *thirdness* of the referent object and signify the object by virtue of mediated associations and relations. The representation stands for its referent object because there are conventions, regulations or inferential rules supporting the signification of the latter by the former (consider using a gift to signify appreciation, using certain clothes and manners to signify psychological attitude or social status, etc).

Icons, indices, and symbols are fundamental sign classes in Semiotics. Semiotic engineering, however, has its own specific object of investigation, namely the designer–user metacommunication through computer-based artifacts. Thus, it requires sign classes rooted in this specific semiotic phenomenon. In other words, semiotic engineering sign classes should help investigators analyze the nature, the structure, the process, the effects, and the conditions of computer-mediated human communication where one communicating party (the designers) is speaking through the very message being communicated (the systems interface and all the interactive patterns it supports). For this reason, in semiotic engineering we use three different sign classes: *static* signs, *dynamic* signs, and *metalinguistic* signs. The classification refers to the interactive conditions that express the representation of the sign and thus enable its signifying structure.

Static signs are those whose representation is motionless and persistent when no interaction is taking place. These representations can be perceived (and interpreted) in snapshots of the system’s interface before or after interaction occurs. All signs in Figs. 1 and 2, for example, are *static* signs. Although we interpret them in view of the interactions that have led to them or that may follow, they can be expressed *statically*.

Dynamic signs are those whose representation is in motion regardless of users' actions or whose representation unfolds and transforms itself in response to an interactive turn. They can only be entirely actualized over time, and lose their substance outside the temporal dimension. For example, the sign used by EclipsePalette designers to signify color selection across the entire display screen is a *dynamic* one. The snapshot in Fig. 2 represents only a momentary state of the actual sign that represents the possibilities envisioned by the designers. As explained when discussing the behavior of the eyedropper tool, the designers' communication is achieved by coupling the eyedropper shape to the cursor, keeping a square box at constant distance from the cursor as it moves across the display, and painting the square box with the same color as that of the region under the eyedropper tip. Together, these features *represent* color selection possibilities, and they can only be fully achieved and perceived over time.

Finally, *metalinguistic* signs, as their name suggests, are signs that represent other static, dynamic, or metalinguistic signs. Representations of metalinguistic signs depend on the separation between two representational levels: one where the action is performed and the other where information, instructions, descriptions, or explanations *about* the action are provided. These levels may be accessed by specific types of interaction (e.g. pressing a certain key to get help), or they may be co-present in the same space and time (e.g. there may be embedded tips in the interface to help the user interact with the system). In EclipsePalette, for instance, when the user clicks on the 'About' link, a dialog box is opened and the user can ask for *help*. Help communication involves signs like the ones shown in Fig. 3. Signs like 'type the actual RGB values', 'sliders below the boxes', 'color swatches', and the like are *metalinguistic* because they represent interactive elements and actions that belong to a separate level of interaction.

The users experience with interactive systems is deeply affected by the way in which designers combine static, dynamic, and metalinguistic signs in order to communicate their vision to users. The content of metacommunication thus achieved can be paraphrased by an instantiation of a metacommunication template (de Souza, 2005) with which designers essentially tell users the following:

“Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it in

order to fulfill a range of purposes that fall within this vision.”

In the next sections of this paper, we will show how these concepts are used in different steps of a semiotic engineering inspection method and the kinds of results that can be achieved with it.

3. An overview of the Semiotic Inspection Method

The primary purpose of SIM is to evaluate the *communicability* of interactive computer artifacts. SIM focuses on user interface meanings expressed *by design*, and takes the perspective of metacommunication senders. In other words, it is centered on the *emission* of the metacommunication message. Another Semiotic Engineering Method, the Communicability Evaluation Method (Prates et al., 2000; de Souza and Leitão, 2009) complements SIM and focuses on the *reception* of the metacommunication message by users.

As is typical of HCI inspection methods, SIM does not require an observation of users interacting with the system. Rather, the method helps inspectors anticipate the kinds of consequences that design choices may bring about when users interact with the system. It can be carried out by a single inspector or a group of inspectors.

In both technical and scientific contexts of use, SIM is remarkably epistemic. By inspecting the efficiency and effectiveness of the designers' communicative strategies, the analyst using it to improve the design of a specific product will increase his technical knowledge of how well (and/or in which interactive contexts) different means and modes of expression communicate design intent. Likewise, HCI researchers using it to identify and explore different kinds of potential communication with/through computer technology will increase their scientific knowledge with an enriched account of their object of study, and/or with formulations of new research questions and challenges related to it.

SIM is an interpretive method applied in five core steps. The first three steps are iteratively carried out, and the last two steps can be carried out in one pass each. In all the five steps the inspector is analyzing signs, but the class of signs under analysis and the analytic operations in each step are substantially different. The first three steps *deconstruct* the design message by means of a strictly segmented analysis of distinct classes of signs, whereas the last two steps *reconstruct* the design message by integrating and interpreting deconstructed signs into an overall designer-to-user communication schema.

Like other inspection methods, SIM requires a preparation step before we begin the deconstruction and reconstruction procedures. Additionally, in scientific contexts of application, SIM requires a final validation step to qualify its results, which is optional in technical contexts. As is the case with interpretive methods used in research, the validation step is a triangulation of results (Cresswell, 2009;



Fig. 3. EclipsePalette help message.

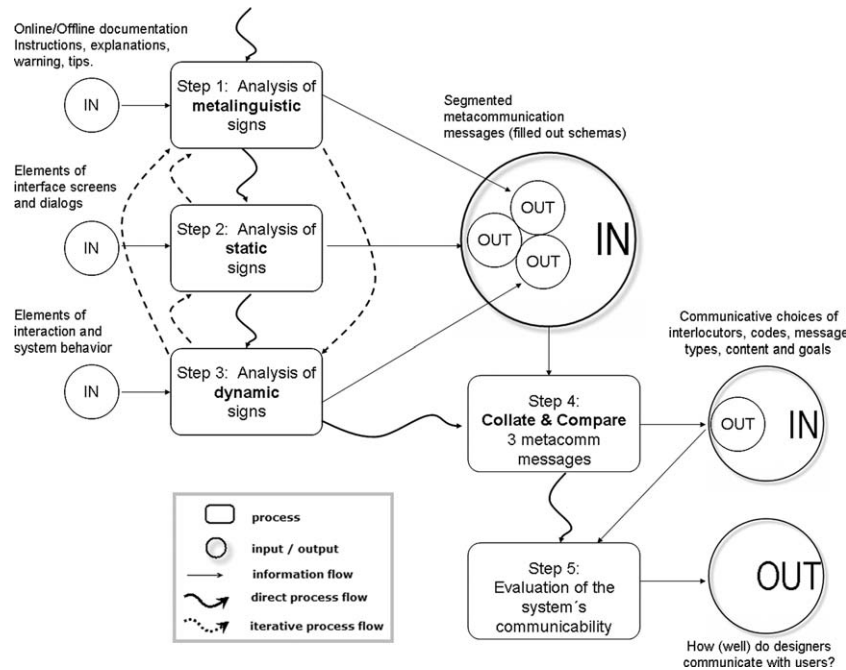


Fig. 4. The five core steps of the semiotic inspection method.

Denzin and Lincoln, 2000). Preparation and validation are general methodological steps. The real essence of SIM is what happens in its five core steps, shown in Fig. 4:

- **Step 1**- the analysis of metalinguistic signs;
- **Step 2**-the analysis of static signs;
- **Step 3**-the analysis of dynamic signs;
- **Step 4**-a comparison of the designers' metacommunication message generated in the previous steps; and
- **Step 5**-a final evaluation of the inspected system's communicability.

Preparation is carried out in four sub-steps. First, the inspector defines the purpose of the inspection, given the specific context where the method is being used. In scientific contexts, researchers must choose an application instance that serves the purpose of investigation. In technical contexts the application is given. The second step is to do an informal inspection of the system in order to define the intended focus of the evaluation. The analyst reads summarized project documentation or promotional material associated with the object of inspection (websites, information on the product's package, quick guides, and the like). The third step is to navigate through the system, tracing evidence to confirm the intended users of the system and the top-level goals and activities that the system supports. Finally, in the fourth step, knowing the purpose of the analysis, the interlocutors involved in communication, what contexts are significant, and what types of effects communication should achieve, the inspector elaborates *inspection scenarios*¹ that provide

the necessary contextual structure required for analyzing communication.

In technical contexts of use, where the main purpose of inspection is typically to inform and improve the professional design and development of a specific system, the first preparation step typically involves conversations with its designers and developers. Talking to other stakeholders, like the system's owners and members of the user population, will give the inspector additional interpretive clues to be used in the core steps of the method. In scientific contexts of use, however, where the main purpose of inspection is typically to identify and explore HCI research issues, talking to the various stakeholders may be more useful for triangulating SIM results. Instead, the first preparation step should produce a clear statement of research goals, as well as a careful examination of how SIM results can contribute in achieving them. The other three preparation steps are the same in both technical and scientific contexts of use.

After preparation, come the five core steps of the method. As already mentioned, the first three of these steps correspond to a *segmented* analysis of signs—each step focuses on a single class of signs (metalinguistic, static, and dynamic). These steps are also *iterative* because given the very wide spectrum of possible interactions that systems usually enable, inspectors are likely to encounter in core step 3—while analyzing dynamic signs—metalinguistic or static signs that escaped him in previous steps. This will cause him to go back to core step 1 or 2 to revise

(footnote continued)

performed. Scenarios explicitly and implicitly indicate the context of the action, and have a central character (the inspector, playing the role of the user) that performs the action implied by the narrative.

¹Based on the definition of Carroll (1995), these scenarios are narratives about one or more characters and a set of focused activities that will be

his findings in those steps. At the end of each of these steps, the inspector must be able to express his *segmented* interpretation by filling out an instance of the metacommunication schema shown on page 5.

The **analysis of metalinguistic signs** is achieved when the inspector finishes his examination of all the metalinguistic signs encountered while running various interactive possibilities related to the inspection scenario. The input for this step, as shown in Fig. 4, is the collection of signs appearing in online and offline documentations, as well as online instructions, explanations, warnings and error messages, tips, and the like. The output of this step is an instantiated version of the metacommunication template, based solely on meanings expressed by metalinguistic signs.

The template may have gaps and ambiguities in it. For example, by relying solely on metalinguistic signs the analyst may not be able to produce a clear characterization of who the designers think the users are. Many help systems and online documentation resources focus almost exclusively on how-to instructions, omitting important information about why the system was built in one way instead of another, what user needs the designers were trying to meet, what benefits the users are expected to have if they adopt the system, etc.

The **analysis of static signs** is achieved when the inspector finishes his examination of all the static signs encountered while running various interactive possibilities related to the inspection scenario. The input for this step, as shown in Fig. 4, is only motionless elements appearing statically and persistently on various screens and dialogs of the system. The output is an instantiated version of the metacommunication template, based on meanings expressed by static signs only.

Again, the analyst may not be able to complete or disambiguate his interpretation of the designers' message by looking solely at static signs. For example, some meanings may only be communicated by dynamic signs. The analyst should rely on the method to be able to isolate his reading and interpretation of each class of signs. So, even if preceding steps have led him to complete and consistent interpretations of extensive portions of the designers' metacommunication message, he must start anew at each of the three deconstruction steps of the method and reinterpret systematically the signs that contribute to the instantiation of the corresponding metacommunication template. This is crucially important for achieving the reconstruction steps appropriately.

The **analysis of dynamic signs** is achieved when the inspector finishes his examination of all the dynamic signs encountered while running various interactive possibilities related to the inspection scenario. The input for this step, as shown in Fig. 4, is a collection of signs whose representation unfolds over time. The output is an instantiated version of the metacommunication template, based solely on meanings expressed by dynamic signs.

Once again the analyst may find gaps and ambiguities resulting from his segmented analysis. The effort of reading

and interpreting dynamic signs as units, separable from the backdrop of metalinguistic and static signs that concurrently integrate the whole of a system's interface, is crucially important.

In core step 4, we **collate and compare** the results of segmented metacommunication analysis. This is one of the key steps of the method because in it the analyst can capture important elements for the final step of analysis. First, the analyst must decide whether the templates instantiated at the end of each step are *consistent* with each other. In other words, the analyst must look for communication conveyed by one class of signs that is contradicted by communication conveyed by another. Second, the analyst must detect if gaps in the instantiated template at the end of one step are filled out by meanings represented in the instantiated template of another step. If they are not, the emission of metacommunication is incomplete *by design*, and the analyst must investigate what effects are achieved by such incompleteness. Third, the analyst must examine the distribution of metacommunication message components across sign classes. Message components appearing in all three templates of segmented analysis achieve redundancy and mutual reinforcement in communication. Message components appearing in a single template rely heavily on the users' ability to identify and interpret the corresponding class of signs appropriately.

Other kinds of results can be achieved in addition to the ones just mentioned. Specific domains of applications (e.g. computer games), specific kinds of technologies (e.g. mobile devices), or specific user populations (e.g. users with motor skills disorder) may prompt the analysts to examine specialized dimensions while comparing metacommunication produced by different classes of signs. For example, communication achieved through metalinguistic signs may require more screen space and multiple object manipulations than is possible in mobile devices. Likewise, if the purpose of the system is to challenge users, some kinds of communication may be intentionally ambiguous. Also, motor skills disorders usually mean that physical interaction with the system is costly. Therefore, the role of static signs for accessibility purposes may be different than in interfaces designed for the average user.

In core step 5, we finally **evaluate the communicability of the system** by reconstructing a unified metacommunication message. This allows the analyst to assess the adequacy, the costs and the benefits of communicative characteristics, and the strategies identified in previous steps. This is the time to evaluate the effects of inconsistencies, gaps, and lack of redundancy in metacommunication, if found in step 4. It is also the time to detect the adequacy of communicative strategies for special domains of application, special technologies, and special user populations, for instance.

As discussed in Section 2, every interactive system contains unique metacommunication ingredients, resulting from unique sign combinations used by its designers to convey a design vision that is specific to that particular

system. The analyst's account of the system's metacommunication in this final core step of the method is thus *new knowledge*. Hence the epistemic nature of SIM alluded to at the beginning of the current section. The analyst learns new facets and possibilities of metacommunication every time he inspects a new system. At this step, the analyst also uses his background to frame his systematic interpretation. In doing so, he unveils unknown associations between phenomena or gives new meanings to already known problems.

In professional practice and technical contexts of application, this characteristic of the method is an important asset, because it continually expands the competence of the analyst. New analytic dimensions, opportunities, challenges, effects, elements, and contextual factors involved in the semiotic engineering of human–computer interaction are constantly added to the analyst's stock of knowledge.

In research practice and scientific contexts, new knowledge generated at the fifth core step of the method can only be added to the stock of knowledge in the discipline after validation. We adopt validation procedures and criteria used by qualitative research methods in general (Cresswell, 2009; Denzin and Lincoln, 2000) and proceed to a **triangulation of results**. In this final step, we can use endogenous and/or exogenous sources. Endogenous sources refer to the same design artifact or artifacts that share the same domain model. That is, we can triangulate our interpretation with what other analysts and interpreters, in different kinds of contexts, say about the same artifact as we are inspecting, or about artifacts of the same sort. Exogenous sources refer to design artifacts that do not share the same domain model, yet share certain relevant design features. That is, the artifacts whose interpretations are being triangulated can do completely different things, but they must have something in common that is directly related to the interpretation we are trying to validate.

The goal in triangulation is to promote diversity in the conditions of knowledge discovery and to expose our findings to contradiction. Commonly used diversification techniques include: having different analysts use the same method and contrast their results; using different methods to analyze the same data; and invoking different theories to explain (aspects of) the findings. Findings are validated when triangulation procedures generate consistent (although diverse) knowledge. When inconsistent interpretations are found, interpretation processes start again. In other words, self-correction takes place as part of ongoing abductions. New hypotheses are formulated and the analyst engages in a new series of abductive inferences.

As was the case in step 5 of SIM itself, triangulation procedures adopted to validate the results of interpretive methods continually expand the analyst's stock of knowledge, but now in a different direction. Whereas in step 5 the analyst gains new knowledge about analytic dimensions, opportunities, challenges, effects, elements, and contextual factors involved in the semiotic engineering of human–

computer interaction, in the final validation step the analyst gains new knowledge about his own process of interpretation and how this process is connected to other processes of knowledge discovery. Triangulation with exogenous sources is a particularly powerful means of, both, validation and knowledge discovery. It forces the analysts to frame problems in more abstract terms and to explore invariants across diverse (yet related) contexts.

In Section 4 we report a case study where SIM was used for scientific purposes. For sake of conciseness and legibility, only the relevant findings and results in each step will be presented, along with selected illustrations of supporting evidence.

4. Case study

The case study reported below was carried out for scientific purposes in order to illustrate all the steps and sub-steps involved in SIM. The primary object of this study was *Simple CSS (SCSS)*, a Cascading Style Sheet editor freely distributed by HostM.com (<http://www.hostm.com/css/>). SIM was jointly applied to SCSS by two inspectors (two of this paper's co-authors). Inspections were carried out in August–September, 2008.

4.1. Preparation

Sub-step a—Purpose of the inspection: The research context where the inspection of SCSS was carried out is an investigation of how a system's feedback is communicated to users, what kinds of signs are used, what interpretations they trigger, how these interpretations relate to the system's semantics. Feedback is a critical element in human–computer interaction. It supports all users' interpretations and decisions about interactive goals and opportunities, and to a very large extent determines novice users' learning cycle.

Sub-step b—Informal inspection of a chosen system: SCSS was chosen for a number of reasons. First, it supports a specification task, which represents a particularly interesting case for semiotic analysis. Second, it figures on the list of CSS editors recommended by W3C in their official Cascading Style Sheets page (<http://www.w3.org/Style/CSS/>). Third, as its name suggests, SCSS is a *simple* tool, which is easy to learn with only a few interactions. And, last but not the least, the object of specification supported by SCSS (a style sheet) causes distinctive visual effects on other objects (web pages viewed with a browser) that constitute the ultimate target of the user's activity.

Sub-step c—Focus of the inspection: After visiting the SCSS website, inspectors concluded that HostM very strongly advertises the technical advantages of using CSS technology when developing websites (ease of maintenance, less data transfers, and faster page downloads). After downloading, installing, and using SCSS for 20 min or so, inspectors also conclude that HostM strategy is to help

non-professional web developers to use CSS technology by concentrating on the definition of styles, rather than on the **codification** of the CSS files required for appropriate visualization of HTML pages with web browsers. Style definition is easily carried out with point and click interaction. CSS codification is automatically generated by *SCSS* when the user presses a button.

There are many aspects to evaluate in the *SCSS* interface, but given our context of research we decided to focus on feedback signs generated after style-defining actions. In other words, we are examining feedback presented by the system when the user defines that the font family of text tagged with `<p>` is ‘Verdana’, for instance, but we are not examining feedback presented by the system when the user saves a project or deletes a style.

Our inspection scenario includes the characterization of a non-professional web developer and expands the social context of use, so that we can plausibly accommodate what seems to be a contradiction—being a ‘casual’ web developer (which is the only reason why a user would choose a tool that does not support CSS code editing), and having a good reason to go through the trouble of using CSS technology instead of styling text spans directly into the HTML code.

Sub-step d—Scenario of the inspection: The inspection scenario elaborated by inspectors was the following:

“Valerie teaches English Composition in a Community College. The College has its own web server and requires that teachers and students use it to host their school-related websites. Valerie has a lot of experience using the Internet for professional and personal purposes, and she has even managed to learn HTML by herself. Lately, she has read a number of articles about the advantages of using Cascading Style Sheets (CSS), especially as a substitute for frames. She was surprised to learn that people with impaired sight, for example, can use their own CSS to view a page designed with different CSS. Because she has a student with severe sight problems, she is motivated to use CSS in the English Composition website. The problem is that she doesn’t have much time to learn CSS syntax, but her friend Oscar says she shouldn’t worry. *Simple CSS* is an easy-to-use CSS editor. She can use it without having to learn CSS code. So, Valerie downloads *Simple CSS* and starts using it to build the new website. She doesn’t expect to find difficulties. Her pages typically have a header, a left menu, and a main area. She hopes that she can use CSS to give the pages a uniform look, and also allow students with sight problems to use their own CSS to read the pages.”

4.2. SIM steps

Step 1: The analysis of **metalinguistic signs** allows us to segment metacommunication and identify important elements in the designers’ discourse about *SCSS*. Because we



Fig. 5. The ‘Source’ tab in Simple CSS.

are focusing on feedback issues, we will only present metacommunication message contents that are related to this focal point.

Online documentation in HostM.com web site is very scarce. Minimal ‘how-to’ instructions are given on the *Simple CSS* documentation page, and the *Simple CSS* overview page talks more about the advantages of CSS (ten lines of text) than the advantages of *SCSS* (six lines of text). Also, tool tips are not used on screen. However, error messages and dialogs in general tend to be a little more informative. Because ‘point and click’ definitions typically do not cause errors and follow-up dialogs are not needed, metacommunication achieved with metalinguistic signs is virtually non-existent. The only exception is the content of the ‘Source’ tab, shown in Fig. 5. Although the signs on screen take the same form as other static signs (which will be discussed shortly), they are truly metalinguistic.

The label on top of the text area says: ‘This is what your exported CSS style sheet will contain’. In the text area we see CSS code, including color-coded syntax. This is the **only** explicit representation of CSS **codification** to be found in all interactions with *SCSS*. Thus, the designers’ message is clear: style definitions are not to be made by writing CSS code. The Source tab contents are just ancillary information about the effects of users’ interactions with the rest of the interface.

Step 2: The analysis of *SCSS* **static signs** is more interesting and informative for our research purposes. In Fig. 6 we see a screen shot of *SCSS* main screen. The ‘point and click’ style of interaction is clearly communicated. The values for a style being defined are typically selected from a list of pre-defined options (appearing in drop-down lists or presented as labeled check boxes). There are very few instances in all style-defining tabs where the user has to type in text. In Fig. 6, next to ‘List Style Image’, at the bottom of the tab, there is a disabled text entry box. If the selected value is set to ‘URL’ instead of ‘Unchanged’ (the default value), the box is enabled and the user must type in the URL where the list style image is to be fetched.

The most important static sign for the purposes of this study is shown at the bottom of the screen. In a scrollable area the system communicates that ‘This is an instant preview’ of the selected style (as currently defined, we anticipate). At this point, the whole spectrum of communication achieved by this sign is hardly evidenced. Because all style-definition parameters are set to ‘unchanged’ values, the previewed text is really a representation of ‘unstyled’ text tagged by `<body>` in HTML. If parameter

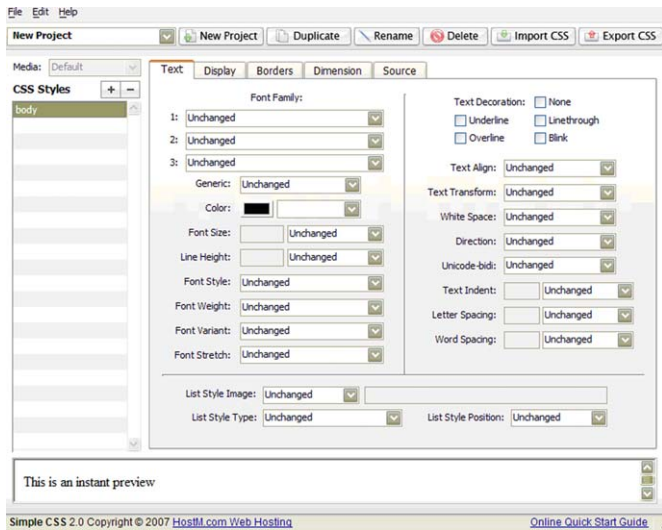


Fig. 6. Main screen of Simple CSS.



Fig. 7. Source tab contents (CSS code) after 5 styles are added.

'1:' of 'Font Family' is set to 'Courier', in the preview area the user will see "This is an instant preview". However, this sort of communication is only fully conveyed through synchronous changes in elements during interaction, with the support of dynamic signs.

Another curiosity of static signs analysis is that until the user chooses a non-default value to define a style, the user does not see any metalinguistic signs in the text area of the 'Source' tab. The CSS source content is null. This particular communication is difficult to understand for a novice, especially if the user has already selected a number of styles (for example, 'body', 'h1', 'h2', 'p' and 'a', as shown in Fig. 7). Why shouldn't she see syntax like 'body { }; h1 { }; h2 { }; p { }; a { };'? The only feedback the system communicates is the list of five styles on the left-hand side of the screen. They are there, in SCSS. Will they be there in HTML pages if a CSS file is generated (exported)? At this point, metacommunication about feedback is confusing. It will only become clearer when we examine dynamic signs.

Step 3: The analysis of **dynamic signs** yields even more interesting results. We realize that the preview area supports crucially important feedback communication. In addition to confirming the effectiveness of the user's decisions (as exemplified with the font family change mentioned while discussing SCSS static signs), this sign also achieves a metalinguistic purpose. For instance, if the user does not know the meaning of a given parameter value—say, 'monospace' or 'cursive' for the 'generic' font

family definition—the instant preview sign actually demonstrates the meaning of the value. The problem is that certain value changes cause no change in the preview text.

At this stage of analysis the lack of metalinguistic signs explaining CSS technology becomes hampering. Although the text preview sign supports productive communication in many cases, in other cases they are totally ineffective. For example, if a particular parameter value is set to 'inherit' (a context-sensitive style rule), the preview text (whose context, or scope, is always just local to the style) shows a very misleading representation of what the style may look like on a web page.

Although dynamic signs in SCSS support the 'trial and error' strategy illustrated with EclipsePalette in Section 2, miscommunication gets in the way of the user's learning process. First, the CSS code feedback in the Source tab proves to be open for certain kinds of editing operations like 'select', 'cut', and 'clear'. So, once the user learns the regular correspondences between style definitions established interactively and the code showing in the Source tab, she may well choose to change certain definitions by editing the CSS code directly. For example, suppose that she discovers that she has made a mistake using the 'inherit' value in the definition of a given style, and she wants to eliminate certain features from the parent style and transfer it to the child style. This is a simple cut and paste operation on the code, which is actually much more laborious to carry out with 'point and click' interaction. Because she can select the parent style features in the Source tab text area and cut it, she may think that a direct code editing strategy will work and help her be more efficient. She will be surprised, however, to see that unlike in other cases the text preview sign does not reflect the editing just made. Even more surprisingly, although she can manage to position the cursor where she wants to paste the portion she cut, 'paste' commands are completely ineffective, both through menu selections and shortcut keys. The system is communicating that something is going wrong. But the real surprise comes when the user clicks on another tab after having cut portions of the CSS code and then returns to the Source tab. The code has been restored to its previous stage—the text she cut out of the parent style definition is back into place. In other words, the Source tab should contain output-only signs, but some input–output features are signified in it.

Although this dynamic feedback problem might be considered an easy-to-fix programming bug, there is more to it than meets the eye as we will see in the next steps. We take this problem to communicate serious design ambiguity that relates both to system feedback signs, on the one hand, and to a poor definition of who the users are (in our terms, a poor definition of the designers' interlocutors in metacommunication).

Step 4: As we **collate and compare** metacommunication conveyed by metalinguistic, static and dynamic signs, focusing on system feedback messages, we notice that the *simplicity* of SCSS may have come *at the expense of*

consistency. In the first core step of analysis, metalinguistic signs suggest that CSS codification is not a topic of designer-to-user metacommunication. Style codification is replaced with style definition, and we are led to believe that CSS code is completely irrelevant for designers and users of SCSS. As we move to the analysis of static signs, we realize that although there seems to be appropriate feedback for communication about style definitions in the instant preview area at the bottom of the screen, CSS code is introduced as a topic of conversation. In fact, the sign used to communicate this idea is of the same class as other signs communicating style-definition conversations—an interface *tab*.

Although this topic of conversation appears during static signs analysis, there aren't other signs communicating how users can engage in conversation about CSS codification (no links to CSS tutorials, no code-related dialogs or menus, no hints about what the user can do with the code or about the purpose of showing it). In other words, there is no redundancy in this respect between static and metalinguistic signs, and metalinguistic communication is poorly distributed (and arguably poorly expressed as well). In the subsequent stage of analysis, dynamic signs help the users learn CSS code by, first, observing systematic cause-effect relations between synchronized interactive style definitions and code appearing in the Source tab, and, second, making generalizations from observed situations. Here, dynamic signs complement metalinguistic communication consistently. The efficacy of feedback expressed in the Source tab is arguably more significant than that of the feedback expressed in the instant preview area (although possibly not more meaningful or understandable to the user). For example, whereas values like 'inherit' have no synchronized expression in the instant preview area, they have very clear expression in the CSS code area. Although users may not know what 'inherit' really means on styled HTML documents, they can see that the definition has been made. Possibly, some curious users will build dummy HTML pages to test hypothetical meanings of 'inherited' styles, following an abductive line of reasoning.

Furthermore, as seen above, certain editing operations are allowed in the CSS code text area. However, only one or two attempts are necessary for the user to realize that user-system conversations *about CSS code* are virtually non-existent. On the one hand, the system helps users to learn some CSS coding by communicating style-definition feedback with three interrelated kinds of signs: icons (instant text preview), indices (cause-effect associations between parameter setting and instant text preview, on the one hand, and CSS code, on the other), and symbols (CSS grammar and coding). But on the other hand, not many users can do it in SCSS with the code they have learned. In other words, some dynamic and static feedback signs are pointing in directions that are incongruent with the designers' message expressed through metalinguistic signs.

Step 5: At the final **communicability evaluation** step of analysis, we notice that designers have apparently been unable to decide whether they were talking to users who will or will not (want to) learn CSS code. In our scenario, Valerie, the central character, may eventually learn the basics of CSS coding and be tempted to edit it directly. However, if she tries to do it in SCSS, she will be frustrated. SCSS leverages the users' knowledge but does not give them the opportunity to benefit from it *within* the scope of interactions it supports. *Expert SCSS* users have to interact with the system very much in the same 'point and click' style as novices. And, if they want to reap the benefits of their learning, they must use another CSS editor.

At this stage, the analyst is led to frame the problem in broader terms, generating new knowledge not only about SCSS but also about HCI design in general. The presence of feedback signs evoking *thirdness* is the origin of the miscommunication we detected. Therefore, it is tempting to eliminate them from the interface and communicate feedback through signs evoking *firstness* (in this case the visual appearance of styles) and *secondness* (in this case the systematic associations between parameter value changes and synchronized visual appearance changes). The problem is, as we have already mentioned, that certain signs of *secondness*, namely *indices*, breakdown because they have no visual effect out of context.

On closer examination the problem, however, is considerably more complex. The *icon* chosen to represent styles is not adequate to the task. The way it signifies styles is by assuming that the selected style is applied exactly to the scope of 'This is an instant preview' (like this: `<style> This is an instant preview</style>`). This choice of *firstness* is misleading in a number of cases. For example, when `<p>` is styled as a justified paragraph, the quality of justification cannot be perceived within the scope of 'This is an instant preview'. The same is true for `<table>` and many other HTML tags, especially those referring to structures (like lists and divisions, for example). A more appropriate *icon* would be a default HTML page containing at least the most frequently used tags. Only then would the preview text really communicate aspects of *firstness* in the referent object of this representation. But the object of the iconic representation is, in fact, a cascading *code* (HTML code structurally linked to CSS code).

The corollary of the analysis carried out so far is that the notion of a *Simple* CSS editor is almost a contradiction in terms. Because the referent object of editing is a set of symbolic representations (CSS code) that affect yet another set of symbolic representations (HTML code), choosing appropriate signs of firstness and secondness to communicate all the ranges of available actions and relevant feedback is extremely difficult, if not impossibly complex. The 'point and click' strategy leads us to question *what object* is being pointed at. In SCSS the objects that can be 'pointed at' are style 'names', 'attributes', and 'values'. Feedback is communicated by means of a 'prototype

object' (a fixed sentence whose appearance changes when certain values are set). But the sentence *is not* the ultimate object that the user wants to affect, and in addition it *cannot* express the whole range of effects caused by 'point and click' actions performed by the user. Certain parameter values do not affect the appearance of the sentence. A strict '*what you see is what you get*' style of feedback would require the choice of a prototype object that is a good metaphorical representation of generic HTML pages. Because this metaphor is difficult to find, SCSS designers have chosen a metonymical representation (they use one part to represent the whole). But the metonymy may lead users to make wrong inferences about the meaning of style definitions that the prototype object cannot represent. So, using only signs of firstness and secondness is a risky strategy for metacommunication about referent objects that are in fact specifications for other objects. Specification tasks are fundamentally symbolic in nature, and can only be achieved through signs of thirdness.

SCSS designers have realized the importance of signs of thirdness in the task they want to support. They show the Source code of the CSS file in the interface. However, they do not allow users to use thirdness signs to communicate desired specifications back to the editor. Through their style of metacommunication, they help SCSS users learn a considerable share of CSS codification, but in order to *test* their learning and *use* it productively in style sheets tasks, users must move to another program. Thus, SCSS becomes a transition tool, a scaffold, something to be used and left behind as soon as users gain some expertise. We do not know if this is the designers' intent, but according to our analysis, this is what they are communicating.

In technical contexts of use, the findings achieved with SIM may be useful to redesign SCSS. They stimulate the designers to think about communicative strategies related to feedback. Moreover, by bringing semiotic knowledge to bear in the analysis of sign classes (associated with Peirce's definitions of firstness, secondness, and thirdness), these findings may also generate knowledge for scientific research about system feedback requirements and possibilities. Specifically, SIM findings suggest that a combination of iconic, indexical, and symbolic feedback signs referring to the *same* referent object or action may be more epistemic *to the users* (i.e. they may help users gain new knowledge and skills). In order to validate this suggestion and turn it into legitimate scientific contribution, we must triangulate SIM results with other endogenous or exogenous sources (see page 8) of knowledge related to the results we achieved with SIM.

4.3. Triangulation

We triangulated SIM results with empirical evidence provided by both endogenous and exogenous sources. The endogenous source was web material content about SCSS itself and other freely distributed CSS editors (users' opinions, FAQ's, publicity, and the like). The exogenous

source was the result of a semiotic inspection of a portion of *Google Groups* (<http://groups.google.com.br>). We used discourse analysis techniques (Seidman, 1998) to analyze Web material content, and used SIM to inspect *Google Groups*. We now highlight only the main results of endogenous and exogenous triangulation.

A discourse analysis of web material about SCSS and other equivalent CSS editors, provided concrete evidence that novice SCSS users initially like the editor but do not find it as useful once they have reached a certain level of technical knowledge. We also found evidence that designers of equivalent CSS editors think that users should be able to manipulate CSS code directly, even if they know very little about it when they start to use the editor. Here are excerpts of discourse providing such evidence (underline shows especially expressive portions of the evidence). The first three show us SCSS users' perspectives, and the last two tell us the perspectives of the designers of similar editors about the users' abilities and expectations.

"Simple CSS is a great little application from the folks at Hostm.com It is more aimed at the beginners to CSS as it allows you to pic [sic] from a list of common elements and style them with ease, you can also make your own just as easily. The application is free so it's worth checking out." [2]

"Not all selector attributes or properties are available, limited options and customization, when they called it Simple CSS they meant exactly that; it's simple and therefore limited, it's a little buggy at times, not the best or ideal tool for complex or elaborate projects, good tool for beginners to learn with but not all that useful after that[...]." [3]

"I use a lot of CSS in my website design.

This app works as listed and is easy to use.

If it had a preview function like CSS Edit has I would rate it higher.

It doesn't work directly on CSS files, it imports them and then you have to export them to CSS..." [4]

{From the developers of EclipseStyle} *"Sure, a few style sheet generators are out there, and they are perfectly fine, if you never want to make any changes, already know the names and values of all of the CSS attributes, and love working with a grid of 100 lists. EclipseStyle puts other style sheet editors to shame with its features such as a preview window, intuitive tabbed interface, and much more. It even includes an introduction to style sheets if you are unfamiliar, as well as complete tutorials and references to the EclipseStyle interface. EclipseStyle also features a code window for seeing what your style looks like in good old CSS code, [...]"* [5]

²<http://noble-designs.org/main/workflow-and-productivity/7-applications-to-make-developing-on-a-pc-easier>.

³http://www.download.com/Simple-CSS/3000-2068_4-10349478.html?hhTest=1.

⁴<http://www.versiontracker.com/dyn/moreinfo/macosx/25884&page=2>.

⁵<http://www.greeneclipse.com/eclipsestyle.html>.

{From the developers of A Style} “A Style is a visual CSS editor.

CSS (Cascading Style Sheets) allows to separate the representation definitions of such structured documents as HTML, XHTML, XML from their content, which is an effective principle in the design of web sites. Key features:
 * Visual easy-to-use interface; * Graphic tree-type view of attachment files and the CSS structure; * Grouped view of properties and selectors; * Automatic selection and grouping of CSS selectors from a markup language document; * Source CSS, HTML, XML highlight code editor” [6]

The semiotic inspection of *Google Groups* was carried out in September–October of 2008 by three authors of this paper, those that were not involved in the inspection of *SCSS*. The focus of their inspection, like that of *SCSS*, was the system feedback for user actions. For practical purposes, we decided to constrain our analysis to the context of a crucial short-time activity in *Google Groups*: the configuration of the group by the group owner. This activity has the advantage of allowing inspectors to investigate feedback about certain group communication processes (e.g. what feedback does the system show to users if messages they post are visible to anyone in the Internet compared to being visible by only a limited group of users), without requiring that they participate in actual group discussions for a long period of time. Also, the activity is directly comparable with the specification tasks examined with *SCSS*.

The analysts carried out their inspection separately but discussed their findings together in order to produce an enriched and integrated conclusion about *Google Groups* communicability, in the last step of SIM. Their separate analyses were *per se* an internal triangulation of results. Nevertheless, they looked for further validation of their conclusions among evidence spontaneously provided by users in the *Google Groups Help Forum*.

The inspection scenario was the following:

“Walter is a teacher in the Telecomm Department. He is preparing his term course that starts within a few days. A colleague from the Computer Science Department commented that she had a good experience using *Google Groups* to support class discussions and distribute extra material to the students. Walter decides to try and do the same. Because he has never used *Google Groups*, he plans to create a new group, with all students in his class (whose emails he already knows), but not to publicize the group and launch its activities till he has finished designing the main web pages and uploading course material for the first 4 weeks of class. In this way he will be able to explore *Google Groups* beforehand, and make sure that the system will really help him work online with the students. If all goes well, he will tell the students about

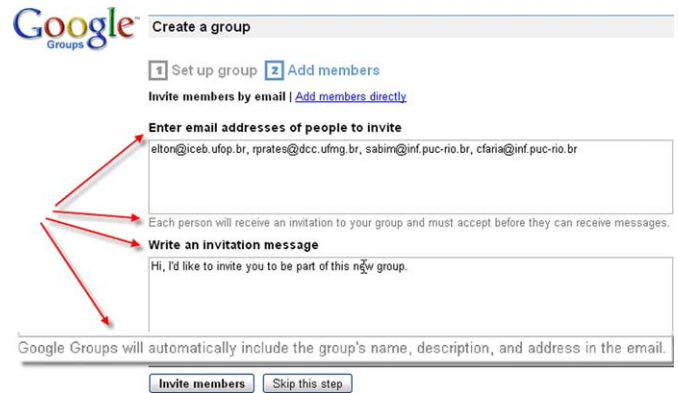


Fig. 8. Source Who is inviting the group members?

the online group on their first day of classes. Only then he will publicize the group and launch the activities.”

The semiotic inspection showed that *Google Groups* communicability is problematic especially because of insufficient or occasionally inconsistent explanations and feedback about the meaning of interface signs. We illustrate communicability problems with only two instances of miscommunication. One has been selected among the results achieved at the end of step 5 in SIM, and the other has been selected from the results achieved while triangulating SIM results with endogenous sources of knowledge about *Google Groups*.

First, when creating a group, users are told that they can invite members. When the group owner invites a member, an invitation message is sent to this person. Static and dynamic signs in the interface lead group owners to believe that *they* are sending a message to the invitee and writing the invitation text (see Fig. 8), although metalinguistic signs hint discretely to something different. The duplicity of grammatical subjects in such phrases as ‘{YOU} Invite members by email’, ‘{YOU} Enter email addresses’ and ‘{YOU} Write invitation message’ versus ‘GOOGLE GROUPS will automatically include the group’s name, description, and address in the email’ creates confusion about *who* is actually writing and sending the invitation to members. The metalinguistic (explanatory) sign at the bottom of Fig. 8 communicates that *Google Groups* acts as a mediator between the owner of the group and the members, but the static signs on screen (text box labels and links) support the interpretation that the owner is sending communication directly to the invited members (‘write an invitation message’).

The owners cannot preview the invitation message, and they would very probably be surprised to read what it says. Not only does the text unexpectedly introduce the *Google Groups* perspective on what the group owner is doing (see the reported speech style of the message content below), but it also communicates the possibility that the invited member is a victim of electronic abuse (which of course would not be part of the message sent by the owner). This note of suspicion about the group owner is

⁶<http://www.athlab.com/Astyle/index.html>.

not in tune with the otherwise friendly and trustful tone of interaction between group owner and *Google Groups*. The message sent when the user presses the ‘Invite members’ button shown at the bottom of Fig. 8 is the following (formatting and URL’s abbreviated for lack of space):

serg.group1@gmail.com has invited you to join the SERG-Group1 group with this message:

Hi, I’d like to invite you to be part of this new group.

Here is the group’s description:

This group is created to test how the system works.

--Google Groups Information--

You can accept this invitation by clicking the following URL:

<http://groups.google.com/> [...]

--If This Message Is Unwanted--

If you feel that this message is abuse, please inform the *Google Groups* staff

by using the URL below.

<http://groups.google.com/> [...]

Lack of feedback (like a preview of *Google Group*’s message to members) reinforces the owner’s belief that *he* is communicating with his invitees, in the terms he chooses. Actually, without feedback, owners may never realize that *Google Groups* is saying something else in their behalf (and something else about them). Because their role in the system is not that of a regular member—they are the owner—they do not have access to the members interface, view, and experience of the system.

These and other metacommunication inconsistencies regarding feedback were found when the three analysts collated and compared the metacommunication templates reconstructed in steps 1, 2 and 3 of SIM (the analysis of metalinguistic, static, and dynamic signs). Analysts also found feedback problems with decisions regarding the configuration of group privacy and communication structure. Their findings led them to conclude that lack of feedback and/or inconsistent metacommunication in *Google Groups* may lead to important communication breakdowns among members of groups supported by this system. In particular, because group owners do not really know the extent and consequences of their decisions, the inspectors concluded that they may inadvertently cause considerable sociability problems for the group.

At the triangulation step, inspectors found an eloquent piece of evidence of the extent of miscommunication in this system’s interface. When they examined *Google Groups Help Forum*, they found a message from a user asking what she should do to attach a ‘how-to’ page to the invitation she was sending to her friends. The answer she got clearly points to how inconsistent the users’ interpretation of interface signs may be compared to the system’s semantic model. The answer says:

“You cannot attach anything to an Invitation to a Google group. You actually do not sen[d] the Invitations, you request that Google Groups sends them, so you act[ua]lly

*never see the invitations e-mails and so cannot attach anything to them.”*⁷

Insufficient feedback and inconsistency between signs can cause yet more trouble. For example, when creating a group, users can decide whether the group is private, public, or restricted. Explanations about what these terms mean are provided on screen, but static, dynamic, and metalinguistic signs generated in the sequence of interaction may be inconsistent with the user’s choices. We found evidence of the bewilderment experienced by users who did not spot this inconsistency. A user posted the following message in *Google Groups Help Forum* ⁸:

“I thought that in a restricted group, one had to have a password and name (email address) in order to get into a restricted Google group. I am an owner of the India 101 Peace Corps 2008 Google group. After telling everyone in our PC group to join, that it was restricted and safe to put private information there, I found while checking on things at my sisters place in Seattle, that all I had to do was look up Google groups, and then type in India 101 and the site opens right up. No password, no restricted Google group, one feels very vulnerable, misled, and distrustful. Now I have to alert everyone and shut it down. [...]”

Moreover, the inability to anticipate how the owner’s decision will affect other members and non-members leads users to think about laborious strategies to test the meaning of group configuration parameters. Two posts in the Help Forum explicitly address this point. One user asks:

*“Can someone tell me what the email delivery option ‘Users decide where their replies are sent’ means? I would test it but I’d like to avoid spamming the group with my test emails.”*⁹

Another user reports the strategy he used to test how private his group was:

*“I have a “private” group, that is, I have set all the access options to be most private. I have uploaded a file. When I view this file on my computer, when logged in with my Google account, I get a very long URL in the browser. If I send that URL to my wife, who is neither logged in nor a member, she can see the contents of the file, at least for some time.”*¹⁰

These and other *Google Groups*, triangulation results not only reinforced the inspectors’ conclusions about issues with

⁷http://groups.google.com/group/Managing-Your-Group/browse_thread/thread/1cfeab1b20eeebc/6352b2372ed26734?lnk=gst&q=inviting#6352b2372ed2673.

⁸<http://groups.google.com/group/Google-Groups-Basics/topics>.

⁹http://groups.google.com/group/Managing-Your-Group/browse_thread/thread/d22ce18542a3171f/83c42e173870a2c0?lnk=gst&q=Users+decide+where+their+replies+are+sent#83c42e173870a2c0

¹⁰http://groups.google.com/group/Google-Groups-Basics/browse_thread/thread/d48cca980f5d015/295bf34cacc3abb9?lnk=gst&q=Is+a+private+group#295bf34cacc3abb9.

metacommunication discourse in this system but also extended the inspectors' knowledge about the consequences of the problem. The final results of a semiotic inspection of *Google Groups* configuration tasks showed that group owners have to face serious communication problems, with the system and with the group members, mainly because of two communicability problems. First, metalinguistic signs are not always effective and consistent in helping users make group configuration decisions. Second, users have the need to test the effects of their own communication with the system, and with other users, *outside* the context of the task that they are trying to achieve using elaborate role-playing strategies to find out what the system 'means'. *Google Groups* provides virtually no feedback for group owners at the group configuration stage. Hence, some users are angry to realize that *Google Groups* does not work as they 'thought' it would, while others try to probe the meaning of their decisions for other users by asking them, by using their machines and accounts, by creating *alter egos* with different roles in the group, etc.

This points to a feedback issue that is actually more complex than it looks. In order to probe the implications of group configuration decisions, group owners should be able to *simulate* the communicative and social processes affected by such decisions. In other words, not only should they be able to put themselves in the *role* of group members, public visitors, etc., but they should also be able to appreciate how their decisions influence group processes in different contexts. For example, it may not suffice to be able to preview what the owner's invitation message will look like to an invitee. Probably, it would be better to let owners experience (through signs of firstness) what happens in different situations: What if the user wants to decline the invitation? What if the user wants to ask further questions before accepting the invitation? What if the user wants to add a note to the group owner in the act of accepting the invitation?

The use of *alter egos* (or different 'personas') to test what *Google Groups* really means by interface signs is costly for users and fortuitous. A systemic solution, however, requires complex computations, capable of simulating social processes triggered by online communication, for which special sociability and communicability models are necessary.

5. The nature and value of case study results

Having carried out a semiotic inspection of *SCSS* with the purpose of finding new scientific knowledge about the communication of system feedback in HCI, we now use the results of the semiotic inspection of *Google Groups* to validate our conclusions and express what we can learn from semiotic inspections in the context of scientific research.

A comparison between results achieved with *SCSS* and *Google Groups* inspections reveals that, although completed by different inspectors and pertaining to totally

different domains, both have much in common. To begin with, the importance of feedback evoking the perceptible qualities that a configurable object may acquire (be it a web page or an online community) as a result of different parameter value choices is backed by empirical evidence presented in both studies. These perceptible qualities are effectively signified by 'iconic' signs, that is, signs that bring up the *firstness* of their referent. In *SCSS*, an example of such sign category is a prototype HTML page, where styled elements correspond to the CSS specifications. In *Google Groups*, however, the equivalent of such iconic signs would be the unfolding of prototype social processes demonstrating online the qualities associated with various parameter settings.

The systematic association between parameter values and prototype object qualities, as seen in preceding sections, is a sign of *secondness*. It supports an important learning process, by which users become skilled in anticipating the correct effects of using conventional symbols (namely, the attribute values that different type of parameter can take) upon the ultimate object that they want to configure. Once they dominate the symbolic representations that must be used to achieve their specific configuration goals, users can be said to have learned a conventional *configuration language*, an unmistakable sign of *thirdness*.

In *SCSS* signs of *thirdness* are explicitly communicated through the interface (in the list of values that can be chosen for different types of parameter, which are reproduced in the Source tab code that users can view). In *Google Groups*, however, the conventional stance of parameter values is not clearly expressed. On the one hand, there are certain values (like 'public', 'private', and 'restricted') that appear in different parts of the configuration process, suggesting that they belong to a configuration *code*. But on the other hand, unlike style sheet codification that refers to mostly structural qualities of static objects, group configuration decisions refer to mostly procedural qualities of dynamic objects like computer-mediated communication, social interaction, online security and trust, etc.

Representing these with signs of *thirdness* would amount to providing a *simulation language*. If users could learn this language and edit group process specifications directly in this language, they would eventually be in a position to *program* group processes. In other words, designers would have entered the territory of end user programming. We should remark at this point that whereas *SCSS* users end up learning a considerable part of CSS codification, and gain enough expertise to *program* style sheets directly, *Google Groups* users cannot do the same. There is no conventional language for group process configuration as there is for HTML style configuration. And of course because such language is available for *SCSS* designers, they can use it to provide feedback in style sheet configuration tasks. *Google Groups* would have to design this language if they wanted to give similar kinds of feedback for group owners while configuring group

processes. But the challenge for *Google Groups* designers would be some orders of magnitude greater than with SCSS. Appropriate feedback for group configuration processes, reinforcing the perceptual qualities they refer to and the systematic associations between selecting conventional symbols and causing such qualities to be present in group processes, involves dynamic referent objects. Moreover, these dynamic object referents include a role structure whose configuration affects their final behavior. Thus, the semiotic engineering of feedback signs of firstness, secondness, and thirdness in group support systems is very complex. So, we are not surprised to see that these signs are virtually absent from the system we inspected.

The main scientific contribution of our study is thus to provide a new account of a known problem (de Souza and Leitão, 2009). In essence, we have found that feedback provided by systems supporting specification and configuration tasks of type-level referent objects (i.e. objects whose final instantiation depends of factors that are not defined, and possibly not definable, at specifications and configurations time) requires three kinds of signs. Symbolic signs are necessary to express the conventions by which certain perceptual qualities of actual object instances are named and defined. Iconic signs are necessary to express what these perceptual qualities ‘mean’ when present in actual object instances. And finally, systematic associations between conventions and perceptual qualities have the effect that the presence of certain specifications and configurations indicate (i.e. are indices of) perceptual qualities, and vice versa. Indices play a fundamental role in abductive reasoning processes that eventually make users learn a signification system. We have also found that iconic type-level representations of static referent objects like HTML pages are very difficult to produce. Metaphors and metonymies used to express the qualities being defined might not always exhibit the entire scope of dimensions affected by the values assigned to specification or configuration parameters. An additional unexpected finding of our study was that when specification and configuration tasks refer to dynamic type-level objects like processes, providing iconic and indexical feedbacks involve simulation procedures, and thus becomes an order of magnitude more complex than feedback for static type-level objects. If processes involve different roles, as was the case of group processes in *Google Groups*, than the communication of feedback becomes critically difficult.

Interface feedback is the object of voluminous research work in HCI, of course. However the way in which SIM has allowed us to frame it, with the connections we have been able to establish, is new. To the best of our knowledge, the simultaneous articulation of feedback challenges with representation systems that support abductive reasoning, token- and type-levels iconic signs, iconic representation of processes, and end user programming language issues has not been made to date. Previous work in programming by demonstration and other end user programming techniques based on artificial intelligence

(Cypher, 1993; Lieberman, 2001), for instance, has discussed representational issues and system feedback for learning, but has not systematically analyzed the types of signs that can be used in representations systems, the expressive and epistemic functions they can support, etc. Likewise, previous work in computer-supported learning, as that by Sedig et al. (2001), for instance, has explored the role of interface languages as scaffolds in learnware. Although their results can be cast in semiotic terms (de Souza and Sedig, 2001), they were not generalized to the extent our case study has allowed us to do. Finally, previous work on cognitive dimensions of representation systems (Blackwell and Green, 2003), has produced insightful results for both HCI and end user programming activities (Blackwell, 2006), but some important aspects of the metaphors and metonymies that can be used to express feedback in type-level specification and configuration tasks have not been addressed.

We can thus say that, as is the case with other qualitative methods, SIM induces the generation of *new knowledge*, and opens the avenue for new kinds of research. Interpretations produced during semiotic inspections are rigorously systematic and can be validated by means of triangulation. Therefore, although they refer to situated instances and contexts of human–computer interaction, they are a solid platform for abductive conclusions, tested against various kinds of evidence collected in the triangulation process.

6. Conclusion

In the beginning of this paper we said that, in spite of considerable resistance encountered in the HCI community, non-predictive inspection methods *can* be safely used in scientific research and extend their advantages beyond the territory of professional practice. We also argued that interpretive results of qualitative methods are objective, can be validated, and produce new scientific knowledge comparable to that generated by more widely accepted methods.

After presenting a detailed illustration of the use of SIM, an inspection method proposed by semiotic engineering, we now address very briefly some methodological issues that are often invoked against the use of inspection methods in scientific research.

The first issue is the fact that inspection methods are not good because they do not involve the users, and do not examine empirical ‘evidence’ of human–computer interaction. A semiotic engineering perspective shows, however, that the *human* side of HCI is shared by users and designers alike. Both are brought together at interaction time: designers communicate their design intent to users through the system’s interface, and users communicate back with the system according to the message they get. So, an inspection of the material that designers produce seems to be a critical requirement for a comprehensive analysis of the users’ behavior while interacting

with computers. In other words, inspection methods have a precise role in HCI research, and the semiotic engineering inspection method presented in this paper amounts, in fact, to an observation of evidence produced by *human behavior*, although not by *user behavior*.

The second issue is whether knowledge generated by SIM is objective, and not just an expression of the inspectors' subjective judgment and attitude. As shown in our case study, there is no doubt that SIM does not generate neutral knowledge. As other non-predictive interpretive methods (like, grounded theory (Glaser and Strauss, 1967), for instance), the semiosis of HCI practitioners or researchers is by necessity the basis of the abductive process on which SIM relies. But, if on the one hand each interpretation is unique and not replicable, on the other, SIM interpretation processes are systematic and strongly committed with and determined by the semiotic engineering ontology. As in other theory-driven methods, a set of concepts guides the inspection and provides traceability to interpretation results. Moreover, SIM inspections refer to objective and identifiable entities: computer interface signs. Its analytical steps focus on metalinguistic, static and dynamic signs that can be traced by other inspectors (and also by users). Hence, although unique, the systematic interpretation process involved in SIM contains fundamental verifiable elements stemming from the ontology of semiotic engineering as well from the objective interface signs. These elements provide an objective grounding for interpretation processes, and prevent us from confusing them with subjective *expert opinions* about the inspected system.

A third methodological issue often invoked against inspection methods is generalization. Can knowledge obtained with interpretive methods be generalized in order to predict certain HCI phenomena? Knowledge generated by interpretive methods in general and by SIM in particular cannot be generalized. Qualitative methods belong to a non-predictive and interpretive territory of science. In it, knowledge is conceived as a unique situated construction, referring to specific contexts of interpretation. Thus, results achieved with SIM in one context are not 'replicable' in other contexts. Therefore, results cannot be used to predict HCI phenomena. Non-predictive methods like SIM value abductive reasoning as a scientific knowledge-discovery tool (Santaella, 2004), whereas predictive methods tend to value deductive and inductive reasonings, which support generalizations and produce solutions that can be replicated in a wide range of contexts. Abductive reasoning, however, can provide epistemic seeds to 'unpredictable' and potentially innovative paths in scientific discovery.

A related methodological issue is the perception that because their results cannot be generalized, qualitative methods cannot advance scientific knowledge, even though they can be validated through triangulation. As we showed in Section 5, the level of abstraction that the very process of triangulation (especially exogenous triangulation) leads

researchers to achieve while looking for invariants and relations among the diversity of evidence is very high. So, although not generalizable, the results of qualitative methods can be widely applicable. SIM gives us the opportunity to appreciate this feature of qualitative methods in considerable depth.

In view of the above, and given the facts of SIM demonstrated in our case study, we argue that this method is fit for scientific research as well as for technical purposes centered around the improvement of specific system designs.

We would like to finish this paper by addressing a challenge often proposed to semiotic engineering as a whole, and hence also to the use of SIM in HCI research. It is about whether or not designers actually intend to communicate design rationale to users, and consequently about the validity of evaluating this sort of communication regardless of what the designers consciously want to do, or think they are doing.

In his early formulation of the science of design, Simon (1996) defines design as a purposeful activity. He says: "everyone designs who devises courses of actions aimed at changing existing situations into preferred ones" (p. 129). Although intentionality is there, the idea of communication in/of design is not. It comes through more explicitly in later work, like Rheinfrank and Evenson's for example, who talk about design languages (Rheinfrank and Evenson, 1996), tracing this tradition back to Christopher Alexander (1977) and others famous designers. According to Rheinfrank and Evenson (1996) "natural languages are used to generate expressions that communicate ideas; design languages are used to design objects that express what the objects are, what they do, how they are to be used, and how they contribute to experience" (p. 68). Simon (1996) comments that designers synthesize (or create) artifacts "not always or usually with full forethought" (p. 8). Likewise, Rheinfrank and Evenson (1996) note that design languages are often used unconsciously, but "when consciously understood, developed, and applied, design languages can build on and improve this natural [creative] activity, and can result in dramatically better interactions, environments, and things of all kinds" (p. 65).

In this light, we believe that we can safely assume that designers do want the users of the artifacts they produce to understand and experience the benefits of their design. Thus, a number of design features are deliberately meant to suggest and promote, to express and communicate, such experience and benefits. More recently, Norman also revised some of his previous views on HCI, stressing the importance of viewing design as communication (Norman, 2004, 2007)—a perspective that was deemphasized in his early formulations of user-centered systems design (Norman, 1986, 1988, 1999).

A more fundamental question, however, is about the validity of evaluating the quality of communication that may not be fully conscious. Establishing the limits of conscious decision-making, however, especially

a posteriori, when the product of design is deployed, is a difficult task. For example, suppose that a team of HCI designers consciously decides to allow users to reconfigure a system's interface at their discretion. Thus, users can change the original *default* interface into radically different interface versions, resembling only remotely what the original design looked like. Can we say that the designers *intend* that users will be able to create customized interfaces that cancel or change so much of their original design to the extent that it may become unrecognizable?

Philosophical debates about intentions in action and communication (Bratman, 1987; Cohen et al., 1990) show that both 'yes' and 'no' answers have their advocates. Very briefly, we can say that 'yes', they do, because they *intend* (or *mean*) to let users reconfigure the interface freely. So, although they might not expect (or might never have thought) that anyone would change a text editor interface in such a way that it would look like a form layout editor, one of the consequences of their conscious design choices and decisions is that such customizations are possible. On the other hand, we can say that 'no', they don't, because turning a text editor into a form layout editor destroys the very identity of the original application. It does not make sense to *customize* the system to the extreme of depriving it from agile access to its most basic and fundamental functionality, and thus the designers most probably do not *intend* (or *mean*) to communicate *that*.

Asking the designers about their design intent will necessarily lead them into rethinking the whole decision-making process, and elaborating (now) on the reasons for their choices (then). The whole spectrum of meanings that a designer can assign, at any point in time, to his design decisions cannot be fairly conflated into a single stable and definitive expression. The more they think about it, the greater the chances that they will find new meanings in what they did. So, although by asking the designers we will obtain empirical evidence of design intent, we will not obtain unquestionable empirical proof that a certain range of meanings was intended *at design time*.

One of the most interesting features of SIM is its ability to shed light on 'unconscious' communication implied by the designers' choice of interface signs and interactive patterns. Such unconscious communication affects the users experience as much as conscious communication does, and the method systematically leads inspectors to examine communication *with reference* to design intent, but not *limited to* intentful communication. This is probably the main mechanism leading inspectors to produce knowledge that is truly *new*, knowledge that is implicit in interactive computer artifacts and can be appropriated by users in indefinitely many ways.

Acknowledgments

Clarisse de Souza thanks CNPq and Raquel Prates thanks FAPEMIG for financial support to do this research. Silvia Bim thanks CNPq for a Ph.D. scholarship.

References

- Alexander, C., 1977. *A Pattern Language*. Oxford University Press, New York.
- Blackmon, M.H., Polson, P.G., Muneo, K., Lewis, C., 2002. Cognitive Walkthrough for the Web. In: CHI 2002 Proceedings, vol. 4, no. 1, ACM Press, pp. 463–470.
- Bratman, M.E., 1987. *Intentions, plans, and Practical Reason*. Harvard University Press, Cambridge.
- Blackwell, A.F., 2006. Ten years of cognitive dimensions in visual languages and computing. *Journal of Visual Languages and Computing* 17 (4), 285–287.
- Blackwell, A.F., Green, T.R.G., 2003. Notational systems-the cognitive dimensions of notations framework. In: Carroll, J.M. (Ed.), *HCI Models, Theories and Frameworks: Toward a multidisciplinary science*. Morgan Kaufmann, San Francisco, pp. 103–134.
- Carroll, J.M. (Ed.), 1995. *Scenario Based Design: Envisioning Work and Technology in System Development*. John Wiley and Sons, New York.
- Cypher, A. (Ed.), 1993. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge.
- Cohen, P.R., Morgan, J.L., Pollack, M.E., 1990. In: *Intentions in Communication*. Massachusetts Institute of Technology, Cambridge.
- Cresswell, J.W., 2009. *Research design: Qualitative, quantitative, and mixed methods approaches*, second ed. Sage Publications, Thousand Oaks.
- de Souza, C.S., 2005. *The semiotic engineering of human-computer interaction*. MIT Press, Cambridge.
- de Souza, C.S., Leitão, C.F., Prates, R.O., da Silva, E.J., 2006. The Semiotic Inspection Method. In: *Proceedings of the Seventh Brazilian Symposium of Human Factors on Computer Systems (IHC'2006)*, vol. 1, Porto Alegre, SBC, pp. 148–157.
- de Souza, C.S., Leitão, C.F., 2009. *Semiotic Engineering Methods for Scientific Research in HCI*. Synthesis Lectures Series. Morgan & Claypool, San Francisco.
- de Souza, C.S., Sedig, K., 2001. Semiotic considerations on direct concept manipulation as a distinct interface style for learnware. In: *IHC2001-IV Workshop de Fatores Humanos em Sistemas Computacionais*. SBC, pp. 229–241.
- Denzin, N.K., Lincoln, Y.S. (Eds.), 2000. *Handbook of Qualitative Research* Second ed. Sage Publications, Thousand Oaks.
- Eco, U., 1976. *A Theory of Semiotics*. Indiana University Press, Bloomington.
- Glaser, B.G., Strauss, A.L., 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine de Gruyter, Hawthorne, NY.
- Lewis, L. Polson, P., Wharton, C., Rieman, J., 1990. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In: *CHI '90 Proceedings*. ACM Press, pp. 235–242.
- Lieberman, H. (Ed.), 2001. *Your Wish is My Command: Programming by Example*. Morgan Kauffman, San Francisco.
- Nielsen, J., 1994. Heuristic evaluation. In: Nielsen, J., Mack, R.L. (Eds.), *Usability Inspection Methods*. John Wiley, New York.
- Nielsen, J., Molich, R., 1990. Heuristic evaluation of user interfaces. In: *CHI' 90 Proceedings*. ACM Press, pp. 249–256.
- Norman, D.A., 1986. Cognitive engineering. In: Norman, D.A., Draper, S.W. (Eds.), *User-Centered System Design*. Laurence Erlbaum, Hillsdale, pp. 31–61.
- Norman, D.A., 1988. *The Design of Everyday Things*. Basic Books, New York.
- Norman, D.A., 1999. Affordance, convention and design. *Interactions* 6 (3), 38–42.
- Norman, D.A., 2004. Design as communication. <http://www.jnd.org/dn.mss/design_as_comun.html>. Last visited in Set 2008.
- Norman, D.A., 2007. *The Design of Future Things*. Basic Books, New York.
- Peirce, C.S., 1992, 1998. Houser, Nathan, Kloesel, Christian (Eds.), *The Essential Peirce*, vols. I and II, Indiana University Press, Bloomington.

- Prates, R.O., de Souza, C.S., Barbosa, S.D.J., 2000. A method for evaluating the communicability of user interfaces. *ACM Interactions* 7 (1), 31–38.
- Rheinfrank, J., Evenson, S., 1996. Design languages. In: Winograd, T. (Ed.), *Bringing Design to Software*. ACM, New York, pp. 63–85..
- Santaella, L.B., 2004. *O Método Anti-Cartesiano de C. S. Peirce*. UNESP/FAPESP, São Paulo, Brasil.
- Sedig, K., Klawe, M., Westrom, M., 2001. Role of interface manipulation style and scaffolding on cognition and concept learning in learnware. *ACM Transactions on Computer–Human Interaction* 1 (8), 34–59.
- Seidman, I., 1998. *Interviewing as Qualitative Research: A Guide for Researchers in Education and the Social Sciences*. Teachers College Press, New York.
- Simon, H., 1996. *The Sciences of the Artificial*. The MIT Press, Cambridge.
- Spencer, R., 2000. The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. In: *CHI 2000 Proceedings*, vol. 2, no. 1. ACM Press, pp. 353–359.