

Finding Boundary Objects in SE and HCI: An Approach Through Engineering-oriented Design Theories

Andrew Walenstein
Center for Advanced Computer Science
University of Louisiana at Lafayette
walenste@ieee.org

Abstract

This paper outlines an approach of using engineering-oriented design theories to build bridges between software engineering and human-computer interaction. The main thrust of the approach is to try to use design theories to create “boundary objects”, which are intellectual tools that are shared by both disciplines and enable cooperation. The approach is illustrated and supported by relating it to ongoing domain-bridging work which is exploring the application of cognitive support theories to SE research problems.

1. Introduction

Gaps currently exist separating human-computer interaction (HCI) and software engineering (SE) research and practice. These gaps are due, in part, to differing terminology, concepts, education, and methods. In the past, such gaps have been identified as being problematic, and efforts have been made to try to bridge these gaps (e.g., John *et. al* [13], Carroll *et. al* [4]). These advancements are encouraging, but new methods for bridging the gaps are still needed. In this paper I shall argue that, over the long haul, critical struts in key bridges between SE and HCI will be in the form of *engineering-oriented design theories*, and in the intellectual tools these make possible.

Pragmatic, engineering-oriented theories have historically been key contributors to the maturation of engineering disciplines. Why should HCI engineering be any different? In other engineering disciplines, explicit theories—and the intellectual tools they make possible—have served to reduce the reliance on skill and craftsmanship, and thus paved the way for more deliberate and systematic engineering. For example, Vincenti recounted the way in which airplane wing design was transformed by the emergence of theories of wing performance [19]. After the initial amateur exper-

imentation phase but before the advent of the design theories, wing design relied critically on special labs. These labs had plenty of experience in testing and developing various known wing designs, relied to a great degree on this experience to generate plausible designs, and leaned heavily on their capabilities to use wind tunnels to test prototypes. The similarity to modern-day HCI design consultants and user testing facilities is readily apparent. New wing performance theories facilitated a deeper and broader shared understanding of both the overall design problems and their solutions. This allowed wing design decisions to be more closely integrated into the other engineering decision processes. These processes typically involved various tradeoffs such as flight range, load limits, and engine power requirements [19]. With suitable theories available, the primacy of wind tunnel tests in wing design abated. Such changes represent important transformations away from primarily craft-based empirical practices using post-design prototypes, and towards significantly theory-based practices which can work in a more deliberate, anticipatory manner.

I believe that design theories should be able to play a similar role in transforming SE practice by bridging gaps between HCI and SE activities. HCI theories can help transition “invent and test” practices into “specify then implement” practices [6]. It might take a long time to complete this transition, but it is unlikely to be sped along by avoiding the challenge. Overall, then, I agree with the assertions by Newell *et. al* [16] and John *et. al* [13] that “design is where the action is”. They argued that HCI has to a great extent organized itself around evaluation and user testing, and that by the evaluation stage HCI is already in a sense too late. Although I concur with this argument, I would also point out that nothing appears to stop us from bringing theory to bear on the whole spectrum of SE activities (engineering requirements, assuring quality, etc.). Their argument also does not settle the question of how SE and HCI will relate to each other as the fields evolve. The maturation of engineering practice is likely to involve transitioning HCI theory into HCI practice and SE theory into SE practice. Moreover—

and this is critical—it might involve building *both* SE and HCI theories into structures that *bridge* SE and HCI.

This paper explores the use of engineering-oriented design theories to bridge gaps between SE and HCI. First, the particular theory-based route being advanced will be clarified. The central theme is that the bridging can be done by building so-called “boundary objects” in the form of practitioner-oriented intellectual tools. Second, I will try to both illustrate and support the argument by describing an ongoing effort to bridge gulfs between SE and HCI using *cognitive support* theories. These are theories that say why artifacts can help or aid cognition. Such theories can form the foundation for bridging structures. Implications for future research directions are outlined.

2. Bridge building with design theories

It is not *a priori* clear what sort of bridgework is needed between SE and HCI, if any. For example, one may argue that merely *bridging* the two does not go far enough, and that HCI and SE should, in fact, be *merged* into a single whole. After all, the overwhelming majority of software is made ultimately for nobody but humans. Thus it is not beyond reason to claim that all computing invariably needs to conform to human requirements, goals, and needs, be they cognitive, social, economic, political, organizational, emotional, spiritual, philosophical, etc. Surely the problem of effectively building software to satisfy human requirements is a theme which is central to both HCI and SE.

Curious, then, that normally dispassionate people can be made to bristle when it is coyly suggested that HCI be positioned as a part of SE or vice versa. I just may have enough personal experience on this matter to credibly offer the following etiquette guidance for how to behave at future SE–HCI social functions. Beware when suggesting that HCI is merely a part of SE—the part most directly connected with humans. Avoid suggesting that HCI can be comfortably slotted as parts of requirements engineering and quality assurance; do *not* call it “the annoying part most easily cut if pressed for time”! Likewise, bite your tongue before proposing that SE is simply that expensive but basically unproblematic portion of HCI consisting of the final step from idea to silicon. And leave out suggesting, even if you are merely waxing philosophical, that SE is actually the junior partner to HCI since HCI is the final arbiter of quality and suitability.

Playful ribbing and turf wars aside, there are good reasons why HCI and SE are not comfortably subsumed by one another, and why they may forever maintain quite distinct journals, conferences, courses, development departments, and value systems. In my mind the main argument in favor of perpetual separation is that they break off different portions of the overall challenge of building computing so-

lutions fitting human needs. For to build software that truly fills human needs one must solve two frequently baffling—and separate—riddles. First, *what satisfies the human needs of the clients or users?* This is HCI’s home turf. Software is only part of the answer. And even if you assume you can perfectly write the software for free, you are effectively no closer to answering the question. Second, *what software is feasible and how do you actually build the software effectively, efficiently, and reliably?* This is what SE can arguably claim is in its foveal view. Even if HCI can be relied upon to independently figure out the human suitability issues, some software artifacts must be engineered. And not all envisionable software can be feasibly built, so the parameters for HCI solutions are at least in part determined by SE’s software factors such as reliability, cost, safety, etc. So, really, to a great extent SE and HCI have different scopes that force them to specifically concentrate on different aspects of software development.

There may be good reasons to wish this separation to persist. The loose coupling between the two permits specialized training, gives practitioners the freedom to use their own specialized techniques and methods when they work best, and allows researchers and innovators to explore new directions unencumbered (at least initially) by requirements from the other domain. The same basic argument is used in SE to defend personnel selection to ensure mixed and synergistic expertise in software development teams. It also underlies the advantages of heterogeneous systems in AI and cognitive science (e.g., Newell’s blackboard system argument [15]). Thus it is reasonable to ask: need we bridge these two disparate disciplines?

Absolutely. Experiences with the problems in current development practices seem to demand it. Furthermore, admitting that HCI and SE are distinct in essential ways can, at most, argue only that they should not be merged into a truly monolithic entity. Some sub-structure is still admissible; and their distinctness in no way absolves them from needing to smoothly inter-operate. In large organizations it might be feasible to more-or-less forcibly separate the HCI and SE into different people, units, and sub-processes. Then, perhaps, the main issue is merely to get SE and HCI people, units, or processes to adequately mesh when they cross paths (which, thinking optimistically, is rarely). But surely is not always possible or desirable for HCI and SE to be highly decoupled; indeed it seems quite likely that the *same people* will sometimes need to wear both SE and HCI hats *at the same time*. SE and HCI are therefore tied together by their shared problem and, indeed, are often bound up in the same people, activities, and events. And in both research and practice, SE and HCI are currently too loosely coupled to adequately facilitate the needed interplay. Bridges—or *more or better* bridges—need to be constructed. But how?

I propose that engineering-oriented design theories can

be effectively employed to bridge the gaps between SE and HCI. I am certainly not the only one to propose a theory-laden prescription for delivering HCI to SE, or for imbuing HCI with more of SE's sensibilities such as its focus on tools, processes, and formal methods. In fact, my argument will take significant cues from authors such as Newell and Card [2, 16], Green *et. al* [8–10], and Barnard [1]. Although I have gratefully borrowed from this existing heritage, it is still important to clearly lay out a position on what sort of bridges can be built, and how to do it using design theories. This is done in two stages by first explaining what *boundary objects* are and how they might bridge gaps between HCI and SE, and then by proposing how design theories might be helpful as a foundation for organized collections of boundary objects.

2.1. Boundary objects as domain bridges

The launching point for this proposal is Star's analysis of the workings of scientific communities [17]. Star employs her concept of "boundary objects" to explain how it is that a loosely coupled heterogeneous network of scientists can coordinate to solve problems. Green [8] pointed out that Star's concept of a boundary object is also useful in describing how to bridge between different domains such as cognitive psychology and HCI or SE.

Based on historical case studies of scientific work involving both professional scientists and amateurs, Star and her colleagues found that the participants:

- (1) cooperate without having good models of each other's work;
- (2) successfully work together while employing different units of analysis, methods of aggregating data, and different abstractions of data;
- (3) cooperate while having different goals, time horizons, and audiences to satisfy. [17, pg. 46]

The reader may immediately recognize the similarity between that description and the gaps which exist between SE and HCI. SE and HCI people often fail to completely understand each other, are accustomed to different representations, and are often focused on different goals, audiences, etc. HCI and SE nonetheless must work together as a coordinated whole to solve a common problem. Specifically, they must work smoothly together to build good software.

It makes sense, therefore, to look at Star's analysis to try to see if lessons learned there can apply to build better bridges between SE and HCI. Star studied heterogeneous and loosely coupled communities having divergent conceptions, goals, methods, and knowledge. If these gulfs *truly* separated the heterogeneous communities, would coordination not break down? Something must have been bridging the gulfs.

Star suggested that in the activity she observed it was the *boundary objects* that made cooperation possible. Boundary objects are "objects that are both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identity across sites," [17, pg. 46] and that they sit "in the middle of a group of actors with divergent viewpoints" [17, pg. 46]. From her description it seems that their value stems partly from their ability to provide a common structuring resource for the communities on either side of it, and partly also from their ability to make it possible to map data or knowledge from one domain onto another.

This idea of a boundary object between communities is similar to Barnard's conception of "bridging representations" for bridging gulfs between theory and practice [1]. His bridging representations may be thought of as representations of knowledge from one community that have been transformed and tailored for use in another. Compared to Star's conception of boundary objects, Barnard's description of bridging representations makes them appear less shared, and more one-dimensional in the way they facilitate coordination. Thus Star's boundary object model may be a more suitable starting point for discussing how to bridge gulfs *within* practice or theory. Barnard's emphasis on transformation may be more helpful in understanding how to bridge gaps *between* practice and theory.

If we take these ideas of boundary objects and import them into the debate about how to bridge between SE and HCI, then we should be looking to create boundary objects that allow loose coordination in similar ways. This attempts to go beyond merely packaging contents from HCI for use by SE, improving cross-discipline training, or massaging HCI practice to make it follow or interleave with typical SE processes. It would involve creating objects that are expected to sit between loosely-coupled HCI and SE processes and coordinate them. These boundary objects would leave flexibility in interpretation and usage for parties on different sides of the SE/HCI divide.

Star emphasized that there should be a plurality of boundary objects, each suited to sharing different types of information from the different communities. In fact, she documented a taxonomy—i.e., a collection of boundary object *categories*—which she extracted from her historical case studies. Boundary objects from any category shared similar characteristics and coordination benefits. These categories included:

Label. Keywords or other conceptual categories are shared between two communities. These are "indexes" that can be used to refer to other entities of interest within the domain. The labels establish common terminology even though implications and interpretations of the terms may vary between communities.

Repository. An indexed collection of units such as a library or museum. Multiple domains can import these atomic units for use in work done in each domain. This implies that the units combine ideas and issues from multiple domains, and do so in suitably fine-grained units.

Ideal Type/Platonic Object. Star described a Platonic object as “an object such as a map or atlas which in fact does not accurately describe the details of any one locality. It is abstracted from all domains, and may be fairly vague. However, it is adaptable to a local site precisely because it is fairly vague” [17, pg. 49].

Terrain with Coincident Boundaries. These occur when objects from different domains share commonalities. I find it helpful to de-emphasize Star’s physical description (e.g., “terrain”), and think of them instead in terms of homomorphic structures which are utilized differently within different domains. Her example is of geographic maps which are tailored to show different information for different communities, but which can nonetheless be correlated because of their common geography.

This derived taxonomy of boundary object types may also apply well to SE-HCI. A short survey of previously proposed bridges between HCI and SE yields several candidates which match qualities of Star’s categories. For instance, the now ubiquitous GUI toolkit may be considered a repository since it usually contains an organized library of reusable interface components. GUI toolkits simultaneously binds SE concerns (reuse, framework design, etc.) and HCI concerns (interaction primitives, style, etc.). For another repository example, we might also point to Sutcliffe’s proposal for building an indexed collection of reusable HCI design analyses [18]. Star’s description of Platonic objects is highly reminiscent of the aims of design patterns. HCI design patterns (e.g., Erickson [7]) might therefore reasonably be called Platonic objects. It might also be argued that “usability architectures” [13] match Star’s intent for Platonic objects since architectural-level descriptions abstract over lower-level details. In addition, the role sometimes envisioned for scenarios in coordinating HCI and SE [4] seems consonant with Star’s description of terrain with coincident boundaries.

Star’s particular taxonomy of boundary objects may not be exhaustive. It may also not apply well enough to SE-HCI. However it at least hints at the possibility of understanding SE-HCI gulf-bridging in abstract, principled ways. It suggests also that it should be feasible to inform SE-HCI bridging by studying and modeling HCI-SE coordination in real organizations.

2.2. Design theories and boundary objects

When I propose to use “engineering-oriented theories” I have in mind theories that are explicitly designed with effective engineering decision making and judgment in mind [14]. Typically this means an eager acceptance of approximation, simplification, and general rules. Even theories that have been proven inaccurate and false can be good engineering theories if they answer engineering questions adequately well in known circumstances [14]. Thus the widely known GOMS [14] framework qualifies as an engineering-oriented theory.

By “design theory” I mean to include only those theories that can make direct contributions to the design of artifacts. Metaphorically speaking, the designer gives such theories information about engineering goals and tradeoffs, and the theory responds with hints (i.e., predictions) as to which artifacts might satisfy these goals. Significantly, my definition explicitly requires that the theories need no existing tool, prototype, or design in order to be invoked. This definition excludes theories such as GOMS since it requires a design to be given before it can be invoked. Design theories can be engineering-oriented. “HCI design theories” facilitate the design of interactions between humans and computers. Thus the Cognitive Dimensions framework (CDs) [8] can justly be called an engineering-oriented HCI design theory. CDs give names to common HCI-related design tradeoffs and choices. The CDs are explicitly advertised as a “broad-brush” intellectual tool so they are of a different character than GOMS [9]. As such, they do not fall into the calculational style of “hard science” theory thought by Newell and Card [16] to be necessary. They are engineering-oriented, nonetheless, in that they can be used proactively by designers to reason about design decisions and tradeoffs. Therefore they can do just what Newell and Card wished for in a practical theory for HCI.

Now there is, I believe, an important relationship between design theories and desirable boundary objects for SE-HCI. Many—if not all—key boundary objects for SE-HCI will embed within them design theories. These theories may not be very explicit. The situation is directly analogous to the idea from Carroll and Campbell [3] that artifacts embed theories. For instance a skyscraper embeds a theory of local weather since it reliably predicts the expected maximal wind speeds (they are usually built to withstand expected storm forces). In the same way SE-HCI boundary objects can be expected to embed design theories from SE and HCI. More specifically, they can be expected to *simultaneously* embed theories from both SE and HCI.

An example of such a boundary object is a usability architecture [13]. John and Bass argued that usability architectures closely weave SE and HCI design issues together in its architectural description. For instance, their descrip-

tion of a usability architecture highlights the frequent need and utility of an undo mechanism and also indicates the need to consider undo mechanisms at the architectural level. Such architectures embed an HCI theory regarding the usability impact of an undo mechanism (including its likely uses, etc.). It also embeds a SE design theory as to what software architectures are suitable for implementing flexible and reusable undo mechanisms. The particular theories might be neither explicit nor obvious, but it seems to me that they must exist or else the design results would likely be unacceptable to one or both of the SE or HCI audiences.

At this point it seems plausible that most or all of the boundary objects one could point to would similarly rest on fundamental design theories from both domains. It also seems likely that certain theories would be able to contribute to the foundation of several different boundary objects. Furthermore, it seems eminently sensible to desire whole *families* of boundary objects which apply across the entire software lifecycle and which are related by a common theoretical framework. It is reasonable to expect that SE processes would generally be aided by a coherent collection of boundary objects for requirements engineering, design, implementation, and testing. Thus instead of merely pursuing an *ad hoc* collection of boundary objects, it might be feasible to generate suites of related boundary objects to coordinate HCI and SE at several levels and during several activities.

These thoughts lead to an ambitious vision for bridging SE and HCI. Imagine, for example, that one has a general theory of the usefulness of undo mechanisms. Suppose the theory states when they are useful, what features are useful, provides quantities for their impact, and so on. Presumably the theory relies on a reasonably well defined ontology and a vocabulary to match. Further, suppose that the theory is found useful in generating usability specifications [5], as well as for defining a related usability architecture. An object-oriented framework might be built to implement the architecture. These (specs, architecture and OO framework) might well be used in a variety of HCI and SE activities related to requirements engineering, design, and evaluation or quality assurance. Presumably the theory's vocabulary, ontology, and statements about usability would be preserved in these, which should help ensure coherence within the family. The result is a purposefully-designed, coherent collection of boundary objects built from a common theory. What is more, the theory is exposed to SE and HCI scientists for independent scientific validation. With such a theory pervasively backing the boundary objects, this could lead to an engineering discipline built more solidly on a scientifically sound foundation.

This vision is ambitious and the route to fulfilling it is likely full of serious hurdles. But I strongly feel the potential payoffs make it worth trying. The main catch, of

course, is having useful design theories and knowing how to weave these into various boundary objects. Are useful theories even possible? These questions are addressed in the next section.

3. The case of cognitive support theories

This section attempts to illustrate the envisioned bridge-building approach. The illustration is intended to help convey the overall vision and to help inspire new ways forward. This section contains no definitive results or conclusions as the work being described is very much in its beginning stages. However if it succeeds in providing grist for current SE–HCI debates then it will have served its purpose.

The illustrations in this section are based on the work stemming from my own dissertation work [20]. The main thrust of the dissertation was to try to set SE on a firmer scientific footing regarding cognitive issues in HCI. Specifically, the dissertation worked towards establishing a comprehensive, applied theory of *cognitive support* which could be used to develop pragmatic resources for SE and HCI practitioners. This background is briefly described, and the way it is proposed to be used is then overviewed and related back to Star's boundary object ideas.

3.1. Cognitive support background

Cognitive support can be defined as the capacity an artifact has for assisting in cognitive work. Artifacts that support cognition make cognition easier, faster, or less errorful—i.e., better in some way. Cognitive support is only one HCI issue in a sea of many, but it is a crucial one: it is a critical factor in the usefulness of essentially all software designed to support knowledge work. A design environment, a library interface, a financial forecasting tool, or a cockpit would need to consider how cognitive performance is improved by using the software tools. If any of these failed to adequately support the cognition of its users, it would likely be considered to be a failure.

An important cog in my dissertation is the definition of a high-level theory of how artifacts can support cognition. This theory is called RODS [20,21]. It assumes an umbrella framework for understanding HCI in terms of theories of “distributed cognition” (DC). DC treats cognition as a type of computation: one that is distributed between humans and artifacts. For example, in a cockpit human pilots rely on external media such as marks on dials to hold data that, without such external aids, would otherwise need to be held in their heads. DC effectively treats such a cockpit as a heterogeneous distributed computing system where pilots are computing nodes and the dials are external memories they can access [11].

The basic insight behind RODS is that cognitive assistance can be traced to rearrangements in computation. For instance, if a dial marker is introduced into a cockpit, it can offload data from internal memory onto external memories, thereby reducing the cognitive burdens of the pilots. The support introduced reengineers the cognition in the cockpit according to principles of distributed memory computing optimization. RODS claims that all cognitive supports are instances of computational reengineering. This is a significant statement. It asserts that improvements in cognition can be related back to established computational theories. To put it another way, psychological effects are explained in computing sciences' own terms.

Another important claim that RODS makes is that many different types of cognitive support can be decomposed into just four primitive types of computational rearrangements. These four transformations give RODS its name. For instance, an external memory such as a bookmark list in a browser falls into the support category termed "distribution" because it distributes data (the bookmarks). The "D" in RODS stands for "distribution". The remaining letters of the RODS acronym name the other support categories. All four RODS categories are transformations commonly known to computer scientists familiar with standard programming tricks. It does not add enough value to the present argument to discuss the theories in more detail here. Details are in the dissertation [20].

Another key part of the work is the realization that one can apply the four generic computational transformations defined by RODS to any DC model of human-computer systems. In particular, if one considers a particular model of a human interacting with a computer (or other artifact), then RODS transformations on that model can be systematically considered. These transformations correspond to promising design moves. For instance, the data and processing in the model can be analyzed for opportunities for distribution onto external memory or processing. By using a generic cognitive model, a taxonomy of 16 different classes of cognitive support was generated and considered. Each class of cognitive support abstractly identifies a way to support cognition. For example the browser history kept in a web browser was identified as a specific type of cognitive support, namely the distribution of history states. Any one tool could involve multiple support types.

RODS is not quantitative: it does not model or predict quantities such as the amount of memory that might be offloaded, or the relative importance of various cognitive supports. It also does not directly address tradeoffs. For instance, an external memory is normally expected to have update costs. It is possible to have external memories that are too costly to update to make them worthwhile. These issues are not factored into RODS.

3.2. Boundary object considerations

Although the above introduction is a terse description of the theories, it covers enough to discuss the sort of boundary objects that can potentially be derived from them. Some of these are under investigation. Others are merely speculative at this point however they help convey the current general direction of the research. These boundary objects are described below and their relation to SE and HCI theories are briefly explained. How they coordinate HCI and SE work is also indicated.

Vocabulary and conceptual framework.

RODS and its overall framework identifies and names concepts relating to cognition and its support in artifacts. For example, the terms "external memory", "distribution", "data distribution", and "history state" are defined in order to be able to define the category of cognitive support that involves the distribution of history states.

An extremely important aspect of this vocabulary is that the definitions bind cognitive issues and models in HCI directly to computational terminology and common computing science theories. In this sense, they can be classified as "Labels" in Star's taxonomy of boundary objects. For instance "data distribution" indexes into well-understood concepts from computing science. The notion will be familiar to most software engineers, and implications for design (e.g., storage methods, search techniques, caching, etc.) is likely to be keyed into. The terms "external memory" and "history state" will also have implications for HCI specialists, who might wish, for instance, to focus in on the how users rely on external encoding of history states to simplify backtracking.

Given the above, the vocabulary of RODS and its associated conceptual framework serves as a type of boundary object between SE and HCI, embedding and combining theories from both computing and HCI. It is well-positioned to be adopted by SE and HCI people alike. Much of the terminology and concepts are familiar to SE people; getting SE people to be aware of HCI concepts and meanings is known to be frequently difficult due to educational differences.

A shared vocabulary and conceptual framework are likely to be important for all phases of software construction. The fact that RODS is not quantitative or does not address tradeoffs does not adversely hinder its applicability. HCI and SE people frequently communicate in informal design meetings, including hallway conversations. A suitable shared vocabulary tied to appropriate abstractions and theories can help lift the level of conversation. This motivation is nicely described by Green and Petre when they say

Explicitly presenting one's ideas as discussion tools... is doing nothing more than recognizing

that discussion among choosers and users carries on interminably, in the corridors of institutes and over the Internet. Our hope is to improve the level of discourse and thereby to influence design in a roundabout way. [10, pg. 132]

Cognitive support patterns.

RODS is created as a decompositional analytic framework in which complicated forms of cognitive support can be broken down into combinations of various primitive types of cognitive support. Certain compositions of cognitive support appear to work particularly well together. For example, consider some of the advantages offered by a typical hierarchical outline processor such as one finds in Microsoft PowerPoint in Outline View mode. The outline processor allows items to be stored externally, alleviating internal memory bottlenecks and making it possible to directly work with larger presentations. Once externalized, the items in the outline can be manipulated externally by directly manipulating them by dragging them with the mouse. This more visceral manipulation can replace mentally juggling the outline. Once the outline is finished it acts as a plan for writing out the contents of the slides. Each of these features is classifiable by the cognitive support theories. Their composition resolves several problems simultaneously.

The above sort of synergistic compositions—and many others—occur frequently in a variety of contexts and implementations. The compositions are abstract in critical ways. They are independent of the implementing technologies. For instance, paper-and-pencil based “outline processors” share most of the same features as electronic ones. The compositions are also at least partly task-independent. For instance, the utility of the above sort of outline view support can be the same for nearly any task where planning the authoring of a hierarchical document is required, such as in programming or proof writing.

Because they are general, can occur in many contexts, and solve multiple issues simultaneously such compositions might be called *patterns* of cognitive support [20, 21]. This would be a form of HCI design pattern. As it was mentioned before, HCI design patterns are likely candidates to be considered “repositories” in Star’s taxonomy of boundary objects. In this case the patterns are specified using combinations of cognitive support terms (i.e., labels) defined by RODS and its framework.

Cognitive support reference designs.

For any specific domain or problem category, one particular generic composition of cognitive support patterns may be prototypical. For example many software reverse engineering tools can be seen as implementing a medium for externalizing and manipulating imperfect knowledge [12].

By this I mean that the tools allow reverse engineers to externally represent and manipulate knowledge about the system, and that this knowledge may be uncertain, contradictory, incompletely, or otherwise imperfect. The reverse engineering process then seeks to repair this imperfect knowledge. As in the case with outline processors, several types of cognitive support combine to make this general solution pattern desirable. The tools allow offloading of memory, make it possible to externally manipulate the knowledge instead of needing to do it internally, and can be made to automatically process the knowledge [12]. The cognitive support analysis of such tools provides a high-level understanding of them [22].

These considerations present the possibility for describing new boundary objects. First, it may be possible to define a type of reference, high-level design for a given task or problem domain (as indicated by Jahnke *et. al* [12] in the domain of reverse engineering tools). This would be useful for kickstarting development, or for generating a reusable framework for building reverse-engineering tools. This reference design can be associated with a pre-existing analysis of the cognitive support provided by the implementation (as in Walenstein [22]). The reference design and its associated cognitive support analysis can serve as reusable boundary object. Since the analysis and reference design can be expected to use the same feature and cognitive support vocabulary, it appears to be much like what Star described as a “Terrain with Coincident Boundaries” (the common structure of the design and analysis is the coincident part). Furthermore, a collection of such reference designs+analysis combinations may be considered to be a repository.

4. Summary and discussion

This paper examined the possibilities for bridging rather than eliminating the gaps between HCI and SE. The analysis focused on a proposal to view the bridging problem in terms of building so-called “boundary objects” that mediate activities between SE and HCI practitioners or researchers. A specific proposal for building such boundary objects was advanced, and was argued to have promise. The proposal involves embedding engineering-oriented design theories in a collection of related boundary objects. Embedding theories in this manner should help ensure coherence and continuity, and assist in the push to solidify the scientific foundations of SE practice.

As an illustration of this overall bridge-building approach, a research effort closely following this proposal was summarized. The theories being employed to build boundary objects were cognitive support theories. From a relatively simple cognitive support theory, several potential boundary objects were described. It was argued that these could help in the coordination of HCI and traditional SE

streams. For example, a vocabulary, pattern catalog, and design/analysis repository may collectively form a family of boundary objects related by being created from a common cognitive support design theory. For instance imagine a case where, during design brainstorming, conversation informally drifts toward the tradeoffs between the user's cost of using an external memory versus the potential benefits for remote collaboration. During design elaboration, HCI participants might consider various cognitive support patterns for resolving such tradeoffs. The pattern candidates are used by SE teams to select usability architectures implementing the identified patterns. While the software is being developed the HCI team can use the cognitive support analysis of a reference architecture to plan evaluations and run initial walkthroughs. The theory and its concepts form a theme running through the entire development cycle.

History has shown that engineering disciplines get ideas essentially right even before explicit theories are developed to explain the reasons why. It seems prudent to empirically study real SE-HCI coordination. Latent theories may be discovered, and successes could be replicated in new or updated boundary objects. In short, the work to bridge SE and HCI can proceed by deliberate steps of investigating design theories and then empirically examining practice for opportunities for improving current practice.

References

- [1] P. Barnard. Bridging between basic theories and the artifacts of human-computer interaction. In J. M. Carroll, editor, *Designing Interaction: Psychology at the Human-Computer Interface*, chapter 7, pages 103–127. Cambridge University Press, 1991.
- [2] S. K. Card. Theory-driven design research. In G. R. McMillan, D. Beevis, E. Salas, M. H. Strub, and R. Sutton, editors, *Applications of Human Performance Models to System Design*, pages 501–509, New York, 1989. Plenum.
- [3] J. M. Carroll and R. L. Campbell. Artifacts as psychological theories: The case of human-computer interaction. *Behaviour and Information Technology*, 8(4):247–256, 1989.
- [4] J. M. Carroll, R. L. Mack, S. P. Robertson, and M. B. Rosson. Binding objects to scenarios of use. *International Journal of Human-Computer Studies*, 41(1/2):243–276, 1994.
- [5] J. M. Carroll and M. B. Rosson. Usability specification as a tool in iterative development. In H. R. Hartson, editor, *Advances in Human-Computer Interaction*, volume 1 of *Human/Computer Interaction Series*, pages 1–15. Ablex, Norwood, NJ, 1985.
- [6] J. Dowell and J. Long. Conception of the cognitive engineering design problem. *Ergonomics*, 41(2):126–139, 1998.
- [7] T. Erickson. *Lingua Francas* for design: Sacred places and pattern languages. In *Proceedings on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, pages 357–368. Association for Computing Machinery, 2000.
- [8] T. R. G. Green. Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay, editors, *People and Computers V: Proceedings of the Fifth Conference of the British Computer Society Human-Computer Interaction Specialist Group*, pages 443–460. Cambridge University Press, 1989.
- [9] T. R. G. Green, S. P. Davies, and D. J. Gilmore. Delivering cognitive psychology to HCI: the problems of common language and knowledge transfer. *Interacting with Computers*, 8(1):89–11, 1996.
- [10] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: A ‘cognitive dimensions’ framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.
- [11] E. Hutchins. How a cockpit remembers its speed. *Cognitive Science*, 19:265–288, 1995.
- [12] J. H. Jahnke and A. Walenstein. Reverse engineering tools as media for imperfect knowledge. In *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE’2000)*, pages 22–31. IEEE Computer Society Press, 2000.
- [13] B. E. John and L. Bass. Usability and software architecture. *Behaviour and Information Technology*, 20(5):329–338, 2001.
- [14] B. E. John and D. E. Kieras. Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction*, 3(4):320–351, Dec. 1996.
- [15] A. Newell. Some problems of the basic organization in problem-solving programs. In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Proceedings of the Second Conference on Self-Organizing Systems*, pages 393–423, New York, 1962. Spartan Books.
- [16] A. Newell and S. K. Card. The prospects for psychological science in human-computer interaction. *Human Computer Interaction*, 1(3):209–242, 1985.
- [17] S. L. Star. The structure of ill-structured solutions: Boundary objects and heterogenous distributed problem solving. In M. N. Huhns and L. Gasser, editors, *Distributed Artificial Intelligence 2*. Morgan Kaufmann, 1989.
- [18] A. Sutcliffe. On the effective use and reuse of HCI knowledge. *ACM Transactions on Computer-Human Interaction*, 7(2):197–221, June 2000.
- [19] W. G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, Baltimore, 1990.
- [20] A. Walenstein. *Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework*. PhD thesis, School of Computing Science, Simon Fraser University, May 2002.
- [21] A. Walenstein. Foundations of cognitive support: Toward abstract patterns of usefulness. In *Proceedings of the 14th Annual Conference on Design, Specification, and Verification of Interactive Systems (DSV-IS’2002)*, volume 2545 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, 2002.
- [22] A. Walenstein. Theory-based analysis of cognitive support in software comprehension tools. In *Proceedings of the 10th International Workshop on Program Comprehension (IWPC’02)*, pages 75–84. IEEE Computer Society Press, 2002.