

# Foundations of Cognitive Support: Toward Abstract Patterns of Usefulness

Andrew Walenstein

Department of Computer Science,  
University of Victoria, Victoria, B.C., Canada  
walenste@csr.uvic.ca

**Abstract.** Computer tools for cognitively challenging activities are considered useful, to a great extent, because of the support that they provide for human thinking and problem solving. To analyze, specify, and design cognitive support, a suitable analytic framework is required. Theories of “distributed cognition” have been offered as potentially suitable frameworks, but they have generally failed to plainly articulate comprehensive theories of cognitive support. This paper seeks to clarify the intellectual foundations for studying and designing cognitive support, and aims to put them in a form suitable for design. A framework called RODS is described as a type of minimal, lightweight intellectual toolkit. Its main aim is to allow analysts to think in high-level cognition-support terms rather than be overwhelmed by task- and technology-specific implementation details. Framing usefulness in terms of cognitive support makes it possible to define abstract patterns of what makes tools “good”. Implications are drawn for how the framework may be used for the design of tools in cognitively challenging work domains.

## 1 Introduction

A universal and critical design goal for tools is to make them *useful*. In the realm of physical labour, critical aspects of usefulness are understood in terms of *mechanical advantage*. For example a lever is understood to be useful primarily due to its ability to reduce the force needed to raise a mass. For cognitive work domains such as software debugging, financial forecasting, and writing composition, a key aspect of a tool’s usefulness relates to how it improves cognition. For instance, we should expect that useful software visualization systems will make program comprehension easier, faster, or better in some way. Thus in highly cognitive work domains, usefulness is closely associated with the provision of *cognitive support*—i.e., with the ways in which cognition is helped, aided, or otherwise assisted by artifacts. Tools for cognitive work domains may fail to be usable in many ways (e.g., they may be hard to learn), but they cannot fail to provide cognitive support.

In order to systematically analyze, specify, and design cognitive support in tools, the fundamental principles of cognitive support must be known and effectively wielded. This is not easily done. Many common shortcomings of HCI theories and guidelines are by now well known (see e.g., Carroll [6]). Four inadequacies are of particular interest in this paper. First, existing theories may be too costly to learn and apply [4]. Second, they can be too task- or technology-specific. This greatly hinders the specification of reusable and science-grounded patterns of successful design [25]. Third, the theories can fail to adequately address usefulness in terms of cognitive support. Frequently the objective adopted for HCI theories is to identify *usability problems*, instead of identifying the forms of cognitive support which render a tool worthwhile using in the first place. Fourth, the theories may be ineffective for generating design advice. The theories must work in the absence of a prototype solution if they are to be especially useful in design; they must identify steps one might take to add cognitive support.

These four shortcomings currently hamper the application of distributed cognition (DC) theories. DC has been portrayed as a promising general theoretical framework for analyz-

ing cognitive work domains [10]. Several researchers have recently adopted DC as an umbrella approach for HCI research and design (e.g., Wright *et al.* [30], Hollan *et al.* [10], Rogers *et al.* [20]). The key tenet in DC is that cognition is not localized to a single individual, but is instead a process distributed amongst various artifacts and humans. But knowing how any cognitive system works is only a prelude to changing it by introducing new tools. It is reasonable to expect that theories of *cognition* (even *distributed cognition*) will primarily explain or predict how existing cognitive systems work. In contrast, a theory of *cognitive support* is a theory of intervention: it tells one how cognitive systems are improved via the introduction of artifacts. It is important to be able to clearly articulate the principles underlying these interventions in a way that is technology- and task-independent, and such that learning and application costs are minimized.

This paper outlines an attempt to satisfy these goals by proposing a high-level cognitive support theory. At its heart is a general cognitive support analysis framework called RODS. RODS integrates and adapts existing theories from DC and elsewhere. It is essentially a high-level and general theory of cognitive support together with an overarching framework for analysis. The central resource within RODS is a list of four support principles identifying distinct classes of cognitive support. These four classes of support give RODS its name.

RODS is proposed as a resource for designers to use throughout the design and development process, but it is expected to be especially helpful during design envisionment. Our long range goal is to facilitate various phases of design, but our current focus is on relatively informal design reasoning such as might occur during design brainstorming sessions. Our aims are thus similar to those of the cognitive dimensions (CDs) work which “raise the level of discourse” [9, pg. 132] of designers. In particular, we wish to specify theory-derived patterns of cognitive support which are task- and technology-independent. These would allow designers to avoid “death by detail” [9, pg. 131] and yet still proceed towards essential design insights. If we are successful, we expect such “broad-brush” theories of cognitive support to have broad implications for HCI development.

The rest of the paper is structured as follows. First, requirements for generating a suitable theoretical framework for cognitive support are outlined in Sec. 2. Six desirable traits are extracted from this analysis. Section 3 describes RODS and shows how it embodies these six desirable traits. Section 4 briefly overviews how RODS has been used to view various tool implementations as instances of patterns of cognitive support. Related work is overviewed in Sec. 5, and implications and conclusions are drawn in Sec. 6.

## 2 Leveraging Mechanical Support Theory

Identifying fundamental principles is a critical undertaking for any field of research. Fundamental principles frequently provide deep and general insights that are as important as the fine details. This fact was noted by Newell and Simon [16] in their 1975 ACM Turing Award lecture. They noted that the essential characteristics of a discipline can often be stated in short, general sentences. They highlighted, in particular, the importance of the cell doctrine in biology, the theory of plate tectonics in geology, and the germ theory of disease. These are all gross qualitative theories which are critical for understanding a domain. They tie together, relate, and organize multitudes of facts. The fact that they are not specific theories capable of generating precise predictions does not diminish their importance. High-level, generalized truths can be enormously valuable in understanding a broad range of phenomena at a high level.

The arguments raised by Newell and Simon likely apply to cognitive support theories. For designers in HCI, there is, in fact, a particular reason for needing a high-level, qualitative theory of cognitive support: cost of application. There exists a “cost gulf” [4] which can prevent otherwise helpful theories from being applied. High-level, qualitative theories are usually “lightweight” and widely applicable. Consequently, they can be expected

to provide the analyst with the most benefit for the least investment in theory learning and application. Colloquially speaking, high-level, qualitative theories may yield the best “bang for the buck” for the analyst. Thus, articulating such an overarching qualitative theory for cognitive support should be given a high priority—ahead of, say, specific theories pertaining to one form or implementation of cognitive support.

Where will such a general, qualitative theory come from? Distributed cognition (DC) is one possible source. DC theory contains many insights into the ways in which tools affect and support thinking. It identifies key principles underlying DC such as the fact that artifacts are key resources used to represent and propagate knowledge. Our concern here is particularly on generalizable principles for how artifacts support cognition. Although DC is a prime candidate to be able to describe such principles, they have yet to be clearly and comprehensively elucidated. What are principles of cognitive support? How are these foundations best highlighted in a designer-oriented framework?

One way to approach these questions is to look to previous successful frameworks for inspiration. Here we shall turn to mechanical support theory for insights. Compared to cognitive work, the realm of physical work is very well understood. We now understand a great deal about how to assist physical work.

Consider our understanding of *levers*. A lever is a *simple machine*—an “atomic” machine of sorts. It can be defined in structural terms as a movable *pivot* rotating about a fixed *fulcrum*. The total amount of work done is not reduced when using a lever. Instead, the lever is merely a *force-amplification* device, meaning that it reduces the force needed to move a load. However the assistance is not completely free: overheads are introduced by the fact that the lever must be depressed a greater distance than the mass needs to be raised. These overheads are perfectly acceptable when the leverage is needed.

There is much to admire about our relatively mature understanding of mechanical advantage. We know that the key principle for a lever’s usefulness is *mechanical advantage*. The concept of mechanical advantage can be stated simply as a way of trading off distance traveled for a reduction in required force. It is a general concept which is defined in abstract terms, and is thus removed from the particulars of its *implementation* (materials, sizes, etc.) or its use (lifting people, water, planets, etc.). Furthermore, the fundamental concept has a succinct and memorable name: “leverage”. Catalogues of various simple machines have been described (inclined plane, pulley, etc.). These, in combination, form a type of “language” for building more complicated physical labour saving devices [18]. With such a language we can decompose complex machines into their component machines, and compose new complicated machines out of simpler ones.

Our scientific understanding of the principles of mechanical support is now well-established. Designers of new machines can rely on these theories as justification for their design decisions without having to reestablish their veridicality. Even without complicated materials models and physics equations, the gross, qualitative theories are valuable. The basic concepts of simple machines are easy to learn and readily applied without deep analysis or scientific knowledge. It does not take a PhD in physics to apply the concepts during real design.

It would be highly desirable to be able to tell a similar tale for understanding cognitive support at the broadest levels. We can use our understanding of *mechanical* support as an intuition pump for deriving desiderata for understanding *cognitive* support. In particular, we can expect it to be helpful to have the following:

1. **Core theory:** this would explain the basic idea behind cognitive support. It would be the analogue to concept of mechanical support.
2. **Small vocabulary of advantages:** this would describe “atomic” principles of cognitive advantage. Ideally, this would identify *orthogonal* principles that can be discussed independently. These principles, in turn, would generate equivalence classes identifying tools implementing a common type of cognitive support. These cognitive support classes would be analogues of the classes of simple machine (lever, pulley, etc.).

3. **Composition language:** this would explain how the various primitive types of support can be composed. It would be an analogue of mechanical composition (e.g., attaching a pulley to a lever).
4. **Mnemonic, evocative names:** the names of the cognitive support principles would ideally have helpful names that are easy to understand and remember. Ideally the names would index into deep expert knowledge concerning related issues.
5. **Abstract, generalizable description level:** cognitive support would be ideally defined at an abstract, functional level which is removed from particulars of the tool or its uses.
6. **Analysis framework:** this framework would serve as a foundation for applying the cognitive support principles during analysis. It would allow the analyst to decompose complicated tools into their component types of cognitive support. It would also allow the designer to do the reverse, i.e., to compose simple supports into more complicated ones.

In the future, as DC and HCI mature, we may expect to someday build detailed models of cognitive support and perhaps even be able to quantitatively predict cognitive benefits. But for now, it would be very helpful merely if the above six desiderata could be addressed in some adequate way such that the basic forms of cognitive support can be loosely analyzed. Ideally, such cognitive support analyses could be done without requiring an advanced degree in cognitive science. The following section describes RODS, an attempt to provide such a framework.

### 3 The RODS Analogue to Mechanical Support

RODS is a high-level, qualitative theory of cognitive support. The name “RODS” comes from the four main classes of cognitive support it identifies: task Reduction, algorithmic Optimization, Distribution, and Substitution. The framework is described in more detail below. It is introduced by arguing how it addresses the six desiderata from the previous section.

#### 3.1 Core Theory of Cognitive Advantage

The central tenet of DC theory is that cognition is not a process localized to an individual human mind, but one that is spread out amongst possibly many humans and artifacts [11]. Various artifacts, including computers, can therefore be viewed as parts of a single cognitive system. Critically, DC argues that a cognitive system will operate better or worse depending upon whether the appropriate external artifacts are available, and depending upon how they are designed. This insight is memorably noted by Norman who said “it is *things* that make us smart” [17].

Since DC generally explains cognition in terms of computation, the explanation of cognitive assistance must surely also be computational in nature. The essential argument is that a computation that utilizes a *computational advantage* may be *substituted* for another equivalent one which does not. By “computational advantage” we mean some way of improving the computation according to some measure (speed, memory use, etc.). Cognitive support can therefore be understood entirely in computational terms: support is the provision of computational advantage. Newly introduced artifacts reengineer the overall computations involved in a DC system. Thus designing cognitive support means reengineering computational systems such that the system’s cognition is improved.

From these considerations, a general, qualitative theories of cognitive support can be stated as follows: *The cognitive support provided an artifact is the computational advantage that it provides.*

### 3.2 Small Vocabulary of Advantages

A staggering variety of cognitive artifacts are used by humans. Software developers, for instance, use a wide assortment of diagrams, compilers, analyzers, visualizers, editors, and so on. Yet it would be surprising if each variation in cognitive artifact would require a wholly different explanation. A more plausible situation is that is that some relatively small set of principles are *in combination* sufficient to account for the many varieties of cognitive support. Variations and combinations of those “atomic” principles might be seen to generate the enormous design space of cognitive artifacts.

Given that cognitive support is considered computational reengineering, the atomic support principles will be associated with distinct principles of computational advantage. The trick, of course, is settling on a suitable set of principles. Ideally, these would identify orthogonal computational advantages. We have found it useful to settle upon just four commonly understood computational principles. Each of these induce a class of support types; these support types are termed “task reduction”, “algorithm optimization”, “distribution” and “substitution”. These support classes are listed in Figure 1, and described below.

<b><math>\mathcal{R}</math></b>	<b>task Reduction</b>
	<p>Cmpt Principle: some functions are easier to compute</p> <p>Substitution Type: substitute simpler tasks for more complicated ones</p> <p>Example (cmpt): removing redundant or unused computations</p> <p>Example (HCI): eliminating unnecessary steps</p> <p>Design Principle: remove unnecessary work; relax task demands</p>
<b><math>\mathcal{O}</math></b>	<b>algorithmic Optimization</b>
	<p>Cmpt Principle: functionally identical algorithms differ in efficiency</p> <p>Substitution Type: substitute equivalent methods, ADTs, or encodings</p> <p>Example (cmpt): changing to doubly-linked list; switching sorting algorithm</p> <p>Example (HCI): switching to Roman numerals</p> <p>Design Principle: optimize cognitive processes for task &amp; infrastructure</p>
<b><math>\mathcal{D}</math></b>	<b>Distribution</b>
	<p>Cmpt Principle: distribution adds memory or computing resources</p> <p>Substitution Type: substitute external resources for internal ones</p> <p>Example (cmpt): caching memory to a hard drive; client-server architecture</p> <p>Example (HCI): writing down a shopping list; automating constraint checking</p> <p>Design Principle: distribute (i.e., <i>redistribute</i> or <i>offload</i>) data or processing</p>
<b><math>\mathcal{S}</math></b>	<b>Specialization</b>
	<p>Cmpt Principle: specialized routines or processors can be more efficient</p> <p>Substitution Type: substitute specialized processors for more general ones</p> <p>Example (cmpt): use a FPU or accelerated graphics card</p> <p>Example (HCI): enable visual search to substitute for “manual” search</p> <p>Design Principle: change representation to make use of specialized hardware</p>

**Fig. 1.** Summary of RODS cognitive support classes.

**Task Reduction.** It is sometimes possible to eliminate work that is unnecessary. For example a pathologically designed programmer’s editor might insist on having the developer

re-read every line of code in a program before each and every edits she makes (e.g., by forcing a line-by-line scroll through the program). In most (all?) circumstances this would be a waste of time and effort. Computationally speaking, the problem is that there are unnecessary computations being performed; from an HCI point of view, the task can be reduced by eliminating unnecessary steps. Removing unproductive work will decrease the amount of cognitive work done and thus should influence performance.

This form of task and performance “enhancement” is quite obvious, but it is included in RODS because in some cases it is helpful to reduce real task demands. For instance, it may be possible to require only “good enough” answers from users. Moreover, it is critical to include task reduction in RODS so that one cannot confuse any of the other support types with a simple reduction in the work done. Thus the remaining principles will all insist on maintaining some form of equivalence in work.

**Algorithmic optimization.** Algorithmic optimization relies on the fact that differences in encoding or procedure in can create differences in performance without changing the essential outcome—i.e., without changing the “function” being computed [13]. For succinctness, we shall gloss over the many possible variations in terminology (e.g., “data structure” instead of “encoding”, “algorithm” instead of “procedure”, etc.). Normally, differences in encoding and procedure go hand-in-hand. For instance, for two different encodings of some data, different processes are normally needed to compute equivalent functions. Because of the intimate relationship between procedure and encoding (e.g., see Rumelhart *et al.* [21]) they are both considered to be variants of a single principle. The term “algorithmic” is adopted because it is a term used in analogous works in computing theory (e.g., see Aho *et al.* [1]). Where the meaning is clear, the term “algorithmic” will be dropped.

The principle identifies the class of cognitive support that works by *optimizing* the algorithm when given some particular task and computing infrastructure. This use is consonant with optimization in computing science, which normally fixes the function being computed and the underlying computational infrastructure [2]. Performance can be optimized according to a variety of measures (e.g., speed, memory usage, etc.). In changing data encoding, the *information content* is presumed to be unchanged [13]. In computing science, a familiar example is the change from singly-linked lists to doubly-linked lists. Such a change can make various list operations simpler and quicker to perform.

In cognitive support terms, optimization normally means the introduction of artifact designs that reduce cognitive burdens on users (memory use, processing, etc.) by changing the encodings that are used. In other words, it frequently involves *re-representation* [31]. One instance of optimization is the difference made to arithmetic tasks when switching from Roman numerals to Arabic numerals or vice versa [17]. For example, Roman numbers are easier to add because the algorithm involved in adding Arabic numbers is complicated (involving symbol substitution).

**Distribution.** In the ordinary sense of the term, computational systems are called “distributed” when they have multiple, loosely-connected distinguishable computing resources. Distributing computations amongst multiple computing resources can have several types of performance advantages. Having multiple processing elements do the computation means that the work can be divided, thereby reducing the work done by each processor. This can also speed up the execution because frequently the work can be done in parallel. Distribution can also mean that computations with resource requirements exceeding the capabilities of one limited processor might still be performed if the excess load can be taken up by other elements. In computing science, a familiar example of distributed computing is a client-server architecture. One important reason for moving to client-server architectures is that the burdens on the client are greatly reduced.

In its application to HCI, distribution of computation can be interpreted in cognitivist terms [30]. Thus, instead of “data”, one can speak of “representations”, “mental state”,

or “knowledge”. Instead of “processing”, one can speak of “reasoning”, “inferencing”, or “thinking”. For example, using an external memory [17, 23] distributes knowledge. Distributing processing analogously means either having the artifact perform or embody the processing, or having a user process symbols externally. An example of the former is a type-checking compiler. Type-checking compilers externalize the test of constraints on a program [5]. An example of the latter is the manipulation of a slide rule to compute a mathematical function [11].

**Substitution.** The principle underlying substitution is the fact that computing facilities can be adapted specially to restricted sets of tasks. The specialization means that they can be made more efficient. In computing terms, it means they compute fewer functions or operate over a restricted input domain. For instance it is common to have specialized processing hardware such as a floating-point unit (FPU), digital signal processor, vector processor, or graphics accelerator. Unlike the CPU (which is a general, reprogrammable processor), such special computing hardware computes only a restricted set of functions and has limited programmability. The analogue in cognitive terms is the existence of efficient but specialized mental capabilities. These may either be “hardware” (built-in) or various forms of optimized “software” (learned or over-learned skills) [29]. A classic example is perceptual operators [7]. These are fast, effort-free, and execute at least partially in parallel [19]. They stand in contrast to deliberate reasoning which is slow, serial, and effortful.

The principle of specialization creates a category of cognitive support that operates by allowing more specialized processing to substitute for more general, deliberate reasoning. This substitution is a staple of the visualization literature. The essential quality of many accounts of visualization efficacy is that specialized perceptual mechanisms substitute for more complicated inferences (see Larkin *et al.* [13], Casner [7]). One standard example is the use of a line chart to enable *visual search* to substitute for deliberate search [7].

### 3.3 Compositional Language

A single cognitive artifact can be associated with multiple cognitive supports. For instance, when knowledge is distributed from the head and onto an artifact, perceptual substitutions might be enabled for operations over that knowledge. This implies that many of the varieties of more complicated types of support may be reducible to compositions of computational substitutions identified by RODS. Although this cannot be exhaustively demonstrated, we can illustrate the compositionality of RODS principles by deconstructing composite instances of support according to their constituent support types. Here we consider here Larkin’s analysis [12] of *display-based problem solving* (DBPS).

Larkin [12] invented the term “display-based problem solving” to describe a form of problem solving that makes extensive use of external displays. It is clear that to Larkin [12] there are three essential qualities of *display-based* problem solving: (1) that all or almost all of the relevant problem solving state can be read from the display, (2) that because of the nature of external displays, perceptual inferences can be used in places where otherwise more taxing logical inferences would need to be made, and (3) that little planning or deliberation is required for any of the steps—the solver employs very local control. Using this conception, DBPS involves a combination of *distribution* and *substitution*. The distribution involves distributing the current state of the problem. Substitution is involved because the solver is able to use perceptual skills to make inferences concerning problem implications, goal selection, or problem constraints. Substitution is also involved because learned rules are cued by rapid recognition of current system state, eliminating the need to deliberately reason about what operations to perform next. Once DBPS is deconstructed in this manner, a variety of different variations on DBPS can be entertained.

### 3.4 Mnemonic, Evocative Names

RODS is couched in terms familiar to most computer scientists. This does not guarantee that the terms will be more understandable, memorable, or more evocative of important implications. However the odds are that, unlike their cognitive science analogues, the computing terms will be at least meaningful to the average computer science student. Furthermore the terms may index into deep computer science knowledge of computing optimization. This may allow non-specialists to reason by analogy about various forms of cognitive support.

### 3.5 Abstract, Generalizable Description Level

RODS defines four categories of cognitive support and their underlying principles in cognitivist terms (knowledge, inferencing, etc.) and computational terms (memory, processing, etc.). Both of these are description levels are abstract and independent of their implementations. In particular, cognitive support categories are defined without reference to a task or even a particular cognitive model. For instance, distribution is defined without referring to the specifics about what are being redistributed. Plans, constraints, goals, and processing history might be distributed (e.g., see Wright *et al.* [30]). This means RODS can be used on whatever issues of cognition the analyst considers important (goals, social roles, etc.). In addition, the computational principles referenced are abstracted away from their implementation. For instance, any number of artifacts may play the role of an external memory (white boards, computers, even other people [23]). Thus RODS is independent of both task and technology: how they are used depend on the analyst's application context.

### 3.6 Analysis Framework

The "output" of a cognitive support theory is an explanation of cognitive benefits provided by artifacts. The analysis of benefit is necessarily comparative: the benefits of an artifact may be understood only in comparison to what is implied by its absence, substitution, or modification. One way of thinking about this comparison is to suggest that there is a continuum of different levels of support that ranges from the completely unsupported (entirely mental) to the completely automated (no human thinking involved). In between are cases where cognition is spread between humans and artifacts. In practice, both extreme ends of the spectrum will be unattainable for interesting tasks. Still, it is instructive to imagine what, in principle, the unachievable extremes of the spectrum would entail. In particular, starting at the completely unsupported end of the spectrum will allow us to use RODS to define a structured space of cognitive support types.

In order for the *entirely* mental end of the continuum to hold up under close inspection, all of the user's problem, evolving solution, and mental state information would need be held internally; all of the processing of such information would also need to be done internally. Distribution begins to change that picture. As distribution of cognition is increased, the locus of cognition expands away from the individual mind and is dispersed. As various parts of the computation are distributed, substitution and optimization changes can be added. That is, once data of any sort is distributed, processing can be distributed, substitutions can be enabled, and optimizations can be performed. For instance, Sharples noted that writing things down unlocks other types of support:

So long as ideas, plans, and drafts are locked inside a writer's head, then modifying and developing them will overload the writer's short-term memory. By putting them down on paper (or some other suitable medium) the writer is able to explore different ways of structuring the content and to apply systematic transformations... [24, p. 135]

The above observations suggest one conceivable method for generating the space of all possible cognitive supports for some task (conceptually, at least). The generation is effectively by *recursive application of RODS transformations*: start with the set of all possible distributions of data and, one-by-one, apply each type of cognitive support to generate new compositions. For instance in Sharples' writing example, one could begin by enumerating the possibilities of distributing ideas, plans, drafts, goals, and so on. Then one might consider various ways of processing these externally, substituting efficient perceptual operators for more complicated ones, and of making changes to encodings and methods.

The procedure described above also suggests a way of improving design methods. Although an exhaustive generation of the design space would be impossible for real tasks, a restricted analysis might be both feasible and helpful. One would need: (1) a particular cognitive task analysis for how an unaided mind would perform a task (i.e., an analysis of the of data and processing required), (2) a list of known perceptual operators that might substitute for deliberate reasoning, and (3) a catalogue of common principles for re-representing data to optimize cognitive processing. Then #1 could be stepped through for ways of distributing data and processing, and then ways of applying #2 and #3 could be envisioned. This sort of analysis has proven feasible even for non-trivial tools such as software engineering tools [28].

In the case of the above sort of analysis, the process described generates a restricted space of design options. This might be used to make design envisionment more systematic. Alternatively, design options might conceivably be analyzed automatically (much as, for instance, how Casner automated perceptual substitution design [7]). The design space analysis might also eventually lead to improved designer resources. For instance, design heuristics and cookbooks might be produced which codify successful implementations of each known type of cognitive support.

## 4 Toward Mining Patterns *and* Theory

A philosophy guiding our development of RODS was that it will be fruitful to crystallize design knowledge by simultaneously considering our rich wealth of successful design within the context of our existing theories of cognitive support. Our belief is that a good place to begin looking for patterns of good design is in the intersection between craft wisdom and scientific knowledge.

Towards this goal we have been applying RODS to extracting design insights from the world of software engineering tools. While doing this we have been building and refining RODS itself. Essentially we have tried to examine successful tools from the field while looking towards DC theory to justify the patterns we have found [27]. Our results are very preliminary, but we have found that RODS is helpful for giving the "gist" of an enormous variety of different types of support. At the same time, we have found that classifying a tool idea as an instance of cognitive support tends to pull in related literature that may be used to justify or refine the idea.

An excerpt of the results so far hints at the type and breadth of analysis. Table 1 lists several exemplars of tool features, a categorization of these as support implementations, and a list of literature that might be linked to the idea based on its association with the support type. Work has been started on using cognitive models to generate a finer classification system for cognitive support types [27].

## 5 Related Work

This work builds upon and synthesizes select aspects of many prior works from HCI, DC and cognitive science. These include works by Hutchins [11], Zhang and Norman [31, 17], Wright *et al.* [30], and Rasmussen [19]. The main point of departure from prior works is the

emphasis on clearly articulating a small, comprehensive, and orthogonal set of computational principles underlying the different support classes.

There exist several attempts at synthesizing a generalized, abstract understanding of cognitive support and related cognitive design issues. Neuwirth *et al.* [15] produced an analysis of how artifacts can assist in several cognitive tasks in hypertext authoring. Scaife *et al.* [22] try also to provide a synthetic account of the cognitive benefits of graphical representations. Both of the above analyses consider several instances of the cognitive support types identified by RODS. Many other works touch on a smaller or more restricted collection of cognitive support types. These include frameworks and summaries by Tweedie [26] and Narayanan *et al.* [14].

Another collection of related works consider generalizable, high-level cognition-related design issues. The most prominent of these is probably the “cognitive dimensions” framework [8, 3] (CDs). The CDs are intended to represent orthogonal dimensions of usability tradeoffs in notational systems [8]. The CDs framework itself seems essentially orthogonal to RODS. Each class of cognitive support identified by RODS generates design considerations for notational systems. Thus the CDs may be a very helpful resource for identifying design tradeoffs while reasoning about cognitive support.

A third stream of related work is the proposal by Sutcliffe *et al.* [25] for producing a reusable federation of psychological claims together with their theoretical justifications. The most obvious surface difference between their work and RODS is that their claims libraries are proposed to be built up through a taxonomy-driven claims abstraction and collection process, whereas RODS is borne directly from prior theories. Nevertheless their approach aligns well with RODS. Notably, it might be possible for each of the cognitive support classes identified by RODS to be considered as abstract, reusable claim in some future extension of the work of Sutcliffe *et al.*

## 6 Conclusions and Implications

DC is a theoretical framework which appears to have significant promise for application in HCI. One need which has to date been inadequately met is for a clear articulation of a general framework for analyzing cognitive support. RODS takes steps towards providing such a framework. The framework defines primitive support classes and outlines a general method for generating a restricted design space consisting of various compositions of cognitive support types (subject to some particular analysis of a cognitive task). This capability has far reaching implications for the specification, design, and evaluation of interactive systems.

One set of implications stem from the succinct but inclusive definition of cognitive support. With such a definition it is conceivable to try to specify at design time the cognitive support desired or required and then work systematically during implementation to satisfy

SUPPORT	SAMPLE LITERATURE	EXEMPLARS
distribution (data)	Zhang & Norman [31], Wright <i>et al.</i> [30]	code inspection checklists, diagramming constraints, undo history, link visitation history
distribution (processing)	Zhang & Norman [31], Scaife & Rogers [22]	type checking, compiler error list generation
substitution	Casner [7] Larkin & Simon [13]	browser link colouring (visual search), algorithm an- imations, call graph visualizations

**Table 1.** Examples of the types of RODS analyses explored.

these requirements. For instance, a usability engineer might specify that a certain amount of memory offloading must be provided. Tests might be run periodically to ensure that usability targets (i.e., “usefulness targets”) are being met. We have performed a preliminary exploration of this possibility in the context of a study of observing and measuring plan and planning distribution in commercial software development environments [27].

A second set of implications stem from the fact that cognitive support identifies ways of actually improving cognition. Thus “good” design moves are defined and reified (made concrete). Reifying “good” design moves can be expected to be helpful [30]. It may help systematize design space exploration and thus help eliminate designer oversights, thereby reducing the dependence on early usability testing. Our analysis of a reverse engineering tool [28] (using a variation of RODS) lends evidence to this supposition. Our analysis showed that applications of cognitive support theory (based on RODS) could anticipate design changes made by the tool’s designers only after receiving user feedback.

In conclusion, we must emphasize that the importance of RODS may be in terms of a high-level, qualitative theory for understanding an entire field. If the analogy to mechanical support is apt, RODS may make it possible for HCI designers to forgo talking about usability blunders at the implementation level, and begin to speak directly in terms of usefulness at the cognitive level.

## References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.
2. D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4):345–420, Dec. 1997.
3. A. F. Blackwell, C. Britton, A. Cox, and T. R. G. Green *et. al.* Cognitive dimensions of notations: Design tools for cognitive technology. In M. Benyon, C. L. Nehaniv, and K. Dautenhahn, editors, *Instruments of Mind: Proceedings of The Fourth International Conference on Cognitive Technology*, volume 2117 of *Lecture Notes in Artificial Intelligence*, pages 325–341, Berlin, 2001. Springer-Verlag.
4. S. Buckingham Shum and N. Hammond. Delivering HCI modelling to designers: A framework and case study of cognitive modelling. *Interacting with Computers*, 6(3):314–341, 1994.
5. L. Cardelli. Type systems. In *Handbook of Computer Science and Engineering*, chapter 103, pages 2208–2236. CRC Press, 1997.
6. J. M. Carroll, editor. *Designing Interaction: Psychology at the Human-Computer Interface*. Cambridge Series on Human-Computer Interaction. Cambridge University Press, 1991.
7. S. M. Casner. A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 10(2):111–151, 1991.
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. British Informatics Society, Cambridge University Press, 1989.
9. 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.
10. J. Hollan, E. Hutchins, and D. Kirsh. Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction*, 7(2):174–196, June 2000.
11. E. L. Hutchins. *Cognition in the Wild*. MIT Press, 1995.
12. J. H. Larkin. Display-based problem solving. In D. Klahr and K. Kotovsky, editors, *Complex Information Processing: The Impact of Herbert A. Simon*, chapter 12, pages 319–341. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.
13. J. H. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65–99, 1987.
14. N. H. Narayanan and R. Hübscher. Visual language theory: Towards a human-computer interaction perspective. In K. Marriott and B. Meyer, editors, *Visual Language Theory*, chapter 3, pages 87–128. Springer-Verlag, 1998.

15. C. M. Neuwirth and D. S. Kaufer. The role of external representations in the writing process: Implications for the design of hypertext-based writing tools. In *Proceedings of the 2nd Annual ACM Conference on Hypertext*, pages 319–341, 1989.
16. A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, Mar. 1976.
17. D. A. Norman. *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison-Wesley, Reading, Massachusetts, 1993.
18. H. Petroski. *The Evolution of Useful Things*. A. Knopf, New York, NY, 1992.
19. J. Rasmussen. Skills, rules, knowledge: Signals, signs, and symbols and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):257–267, 1983.
20. Y. Rogers and J. Ellis. Distributed cognition: an alternative framework for analysing and explaining collaborative working. *Journal of Information Technology*, 9(2):119–128, 1994.
21. D. E. Rumelhart and D. A. Norman. Representation in memory. In R. C. Atkinson, R. J. Herrnstein, G. Lindzey, and R. D. Luce, editors, *Stevens' Handbook of Experimental Psychology*, volume 2: Learning and Cognition, pages 511–587. John Wiley & Sons, New York, 2nd edition, 1988.
22. M. Scaife and Y. Rogers. External cognition: How do graphical representations work? *International Journal of Human-Computer Studies*, 45(2):185–213, 1996.
23. W. Schönplflug and K. B. Esser. Memory and its *Graeculi*: Metamemory and control in extended memory systems. In C. A. Weaver III, S. Mannes, and C. R. Fletcher, editors, *Discourse Comprehension: Essays in Honor of Walter Kintsch*, chapter 14, pages 245–255. Lawrence Erlbaum, 1995.
24. M. Sharples. Writing as creative design. In C. M. Levy and S. Ransdell, editors, *The Science of Writing: Theories, Methods, Individual Differences, and Applications*, pages 127–148. Lawrence Erlbaum Associates, 1996.
25. A. Sutcliffe. On the effective use and reuse of HCI knowledge. *ACM Transactions on Computer-Human Interaction*, 7(2):197–221, June 2000.
26. L. A. Tweedie. Interactive visualisation artifacts: How can abstractions inform design? In M. A. R. Kirby, A. J. Dix, and J. E. Finlay, editors, *People and Computers X, Proceedings of HCI'95*, pages 247–265. Cambridge University Press, 1995.
27. A. Walenstein. *Cognitive Support in Software Engineering Environments: A Distributed Cognition Framework*. PhD thesis, School of Computing Science, Simon Fraser University, May 2002.
28. A. Walenstein. Theory-based cognitive support analysis of software comprehension tools. In *Proceedings of the 10th International Workshop on Program Comprehension (to appear)*, 2002.
29. C. Ware. The foundations of experimental semiotics: a theory of sensory and conventional representation. *Journal of Visual Languages and Computing*, 4(1):91–100, 1993.
30. P. C. Wright, R. E. Fields, and M. D. Harrison. Analyzing human–computer interaction as distributed cognition: The resources model. *Human Computer Interaction*, 15(1):1–41, Mar. 2000.
31. J. Zhang and D. A. Norman. Representations in distributed cognitive tasks. *Cognitive Science*, 18:87–122, 1994.