# HASTI: A Lightweight Framework for Cognitive Reengineering Analysis

Andrew Walenstein
*Department of Computer Science*
*University of Victoria, Canada*
*walenste@csr.uvic.ca*

Keywords: POP-I.B. Teaching Design, POP-IV.B. Features of Software Solutions, POP-V.A. Cognitive Theories, POP-V.A. Design Theories.

## Abstract

Many software development tools and environments are considered useful because they beneficially reengineer developer cognition. Consequently, to design useful computer tools in cognitive work domains such as software development, one needs to be able to appreciate the possibilities for improving cognition. We argue that to do this, designers need more appropriate cognitive theories and models. In particular, they need models designed to highlight cognitive reengineering possibilities and issues. To address this need, we propose a lightweight modeling framework called HASTI. The intent of HASTI is to support "quick and dirty" cognitive reengineering analysis during the early stages of design. We illustrate HASTI's applicability in analyzing cognitive support in software development tools.

## Introduction

Software development is a work domain in which human cognition plays a critical role. Computer tools in such a domain are normally considered "good" if they can improve the cognitive ergonomics of the users. Since some sort of cognitive work environment almost invariably exists before design begins, tool design is, in a sense, cognition re-design. It may be worrisome, therefore, that relatively little in the way of psychological theory is used in everyday software development practice (Bellotti et al., 1995), or even by typical computing science researchers (Green, 1989). After all, even if psychological theory is not used, design is still psychological in nature—software designers merely fill the theoretical vacuum with folk psychology (Kirlik, 1995).

Cognitive science has been turned to in hopes of delivering theories to tool designers, but this has proven harder than was perhaps first expected (Carroll, 1991; Card, 1989). Many proposals have been advanced for putting more effective cognitive theories into the hands of designers. These include proposals for simplified, quantitative engineering models (e.g., GOMS (Newell and Card, 1985), ICS (Barnard and May, 2000)), programmable models that embed psychological knowledge in a computational model (e.g., PUMs (Young et al., 1989)), simplified cognitive models to "walk through" for design guidance (e.g., Norman's model (Carroll and Rosson, 1992)), and carefully-selected collections of concepts and terms for highlighting cognition-related design issues (e.g., Cognitive Dimensions (Green, 1989), Resources Model (Wright et al., 2000)). Success, so far, has been mixed.

We propose to add to a different line of exploration. The point of departure begins with a realization about the role of cognitive modeling in HCI activities. The realization is that the main reason for modeling cognition during design is to expose ways of modifying it to make it better. This is a critical realization, but one whose implications for HCI do not seem to be fully realized yet.

Most commonly, the purpose for cognitive modeling in HCI is assumed to be to predict or explain facets of human performance and behaviour. Frequently the aim of such work is to expose usability problems. These are goals suitable for tool evaluation, or for requirements analysis. Designers desire a different capability: they want to go beyond merely being able to identify usability problems in existing tools—they want to use cognitive models to determine how to design new tools which improve cognition. Kirlik insightfully argued this point in terms of a contrast in problem spaces:

> A standard modeling approach in cognitive psychology is to hold a task environment relatively fixed and to create a description of the cognitive activities underlying a person's behavior in that environment. The designer, on the other hand, is faced with the reverse challenge of creating an environment to elicit a desired behavior ... In problem solving terms, the solution space for the scientist is a set of plausible cognitive theories, whereas the solution space for the designer is a set of technologically feasible environments. We can thus characterize the scientist's problem

as a search among possible cognitive "solutions" to a given task and the designer's problem as a search among possible environments to obtain a given cognitive solution. (Kirlik, 1995, pg. 385-6)

Designers indirectly reengineer cognition by engineering technological environments. The main purpose of a cognitive model in this context is to say what should be reengineered, and how. It must be able to suggest cognitive improvements without first having a design in hand.

Once this role of cognitive models is realized, the question becomes: what sort of modeling framework is appropriate for the job? Our suggestion comes in two parts. First, we suggest that primary *content* of the framework is a principled decomposition of design-relevant aspects of cognition. More specifically, the decomposition should highlight how these aspects can be reengineered. With the framework, a designer can create an inventory of things to reengineer, and then reason about how to go about doing so. Second, we propose that the *structure* of the model should match the way that the cognitive aspects were decomposed. This ensures that computational reengineering possibilities are made apparent.

We propose HASTI as a modeling framework built according to the above recipe. HASTI is intended for "quick and dirty" cognitive reengineering analysis. The key proposal in HASTI is a five-fold decomposition of cognitive issues. These five "dimensions" of cognition are expected to help the designer catalog aspects of cognition to reengineer, realize reengineering possibilities, and trace the implications thereof.

The following sections describe HASTI and its construction, and then outline how HASTI may be used in analyzing cognitive reengineering. We show a scenario of applying this sort of analysis to software development tools.

## HASTI

Plato is commonly cited as having said that science must "carve nature at its joints" so that the different parts can be modeled independently. Kirlik appreciated the implications in terms of carving up cognitive models useful for design:

... the central problem that needs to be overcome to make the products of cognitive science more relevant to design is identifying a more productive set of dimensions along which modeling efforts can be decomposed. To support cognitive engineering, the decomposition must be derived from an overall framework capable of ensuring that the resulting research products can be reassembled into a coherent theory useful for design. (Kirlik, 1995, pg. 69)

If Kirlik is correct, then the main outstanding problem to solve in cognitive modeling for design is not the classic concerns of model veridicality, completeness, or simplicity. Rather it is finding a model decomposition structure that exposes design issues. To date, it has been hard to find good solutions to this problem (Green, 1990; Kirlik, 1995).

HASTI is a cognitive model decomposition-and-integration framework. HASTI identifies five different "dimensions" or "aspects" of cognition. "HASTI" is an acronym created from the names of the model structures associated with the five aspects: (1) a **H**ardware model, (2) an **A**gent model, (3) a **S**pecialization layering, (4) a **T**ask or problem partitioning, and (5) an **I**nteraction abstraction hierarchy. These five modeling methods are summarized in Table 1. They are briefly overviewed in the following subsections. The description highlights the way each aspect is decomposed, and the modeling techniques used. Further details appear in (Walenstein, 2002a). Since HASTI is meant for lightweight analysis, the emphasis will be on simplifying and idealizing the existing science base. Links to the underlying science base are made afterwards.

| | ***Cognition Aspect / Dimension*** | ***Decomposition*** | ***Model technique*** |
|---|---|---|---|
| H | invariant capabilities/constraints | memory, processors | Hardware model |
| A | goal-related behaviour | agents, resources | Agent model |
| S | specialization (adaptation) | SRK taxonomy | layering |
| T | task/problem | 4-fold taxonomy | partitioning |
| I | interaction | virtual architectures | layering |

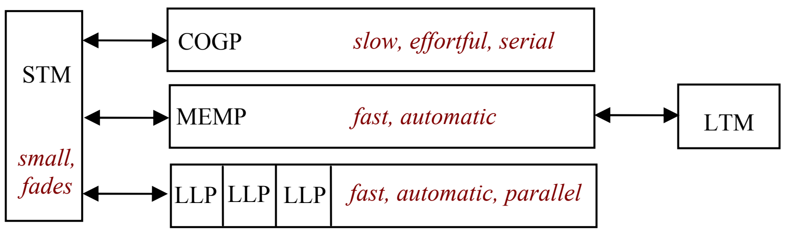

*Figure 1—Hardware model of invariant capabilities*

| ***Resource*** | ***Description of Role*** | ***Resource Data Types*** |
|---|---|---|
| Problem | track problem/task and how to solve it | ends, operations, constraints |
| Agenda | track goals to achieve | goals |
| Control | structure action sequencing | plans |
| Progress | track problem solving status | current state, history |

*Table 1—Five dimensions of cognition, and the way they are modeled*

## Hardware Model: Invariant Cognitive Capabilities and Constraints

**Dimension & Decomposition.** Humans uniformly possess cognitive capabilities and constraints which are independent of tasks, training, and environment. These are thus suitable candidates for modeling in a fixed model. The universal cognitive capacities include processing and memory capacities. Memory can be divided into long- and short-term memory. Short-term memory is small and transient; long-term memory is capacious and permanent. Processing is divided into high-level, low-level, and memory-based processing. High-level processing can be equated to deliberate reasoning. It is serial, effortful, and slow. Low-level processing corresponds to low-level perceptual abilities and learned skills. There are independent processors for each ability or skill. They run in parallel, and are automatic and fast. Memory-based processing is also automatic. It relies heavily on pattern recognition and expertise to be able to recall and reuse solutions to problems. Thus it differs from both low-level cognition (largely independent of deep expertise) and high-level cognition (uses reasoning to deliver solutions).

**Model.** The implementation is as a simple, labeled cognitive architecture diagram shown in Figure 1. Processing and memory types are implemented as computing elements, and their capacities and limitations are denoted. COGP is associated with cognitive processing, LLP with perceptual and skill processing, and MEMP with memory-based processing. STM and LTM indicate short-term and long-term stores respectively.

## Agent Model: Behaviour Decomposition

**Aspect & Decomposition.** Human cognition does not flit from moment to moment: it is characterized by episodes of coherent, purposive, goal-driven, semi-organized activity. This cognitive behaviour may be decomposed into locally-coherent goal-directed activities. This could be done at a relatively large granularity (reading, typing, writing, etc.) or at finer granularities (proposing an hypothesis, defining a variable, entering a command, looking up a function name, etc.). Although these activities are often decomposable in this manner, the particular activities depend upon the user, task, and environment, so they cannot be specified in advance. The modeler must take a hand in defining these activities.

Regardless of the decomposition of behaviour chosen, several situation-independent *resources* can be identified and categorized. These resources and their associated data values are summarized in Table 2. The first resource is a *problem definition*. Human behaviour is commonly modeled as motion within a problem space. The structure of this space must be known in order to how to move within it. One needs to track *ends* (desired outcomes), *operations* (possible moves) and *constraints* (restrictions on moves). *Agenda* management is a way of juggling multiple *goals* during the search. *Control* management is a way of organizing action by making and following *plans*, i.e., action ordering structures. In addition, *progress* in the problem solving needs to be tracked, both the *current* solution state, and *history* of previous states (for backtracking, avoiding repetition, etc.).

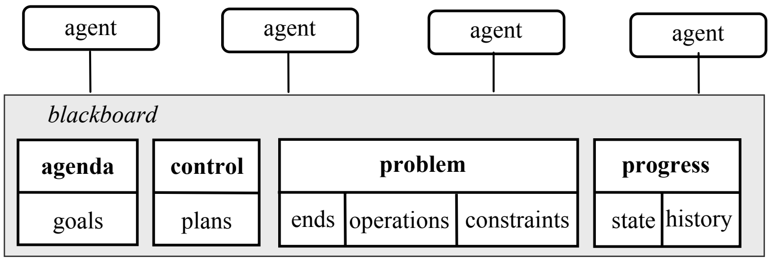

*Figure 2—The Agent model.*

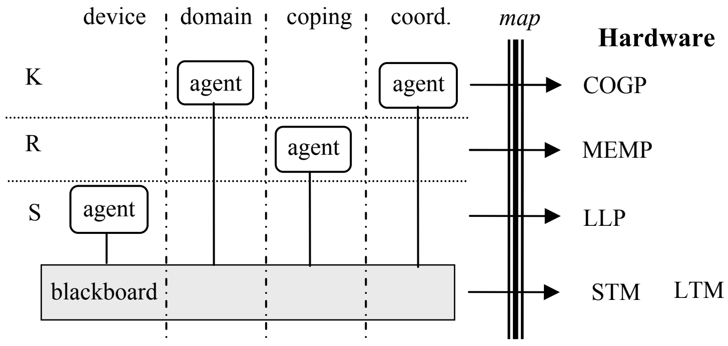

*Figure 3—Agent model mapping, stratification and partitioning*

**Model & Mapping.** A schematic model is shown in Figure 2. The overall model borrows from the blackboard and agent modeling literature (e.g., (Craig, 1997)). Distinguishable coherent behaviours are identified with individual agents generating that behaviour. These agents are presumed to coordinate through the shared blackboard, and to interleave processing opportunistically. The various resources identified above are stored on distinct panels in the blackboard. The Agent model is mapped onto the Hardware model, that is, the Hardware model is seen to "run" the "software" of the Agent model. Agents map down onto COGP, LLP, and MEMP, and blackboard panels map down onto STM and LTM. This mapping is shown in Figure 3. When multiple agents map onto the COGP, they are interleaved (i.e., the user seems to opportunistically switch between activities). The Hardware model in this diagram has been simplified to the point of only naming the relevant architectural modules.

The analyst is allowed considerable leeway in interpreting cognition in Agent terms. For instance it is relatively easy to view the software comprehension model of (von Mayrhauser and Vans, 1995) as a variant of such an agent model. Their model integrates three prior models of program comprehension by proposing that three different comprehension processes (top-down, bottom-up, bottom-up/systematic) take turns building knowledge in a shared knowledge base. This can be cast into the Agent model by portraying the three processing components as separate agents, and modeling the knowledge base as being stored in the problem state panel.

## Specialization Hierarchy and Its Stratification

**Aspect & Decomposition.** Human cognition is frequently specialized or *adapted* to particular tasks and environments. Rasmussen proposed the "SRK" taxonomy to distinguish three categories of behaviour resulting from these differences in adaptation levels (Rasmussen, 1983). The three behaviour categories are called "skill-based", "rule-based" and "knowledge-based". Skill-based behaviour is automatic, quick, and subjectively easy. It involves the use of highly-adapted perception and action skills. Rule-based behaviour is also automatic, fast, and easy. It involves deep expertise and can be done without deep reasoning. Knowledge-based behaviour is slow and serial and normally requires conscious mental effort.

**Mapping.** The SRK taxonomy can be mapped by imposing structure on the Agent model and its mapping to the Hardware model. First, the S, R, and K, behaviours can be seen to impose layers or strata on the agents of the Agent model. Agents are modeled as existing on one specific layer. For instance, recognizing familiar programming idioms (Brooks, 1983) might reasonably be said to exist solely on the R level (the recognition act is fluid and not introspectable). The SRK decomposition refines the mapping of the Agent model to the Hardware model. Specifically, the S, R, and K agents are mapped down onto LLP, MEMP, and COGP processors respectively. This mapping is illustrated in Figure 3. The mapping allows one to trace the agent level to the properties of processing on that level. So, for instance, if an idiom-matching agent is modeled on the R level, it implies the user calls on deep expertise to rapidly recognize the idiom and its implications.

### Task Taxonomy and Its Partitioning

**Aspect & Decomposition.** The notion of a problem space is powerful, but it is often worthwhile decomposing this into independent sub-spaces. The trick is to derive a suitable typology of sub-problems, or task types. Currently four task types are considered by HASTI:

- Domain: related directly to work activities
- Device: related to operating the device (i.e., computer)
- Coping: work done to avoid limitations/contingencies
- Cooperation: work done to coordinate in shared work

The aim of this task taxonomy is to allow the analyst to slice up the overall problem space into these subspaces so that they can be associated with design issues. For instance, an analyst might model writing code as a *domain* task, invoking actions in a programmer's editor as a *device* task, writing down a reminder to implement a function as a *coping* task, and checking in code to a repository as a *cooperation* task. Each of these classifications brings a host of related design issues to the fore.

**Mapping.** Because agents in the Agent model are responsible for performing various tasks, they can be typed or stratified accordingly. Since the SRK taxonomy decomposes behaviour in an orthogonal way, this can be implemented as orthogonal layering on the agents. For instance a skilled agent that can define a program function in a text editor can be assigned to the intersection of the *S* and *device* layers; the idiom recognizer could be classified as existing on the *domain* layer. This stratification is illustrated in Figure 3. The two agent orthogonalities generate a grid to classify the agents. An agent's classification can thus be referred to in their coordinate position (e.g., the idiom recognizer would exist in (knowledge, domain)).

### Interaction Abstractions and Virtual Architectures

**Aspect & Decomposition.** Interactions between human and computer can be analyzed at various levels (e.g., see (Moran, 1981)). One might, for example, differentiate between low-level device interactions and higher-level device actions. For instance, interactions with a notepad may be analyzed in terms of keystrokes and mouse motions, in terms of device actions such as loading and saving, or at a cognitive level in terms of saving and recalling knowledge from an external memory (see e.g., (Flor and Hutchins, 1991) for such an analysis in programming). Activities on the cognitive level are *implemented* or *simulated* at the device interaction level. This induces a simulation hierarchy on the interactions. HASTI currently defines two simulation levels: simulating a virtual memory and simulating a shared-memory computer. These are described below.

**Model & Mapping.** Cognitive level interactions can be modeled using a standard trick in computing science: a virtual architecture. A virtual architecture is much like a regular cognitive architecture, but it is simulated by other hardware. Using this concept allows the cognitive models to encapsulate device-level interactions by mapping the Agent model onto a virtual architecture instead of directly onto the Hardware model. A *virtual memory architecture* (Card et. al., 1986) can be used to encapsulate external memory use. Then, memory actions like *load* and *store* are simulated by the user. Such a model allows analysts to consider the Agent's blackboard as residing partially on the shared memory. A simple extension is a *shared memory parallel processing architecture* which allows the analyst to consider the agents in the Agent model as being located either in the human's head or on a computer.

## Reasoning about Reengineering Cognition with HASTI

HASTI is proposed as a lightweight modeling framework that highlights possibilities for reengineering cognition. This ability relies on the framework's *computational model structure*. If we follow Plato's carving metaphor, then a good decomposition framework should identify *joints*. Joints correspond to *degrees of freedom*, that is, places where motion or rotation can occur.

This metaphor leads us to the crux of HASTI: providing an articulated model to "play with" during design reasoning. The metaphor is that design reasoning corresponds to exploring ways of "rotating" a HASTI model along one of its joints. Each such rotation identifies a class of design moves. An analogy might be made to playing a Rubik's cube game: the cube can be rotated to explore the solution space. In an analogous way, the would-be cognitive engineer would seek to explore changes to a HASTI model in order to discover or reason about potential cognitive arrangements. This is precisely what Kirlik suggests that designers do—search the solution space of cognitive arrangements.

What are the "joints" of HASTI? Since HASTI is a computational model, cognitive reengineering corresponds precisely to *computational reengineering*. Computational reengineering may be defined as reorganizing the structure of a computational system without changing its essential function (Chikofsky and Cross, 1990). Thus the ways of improving cognition can be identified with different computational reengineerings of HASTI. With a HASTI model and a list of computational reengineerings, the designer may explore various reengineerings of cognition.

In conjunction with HASTI, we have been developing a list of orthogonal reengineerings called RODS (Walenstein, 2002a). The full details are outside the scope of this paper. However, two relevant cognitive reengineering principles identified by RODS can be briefly described: (1) *distributing* cognition onto artifacts (e.g., Zhang and Norman, 1994), and (2) designing representations that allow more specialized processing (skill, rule) to substitute for less specialized processing (rule, knowledge) (e.g., Rasmussen, 1983; Casner, 1991). We call these "distribution" and "specialization" transformations. Both transformations identify potential improvements to cognition.

To apply RODS to HASTI, the analyst first defines an Agent model of how users perform their tasks. Then the model can be inspected for opportunities for distribution and specialization transformations. Distribution moves agents or resources onto tools. The list of generic resource data types provide a list of common distributions (e.g., goal offloading, plan offloading, etc.). The agents direct attention to potential task-specific processing distributions. The agents also can highlight possibilities for specializing the processing. Any of these transformations can add or move agents between the four task types (domain, device, etc.).

We have performed several such analyses of software development environments, and have found the analyses to be useful. We have reconstructed key design rationales for two reverse engineering tools (Walenstein, 2002b), and have analyzed program error tracking support in a commercial software development environment (Walenstein, 2002a). Although these have been relatively complicated, a simple imaginary design scenario can be used to convey the essential nature of the analysis.

**Scenario**. Lorena is trying to support the task of fixing a syntactically invalid program. She models the fixing process using two agents, "search" (look for a syntax error), and "repair" (fix the error). She plots these in (knowledge, domain) since the user must deliberately reason about the errors. She decides to model the human and computer as agents sharing a blackboard (i.e., the computer display). She then considers distributing "search" onto the tool (e.g., a compiler). She then reasons that "search" needs to communicate the results to "repair". Lorena reasons that this can be as a plan of repairs (i.e., an error list). Now "repair" must coordinate by following the plan. She draws a "current plan item" in the Progress panel, a "get next" agent in (knowledge, coordination), and renames "repair" to "repair current plan item". She then realizes that "current plan item" could be forgotten because it is stored in STM, so she reasons about how to externalize it (e.g., keeping a "current error" pointer in the interface). Once externalized, the "get next" can be moved into (skill, coordination) if a simple "next error" action is made available. Next, she turns attention to "repair current plan item". She cannot figure out a way to distribute it, but she tries to figure out a way of moving it onto a lower specialization level. She decides that at least the sub-task of realizing what the current plan item is might be lowered. She reasons this could be done by coding the "current error" item in a different colour. This would allow visual search to replace effortful search.

The resulting analysis aligns with sequential error tracking features common to modern compilation environments. The point here, though, is not the result, but the account of how various design options can be cued by examining the evolving cognitive model for distribution and specialization opportunities. HASTI helps by relating tasks and cognitive resources to reengineering issues. For instance, once Lorena decides to plot a "repair" agent, issues of coordination, adaptation level, distribution and specialization possibilities are brought up by the way the model is structured. The analysis in the scenario above borrows from analysis frameworks such as the Resources model

(Wright et al., 2000) and Casner's perceptual substitution analysis (Casner, 1991). HASTI's model integration, however it permits smooth transition between reasoning about the various reengineerings.

## Relating HASTI to Other Resources

For many types of design, a HASTI analysis may not be enough. However HASTI was built as simplifications of other theories and resources. These or other resources could be used to iteratively deepen a HASTI analysis. Table 4 shows a list of some of the theoretical resources that HASTI was built from. Future work involves expanding this list, and making it easier to transition to the more detailed techniques.

Table 4 also identifies some of the prior work closely related to the various parts of HASTI. Besides these specific points of contact, many works have tried to integrate cognitive theories (e.g., (Newell, 1990). Most of these, however, are not aimed at broad-brush, design-oriented theories (although see (Green, 1990)). Several researchers have also pursued the issue of decomposition of cognitive issues (e.g., (Rasmussen, 1983), (Green, 1989), (Green, 1990), (Kirlik, 1995)), however to the best of our knowledge, comprehensive integrations like HASTI are absent. In addition there are various other attempts at making designer-oriented theory representations (e.g., (Barnard and May, 2000), (Blackwell et al., 2001)), however we know of none that effectively help highlight how cognition may be helpfully rearranged.

| | |
|---|---|
| **H**ardware | (Kieras and Meyer, 1997), (Newell, 1990) |
| **A**gent | (Wright et al., 2000), (Howes, 1995), (Craig, 1997) |
| **S**pecialization | (Rasmussen, 1983), (Casner, 1991) |
| **T**ask | (Moran, 1981), (Freed and Shafto, 1997), (Baecker et al., 1993) |
| **I**nteraction | (Card and Moran, 1986), (Flor and Hutchins, 1991) |

*Table 4 — major building blocks of HASTI*

## Conclusions

Computer tools are designed daily with little help from the thousands of cognitive science results that lie waiting. As Vicente laments:

> There are many theories and models that are potentially relevant to systems design ... [but] it is not at all obvious how to identify the knowledge relevant to any one design problem and then package this knowledge into a form that makes the practical implications for systems design more obvious. (Vicente, 1999)

We have proposed HASTI as a simplified integration framework for many existing models. Vicente's concern for making "practical implications for system design more obvious" resonates with HASTI's construction philosophy. HASTI provides a model structure that can be inspected for cognitive rearrangements, thereby generating design implications.

## References

Baecker, R. M., Nastos, D., Posner, I. R., and Mawby, K. L. (1993). The user-centred iterative design of collaborative writing software. In Conference on Human Factors in Computing Systems, pages 399–405, ACM Press.

Barnard, P. J. and May, J. (1993). Cognitive modelling for user requirements. In Byerley, P. F., Barnard, P. J., and May, J., editors, Computers, Communication and Usability: Design Issues, Research and Methods for Integrated Services, chapter 2.2, pages 101–145. Elsevier Science Ltd., Amsterdam.

Barnard, P. J. and May, J. (2000). Toward a theory-based form of cognitive task analysis of broad scope and applicability. In Schraagen, J. M., Chipman, S. F., and Shalin, V. L., editors, Cognitive Task Analysis, Expertise: Research and Applications, chapter 10, pages 147–163. Erlbaum.

Bellotti, V., Buckingham Shum, S., MacLean, A., and Hammond, N. (1995). Multidisciplinary modeling in HCI design…in theory and in practice. In Proceedings of Conference on Human Factors in Computing Systems, pages 146–153, ACM.

Blackwell, A. F., Britton, C., and et. al. (2001). Cognitive dimensions of notations: Design tools for cognitive technology. In Benyon, M., et. al., editors, Instruments of Mind: Proceedings of The Fourth International Conference on Cognitive Technology, pages 325–341, Springer-Verlag, Berlin.

Brooks, R. E. (1983). Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, 18(6):543–554.

Card, S. K. (1989). Theory-driven design research. In McMillan, G. R., Beevis, D., Salas, E., Strub, M. H., and Sutton, R., editors, Applications of Human Performance Models to System Design, pages 501–509, New York. Plenum.

Card, S. K. and Moran, T. P. (1986). User Technology: From Pointing to Pondering. In Proceedings of the ACM Conference on History of Personal Workstations, pages 183–198, ACM Press.

Carroll, J. M., editor (1991). Designing Interaction: Psychology at the Human-Computer Interface. Cambridge University Press.

Carroll, J. M. and Rosson, M. B. (1992). Getting around the task-artifact cycle: How to make claims and design by scenario. ACM Transactions on Information Systems, 10(2):181–212.

Casner, S. M. (1991). A task-analytic approach to the automated design of graphic presentations. ACM Transactions on Graphics, 10(2):111–151.

Chikofsky, E. J. and Cross, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, 7(1):13–17.

Craig, I. D. (1997). From blackboards to agents. In Online Proceedings of the VIM Project Spring Workshop on Collaboration Between Human and Artificial Societies.

Flor, N. V. and Hutchins, E. L. (1991). Analyzing distributed cognition in software teams: A case study of team programming during perfective software maintenance. In Empirical Studies of Programmers: Fourth Workshop, pages 36–64. Ablex Publishing Corporation, Norwood, NJ.

Freed, M. A. and Shafto, M. G. (1997). Human-system modeling: Some principles and a pragmatic approach. In Harrison, M. D. and Torres, J. C., editors, Design, Specification and Verification of Interactive Systems '97: Proceedings of the Eurographics Workshop. Springer-Verlag.

Green, T. R. G. (1989). Cognitive dimensions of notations. In Sutcliffe, A. and Macaulay, L., editors, People and Computers V, pages 443–460. Cambridge University Press.

Green, T. R. G. (1990). Limited theories as a framework for human-computer interaction. In Ackermann, D. and Tauber, M. J., editors, Mental Models and Human-Computer Interaction 1, chapter 1, pages 3–40. North Holland.

Howes, A. (1995). An introduction to cognitive modelling in human–computer interaction. In Monk, A. F. and Gilbert, G. N., editors, Perspectives on HCI: Diverse Approaches, chapter 5, pages 97–119. Academic Press Limited.

Kieras, D. E. and Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human Computer Interaction, 12(4):391–438.

Kirlik, A. (1995). Requirements for psychological models to support design: Toward ecological task analysis. In Flach, J., Hancock, P., Caird, J., and Vicente, K. J., editors, Global Perspectives on the Ecology of Human–Machine Systems, chapter 4, pages 68–120. Lawrence Erlbaum Associates.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 15(1):3–50.

Newell, A. (1990). Unified Theories of Cognition. Harvard University Press.

Newell, A. and Card, S. K. (1985). The prospects for psychological science in human-computer interaction. Human Computer Interaction, 1(3):209–242.

Norman, D. A. (1993). Things That Make Us Smart: Defending Human Attributes in the Age of the Machine. Addison-Wesley, Reading, Massachusetts.

Rasmussen, J. (1983). 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.

Vicente, K. (1999). Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work. Lawrence Erlbaum Associates, Mahwah, NJ.

von Mayrhauser, A. and Vans, A. M. (1995). Program comprehension during software maintenance and evolution. Computer, 28(8):44–55.

Walenstein, A. (2002a). Cognitive Support in Software Engineering Environments: A Distributed Cognition Framework, PhD thesis. School of Computing Science, Simon Fraser University, May.

Walenstein, A. (2002b). Theory-based cognitive support analysis of software comprehension tools. In Proceedings of the 10th International Workshop on Program Comprehension (to appear).

Wright, P. C., Fields, R. E., and Harrison, M. D. (2000). Analyzing human–computer interaction as distributed cognition: The resources model. Human Computer Interaction, 15(1):1–41.

Young, R. M., Green, T. R. G., and Simon, T. (1989). Programmable user models for predictive evaluation of interface designs. In Proceedings of Conference on Human Factors in Computing Systems, pages 15–19. ACM.

Zhang, J. and Norman, D. A. (1994). Representations in distributed cognitive tasks. Cognitive Science, 18:87–122.