Preprint
UCRL-JC-144942

Example Elaboration as a Neglected Instructional Strategy

T. R. Girill

This article was submitted to Association for Computing Machinery,
Special Interest Group for Documentation,
"SIGDOC 2001" Conference, Santa Fe, NM, October 21-24, 2001

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

July 18, 2001

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Example Elaboration as a Neglected Instructional Strategy

T. R. Girill
Lawrence Livermore National Laboratory
University of California
P. O. Box 808, L-72
Livermore, CA 94551 USA
trg@llnl.gov

ABSTRACT

Over the last decade an unfolding cognitive-psychology research program on how learners use examples to develop effective problem-solving expertise has yielded well-established empirical findings. Chi et al., Renkl, Reimann, and Neubert (in various papers) have confirmed statistically significant differences in how good and poor learners inferentially elaborate ("self-explain") example steps as they study. Such example elaboration is highly relevant to software documentation and training, yet largely neglected in the current literature.

This paper summarizes the neglected research on example use and puts its neglect in a disciplinary perspective. I then show that differences in support for example elaboration in commercial software documentation reveal previously overlooked usability issues. These issues involve example summaries, using goals and goal structures to reinforce example elaborations, and prompting readers to recognize the role of example parts.

Secondly, I show how these same example elaboration techniques can build cognitive maturity among underperforming high-school students who study technical writing. Principle-based elaborations, condition elaborations, and role recognition of example steps all have their place in innovative, high-school-level, technical-writing exercises, and all promote far-transfer problem solving.

Finally, I use these studies to clarify the constructivist debate over what writers and readers contribute to text meaning. I argue that writers can influence how readers elaborate on examples, and that because of the great empirical differences in example-study effectiveness (and reader choices) writers should do what they can (through within-text design features) to encourage readers to elaborate examples in the most successful ways.

Keywords

Cognitive psychology, documentation, example design, teaching technical writing, self-explanation, usability

SIGDOC'01, October 21-24, 2001, Santa Fe, New Mexico, USA
ACM 1-58113-295-6/01/0010.

1. INTRODUCTION

Example elaboration is a uniquely effective way to learn from worked technical examples. This paper summarizes 10 years of research that clarifies example elaboration. I then show how example elaboration can make complex software documentation more useful, improve the benefits of technical writing exercises for underperforming students, and enlighten the general discussion of how writers can and should help their readers.

2. EXAMPLE ELABORATION RESEARCH

2.1 Chi's Discovery

Michelene T. H. Chi and her colleagues, mostly at the Learning Research and Development Center in Pittsburgh, PA, have studied the nature of expertise for years. In 1989 they described a new direction in this work, aimed at discovering how students develop expertise by studying examples, sometimes even just a few examples [4]. Although the initial experiment was paid for by the Office of Naval Research and framed in the artificial-intelligence terms popular at the time, the results gave rise to a new research program (on learning from examples) with implications far beyond AI.

As subjects Chi et al. used 10 college students who had not yet taken physics (p. 151), asked them to study worked examples of statics problems (inclined planes, pulleys, etc.), then tested their performance on near and far transfer mechanics problems, all while collecting talk-aloud protocols that revealed their personal study techniques (p. 158). The researchers divided the students into "good" and "poor" groups post hoc, based on actual problem-solving performance, and then they looked for between-subject study-strategy differences in the way the students had used the examples provided.

Chi et al. found that good students and only good students studied worked examples by making numerous inferences about each example step. They

> ... tend to study example-exercises in a text by explaining and providing justifications for each action. That is, their explanations refine and expand the conditions of an action, explicate the consequences of an action, provide a goal for a set of actions, relate the consequences of one action to another, and explain the meaning of a set of quantitative expressions ([4], p. 175).

The researchers concluded that such inferential elaboration of examples is "a crucial phase in skill acquisition" (p. 169).

Good and poor students had the same level of understanding of Newton's laws before example study, they noted, but not afterward. Good students invoked significantly more components of Newton's laws while studying (inferentially elaborating) the example text. And elaborating the example steps "triggered and allowed" good students better access to their knowledge in a way unavailable to poor students (p. 168). When later using the (previously studied) examples to solve far-transfer problems, good students returned to the worked cases "with a very specific goal" (p. 175) and extracted a specific equation, feature, or subprocedure. Poor students, on the other hand, returned with a "general global goal" (p. 175) and usually just reread the whole example.

Chi et al. interpret effective learning from worked examples as "self-explanation." This interpretation begins in the very title of their focal paper and continues throughout its analysis of their experimental methods and their research results (even though "inferential elaboration," which I prefer here, is perhaps a more accurate alternative). For instance,

> ... we examine the explanations that students spontanneously produce [p. 176] ... we think self-explanations provide the means for the construction of inference rules which can be later proceduralized into usable skills [p. 178].

2.2 Renkl's Refinements

Alexander Renkl was aware of and intrigued by the "self-explanation" results of Chi's study of learning from worked-out examples [12]. But he was concerned about generalizing the results because of

- the small original sample size (10, yielding only 4 unambiguously good and 4 poor learners),
- the special domain (physics) covered by the examples, and
- failure to control for potentially confounding factors, primarily the time spent studying.

So he replicated the experiment with 36 college freshmen who studied worked-out examples of probability calculations under more controls than before. Renkl's results clarified and extended Chi's original findings in ways pertinent to documentation and training. The general experimental approach was the same: have all students refresh their mathematical background, take a pretest, study worked examples while verbalizing so their protocols could be categorized by the experimenters, and take a posttest on how well they transferred what they learned to new problems.

2.2.1 Clarified Self-Explanation Inferences

Chi's categories for protocol analysis, for analyzing just what the students did as they studied the worked examples, were never very explicit. Renkl, however, spelled out and illustrated his seven protocol categories clearly at the start ([12], pp. 8-9):

1. "Principle-based explanation" (recognizing an example step as an instance of a general [here, probability] principle).
2. "Goal-operator combinations" (inferring a step from the goal that it pursues).
3. "Anticipative reasoning" (predicting a result before checking the corresponding worked step).
4. Elaboration of a problem situation (inferring details from the givens of a problem).
5. Noting coherence (recognizing several different examples as similar).
6. Negative monitoring (acknowledged nonunderstanding of a step).
7. Positive monitoring (acknowledged understanding of a step).

The first three of these inferences (and only those) were reliably correlated (at the 0.80 level or above) with post hoc successful learning. These first three inferences also share the same underlying logical structure. All involve, overtly or covertly, instantiating a general conditional principle or inferring a conditional's consequent from its antecedent (*modus ponens*). Anticipative reasoning, possible here (but not for Chi) because of Renkl's progressive disclosure of the worked-example steps, is really logically equivalent to either of the first two inferences but simply done in advance of seeing the step to which it applies. So given their logical congruence, that just these three proved to be the inferential elaborations actually linked with later learning success is not surprising.

2.2.2 Time Controls

Chi's good students spent more time studying the worked examples than did the poor students, precisely because they were busy inferentially elaborating the steps. But since time on task is a general predictor of learning success, "it could not be definitely ruled out that the effective learners were superior merely because they devoted more time to elaborating the worked-out examples" ([12], p. 2). Renkl's replication controlled this extra variable: he limited all students to the same 25 minutes for example study, regardless of their personal study strategy. This made inferential practice, the phenomenon of interest to us, the *only* independent variable in his experiment.

2.2.3 Domain Flexibility

Because Chi chose physics (statics) for her example domain, the possibility arose that physical intuitions and problem diagrams unique to this domain might work against generalizing the results, even if genuine, to other problem domains with different features. Renkl's use of probability calculations (computing the probability of complex events by adding or multiplying the probability of their component simple events) put this concern about domain brittleness to the test.

His results showed that inferential elaboration of worked examples supports successful learning and subsequent problem solving in this rather different domain as well:

> In summary, successful learners tended to provide many principle-based explanations, to frequently anticipate to-be-computed probabilities, and to seldom state lack of comprehension. The explication of goal-operator combinations and the inspection of a relatively large number of examples seemed to be especially relevant for the

> medium transfer performance ... thus, the quality of self-explanations seemed to be of major importance for the acquisition of well transferable knowledge ([12], p. 17).

These findings increased the likelihood that the inferential elaborations first noted by Chi among good physics students would also apply to at least some of the topics routinely covered in software documentation and training. Numerical methods, program development, and code optimization (all very important in current large-scale simulation projects) are among the topics sufficiently like Renkl's probability examples to suggest that encouraging inferential elaborations (of the kind he listed) could be an effective, underused teaching strategy here too. But what about their relevance to learning more interactive, menu-oriented, commercial software? To this question P. Reimann (an early Chi colleague) and C. Neubert addressed themselves.

2.3 Reimann and Neubert's Extension

Psychologists P. Reimann and C. Neubert asked themselves "if self-explaining activities in the area of learning to use end-user software from worked examples are as effective as they are in other domains" ([11], p. 319). They took "self-explaining" activities just slightly more broadly than did Chi and Renkl before them, to mean "monitoring one's understanding as well as engaging in knowledge construction [by inferential elaboration] in order to overcome self-diagnosed problems of understanding" (p. 318). Their study of worked examples for software support was exploratory rather than definitive, using a quasi-experimental design with no control group. Their subjects were 10 adult accounting students just learning to use Microsoft Excel, and the worked examples for study comprised step-by-step Excel tables for dealing with various accounting problems. Reimann and Neubert did, however, follow Renkl and hold time on task constant at 120 min of study for every participant ([11], pp. 319-320).

Reimann and Neubert also paralleled Renkl in spelling out clearly the seven categories of elaboration into which they analyzed the verbal protocols of the students who studied the worked examples ([11], p. 320):

1. Condition elaboration (specifying the conditions under which an operation applies).
2. Structure elaboration (spelling out the role that a specific solution step plays in the overall solution).
3. Syntax elaboration (describing the form of a function or operator).
4. Effect elaboration (detailing the effects of applying an operator).
5. Taxonomy elaboration (classifying an operator or function).
6. Comparisons (comparing several operators or functions overtly).
7. Paraphrasing (repeating example text with no new information).

Their seven categories differ somewhat from Renkl's primarily because the examples in the interactive software domain involve more role recognition and interpretation, but less numerical calculation, than did Renkl's probability cases. For the elaborations near the top of the list, however, inference is still important (much less so near the bottom).

In the now-standard pattern, Reimann and Neubert divided their subjects post hoc into two groups of five successful and five unsuccessful learners, based on how they solved later spreadsheet problems. As in the physics and probability domains earlier, those spreadsheet "participants who self-explain with the goal to discover meaning prove to be better problem solvers than those who do not self-explain or who focus more on syntactic aspects of [worked] examples" ([11], p. 316).

Specifically, the first two elaborations (condition and structure elaborations, or role recognition) separate good from poor learners with a significance of 0.05 and 0.0006 respectively, and these are the only statistically significant differences between the groups (p. 322). Said another way, condition elaborations correlated at the 0.51 level and structure elaborations correlated at the 0.88 level with high problem-solving scores later, and these were the only significant correlations (p. 322). Although the domain is quite different, these two elaborations most resemble the inferential elaborations previously probed by Chi and Renkl because only here "participants relate a step of a worked-out solution to a more encompassing solution plan, in other words, they elaborate on the relation between single solution steps to [*sic*] goals and subgoals" ([11], p. 321). Role recognition thus becomes the primary way that the previous example-elaboration results generalize to cover learning to solve problems with interactive software by studying worked examples.

While acknowledging the exploratory scope of their work here, Reimann and Neubert recognize its potential value for documentation and training. They conclude that "the role of examples for fostering software skills is still underestimated," and that it can be enhanced if care is "taken that the meaning of the example steps—and foremost the subgoal structure—is well explained" by writers to encourage helpful elaborations by readers ([11], p. 323, 324).

3. BIBLIOGRAPHIC NEGLECT

Despite their apparent relevance, these "self-explanation" findings have been almost totally ignored in the documentation (and software training) literature of the last decade.

A few influential writers on documentation design are excused because their works were already in press by the time Chi published the first report on inferential elaboration of worked examples in 1989. Donald Norman's widely read *Psychology of Everyday Things* [9] predated Chi's paper by a year (1988). Likewise, R. John Brockmann's encyclopedic second edition of *Writing Better Computer User Documentation* [2], where such issues are much discussed in psychological terms, had already appeared in early 1990.

Two much more recent works that by design take a broad view of the field, however, surprisingly miss the value of example elaboration altogether. Karen Schriver's *Dynamics in Document Design* [13] (1997) does include Chi's 1989 paper in her long bibliography, along with follow-on work by Van Lehn and Jones published in 1992. But Schriver's only comment on this work in the text falls in her discussion of the extent to which readers of manuals can accurately self-assess their understanding of what they read ([13], p. 226). Chi and her colleagues initially conjectured that good students of examples performed better self-monitoring of their learning than did poor students. But this claim was *not* con-

firmed in the later generalizing studies by Renkl and (separately) by Reimann and Neubert, each of whom tested for it (e.g., [12], pp. 13, 17). So the spurious aspect of Chi's work was passed along by Schriver, while the important underlying insight that inferential elaboration is vital for studying examples well vanished from her analysis.

The contributors to John Carroll's *Minimalism Beyond the Nurnberg Funnel* [3] anthology (1998) likewise remain silent about the place of example elaboration in the design of minimal manuals. The studies reviewed above suggest that instructional text should be "minimal" enough to allow for student elaboration of worked examples, yet nonminimal enough to overtly invite and enable this behavior in readers (more on this below). Carroll's collection has the slimmest of topical indices and no name index at all. But a search through the individual article reference lists fails to turn up any mention of Chi or her later collaborators. Draper, Hackos, Mirel, and van der Meij, all of whom strive to place minimalism in its broad psychological and sociological context, completely omit example elaboration as a document design issue. I think that this shows an incomplete appreciation for the relevance of "educationally oriented" empirical studies to parallel problems that happen to fall outside the arena of formal schooling, namely, in software documentation.

Finally, although *Cognitive Science* (where most of the example elaboration research has appeared) is a strongly interdisciplinary journal, rhetoric and publishing are not among the disciplinary communities that it usually serves. Even more remarkable is the neglect of this work by Gary Perlman's Human-Computer Interaction Resources web site [10]. Sponsored by ACM SIGCHI, this site has bibliographic entries for over 20,000 papers related to documentation design and human-factors aspects of computing, broadly construed. Yet all three core papers on example elaboration, or indeed any papers by any of the authors cited above, are absent from this database.

4. APPLIED TO SOFTWARE DOCUMENTATION

The experimental results of Chi, Renkl, Reimann, and Neubert together suggest that the extent to which software documentation promotes example elaboration by its readers will affect its usability, especially when the examples are important and the topic is complex.

One way to explore the penetration of these example-handling psychological discoveries into software documentation is to compare two major technical publications that are:

- rich in examples,
- published by IBM's International Technical Support Organization on their widely used `www.redbooks.ibm.com` web site, and
- written collaboratively by IBM staff and major IBM customers.

The audience here is experienced, high-end computer users (scientists and engineers) for whom computer science is not their primary professional field. These readers are technically sophisticated but often unfamiliar with the algorithmic or programming features described. Hence, they are prime targets for detailed explanatory examples.

The "redbooks" of interest to us are a pair that address efficient parallel processing on machines with many CPUs, with either of two approaches: (1) using many concurrent processes, with special communication between them (MPI, [1]), or (2) using many concurrent "threads of execution" within a single process, where communication may be easier but synchronization harder (POSIX threads or Pthreads, [5]). Both books are about 200 pages long. And both contain dozens of worked examples in the chapters analyzed here (and hundreds altogether).

Comparative study of four example-related features of these software manuals reveals two trends. First, writer encouragement of inferential elaboration of examples (even abundant examples) by readers is *not* always the default approach to document design. Second, these manuals differ in perceived lucidity and usefulness in proportion as they contain example-elaboration aids (the Pthreads manual manages to be adequate, while the MPI manual sets a high standard for helpfulness and clarity).

4.1 Strategy Summaries

Are examples preceded by strategy summaries that invite "anticipative reasoning" (along the lines explored by Renkl)? Interestingly, both manuals attempt this to some extent, but the intellectual context (designing complex parallel software) works against major practical benefits here.

The Pthreads chapter in [5] has a long, subdivided section on how to synchronize threads, rich in worked examples (C-program excerpts) of five alternative thread-synchronization techniques. Preceding each technique is a short, clear strategy summary that outlines the value and strengths of the technique at hand. But none of these strategic introductions is integrated closely enough with the code example that follows to invite much anticipative reasoning. They do not invite self-answers to the question "how would I program that?"

The MPI manual [1] contains a major section (3.4) on parallelizing DO loops (in Fortran). In a pattern typical of the whole document, each subsection here begins with an overtly stated strategic principle that the body of the subsection exemplifies and unfolds, usually with a combination of prose, code samples, and simple diagrams. These strategic principles range from the very general ("distribute iterations among processes and let each process do its portion in parallel" (p. 54)) to the narrowly focused ("it is more efficient to access arrays successively in the order they are stored in memory than to access them irrelevantly" (p. 62), important because storage order differs between C and Fortran).

Even here, with overt strategic summaries introducing every subsection, actual anticipative reasoning by readers may be fairly rare. The subject is complex and the coding moves often subtle. More prosaically, without progressive disclosure (as used by Renkl) nothing prevents readers from simply skipping to the worked example's details without trying to anticipate them.

4.2 Goal Signals

Is the goal of each step spelled out (to reinforce reader elaborations)?

The Pthreads treatment spells out step goals only in the simplest, earliest examples. For instance, the "mutual ex-

clusion lock" example ([5], pp. 118-120) contains comments noting the goal of every line of code. But this practice evaporates as the thread synchronization techniques grow longer and more intricate. In the read/write lock and semaphore examples few step goals are even hinted at, much less announced. This may reflect the unreconciled work of many hands, with few coauthors aware that overt goal signals help readers effectively elaborate their examples.

Step goals are spelled out routinely in the MPI manual, however. Here each step often involves a cluster of related code lines that together achieve some sought effect. The DO-loop parallelization section (introduced above), for instance, continues ([1], pp. 62-66) with a series of carefully layered comparisons that make clear the goal of each successive step (code cluster):

1. handling small versus large stride size, then
2. data dependencies in one direction, then
3. data dependencies in both (two) directions, then
4. parallelizing inner and outer loops at the same time.

These are added subcase by subcase to always reveal what specific problem (goal) each next cluster of code, each specific programming technique, addresses. So reader elaboration based on step goals is promoted more thoroughly here than in the Pthreads manual.

4.3 Subgoal Structure

Is the example set's subgoal *structure* overt? These manuals also diverge markedly in how much they encourage example elaboration by disclosing the framework of goals that holds the individual cases together in a meaningful way.

In the Pthreads treatment [5] the subgoal structure is implicit. The programming samples are themselves explicit, of course. But the authors provide no systematic map, in prose or graphics, to reveal how the five-part thread synchronization discussion is organized or why the various parts are included (at all, much less in the order chosen). Readers are free to infer goals here, but they get no support or encouragement from cues or statements in the text.

The MPI manual, on the other hand, offers abundant cues to the subgoal structure of its examples. The discussion of how to distribute loop iterations among processors is typical of this more overt approach ([1], pp. 54-58). Here: (1) Line diagrams visually contrast three loop distribution alternatives (block, cyclic, and block-cyclic). (2) Before/after code samples show each suggested parallelization change at the subroutine level. (3) Assessing how well each technique meets a program's (i.e., a reader's) goals is aided by frequent and overt comparisons of the effects of different choices (e.g., "the cyclic distribution incurs more cache misses than the block distribution," (p. 57)). This also promotes recognizing the conditions under which it is appropriate to pursue certain goals, and hence to deploy certain goal-relevant techniques.

Overt MPI goal announcements commonly begin each case studied (e.g., "minimize interprocess communication") so readers can easily and regularly compare their mental model of the goal structure with the structure built by the authors. These extra prompts are not intrusive. But they provide support on almost every page for goal-related elaboration of the examples by those who read them to solve programming problems.

4.4 Role Recognition

Is role recognition of the example parts encouraged?

The Pthreads manual shows the same implicit pattern here that we have seen above regarding example goals. Brief introductory comments indirectly suggest that each of the five thread synchronization techniques discussed (and separately exemplified) has its own intended role (to make a thread wait until a specified other thread completes, wait until a binary variable unlocks, wait until a specified condition is met, etc.). But the text almost entirely omits any within-example role explanation, comparison between roles filled, or comments on the role contribution of specific steps. In fact, there are no cues or prompts even distinguishing essential (role-crucial) from embellishment (role-trivial) features inside the Pthreads programming samples.

By contrast, the MPI manual encourages role recognition and elaboration. Although in-code comments are rare in the MPI programming examples, other kinds of scaffolding show clearly when roles shift within each worked case. For instance, in the section on how to parallelize one standard numerical method, the one-dimensional finite-difference method ([1], pp. 67-69), the nonparallel code example is followed by a parallelized version marked with line numbers. This enables subsequent line-by-line commentary that reveals just which MPI role each set of related lines plays. Elsewhere in the manual (e.g., the data synchronization section, pp. 73-74) simple but ingenious diagrams serve the same purpose. This text thus consistently invites its readers to note, compare, and elaborate on the roles that the examples and their parts fill.

5. APPLIED TO TEACHING TECHNICAL WRITING

A second neglected application for the example elaboration research analyzed here is to the superficially unrelated task of teaching technical writing in high school.

5.1 Cognitive Needs

Over the last decade technical writing has gradually moved into American high schools. The 18 chapters of Mary Sue Garay and Stephen Bernhardt's *Expanding Literacies* [6], for example, "examine the resistance against work-relevant instruction in [high-school] English, describe trends in workplaces that affect literacy, and seek to define best practices" (p. ix) in high-school technical-writing projects. Increasingly, technical writing is not just an advanced-placement adventure, but rather a part of mainstream or remedial English and science classes. The goal here is to offer an alternative path to practical literacy for those students for whom a traditional, literature-only writing program proves inadequate. In such cases, students must learn how to learn, not just how to write. Building underlying cognitive maturity, however slowly, is as important as teaching specific writing techniques.

Meeting this need is all the more difficult because many commercial training materials, even those aimed at the high-school audience, are unsuited to the task. Some are too abstract for students (and teachers) unprepared in science. Others are so unfocused or badly paced that they waste the chance to promote cognitive growth as students practice technical writing. The potential for a different approach

based on the psychological research summarized above is great.

5.2 An Example-Elaboration Response

Example-based technical writing exercises for underperforming high-school students, especially exercises well informed by the example-elaboration research discussed here, offer a highly relevant, highly promising alternative for the high-school classroom.

Because it lends itself to overt principles and checklist guidelines, instruction (as opposed to description) writing affords an excellent first place to develop this alternative approach. While rhetorical techniques summarized in overt instruction-writing guidelines [14] are the primary focus that students see, their own cognitive growth is a latent focus. With some carefully constructed exercises, students can practice what Renkl calls "principle-based elaboration" (recognizing a step in worked-example instructions as an instance of a general principle, in this case, a writing principle listed in student guidelines). Other exercises can practice what Reimann and Neubert call "condition elaborations" (specifying the condition(s) under which a good-instruction technique or writing move applies). Both of these example elaborations are significantly correlated with later problem-solving success. Practicing technical writing through them promotes more sophisticated reasoning in general, as well as just familiarity with the writing techniques that each example step employs.

In a high-school classroom setting I have experimented successfully with this example-elaboration approach. I used kitchen recipes as a concrete but logically parallel surrogate for abstract software instructions. I generated the study examples by introducing intentional flaws into the recipes (such as omitted, misordered, or too-complex steps). Students then studied a spectrum of cases ranging gradually from fully worked examples to similar but open-ended text-revision projects.

For instance, consider student study of a recipe containing the step "add cooked macaroni." Such a recipe should include as a previous step the instruction "cook the macaroni." As a fully worked example, this exercise would be scaffolded with an overt diagnosis of the problem (missing a needed step) and a suggested solution (what to insert and where). As a partly worked example, this exercise would be scaffolded with a pointer to the problem location and an invitation to fill in the missing step. Either way, the exercise directly encourages students to recognize their writing revision here as an instance of the general principle (listed on their guidelines) to "make all hidden steps explicit" when drafting instructions. Fostering such example elaborations thus includes but also goes beyond teaching the specific writing principle that each step illustrates.

5.3 Example Elaboration Extended

For teaching *description* writing to high-school students I have extended this approach to use "worked-example" descriptions of technologically complex but familiar objects (such as the paper clip, compact disk, or fluorescent lamp). Here the students study, then practice identifying, the specific (rhetorical) role filled by each part (first each paragraph, then later each sentence) in previously dissected technical descriptions. Recall that such role recognition and hence subgoal discovery were the "structure elaborations" most highly correlated with successful later problem solving in Reimann and Neubert's exploratory work on examples in software documentation.

Once again, astute teacher commentary and printed scaffolding can promote example elaboration. For instance, consider the worked-example description of a compact disk. Here sentences describe compact-disk structure by noting similarities with the single spiral groove on a phonograph record. Scaffolding prompts students to notice this text feature and (learn to) recognize it as giving an intentional comparison (one role on a provided list of descriptive rhetorical roles). This in turn encourages students to see this part of the description as fulfilling the explanatory subgoal of relating the unfamiliar to the familiar. Repeated exercises all structured and presented in this way make example elaboration a latent pedagogical theme as students learn about descriptive techniques. This prompted role recognition encourages active processing (beyond passive reading) of the worked-example descriptions in ways likely to improve far transfer. And it indirectly teaches students to try this style of studying examples on their own whenever they encounter them.

6. APPLIED TO CONSTRUCTIVISM

All of the "self-explanation" studies discussed above not only acknowledge but thoroughly explore the active role of readers (learners) in interpreting and processing the worked examples that they encounter. So one might well expect general insights from this research program for constructivism, for the cluster of theories that affords readers generous scope in "making the meaning" of what they read.

A recent review paper by Beverly Zimmerman offers a convenient place to start this analysis [16]. Zimmerman first introduces her basic approach to "meaning making":

> Social cognitive theory looks beyond text features like grammar, correctness, and convention to consider writing as a social activity, undertaken while collaborating with other writers and readers ([16], p. 32).

She then reveals the two very different faces that this view presents, each of which receives more extreme expression by other commentators in the same journal.

One interpretation of this theory makes writers *more* responsible than ever for the success of their readers (more to follow on this):

> ... the authors of the instructions have failed to consider the specific needs (education level, experience level, handedness, etc.) of the students who try to complete the process (p. 34)....providing support for usability testing, observing users using software in real situations, communicating the discourse practices and culture of the workplace—all of these are practical implications of the social cognitive theory of writing ... (p. 35).

But a second interpretation of the same theory makes writers *less* responsible, because so much seems out of their hands:

> Adopting a social cognitive theory of design would mean acknowledging that people construct multiple realities through social interchange—realities

> that change across time and culture ... the role of design would be to construct rather than to convey knowledge (p. 34).

Killingsworth and Rosenberg [8] take this hands-off interpretation to its logical extreme by suggesting that under it the writer's responsibility for designing an effective document actually approaches zero:

> [it] allows the user the greatest possible power and freedom ... [it] emphasizes the user's individuality and creativity, placing in doubt the very possibility of predicting user behavior. In one sense, it undermines the whole idea of document design ([8], pp. 33-34).

This second view, the one that nullifies writer responsibility, relies on three often overlooked assumptions. For the case at hand, of writing and reading worked examples in technical text, these three assumptions are:

1. The equivalency of elaborations: Which elaborative technique students of a worked example use to study it does not matter.

2. The democracy of interpretation: Students are as good as anyone at picking the approach to worked examples that best suits those examples.

3. The spontaneity of understanding: Readers construct the meaning of a text, such as a worked example, spontaneously as they read it, regardless of what the writer does.

The example-elaboration research program clarifies the soundness of each of these assumptions. For worked examples, every one of them turns out to be an empirical claim carefully studied during the last 15 years and found to be false.

The falsity of the first assumption (that all elaborations are equal) is of course the basic finding of Chi and all her colleagues. Some example-elaboration techniques (inference of a step from a general principle, inference to a step's goals, and anticipative inference) have repeatedly shown themselves to be much more supportive of later problem-solving success than their alternatives. Example study techniques are simply not created equal.

The falsity of the second assumption (that students can themselves pick good study techniques) follows from the ability of Chi and all her colleagues to easily divide their subjects into successful and unsuccessful learners post hoc. At least half of all subjects in the several example-elaboration experiments were unable to solve subsequent related problems well, despite access to the same examples as successful problem solvers. As Renkl concludes:

> ... learners, left to their own devices, typically fail to show effective learning behaviors when no external support (e.g., teacher guidance or scaffolding) is present ([12], p. 25).

The falsity of the third assumption (that "meaning making" is spontaneous) hinges on its last clause ("regardless of what the writer does"). Reader interpretations of texts, like 'free" economic markets, are almost never really unconstrained. Just as reader background knowledge, motivation, and vocabulary constrain text interpretation, so do the macroscopic (headings, comparisons, lists) and microscopic (clause structure, proleptic words) features of the text itself. And some of the constraints that writers provide in worked examples are intentional scaffolding that signals and facilitates reader use of the best-performing elaboration techniques. The software documentation and high-school teaching cases above have already illustrated this in-text scaffolding; see also Guzdial [7]. That readers "make meaning" never entails that they make it in a vacuum, nor that unconstrained interpretation would somehow be better even if it were possible.

All of this argues for returning to the first (writers are *more* responsible) view of constructivism suggested above. The three hidden empirical assumptions of the "writer irresponsibility" view are unfounded, or perhaps just represent a romanticized overconfidence in reader behavior when using worked examples. It just doesn't happen that way. Patricia Wright overtly draws the responsibilist conclusion when she comments on the (parallel) work of psychologist Richard Mayer:

> ... one way of encouraging appropriate [reading] strategy selection is through careful design of the text ... the onus for achieving successful communication cannot be safely left to the reader. Writers need to see themselves as catalysts for the strategies that their readers adopt; and they need to be aware of the design features that promote the selection of particular strategies ([15], pp. 38-39).

7. CONCLUSION

Example elaboration, not only in science prose but also in software documentation, enjoys solid empirical support as the crucial way to learn from worked examples in technical text. Although still largely overlooked (because of its origins outside the usual literature on rhetoric or instructional design), it promises to improve the usefulness of complex software instructions, to help underperforming students mature intellectually as they learn basic writing techniques, and to clarify the responsibilities of everyone who prepares and publishes examples for others.

8. ACKNOWLEDGMENTS

This work was performed in part under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract W-7405-ENG-48.

9. REFERENCES

[1] Y. Aoyama and J. Nakano. *RS/6000 SP Practical MPI Programming (SG245380).* IBM, Poughkeepsie, New York, 1999.

[2] R. Brockmann. *Writing Better Computer User Documentation from Paper to Hypertext.* John Wiley, New York, 1990.

[3] J. Carroll, editor. *Minimalism Beyond the Nurnberg Funnel.* MIT Press, Cambridge, MA, 1998.

[4] M. Chi, M. Bassock, M. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13(2):145–182, March 1989.

[5] R. Cutler, F. Armingard, E. Conejo, and K. Nagarajan. *C and C++ Application Development on AIX (SG245674)*, Chapter 5, POSIX Threads. IBM, Poughkeepsie, New York, 2000.

[6] M. Garay and S. Bernhardt, editors. *Expanding Literacies*. SUNY Press, Albany, NY, 1998.

[7] M. Guzdial. Supporting learners as users. *Journal of Computer Documentation*, 23(2):3–13, May 1999.

[8] M. Killingsworth and M. Rosenberg. The evolution of document design since 1985. *Journal of Computer Documentation*, 19(3):31–35, August 1995.

[9] D. Norman. *The Psychology of Everyday Things*. Basic Books, New York, 1988.

[10] G. Perlman. Human-computer interaction resources. URL: www.hcibib.org, 2001.

[11] P. Reimann and C. Neubert. The role of self-explanation in learning to use a spreadsheet through examples. *Journal of Computer Assisted Learning*, 16(4):316–325, 2000.

[12] A. Renkl. Learning from worked-out examples: A study on individual differences. *Cognitive Science*, 21(1):1–29, January 1997.

[13] K. Schriver. *Dynamics in Document Design*. John Wiley, New York, 1997.

[14] P. Wright. Editing policies and procedures (Ch. 4). In T. Duffy and R. Waller, editors, *Designing Usable Texts*, pages 63–96. Academic Press, New York, 1985.

[15] P. Wright. Reading strategies, mental models, and text design. *Journal of Computer Documentation*, 19(3):38–45, August 1995.

[16] B. Zimmerman. Linda Flower and social cognition: Constructing a view of the writing process. *Journal of Computer Documentation*, 22(3):25–37, August 1998.