ADAPT^{IT}: Tools for Training Design and Evaluation

☐ Marcel B.M. de Croock
Fred Paas
Henrik Schlanbusch
Jeroen J.G. van Merriënboer

This article describes a set of computerized tools that support the design and evaluation of competency-based training programs. The training of complex skills such as air traffic control and process control requires a competency-based approach that focuses on the integration and coordination of constituent skills and transfer of learning. At the heart of the training are authentic whole-task practice situations. The instructional design tools are based on van Merriënboer's 4C/ID methodology (1997). The article describes a training design tool (Core) that supports the analysis and design for competency-based training programs and an evaluation tool (Eval) that supports the subsequent revision of this training design.*

☐ Traditional knowledge-based training can be characterized as an approach in which the learning goals primarily describe what learners should *know* at the end of training. Hence, instructional design primarily focuses on the identification, ordering and presentation of information about a skill or learning domain. In contrast, learning goals for competency-based approaches describe what learners should *be able to do* after training. Instructional design then focuses on the identification of constituent skills that make up the competency or complex skill and on designing a training blueprint that helps students learn to perform these skills in a coordinated and integrated manner. Various studies have provided evidence that competency-based approaches to training design are more effective in attaining transfer performance than knowledge-based approaches (for an overview, see van Merriënboer, Clark, & de Croock, 2002).

Van Merriënboer developed the four-component instructional design model (4C/ID* methodology, 1997) to support the development of competency-based training programs. This model provides methods and techniques for: (a) analyzing a complex cognitive skill into its constituent skills and their interrelationships; (b) analyzing the different knowledge structures that may be helpful or are required to be able to perform the constituent skills; and (c) designing a training blueprint with, as a base, a sequence of whole task practice situations that support integration and coordination of the constituent skills. The model is fully consistent with cognitive load theory, because its instructional methods ensure that learners are continually confronted with whole tasks that yield an acceptable cognitive load (van Merriënboer, Kirschner & Kester, submitted; for a review, see Sweller, van Merriënboer, & Paas, 1998). Its consistency with cognitive load theory allows training designers to develop training programs that take the limited processing capacity of the human mind into account.

Applying the 4C/ID* methodology for the design of competency-based training is not an easy task. The amount of intermediate and final products that are produced during analysis and design is large and those products are highly interrelated. As a result it is easy to lose the overview over the complete design process, which may impair decision making. Therefore, in a European-Community–funded project called Advanced Design Approach for Personalized Training—Interactive Tools (ADAPT^{IT}), we are currently developing a set of software tools that will help designers apply the 4C/ID* methodology. A first tool, Core, supports the analysis of a complex skill and the design of a competency-based training blueprint. A second accompanying tool, Eval, supports the evaluation of the training program and the subsequent revision of the blueprint on which the training program is based. Note that the actual development of the training program as well as its implementation is not supported by the ADAPT^{IT} tools.

The forthcoming sections first describes what a competency-based training blueprint designed according to the 4C/ID* methodology looks like. Next, the Core and Eval tools are discussed. Finally, the discussion addresses the evaluation of the tools and some directions for future research.

THE 4C/ID* METHODOLOGY

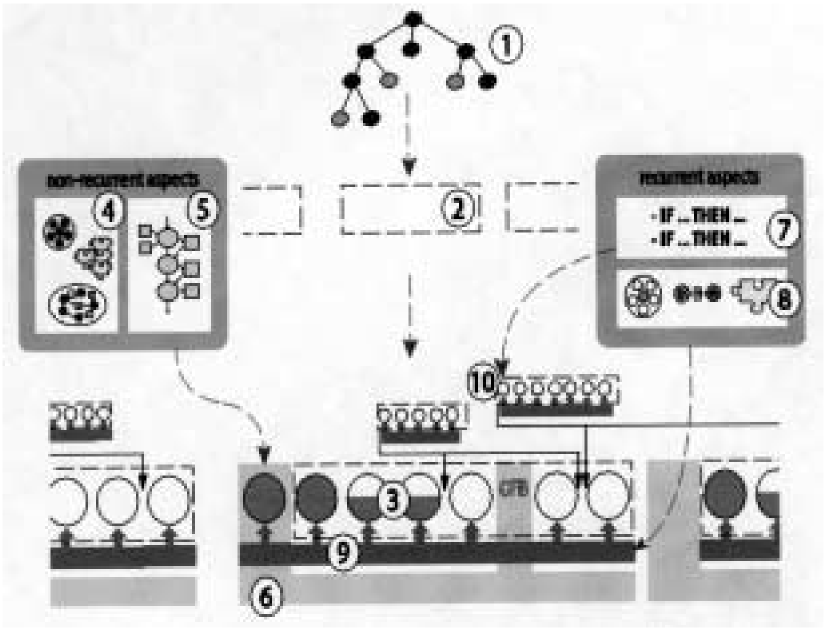
A basic assumption of the 4C/ID* methodology is that environments for complex learning can be described in terms of four interrelated blueprint components (van Merriënboer et al., 2002):

1. *Learning tasks,* which are the actual tasks the learners will be working on during the training program. Learning tasks are organized in a simple-to-complex sequence of task classes, that is, categories of equivalent learning tasks. Learning tasks within the same task class start with high built-in learner support, which disappears well before the end of the task class (i.e., a process of "scaffolding").
2. *Supportive information,* which is helpful to the learning and performance of aspects of the learning tasks that require variable performance over problem situations. It explains how a domain is organized and how to approach tasks or problems in this domain, and provides cognitive feedback on the quality of task performance.
3. *Just-in-time information,* which is information that is prerequisite to the learning and performance of aspects of learning tasks that show invariant performance over problem situations. It provides algorithmic specifications of how to perform those aspects.
4. *Part-task practice,* which provides additional repetitive practice for selected constituent skills that need to be performed at a very high level of automaticity after the training. It is only necessary if the learning tasks do not provide enough repetition to reach the desired level of automaticity.

The bottom part of Figure 1 shows a schematic view of the four components. The learning tasks are represented as circles (numbered *3*), organized in task classes (the dotted boxes around a set of learning tasks), and showing a decrease of support within task classes (indicated by the dark filling of the circles). The supportive information is represented in the L-shaped, light gray figures that are connected to the task classes (numbered *6*) and may also contain cognitive feedback (CFB). The just-in-time information is represented in the dark gray rectangles, with upward arrows that indicate that units of just-in-time information are connected to separate learning tasks (numbered *9*). Finally, part-task practice is represented by sequences of small circles (i.e., practice items; numbered *10*).

The 4C/ID* methodology can be described as an organized set of 10 activities that may help to create a detailed training blueprint. Four activities pertain to the design of the blueprint components described above: the *design* of learning tasks (3), supportive information (6), just-in-

Figure 1 ☐ The ten main activities of the 4C/ID* methodology (see text).



time information (9), and part-task practice (10). The other 6 activities are preparatory and *analytical* in nature. They provide the input necessary for the design activities. The first activity, *decompose the complex skill* (1), is concerned with identifying the constituent skills and their interrelationships. The result is a so-called intertwined skills hierarchy. The goal of the second activity, *sequence task classes* (2), is to make a first rough training design, by specifying a simple-to-complex sequence of categories of learning tasks characterizing authentic problem situations. These first 2 activities form the basis for the design of learning tasks, which completes the skeleton of the training blueprint.

The remaining activities flesh out this skeleton. The activities *analyze mental models* (4) and *analyze cognitive strategies* (5) are concerned with the identification and description of knowledge structures that may be helpful to perform the so-called nonrecurrent constituent skills, that is, the skills that require variable performance over problem situations. The activities *analyze rules and procedures* (7) and *analyze prerequisite knowledge* (8) result in the identification and description of the knowledge that must be presented to the learners because it enables the performance of so-called recurrent skills, that is, the skills that show identical performance over problem situations. The next section will discuss Core and illustrate how the first three activities are supported by this tool.

CORE

The goal of Core is to support designers with analyzing complex skills and designing competency-based training blueprints according to the 4C/ID* methodology. Available ID methodologies and tools are often criticized for the fact that they are not compatible with the way designers perform their job (Rowland, 1992). Therefore, Core allows instructional designers to perform their tasks as they prefer (Goel & Pirolli, 1991; Perez, Johnson, & Emery, 1995). For instance, Core supports "zigzag" design, which includes a top-down approach (starting with task analysis); a bottom-up approach (starting with the design of learning tasks); or any alternative approach, such as starting from the middle by first making a sequence of task classes. Furthermore, Core allows other systemic forms of nonlinear, cyclical and iterative design, giving designers optimal freedom in their approach.

Other functions that Core offers pertain to the management of all the complete or intermediate products that are constructed during the analysis and design process (e.g., skills hierarchy, task class descriptions, learning tasks, etc.). First, Core provides functions for entering, editing, storing, maintaining and reusing analysis and design products. This is achieved primarily by providing templates for easily entering information, ensuring that products are created in a way that is consistent with the 4C/ID* methodology. Second, Core provides functions for examining the products from multiple perspectives. The designer can filter out undesired information, specify different textual and graphical views on products (e.g., zoom in on supportive information for a selected task class, list just-in-time information for all learning tasks, examine differences between task classes, etc.), and use printing-on-demand functions to export the blueprint in any desired format. Third, Core provides functions to check whether the analysis and design products are complete, internally consistent, and in line with the 4C/ID* methodology. Finally, wherever possible Core provides utilities to automate analysis and design activities.

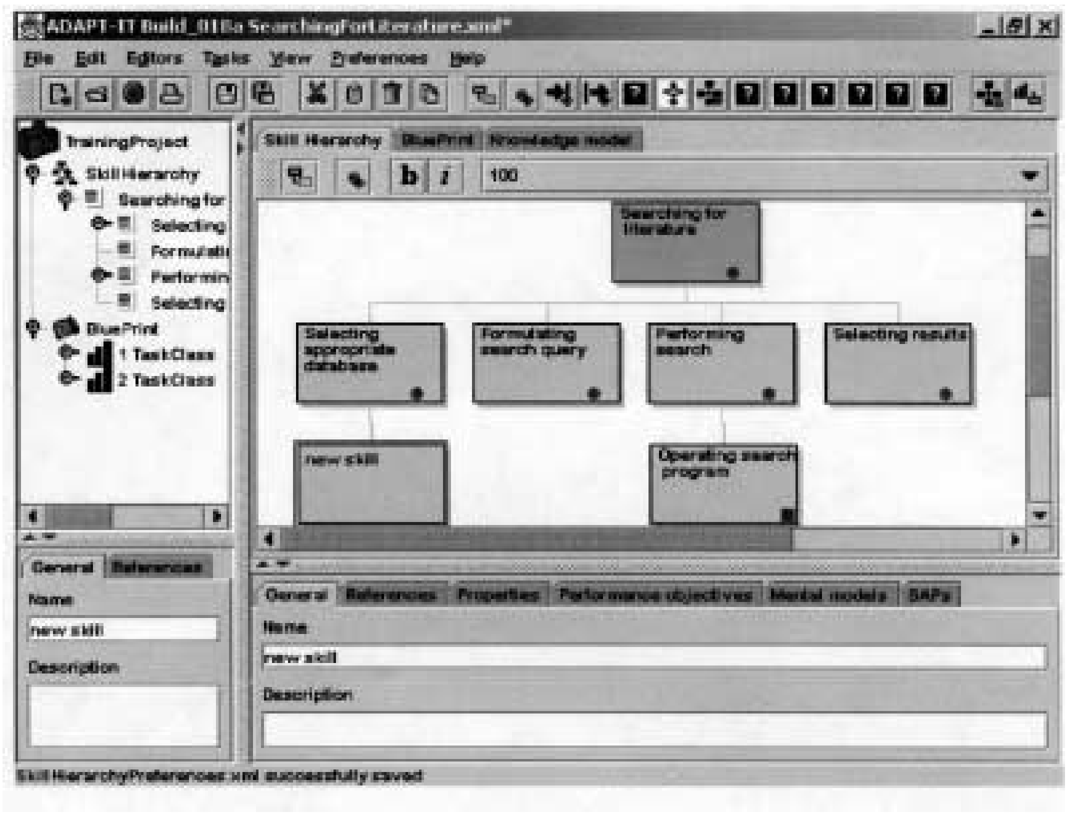
The next section describes the main interface of Core. Then, a scenario is presented of a designer who uses Core to perform the first three activities of the 4C/ID* methodology. Finally, the functions that support the seven remaining activities are briefly discussed.

Core Main Architecture

Figure 2 shows an overview of Core's main interface. The screen is divided into four different regions. In the upper left corner there is the *project browser,* which is used to display all elements of the training project in a hierarchical way. In addition, the project browser gives the designer access to other training projects and, if applicable, to other files with relevant information that can be imported into a 4C/ID* project. To the right of the project browser, in the upper right corner, the *diagram browser* is located. This browser provides access to three diagram editors, which allow for the manipulation of the same elements as in the project browser. However, the elements are now represented in a more intuitive graphical format. The following three diagram editors are available: (a) the skills hierarchy diagram editor, (b) the knowledge model diagram or KMD-editor, and (c) the blueprint diagram editor. At the bottom right, the *detailed editors* are located. These editors are used to specify information for the different types of elements (e.g., particular constituent skills in the skills hierarchy diagram, particular learning tasks in the blueprint diagram, etc.) that can be selected in the diagram editors. In the bottom left corner, the *draft editor* is located. It can be used for the quick naming and description of elements in the training project and for displaying aggregate information about the diagram that is displayed in the current diagram editor.

Both the project browser and the diagram browser have functions that allow a designer to filter out undesired information and display only the products that are needed at a particular moment. Interaction between the different parts of the tool is fully visual, meaning that linking, adding, and rearranging elements is simply done by "drag and drop." Navigation is data driven, meaning that the system always presents the editors and details for the currently selected element. By right-clicking an element, the system will present the user all commands that are available for that element. Because Core also is a client-server application, a designer can log on to a server from everywhere, as long as Internet access is available. This function enables a design team with members at different locations to collaboratively work on a training design project. Finally, Core is implemented in the Java programming language and, as much as possible, based on open standards such as shareable content object reference module (SCORM) and extensible markup language (XML). This allows for the exchange of design products with other applications that also adhere to those standards.

Figure 2 ☐ Prototype of the Core tool showing the skills hierarchy diagram.



Design Scenario

Below, a scenario is presented of a designer who uses Core to perform the first three activities of the 4C / ID* methodology, that is, (a) the decomposition of a complex skill, (b) the sequencing of task classes, and (c) the design of learning tasks. This scenario illustrates the functions of Core that were discussed above. While the activities are discussed in the specified order, it should be noted that a designer is actually free to start with any activity he or she prefers.

Using Core to decompose the complex skill The main purpose of this activity is to create a skills hierarchy that displays all constituent skills that make up whole-task performance and interrelates these skills to each other. Figure 2 shows a part of the skills hierarchy for the complex skill, "searching for literature" (for a complete description see van Merriënboer et al., 2002). The skills hierarchy diagram editor supports the designer, who is building a skills hierarchy. Skills can be added, removed, and rearranged in the hierarchy by means of drag and drop, after which the hierarchy is automatically reformatted. Additional relations can be drawn between skills at the same level to indicate the order in which they are performed: temporal relations for skills that have to be performed in a particular order; simultaneous relations for skills whose performance is highly interrelated or concurrent, and transposable relations for skills that can be performed in an arbitrary order. The user can zoom in on the skills hierarchy diagram to see more details, or zoom out to get a global overview.

Another purpose of the decomposition of the complex skill is to describe performance objectives for each constituent skill and to classify them on a number of dimensions. A detailed editor, the skills editor, is used to describe the performance objectives. The different aspects of a performance objective, such as the starting situation, conditions under which the task must be performed, desired results, and standards for acceptable performance are described by clicking on the tab labeled "performance objectives" and then filling out the associated forms. The tabs of the skills editor always show the forms for the currently selected skill in the skills hierarchy diagram editor. All products that result from 4C/ID* activities and that are logically linked to a particular skill are also labeled in the skills editor. For instance, the tab "SAPs" contains a form that allows the user to list systematic approaches to problem solving that were identified during the observation of an expert; these SAPs may be further analyzed with the KMD-editor as part of the main activity, "analyze cognitive strategies." Classifying skills is done by clicking on the properties tab in the skills editor and then filling out the associated form. In the skills hierarchy diagram editor, properties such as whether a skill is recurrent or nonrecurrent can be directly visualized in the elements that represent skills. For example, in Figure 2 round dots in the elements indicate that a skill is nonrecurrent; squares indicate that a skill is recurrent. By applying filters, the desired information only can be selected for display. Filters are set in the Preferences menu.

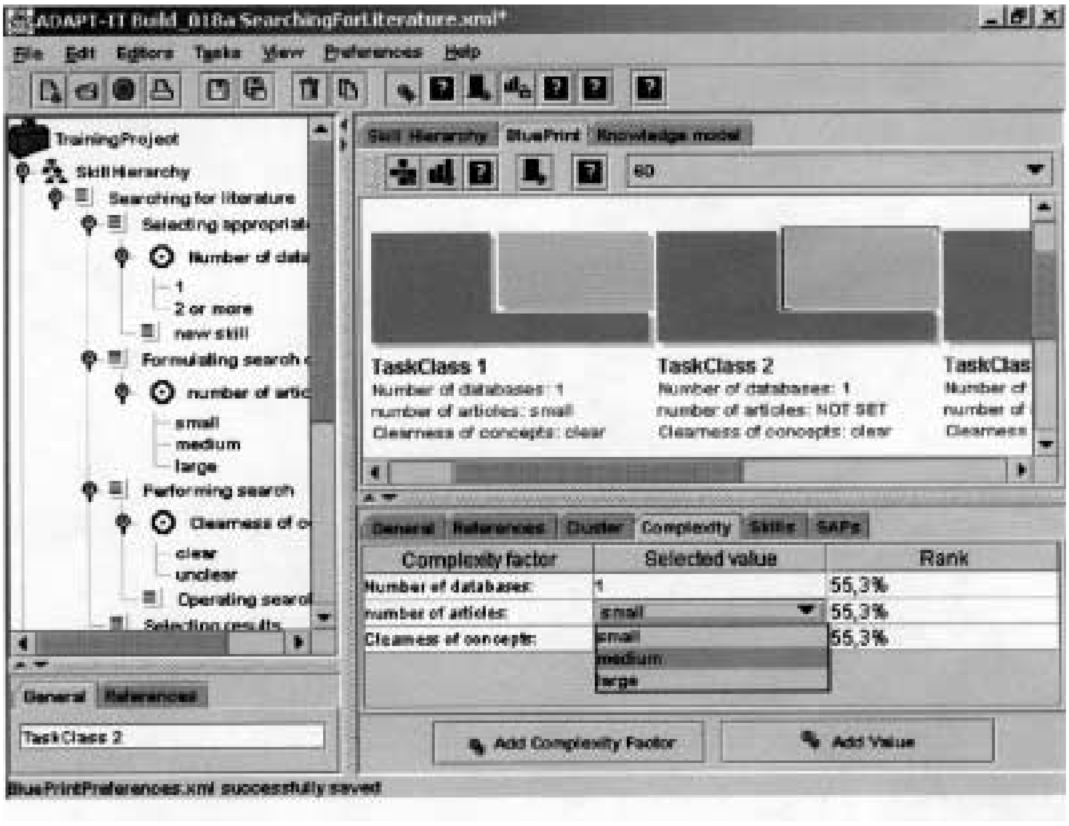
Core can check if prescriptions from the 4C/ID* methodology are consistently applied. An example of this feature can be seen when skills are marked as recurrent or nonrecurrent. If a skill is marked as nonrecurrent, the 4C/ID* methodology specifies that all parent skills are nonrecurrent as well. The user can choose to have the system either automatically apply this principle by updating the hierarchy whenever a classification of a skill is changed, or to graphically indicate where in the hierarchy inconsistencies with this principle appear. Consistency checking for this principle, as well as for many other principles of the 4C/ID* methodology, can be enabled or disabled at the user's convenience.

Using Core to sequence task classes. The main purpose of this activity is to design a global outline of the training program by describing a simple-to-complex sequence of *task classes,* that is, categories of meaningful, whole-task practice situations that will confront the learners during the training. These task classes form the basis for the next main activity, the design of learning tasks, in which the designer describes a number of equivalent tasks for each task class (i.e., learning tasks of roughly equal difficulty that can be performed on the basis of the same body of knowledge). The 4C/ID* methodology describes several approaches that can be used to sequence task classes, such as emphasis manipulation, progressive mental models, and simplifying assumptions. Here, it will be demonstrated only how Core supports task-class sequencing according to the *simplifying assumptions approach.*

The basic idea behind the simplifying assumptions approach is that all conditions that might simplify or complicate performance of—parts of—the whole skill are identified. Training then starts with a task class that contains the simplest but authentic tasks that a professional might encounter in the real world. In subsequent task classes, the simplifying assumptions are relaxed, so that the situations that the training is based on become more and more complex. To apply the simplifying assumptions approach, the designer first has to identify and specify so-called *complexity factors.* A complexity factor is a variable, with values assigned that indicate different levels of complexity for performing the skill. A user can create a new complexity factor in the skills hierarchy diagram by right-clicking on the highest-level constituent skill to which the complexity factor applies, and then choosing the appropriate command from the pop-up menu. In a detailed editor, the task-class editor, the user can then specify the different values for the newly created complexity factor.

After the complexity factors have been identified the user can start to create task classes. Figure 3 shows the blueprint diagram with a sequence of three task classes and the associated task-class editor. The designer can add, edit, and delete task classes in the blueprint diagram. The system shows a list of all complexity factors for each task class. In the task-class editor, the designer can set values for each factor to specify the level of complexity for each task class. If the designer fails to set a value for a complexity fac-

Figure 3 ☐ Prototype of the Core tool showing the blueprint diagram, with a sequence of task classes.



TaskClass 2

TaskClass 1
Number of databases: 1
number of articles: small
Clearness of concepts: clear

TaskClass 2
Number of databases: 1
number of articles: NOT SET
Clearness of concepts: clear

Complexity factor	Selected value	Rank
Number of databases	1	55,3%
number of articles:	small	55,3%
Clearness of concepts:		55,3%

BluePrintPreferences.xml successfully saved

tor, the system gives a warning indicating that the design of that particular task class is incomplete. After all values have been assigned, the tool can automatically conduct a consistency check. If the values for the complexity factors for the different task classes are not set in such a way that each following task class is of a higher complexity than the previous one, the system gives a warning to the user.

Core also offers functions to automatically create a task-class sequence based on the complexity factors specified by the user. If this function is used, the designer must enter a rank order for the complexity factors, the desired number of task classes and, optionally, the desired complexity factor values for one or more of the task classes in the sequence. The system then provides suggested complexity factor values for undefined task classes, while ensuring a smooth increase of complexity for the whole sequence, because the number of combinations of complexity factor values is equalized for each task class.

Using Core to design learning tasks. When a sequence of task classes has been created, learning tasks are designed for each task class. Or, vice versa, the designer may first design learning tasks and then create task classes to which those learning tasks can be assigned. According to the 4C/ID* methodology, learners receive much guidance for the first learning tasks within each task class, but guidance and support is gradually decreased to zero well before the end of the task class is reached. The next task class starts with a high level of guidance and support again, yielding a saw-tooth pattern for the level of support throughout the whole training program. In a detailed editor, the learning-task editor, Core provides a set of templates for different types of learning tasks that can be used to design a smooth decrease of built-in learner support. Currently supported learning tasks are case studies, modeling examples, completion tasks, goal-free tasks, reverse tasks, imitation tasks and conventional tasks (see van Merriënboer et al., 2002, for a full description).

The bottom right part of Figure 4 shows the learning-task editor with a partly filled-out template for a completion task (i.e., a task for which the learner must complete a partially given solution). All templates consist of a problem description and an assignment for the learner. The problem descriptions follow the general problem solving model of Newell and Simon (1972) and contain the elements: *given state,* criteria for an acceptable *goal state,* and a *solution* for bridging the gap between the given state and the desired goal state. In addition, a description may be given of the problem-solving process leading to an effective solution. A designer creates a learning task with a certain amount of learner support by first selecting a template for a particular type of learning task and then specifying the necessary elements. The templates already contain the relevant assignment for the learner; for instance, to complete the given solution (for a completion task), to describe a situation for which the given solution is effective (for a reverse task), or to come up with a new solution (for a conventional task). The user is prompted by the system to complete or omit parts of the problem elements in the template in a way that is consistent with the chosen learning task format (see van Merriënboer, 1997, for a complete discussion).

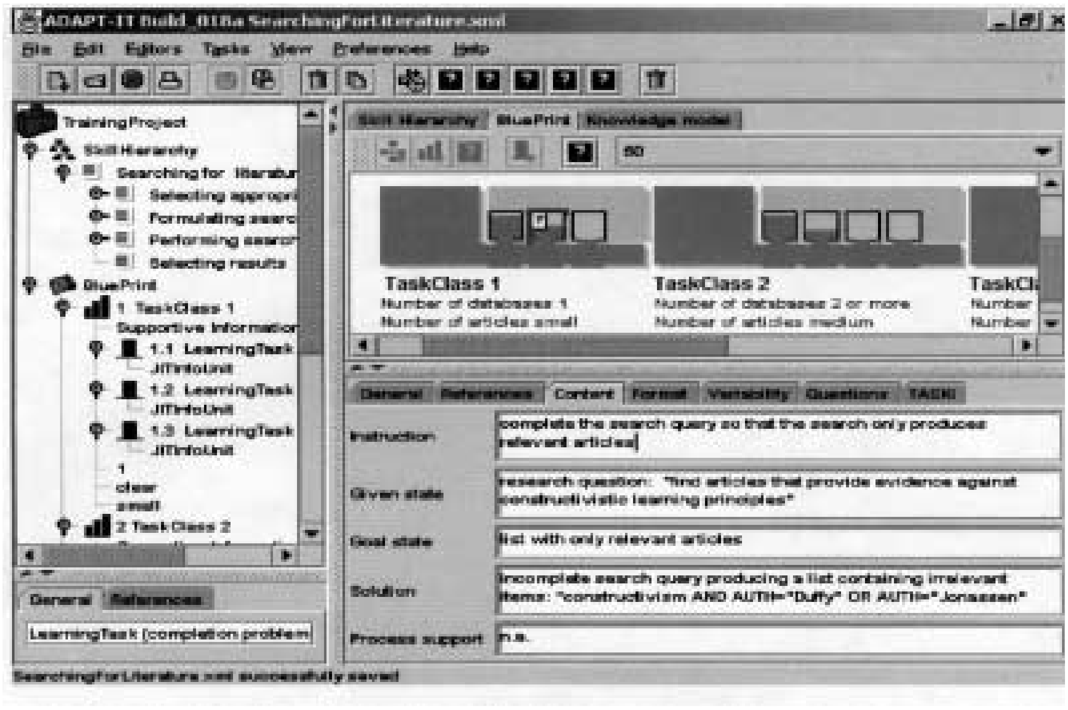
The top right part of Figure 4 shows an example of a blueprint diagram after a series of learning tasks has been created. Core graphically indicates the amount of learner support and guidance for a particular learning task by completely or partly filling up the learning-task element. For instance, a case study is assumed to provide highest support and is thus completely filled up. The amount of learner support for a particular type of learning task that is preset in the system is only a rough indication; the true amount depends on the way templates are filled out. Therefore, the designer can adjust the indicated level of learner support for each learning task. Obviously, there is also the possibility of creating additional templates for new types of learning tasks and to add those to the system.

Using Core for Remaining Activities

For the remaining design activities, that is, the design of supportive information (6), the design of just-in-time information (9), and the design of part-task practice (10), Core offers similar functions as described in the previous section. The user can add additional building blocks to the blueprint diagram by drag and drop. By selecting these elements in the blueprint diagram, associated detailed editors become available that provide templates in which the details for the different building blocks can be entered. Wherever possible, functions for automated design and for consistency checking are made available.

For the remaining analysis activities, that is, the analysis of mental models (4), cognitive strategies (5), rules and procedures (7), and prerequisite knowledge (8), a KMD-editor (see the sector on Core’s main architecture) is available. The purpose of the analysis activities is to describe the knowledge that is helpful or required for the performance of the different constituent skills in the skills hierarchy. The KMD-editor allows for the creation of KMDs, and for listing and graphically representing knowledge analysis products. The editor provides templates for representing different types of knowledge models. For mental models and prerequisite knowledge (products of, in order, activities 5 and 8), concept maps and semantic networks are used. For strategic knowledge (product of activity 4), flow-charts or SAP-charts that indicate a systematic approach in terms of problem-solving phases, and guidelines that may be helpful to complete each of the phases, are used. Finally, for rules and procedures (product of activity 7) a formalism is used in which goals are specified, together with methods to reach those goals, operators that make up the methods, and selection rules to select the best method under given circumstances. To conclude, Core stores all nonordinary verbs, adjectives, and nouns that are entered by

Figure 4 ☐ Prototype of the Core tool showing the blueprint diagram, with sequences of learning tasks for each task class with diminishing support.



Instruction: complete the search query so that the search only produces relevant articles

Given state: research question: "find articles that provide evidence against constructivistic learning principles"

Goal state: list with only relevant articles

Solution: incomplete search query producing a list containing irrelevant items: "constructivism AND AUTH="Duffy" OR AUTH="Jonassen"

Process support: n.a.

SearchingForLiterature.xml successfully saved

the designer in a concept repository that is made available in the KMD-editor. This allows a designer to manage easily all concepts in a learning domain and to build a glossary of relevant terms.

EVAL

Eval is an ID tool that supports the evaluation of an ADAPT^{IT} training blueprint on the basis of dedicated test data that are gathered during the actual use of developed training materials. Eval supports the first three levels of Kirkpatrick's (1994) well-known evaluation model: (a) trainees' reaction to the training program, (b) the measure to which training objectives are reached, and (c) the effects on on-the-job performance. The fourth level, business results, is not supported. Eval helps designers gather relevant data and analyze them in order to assess and, if necessary, improve the effectiveness of the training blueprint.

Data Gathering

For the first level of evaluation, reaction, Eval produces online, web-based questionnaires for gathering subjective data about the training program. Both trainers and trainees fill out the questionnaires. They are tailored to the training blueprint under evaluation and query for information about highly *specific* parts of the blueprint. Because the questionnaires are presented online, the data can be directly stored in a database. For the second level, training objectives, Eval provides online forms that are filled out by the responsible trainers or instructors. The forms gather objective performance data (e.g., accuracy, speed, etc.) and also offer the opportunity to enter, for each trainee, cognitive load data for each learning task and each practice item. Again, the forms are based on the training blueprint under evaluation, so that for each learning task and practice item only the relevant information is gathered. For the third level, on-the-job performance, Eval generates questionnaires and observation schemes based on the analysis of the complex cognitive skill, and focusing on the completeness and correctness of the skills hierarchy. These data are used to assess the suitability of the performance objectives (including standards for acceptable performance) and the relevancy of the knowledge analyzed for each constituent skill.

Data Analyses

In addition to gathering evaluation data, Eval also can be used to analyze data and provide information to improve the training blueprint. The reaction data are analyzed in order to detect parts of the training blueprint that trainees had difficulties with, found unclear, or pointed out as not motivating. Eval provides overviews of all the learning tasks, task classes, practice items, and other blueprint elements that show relatively high or low scores on the aforementioned measures. The designer can use this information either to reauthor some of the training materials or to change fundamentally the training program by redesigning the training blueprint and, only then, reauthor the materials.

The data regarding the training objectives provide more objective information about the effectiveness of the training program. Eval can quantitatively analyze these data and show learning curves for performance and cognitive load. As explained before, task classes represent more or less complex versions of the whole complex skill and are designed to optimize cognitive load. The learning curves are therefore a direct indication of whether or not the training blueprint succeeded in optimizing cognitive load so that trainees could efficiently acquire the whole complex skill. The designer can use this information to alter the training design in the following ways: (a) add learning tasks to a task class, because the trainees did not master the task class version of the whole complex skill after completing all the learning tasks within the task class; (b) split up particular task classes because the increase in complexity was too large; (c) combine particular task classes, because some of them were too easy, or (d) increase, diminish, or alter the timing of part-task practice.

Whereas the data about training objectives provide information about the effectiveness and efficiency of the training blueprint for reaching the training goals, the on-the-job performance data indicate if the training design was really effective in reaching transfer of learning from the training context to the on-the-job situation. The analysis and use of this on-the-job performance data is of a qualitative nature and, as such, not further supported by Eval.

DISCUSSION

This article described tools to support instructional design and evaluation for competency-based training programs. The tools take the 4C/ID* methodology as a starting point. First, the 4 blueprint components of this methodology, learning tasks, supportive information, just-in-time information, and part-task practice were briefly described. Second, the 10 activities that make up the methodology were discussed. Core provides support for each of those 10 activities. Eval feeds dedicated evaluation data back to Core in order to improve and optimize a training blueprint. Together, Core and Eval support the creation of an effective training blueprint by focusing primarily on easing the information management and decision making aspects of applying the 4C/ID* methodology.

ID tools that show some resemblance with Core have been described in recent review articles by Nieveen and Gustafson (1999) and Wang (2001). They include Advanced Instructional Design Advisor (AIDA, later Guided AIDA or GAIDA), Designer’s Edge, Langevin Instructional DesignWare, Electronic Trainer, GUIDE and GOLDIE. There are several important differences with our system. First, other systems tend to be very broad, in the sense that they mainly provide library and information support with regard to the general ADDIE model (analysis, design, development or production, implementation and evaluation), or they focus on conceptual or procedural domains. None of the systems is directed toward the design of whole-task practice and, in our opinion, suitable for the design of training for complex skills. Second, they only allow for a superficial analysis of tasks and knowledge, while Core provides a full-fledged tool (the KMD-editor) for cognitive-task analysis, yielding results that are directly propagated to the related design activities. Finally, we are not aware of any other systems that can use dedicated evaluation data to improve and tune their training blueprint.

The ID tools are currently developed within the ADAPT^{IT} project. Core is available as an early prototype and is currently tested with users (in this project, all users are in the aviation industry and air traffic control). The development of the Eval tool is on its way, but a complete prototype will not become available before Core is completely tested and finished. The tools are being developed following a participatory design approach (Blomberg & Henderson, 1990). In this approach social and organizational requirements for the tools are taken into account by collaborating with the intended users throughout the design and development process. During the development process, prototypes of the tools are continuously tested for functionality and usability. The following usability aspects are taken into account: (a) learnability, indicating that the user interface of the tool can be learned in a short time; (b) memorability, indicating that the user interface does not need relearning if it has not been used for some time; (c) efficiency, meaning that users can quickly accomplish their goals; (d) error minimization, indicating that the user interface prevents and minimizes errors and mistakes, and (e) satisfaction, indicating that users feel comfortable while using the system.

As a next step in the project, a third ID tool, called Task-i, will be developed. This tool will be used during the training process and support the personalized delivery of a competency-based training program. It gathers information on learner performance and learner progress (speed, accuracy, cognitive load, etc.) and uses this information to select the best subsequent learning task for this particular learner. All the data of each trainee that are entered into Task-i are also stored in a log file that can be imported into Eval for further analysis. In a final phase of the project, the effectiveness, usability, and efficiency of the ID tools will be further evaluated by training instructional designers, who are inexperienced with the tools, for use in the design of training programs in the domains of air traffic control and aircraft maintenance. □

Marcel B.M. de Croock [marcel.decroock@ou.nl], Fred Paas, and Jeroen J.G. van Merriënboer are with the Open University of the Netherlands. Henrik Schlanbusch is with the University of Bergen, Norway.

The work presented in this article was carried out within the ADAPT^{IT} project. This project is sponsored by the European Commission under contract IST-1999-11740.

Correspondence concerning this article should be addressed to Marcel B.M. de Croock, Open University of the Netherlands, Educational Technology Expertise Center (OTEC), P.O. Box 2960, NL-6401 DL Heerlen, The Netherlands.

REFERENCES

Blomberg, A.L., & Henderson, A. (1990). Reflections on participatory design: Lessons from the trillium experience. In *Conference proceedings on empowering people: Human factors in computer systems. Special issue of the SIGCHI bulletin* (pp. 353–360). New York: ACM.

Goel, V., & Pirolli, P. (1991). The structure of design problem spaces. *Cognitive Science, 16,* 395–429.

Kirkpatrick, D.L. (1994). *Evaluating training programs: The four levels* (2nd ed.). San Fransisco, CA: Berrett-Koehler.

Newell, A., & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nieveen, N., & Gustafson, K. (1999). Characteristics of computer-based tools for education and training development: An introduction. In J. van den Akker, R. Branch, K. Gustafson, N. Nieveen, & T. Plomp (Eds.), *Design approaches and tools in education and training* (pp. 155–174). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Perez, R.S., Johnson, J.F., & Emery, C.D. (1995). Instructional design expertise: A cognitive model of design. *Instructional Science, 23,* 321–350.

Rowland, G. (1992). What do instructional designers actually do? An initial investigation of expert practice. *Performance Improvement Quarterly, 5*(2), 65–86.

Sweller, J., van Merriënboer, J.J.G., & Paas, F.G.W.C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review, 10,* 251–296.

van Merriënboer, J.J.G. (1997). *Training complex cognitive skills: A four-component instructional design model for technical training.* Englewood Cliffs, NJ: Educational Technology Publications.

van Merriënboer, J.J.G., Clark, R.E., & de Croock, M.B.M. (2002). Blueprints for complex learning: The 4C/ID* model. *Educational Technology Research and Development, 50*(2).

van Merriënboer, J.J.G., Kirschner, P.A., & Kester, L. (submitted). *Taking the load off a learner's mind: Instructional design for complex learning.*

Wang, W. (2001). *Evaluation reports of ISD related EPSS products.* Paper presented at the annual conference of the Association for Educational Communications and Technology (AECT), 7–9 November, Atlanta, GA.