Approaches to Learning Object Oriented Instructional Design

Martin Memmel[1], Eric Ras[2], Klaus P. Jantke[3], Michael Yacci[4]

[1] German Research Center for Artificial Intelligence DFKI GmbH
Erwin-Schrödinger-Straße 57, 67663 Kaiserslautern, Germany
Martin.Memmel@dfki.uni-kl.de

[2] Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
Eric.Ras@iese.fraunhofer.de

[3] FIT Leipzig at HTWK Leipzig
P.O.Box 30 11 66, 04251 Leipzig, Germany
jantke@fit-leipzig.de
and
Hokkaido University Sapporo, Meme Media Laboratory
Kita 13, Nishi 8, Kita-ku, Sapporo, 060-8628 Japan
jantke@meme.hokudai.ac.jp

[4] Rochester Institute of Technology
Rochester, NY 14623, USA
may@it.rit.edu

Learning Objectives

When reading this chapter, the reader will learn:

- the essentials of instructional design,
- the basics of learning objects,
- the challenges of learning object oriented instructional design,
- the multidimensional learning object architecture enabling instructional design, and
- implications for instructional design from the Software Engineering perspective.

Executive Summary

The use of learning objects is an emerging concept proven to be valuable in the areas of knowledge management and e-learning. Learning objects are a key technology building a bridge between these two converging fields (Ras et al. 2005). In spite of their importance, the design of learning objects, conceptually and technically, is often debated with little effort to fully describe the learning or educational process. A lot of effort has been put in the definition of standards by numerous institutions. However, ″key issues related to global content classification such as a common schema, ontology, granularity, taxonomy, and semantics of learning objects which are critical to the design and implementation of learning objects remain unsolved″ (Mohan, Daniel 2004).

Instructional design deals with setting up spaces in which human learners can be directly taught and can guide their own learning experiences. It prepares learning paths such that learners with different prerequisites, with different needs and desires, in varying moods and under widely unforeseeable circumstances can find their way. Learning objects are building blocks for learning spaces; they are constituents of individual learning paths. The field of instructional design and development provides insights into the design of learning objects by enabling learning object designers to use existing terminology, models, and methods from instructional design.

Learning object oriented instructional design is a challenge that has to deal with several issues such as *syntax and semantics*, *granularity,* and *reusability*. The concept of *dynamic annotation* is a step towards a resolution of the *retrieval and reuse dilemma. Dynamic annotation* enables the annotation of individual objects in such a way that these annotations describe sufficiently well the meaning of a learning object. The *granularity* of learning objects has a crucial impact on the ability to adapt, aggregate, and arrange content suiting the needs and preferences of the learner. When deciding on which granularity to choose, the trade-off between the possible benefits of reuse and the expense of cataloging is of crucial importance (Wiley 2000).

The *multidimensional learning object architecture* is a generic approach enabling adaptivity and supporting instructional design. This architecture is based on different dimensions of adaptivity. Smallest building blocks that are ″semantically not dividable″ and ″uniquely classifiable″ are the central element of the presented architecture. They can be combined to other types of learning objects and allow an instructor or a system to choose the variant that suits best the user's goals, needs, and preferences.

Many learning objects can be considered as components, and in some cases as software components. Hence, approaches from the field of Software Engineering have an impact on instructional design. Technologies such as design patterns, frameworks, and product lines are integrated within a component-based approach for instructional design. Design patterns are a good means for making design ideas more explicit and hence applicable, developing learning objects *for* reuse by following the component paradigm will increase their reusability, and the separation of concerns by applying frameworks and product lines will decrease the complexity of instructional design and its embodiment (i.e., reusing concrete learning objects).

Introduction

Learning objects are an emerging technology proven to be valuable in the areas of knowledge management and e-learning and they build a bridge between these two converging fields (Ras et al. 2005). While there are numerous definitions of learning objects, almost all describe the function of a learning object as ″facilitating learning.″ However, *education* and *learning* are themselves terms that carry multiple meanings and unclear definitions, despite a century of research. The design of learning objects, conceptually and technically, is often debated with little effort to fully describe the learning or educational process.

Instructional design and development, as it has evolved in the United States as a discipline since the late 1950's, uses a systems methodology that focuses on clearly defined needs, goals, and testable systems. Instructional design models often are internally consistent: they define goals and methods to achieve those goals. Therefore, the field of instructional design and development may provide insight into the design of learning objects by enabling learning object designers to ″ jump-start″ the design process by using existing terminology, models, and methods from instructional design.

In Europe, instead of ″instructional design,″ terms like ″didactic design″ are preferred. Didactic after the ancient Greek, goes back to Wolfgang Ratke (1571-1635) and, slightly later, but much more productive and influential, to Johann Amos Comenius (1592-1670), who published his Didactica Magna in 1628 (Latin version in 1638). Comenius' work was, as he said, driven by the motivation to guide teaching such that ″the teacher has less to teach, but learners learn more″ (Comenius 1638). So, from its very beginning, didactic design is seen as a pragmatic discipline (Flechsig 1996; Jank, Meyer 2002).

The Basics of Instructional Design

In its simplest form, the goal and purpose of instructional design is to change the knowledge, skills, or attitudes of students. Such a simple and direct purpose is often overshadowed by arguments over learning theory (the exact processes that we are trying to affect) or philosophy (to what extent are we responsible for individuals, groups, society, life-long learning, etc.). Reigeluth describes the difference between instructional design theories and learning theories as the difference between *prescriptive* and *descriptive* theories (Reigeluth 1983, Reigeluth 1999). Instructional design uses theories (or models) in a practical, real-world way; the goals are to be able to diagnose and prescribe instructional solutions that will achieve specific outcomes.

On the other hand, learning theories are *descriptive* theories; their purpose is to describe (scientifically or philosophically) a phenomenon. Of course, learning theory, and our understanding of the learning process should (and generally does) underlie the ″prescription″ of instructional design solutions. However, much like the field of medicine, instructional design is a field that must create solutions even if our understanding of the underlying phenomenon is incomplete. For this reason, learning theory is important, but not absolutely critical to the practice of instructional design. Snelbecker cites Dewey (1929) who points out that education (or instruction in our case) is a mode of practice – and is only indirectly guided by theories of learning. Despite the fact that instructional design methods are based on theories or learning and performance, ultimately, the field of instructional design is pragmatic: the goal is to influence learning in real situations, even if the theories of learning are not fully understood.

As stated earlier, the purpose of instructional design is to change a learner's knowledge, skill or attitude. For purposes of discussion in this chapter, *knowledge* refers to recall or paraphrasing rote material. Gagne and Briggs, for example, referred to this as ″verbal information″ (Gagne, Briggs 1979) while Merrill referred to this type of learning as ″remember-level″ (Merrill 1983). This type of learning generally entails a simple written, pictorial, or verbal demonstration that ″ material has been covered.″ *Skills*, on the other hand, often constitute the most valuable, and difficult, forms of learning. Skills are actionable internal algorithms that guide physical or mental performance. Concept classification, problem solving, or physical procedures are all forms of skill. Finally, *attitudes* refer to values and emotional physical and mental states that are either learned (as in learning a cultural value) or controlled (as in learning to control fear or anxiety). The processes by which knowledge, skills, and attitudes are learned are very different – instructional design attempts to create idealized, general methods that can used to facilitate each type of learning. This chapter limits its discussion to the instructional outcomes of *knowledge* and *skill*, although presumably these ideas may apply to attitudinal learning also.

Is instructional design limited to creating instructional solutions? There has been debate about the range of the field of instructional design. Some feel that the more general field of *performance technology* more adequately describes the field that is concerned with improving human performance, whether it is through instruction or through other methods. While the performance technology movement is still alive and well in the United States, these terms (instructional design and performance technology) are frequently confused and often inappropriately used to describe each other. Performance technology is not concerned with instruction as a sole solution; the field of performance technology was the nexus for electronic performance support systems (EPSS) which are intended to facilitate performance. Learning may or may not occur; instruction is but one possible way to facilitate performance.

Development and Design

Within the discipline of instructional design, there are two major areas that are confused conceptually: the two processes referred to as *instructional development* and *instructional design*. Generally, ″instructional development″ is thought of as the overall process of creating instruction (analogous to a software engineering lifecycle). This process entails several high-level steps and events: needs assessment, specifying performance objectives, designing lessons and materials, building the systems, project management, testing the materials and systems, and evaluating. In contrast, ″instructional design″ often refers to design and creation of specific lessons or modules; essentially, instructional design is a subset of instructional development.

Why is this important? These two processes are at very different levels of scope: one deals with the overall problem, solution, and project leadership in solving that problem in a cost-effective way (instructional development). Instruction design, on the other hand, deals with an isolated step in the overall project: creating good, workable, instructional materials. Generally speaking, learning objects are often described in the context of instructional *design*; that is, there is an assumption that these reusable pieces will be combined to form modules or instructional sequences. Relating these back to an organizational or educational need is rarely discussed; it is assumed (perhaps erroneously) that an individual worker, user, or student will somehow determine what he or she needs. There is much research to show that students are not particularly good at self-diagnosis, and that students often make very poor decisions when given the opportunity to self-select instructional materials. If there is a role for learning objects at the *development* level, it may deal with diagnostic systems that can accurately determine these needs.

A Basic Instructional Design Method

For now, then, let us simplify the discussion and describe the instructional design process in somewhat broad strokes. (This discussion does not rigorously follow any particular design model, but explains the general methods of instructional design.) First, a knowledge, skill, or attitude need is identified. Next the knowledge, skill, or attitude is analyzed (through techniques of task analysis or content analysis). Additionally, the intended audience is described and often analyzed. At this point, given the preliminary analysis stages, an instructional designer is now ready to begin creating instructional materials, and hence ready for instructional design.

There are a variety of instructional design models and approaches. The field supports eclectic approaches from basic tutorial designs through discovery-based constructivist approaches. Despite the range of methods and models, most instructional design models are focused on describing the intended instructional outcomes, and in turn prescribing an approach to achieve these outcomes.

Several instructional design models are based around content classification taxonomies, most notably the work of Gagne & Briggs (Gagne, Briggs 1979) and Merrill (Merrill 1983). Typically, an instructional designer might begin by determining if there is a content classification that might accurately coincide with the desired skill. If there is a ″standard″ content category that matches the desired outcome, then the designer can follow a loose template to create instruction. If the desired skill fits within a content taxonomy, the belief is that there are clear ″conditions of learning ″ for these types of content. (These conditions of learning are generally referred to as instructional methods, techniques, or strategies.) For example, concept classification skills (that require a student to place a new instance in a category, according to the instances attributes or similarity with other instances in the category) require definitions, examples, feedback, and practice. At this level of analysis, most of the instructional design models (despite their differing rhetoric) are very similar in the actual methods for teaching concepts.

While this is not the only way to proceed, this approach lends itself to the automatic (i.e., computer-created) design of instructional materials. If an analyst could determine that a particular learning outcome falls into a ″known″ category (procedural skill using, for example) and the analyst determined a particular method of instruction (constructivist, for example) then learning object components could be automatically selected based upon their function. If properly tagged, these objects could be sequenced with minimal human effort, allowing for the computer-created design of instruction, following a set of pedagogical rules.

Common instructional design theories often speak of the following elements in the design of instruction: generalities, examples, explanations, practice items, test items, overviews, advance organizers, and analogies among others. A brief definition of each is given in Figure 1 (Yacci 1999). These could become the basic object structures for learning objects.

Generality: a statement or diagram that applies to all instances, such as a definition of a concept, or a flowchart of a procedure

Example: a specific instance of a concept, procedure, principle

Explanation: a series of statements that justifies why something works, or why things are done are particular way

Practice Item: an opportunity for a learner to perform a task with the goal of building skill by receiving feedback on performance

Test Item: an opportunity for a learner to perform a task with the goal of showing competency on the task

Course Maps: a verbal or graphic organizer that reveals a learner's place in a given set of content

Advance Organizer: verbal or graphic information presented at a higher level of abstraction than content that is forthcoming

Analogy: a comparison of two objects that states their similarities and differences

Figure 1: Some proposed knowledge components drawn from instructional design theories

Learning objects and components, therefore are essentially the content chunks that are needed to teach any given content type. For example, we might wish to use a discovery approach if we were supporters of constructivism. Alternatively, we may wish to use a totally expository tutorial approach if we were believers in a more teacher-directed approach. (We don't wish to argue the strengths or weaknesses of these approaches at this time.) Tutorial instruction progresses from generalities (abstractions) to examples (actual instances). Discovery learning often provides examples first, assisting the students in discovering the generality for themselves. Tagging objects as examples and generalities, then, might be a reasonable starting point for learning objects. Other instructional components that have been shown to assist in student learning are advance organizers, statements of relevance, and analogies.

Are there any existing instructional design models that might work well with this type of learning object approach? Gagne's events of instruction approach and Merrill's Component Display Theory, are both well-established and researched approaches that use classification taxonomies and a conditions of learning that are associated with each content classification; most tutorial instructional design approaches will be variations on the generality followed by numerous examples. These, then, might be the most well-established starting points for this backward design of learning objects. Structurally, constructivist approaches are variations of the *example to generality* approach. (There are sometimes additional steps or stages that assist the student in deriving the generality.)

Instructional design pedagogical rules could be handled in two obvious places: (1) embedded within a learning object or (2) as a separate object (e.g., using IMS LD). We might suggest that initially, the pedagogical rules of instructional design should be handled *outside* of the learning objects. If, as we have suggested so far, the objects themselves are tagged and structured chunks of content, adequate variants could then be selected and sequenced in a different layer, providing for the most flexibility in use. For example, if our goal was to use a constructivist model, different objects would be selected and sequenced differently than in a more teacher-centered model.

Contributions of Cognitive Psychology and Neurophysiology

Spitzer has recently argued that a variety of deficiencies in the educational system's effect, at least in Germany, may be due to badly misunderstanding or completely ignoring what sciences have brought up about learning (Spitzer 2002). We are not going into detail here, but point into a few directions to make clear that the sciences of cognitive psychology and neurophysiology also have an impact on the social engineering discipline of instructional design.

The human brain is known to be, so to call, a pattern inference machine. The fact that understanding regularities and the triumph of mastering a complex situation results in a very good feeling (through the release of dopamine in the human brain), is widely used in game design (Koster 2005). The nature of student motivation is addressed at a different level through Keller's ARCS model (Keller 1983). The ARCS model deals with attention, relevance, confidence, and satisfaction and provides guidance for including these types of motivational issues in the instructional design process. Further, the cognitive processes involved in learning and mastering educational games has been addressed using eye-tracking techniques (Yacci 2004; Yacci et al. 2004).

Another phenomenon though rather well-known seems to be widely misinterpreted: attention. Attention does not mean–notice the usage of double-negation in the argument–that learners do not think of anything else. Let them think of whatever they want, but let them think! Attention is a ″gate opener″ (Spitzer 2002) for learning, and it is one of the major steps in Gagne's well known events of instruction approach (Ragan & Smith 2005).

Brain sciences are difficult enough to leave always a number of questions open. We know already for decades that there are brain developing processes like myelenisation that change the degree of brain plasticity (Flechsig 1920) such that critical phases in human learning do really exist and, when missed, may result in severe disabilities (Rymer 1994). This is particularly known for language learning (Newport 2002). Generally speaking, stages of child development are considered in a K-12 educational setting. However, corporate training situations (for example) deal with adult learners, and little research of the changing physiology of adults has been used in instructional design. The changing nature of the adult brain is a vital and important area.

Computer-created Instructional Design

As already mentioned above, learning is a very individual and complex process which is not yet fully understood. Therefore, a learner's context, special needs, interests, goals and preferences have to be considered in an instructional design process. E-learning systems might offer the user the option of choosing among several possible instructional options. However, human self-selection of materials is not always an effective process, even though it has much intuitive appeal. An improvement over self-selection might use the computer as an *assistant*. Ideally, this might be done automatically by interpreting information captured about the learner stored in an corresponding user model. The ability of a system to automatically adapt to a user is referred to as *adaptivity*, and it is a key feature in evolving standard systems to assistants (Jantke et al. 2004a).

The attempts to build such intelligent systems are partially motivated by the lack of human teachers and tutors. When there is no one able to provide a certain level of guidance to the learner, an intelligent system will try to support this task. These approaches, carried out in fields like Intelligent Tutoring System (ITS) research or AIED (Artificial Intelligence in Education), are often criticized by groups championing sociocultural approaches to learning. They claim that these efforts exclude humans from instructional design or from providing instructional feedback. A middle course, enabling humans to take part in as many processes as possible and nevertheless offering automatic assistance if wanted or needed by the learner, is required.

A different conception of instruction presents a dramatic point of view, in which learners are the key actors in the learning play. In this approach, teachers as considered actors as well. IT system components may form agents that occur as further actors, whereby there is no need at all to have any anthropomorphic appearance of those agents. Sometimes, they are simply windows on the computer screen. Instructional design may be seen as a way to set the stage for these actors' interplay. Storyboarding is a conventional approach to anticipate what shall happen on the stage (Rabenalt 2004) and it has been taken and adapted to technology enhanced learning (Hagebölling 2004). Storyboarding is already used in other social engineering disciplines like enterprise conflict solution (Forsha 1994).

The Basics of Learning Objects

Before we talk about the relations between instructional design and learning objects, we need to have a clearer idea of what we're talking about. This section explains why learning objects are relevant and discusses terminology and the role of standards.

Motivation

When examining almost any learning material, one will always find numerous relations, links and cross references. Some content is prerequisite to understanding other material; examples are useful for different purposes; some keywords are explained in different sections and so on. In traditional print-based learning material, the content is arranged in a reasonable way—sequentially, of course—to enable the reader to understand it. But if one is interested in additional material that is not presented, a book's index, a bookmark—or even other sources—have to be used to find the respective information. In contrast, e-learning provides the opportunity to use a huge multimedia repository and to adapt to the user's individual preferences. The learning material no longer has to be presented in a static way. The sequencing of topics, next steps, presentation style or difficulty can be chosen by the user or can be automatically selected by the system. To generalize: All relations and links existing within the learning material can be used to help the user and to create or offer individual environments. To realize this opportunity, the content must be structured into relatively small fragments which can then be combined into bigger objects in alternative ways. All fragments and combined objects have to be annotated with adequate metadata to provide information about relations to other objects, technical prerequisites, presentation style and so on.

But if learning material is to be divided into so-called ″learning objects,″ the following features must also be considered:

- Interoperability, i.e., two or more systems or components need to be able to exchange information and to use the information that has been exchanged
- Sharability, i.e., content from several different sources may be accessed by multiple users (simultaneously) with different e-learning systems
- Reusability, i.e., content developed in one context should be reusable in a different context

Terminology and Standards

In many publications about learning objects, terminology issues are discussed, because there is a lack of consensus in the field of e-learning (Self 1992), especially concerning learning objects. Numerous initiatives like AICC (the Aviation Industry CBT Committee), ADL (Advanced Dis-

tributed Learning), IEEE LTSC (the Learning Technology Standards Committee of the IEE) and IMS Global Learning Consortium have made efforts to establish standards. For several years, a number of initiatives agreed to corporate on the field of standards as shown in the network in Figure 2.

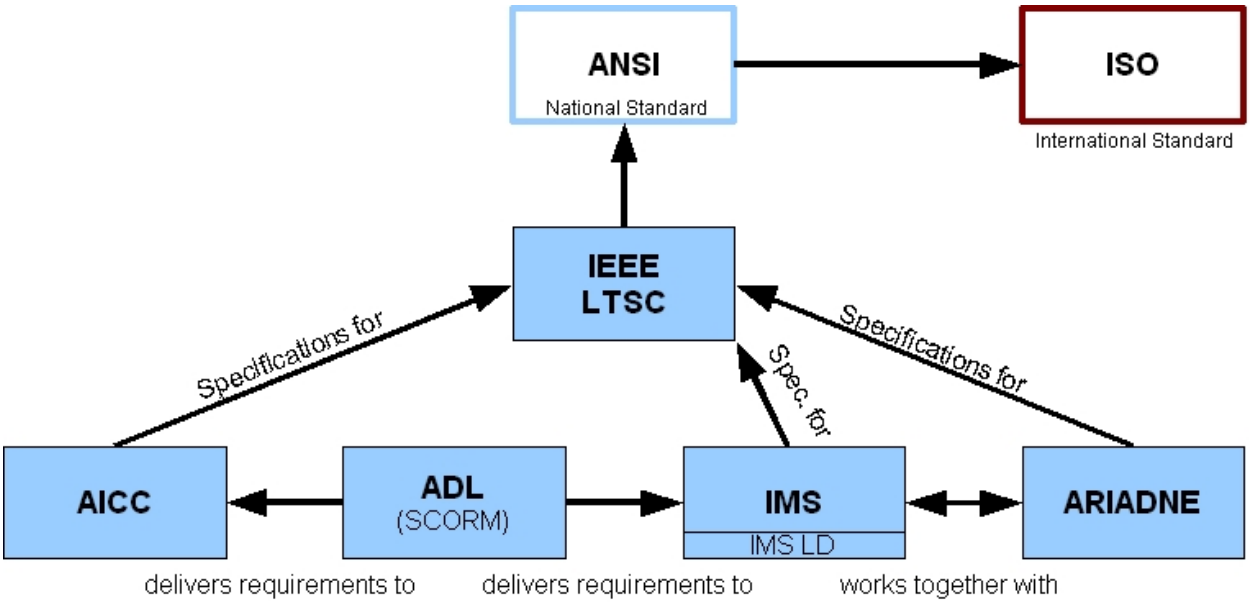

Figure 2: The cooperation network of the standardization initiatives (adapted from (IMC 2001))

Although much effort has been put toward the definition of standards, and although the LOM Standard seems to be widely accepted now, ″key issues related to global content classification such as a common schema, ontology, granularity, taxonomy and semantics of learning objects which are critical to the design and implementation of learning objects remain unsolved″ (Mohan, Daniel 2004).

There is also still no consensus on what a learning object is, or, to be more precise: What constitutes a learning object, and which metadata have to be annotated? One of the most popular and often cited definitions is offered by the Learning Objects Metadata working group (LOM) of the IEEE Learning Technology Standard Committee, where a learning object is defined as ″*an arbitrary entity (digital or non-digital) that can be used, reused or referenced in an electronically supported learning process.*″ This is certainly a very broad definition, and it has been criticized very often because of its breadth.

What, then, constitutes a *good* definition?

- According to Wikipedia, a good definition must describe the ″essential properties of a certain thing″
- The definition must be precise enough to include all concepts wanted and to exclude all unwanted concepts

Keeping this in mind, let us have a look at some very popular criticisms. Friesen, for example, argues that the breadth of the LOM definition means that there are few things that cannot be learning objects (Friesen 2004). But does this really pose a problem? A narrower definition only makes sense if it does not exclude any objects that may possibly be useful in some e-learning scenario, otherwise it is not usable in too many cases. To illustrate this problem, let us have a closer look at such a narrower definition. In (Wiley 2000), the author also criticizes the LOM definition and proposes the following, slightly changed definition: ″*any digital resource that can be reused to support learning*″. But this definition entails at least two main problems:

- Wiley replaces the term ″used″ in the LOM definition with the term ″supported″ because otherwise this would also allow an object like an advertisement banner to be a learning object. Although his criticisim is valid, using the term ″supported″ does not solve this problem. It always depends on the current context whether a certain object supports learning or not! For example, a definition of a decision tree could be considered a learning object in a machine learning course. However, that same decision tree surely doesn't support a user trying to learn about a French lyric from the 19th century. To generalize: (Almost) any object can support learning, it just depends on the context! Or the other way around: Any object can be meaningless! It is hardly justifiable, if at all, to explicitly exclude certain objects in a definition of learning objects.
- The definition, as Wiley says, ″explicitly rejects non-digital (…) and non-reusable (…) resources″. This is also critical, because there are many scenarios in which it makes sense at least to reference ″real objects″ like a teacher or a book by using a URI or the like. This is of special interest if one is interested in defining complete learning processes with approaches like storyboarding (Jantke 2005), where a learning process can also include a lecture or a seminar.

So what could be a solution to the terminology problem? Obviously, the LOM definition provides a kind of superclass of all learning objects, because it includes almost all objects or instructional materials one can think of. Using this as a starting point, any developer or theorist may define certain types or categories of learning objects according to his special needs and interests. What has to be considered is that their meaning must be understandable, by humans (learners as well as instructors) and at least to some extent by a system. Or, as Friesen says in (Friesen 2004): ″In order for the positive potential of learning objects to be realized, they need to be labeled, described, investigated and understood in ways that make the simplicity, compatibility, and advantages claimed for them readily apparent to teachers, trainers and other practitioners.″ This also implies the notion of the context in which the learning object is developed and used! Hopefully, the standardization efforts will focus on defining commonly agreed subcategories of learning objects in the future.

Challenges of Learning Object Oriented Instructional Design

When we have an understanding of how to stimulate the learners' activities which, in the very end, are expected to have some learning effects close to what we would like to achieve, we need to set up a space for the learners' experiences. In the simplest case, their experience might just be reading a text, watching a video or viewing some pictures. In other cases we may prepare interactions of learners with an e-learning system, with other learners and with teachers, tutors and, perhaps, humans in further roles. Learning objects on the one hand, are forming the space of experience and, on the other hand, are among its inhabitants. Learners may meet them there.

Several aspects are relevant when learning objects shall be used in the given context. Especially the challenges syntax and semantics, granularity and reusability have to be considered and will be elaborated in the following.

Syntax and Semantics

Instructional design is a design activity based on a complex corpus of knowledge about the learning task and the audience, driven by clear pedagogical intentions, and based on experience and belief in the effectiveness of various instructional methods. Learning objects are the essential building blocks of instructional design, but they are also determining severe limitations of which pedagogical dreams can be implemented.

When humans get engaged in those ambitious design problems, a key difficulty is the richness of semantics on the side and the rigidity of the syntactic means available. The granularity of learning objects (see below) is crucial, because a finer granularity allows for the design of a larger variety of more fine-tuned learning spaces. Regardless of the granularity, human designers need to know about the building blocks available; they need to be supported in finding or creating learning objects they need.

Therefore, it would be great if developers and designers of learning objects were be able to annotate individual objects in such a way that these annotations described sufficiently well the meaning a learning object. This is the question for a syntactic representation of a learning object's semantics. The dichotomy of syntax and semantics is perhaps the most prominent one in informatics. Due to generally unsolvable problems of computer science (Rogers 1967), there does not exist any universal approach to representing the meaning of syntactic objects sufficiently well in syntactic annotations.

Before pondering about how to annotate learning objects in a way that describes the essentials of the learning object's semantics, one needs to clarify what the meaning of a learning object might be. Questions like that are known to be difficult. Ludwig Wittgenstein investigating the dichotomy of syntax and semantics in natural languages claimed that *the meaning of a word is its usage in a language* (Wittgenstein 1953). So, what is the meaning of a learning object?

The meaning of a learning object is its usage in a community and/or in a learning context.

Though instructional design deals with the anticipation of learning experiences (Flechsig 1996, Reigeluth 1999), one may not expect deterministic forecasting. Therefore, it is a rather difficult task for designers, developers and implementers to annotate sufficiently precise future usage of learning objects because of this ambiguity.

To sum up this discussion, one knows that there is not much hope for a priori annotation of learning objects which are expressively describing the semantics of learning objects from the perspective of the value for and usage in instructional design. The only way out is dynamic annotation. Learning objects must be watched as they are used. A systematic monitoring of ongoing e-learning practice will surely tell a lot about the value of particular learning objects by identifying their role in successful learning paths. Current meta-data concepts do not support dynamic annotation. The future of e-learning has to bring, among other things, extensions of meta-data concepts that go far beyond the current conventional approaches. Satisfying standards are not in sight.

Reuse in E-Learning

Re-using software is a long-nourished dream of computer science. Software reuse has been proposed as a solution that would provide for higher quality software with less effort on the part of developers. For decades, the software industries has strived hard to make this idea work. However, even with the enormous power of global enterprises there has been little success. Therefore, we should dampen our optimism about the reuse of learning objects.

Reuse in e-learning does not mean that someone uses learning objects that he or she has generated some time ago and it also does not mean that someone has access to a friend's learning objects. The true challenge of reuse–both in the software business, in general, and in e-learning, in particular–is to find material that the developer did not know about before. This requires support when searching for learning objects; semantically relevant annotations are seen as a key to successful learning object retrieval. The core difficulty lies in the dichotomy of syntax and semantics.

The e-learning community is facing another particular problem related to the issue of learning object granularity: *the Retrieval and Reuse Dilemma*.

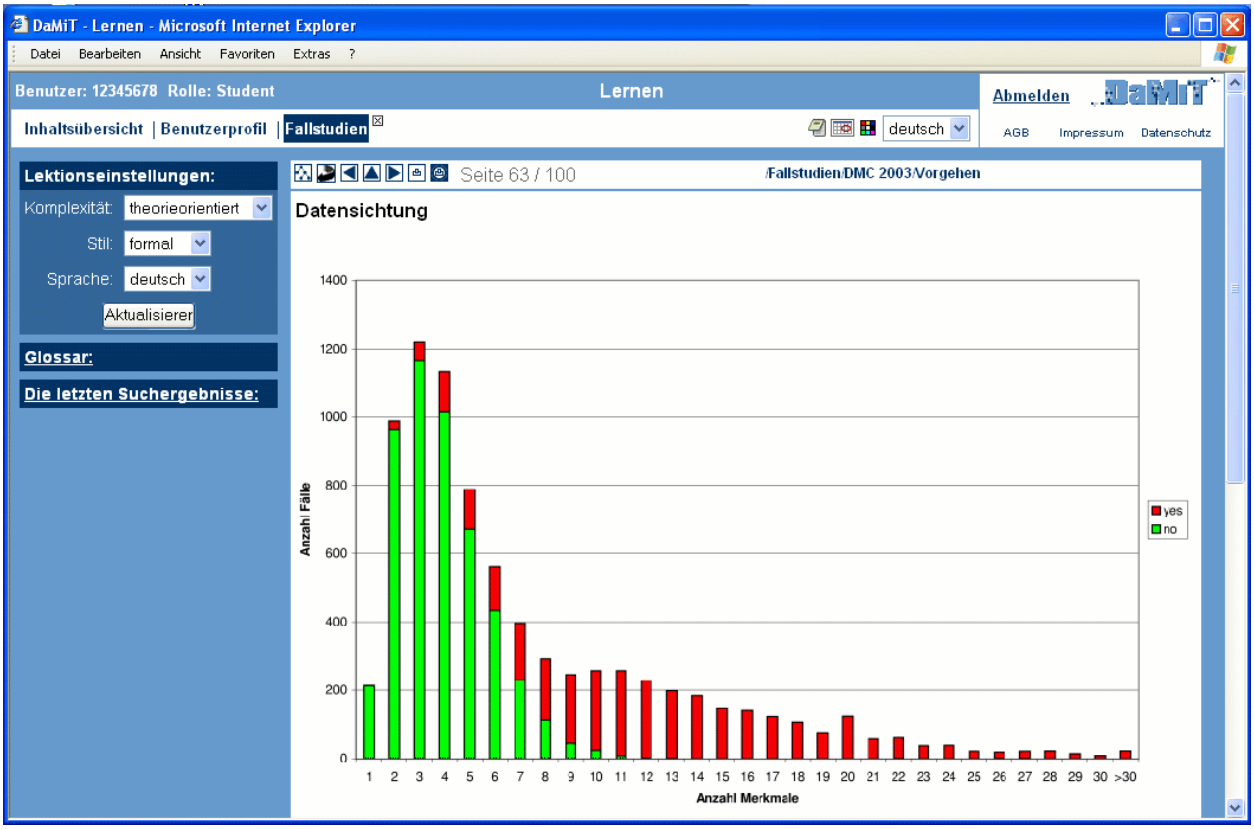

Figure 3: Simplified view at a case study within the DaMiT System; certain learning objects are hidden

For illustration, imagine you deal with data mining, in general, and with the CRISP process model of data mining, in particular. Imagine you find in another system (see Figure 3) a case study with realistic data that fits your needs of illustrating the importance of business understanding and of data preprocessing. Of course, you don't want the whole case study from the other system. Instead, you may wish to use the data set itself or just the illustration which might serve

your purpose. The crux is that the data set is deeply hidden within the other system. It is usually accessed via a download link, within an exercise, for instance. The picture appears as component of some page which is itself a composite learning object. The picture file is not carrying any didactic metadata.

The *Retrieval and Reuse Dilemma* describes this phenomenon: what designers, developers and implementers would usually like to reuse from another system is of such a fine granularity that it is not appropriately annotated. And what they find, in contrast, is too large and contains too much irrelevant detail. The learning objects most appropriate for reuse are generally situated in an e-learning context that is not wholely suitable.

One of the reasons for this lack of suitability is coherence of the existing presentation. Designers and authors have their own writing styles. In principle, fine granularity learning objects like the picture shown as a component of the page displayed in Figure 3 can hardly be annotated a priori with ″the right″ pedagogical terms. The picture may be used in different contexts of instructional design.

One step towards a resolution of the retrieval and reuse dilemma is dynamic annotation (see the discussion of syntax and semantics). Systems that support the reuse of learning objects need to monitor the usage of learning objects and *learn* useful annotations over time–the induction of meaning. This requires corresponding meta-data concepts.

The hierarchical structure of learning objects and, thus, understanding their degrees of granularity, is essential. This is enabled by storyboarding concepts as in (Jantke, Knauf 2005), where the assets that are most probably subject to reuse appear as leaves of a structure which may be seen in two different ways: as a graph, as long as learning paths in spaces of learning experience are focused, and as trees, as soon as composition and decomposition of scenes in a storyboard are crucial.

For the purpose of retrieval, inheritance of meta-data–quite similar to inheritance in class hierarchies–should be introduced. Search for appropriate learning assets takes place on a higher level until desired meta-data are found. Vertical propagation leads to potentially useful assets. Finding useful levels for retrieval and reuse still requires research and experimentation.

Granularity

The granularity of learning objects has a crucial impact on the ability to adapt, aggregate, and arrange content that suits the needs and preferences of the learner. Insufficient granularity of learning objects (i.e., dealing with large and monolithic blocks of content) prevents a flexible integration in new learning contexts and into a learning environment with exchangeable, transferable and reusable components. In contrast to that, dividing the content into more or less small learning objects offers a variety of possibilities:

- The learning objects can be aggregated to different types of new learning objects with varying complexity (e.g., course-grained), and the learning objects can be arranged with methods like storyboarding (e.g., the IMS Learning Design Specification treats this issue).
- Multiple and customizable views of material can be generated. Thus, the same content can be repurposed and used in highly diverse learning contexts, adapted to the learner's needs and preferences.

- It is rather simple to assign the learning objects to a certain classification type (e.g., a definition or a theorem) such that similar types can always be presented in the same way, and filtering functionalities can be offered (e.g., ″just show me all definitions and theorems″).
- The reusability of learning objects can facilitate a consistent notation (this is of special interest in the domain of science).

Of course, the granularity of learning objects also has an impact on the simplicity and straightforwardness of the authoring process. The more strict and detailed the definitions are, the more metadata has to be annotated and the harder and the more time-consuming the design of learning material gets.

Thus, much thought has to be put into the definition and choice of the granularity of learning objects. The decision of which granularity to choose strongly depends on the context in which the system using the learning objects shall be used. It has to be distinguished whether most of the elements will be produced in advance and whether the according repository will only be changed occasionally (this is typical for publicly funded projects in an academic scenario) or whether we have more dynamic systems in which content has to be produced faster, in some cases by the users themselves. To sum up: There is a trade-off between the possible benefits of reuse and the expense of cataloging (Wiley 2000).

Multidimensional Learning Objects

This section describes a generic approach for learning objects that can enable adaptivity and support instructional design. This generic approach is based on atomic and multidimensional learning objects called ″units″ and was first used in DaMiT, a research and development project sponsored by the German Federal Ministry for Education and Research (BMBF). DaMiT abbreviates ″Data Mining Tutor″ and is an Internet-based intelligent tutoring system for the domain of knowledge discovery and data mining. The system is available under http://damit.dfki.de under the link ″Lernsystem″. For more information about the system and implementation details see (Jantke et al. 2004b) or (Rostanin 2004).

The main idea of this approach is to distinguish between two different dimensions of content presentation referred to as *″what″-adaptation* and *“how”-adaptation* (Memmel 2005):

- **″what″-adaptation:** Presenting different learning objects or a different sequence of learning objects.
- **″how″-adaptation:** Only presenting different, possibly automatically generated variants of learning objects.

At the heart of the approach are building blocks called ″units″ which can be assigned to a certain classification type as an example or an advance organizer. Units are multidimensional (concerning ″how″-adaptation) learning objects that are composed from one or more so-called ″assets″. An asset is a media element of a certain technical format, such as a text file, a jpg image or a wav file.

The units themselves can be combined into other, larger types of learning objects called ″containers″. A container is a combination of one or more units and other containers and can also be defined in various dimensions (concerning ″what″-adaptation).

A derivative of the LOM Metadata Standard is used to describe the learning objects – some parts were left out, and some important extensions turned out to be necessary.

In the following sections, the different types of learning objects used in the multi-dimensional approach will be presented in detail, and examples of their use in the DaMiT system as well as their potential use for general instructional design are given.

Assets

An asset consists of a single media element of a certain technical format like a text, a jpg image or an HTML-file. The metadata describing a sample asset is shown in Figure 4. This asset refers to an HTML-file containing the definition of a decision tree. The metadata contain no semantic information (e.g., about the use of the object in an instructional design scenario) to ensure a high degree of neutrality and potential reusability. Only information about the version number, the date of creation, the author and technical information like the file size, format, and technical prerequisites is annotated.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<content>
<general name="dt_definition_informal-0"/>
<lifecycle version="1.0"/>
<contribute author="KL_MM" date="2002-12-511:20:6.000000"/>
</lifecycle>
<technical format="HTML" size="8" location="src/dt_definition_informal-0.html"/>
</content>
```

Figure 4: A sample asset

Units

These multidimensional learning objects are the heart of the presented approach. As stated previously, a unit is assembled from one ore more assets. Normally only one asset is used, but in certain cases, more assets may be necessary. For example, if a designer needs to combine an image asset and an audio file asset, these two assets will combine into a unit.

Units are the smallest building blocks annotated with semantic metadata and they can be defined in multiple dimensions, if these dimensions only have an impact on the way the unit is presented (″how″-adaptation). Units are ″atomic″ learning objects, meaning that they are ″semantically not dividable″ and ″uniquely classifiable″, i.e., a learning object may only be defined as a unit if it satisfies the following conditions:

- It makes no sense to reuse a part of the learning object**.**
- It can be assigned to a certain classification type as a definition, a proof, or a theorem.

This strict definition of course has an impact on granularity – units are relatively small learning objects. As already discussed above, this makes the authoring process harder. But the strictness of the definition and the division into relatively small building blocks facilitates a common understanding of the atomic learning objects – this is essential for the creation of different dimensions of the same unit by potentially different authors.

In Figure 5, the metadata used for a unit in DaMiT is shown. The available classification types can certainly be adapted to the needs of the respective scenario. Grey fields indicate that these fields correspond to a dimension of a unit. In DaMiT, only the dimensions ″language″ and ″

presentation style″ were used, but there are no restrictions to introduce various other dimensions, as long as they only have an impact on how the respective unit is presented.

NAME	DESCRIPTION
general	
name	id of the element
title	title to be displayed (optional) for certain types of units
language	language of the element
description	informal description of the unit (for other authors)
keyword	associated glossary entries
lifecycle	
version	version number
status	status (draft, final, revised, unavailable)
contribute	
author	id of the corresponding author
date	creation time of the element
technical requirement*	
type	type of requirement
name	name of what is required
minimumversion maxim-	minimal version
umversion	maximum version
educational	
presentation	presentation style (formal, informal)
rights	
cost	could payment be necessary? (0/1)
relation*	
kind	type of relation (requires,is-required-by,has-part,is-part-of)
newpage	only relevant for “has-part”: defines pagebreaks
resource	id of the referenced element
version	version number of the referenced element
classification	
type	classification type (algorithm, axiom, corollar etc.)

Figure 5: Unit metadata

The metadata describing a sample unit is shown in Figure 6. Physically, a unit is simply an xml file containing metadata including the references to the corresponding assets (by using the relation-tag), in this case the asset already shown in Figure 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<unit>
<general name="dt_definition" title="Entscheidungsbaum" language="de"
description="Definition eines Entscheidungsbaums" keyword="gl_classification, gl_tree, gl_decision_tree"/>
<lifecycle version="1.0" status="final">
<contribute author="KL_MM" date="2002-12-5-11:20:6.000000"/>
</lifecycle>
<educational presentation="informal"/>
<rights cost="0"/>
<relation kind="has_part" newpage="false"><resource name="dt_definition_informal-0"/></relation>
<classification type="definition"/>
</unit>
```

Figure 6: A sample unit

Figure 7 gives an impression of how the unit shown in Figure 6 is presented in the DaMiT system. This screenshot illustrates one of the benefits of assigning each unit to a certain classification type: a consistent layout is enabled. In this case this is done by automatically creating the header ″Definition″ (with the title ″Entscheidungsbaum″ added), drawing a border around the element and displaying an exclamation mark icon. This layout information is stored separately from the learning objects and can easily be changed.

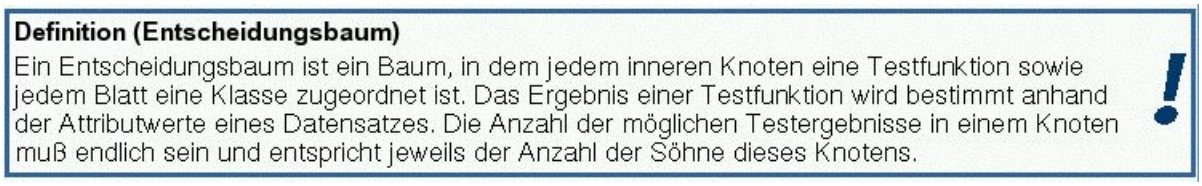

Figure 7: The unit shown in Figure 6 as it is presented in the DaMiT system

Using classification types also offers many retrieval and filtering functionalities to learners as well as instructors. One can show or hide certain types of units, for example and a systematic and fast search for certain types of learning objects in the repository is possible, whether to learn or to build new objects or courses. Furthermore, modes like ″exam preparation″ automatically hiding illustrations or examples and thus delivering only a condensed version of a course can be offered by the system easily.

A unit can be defined in various dimensions, e.g., concerning ″language″ or ″presentation style″. This enables an instructor or a system to just pick the variant suiting best the user's goals, needs, and preferences. Thus, the same content can be repurposed and used in highly diverse learning contexts, supporting sharability and reusability. Figure 8 shows two screenshots from the DaMiT system containing a unit with the definition of a decision tree in an informal (as already shown in Figure 7) and a formal variant.

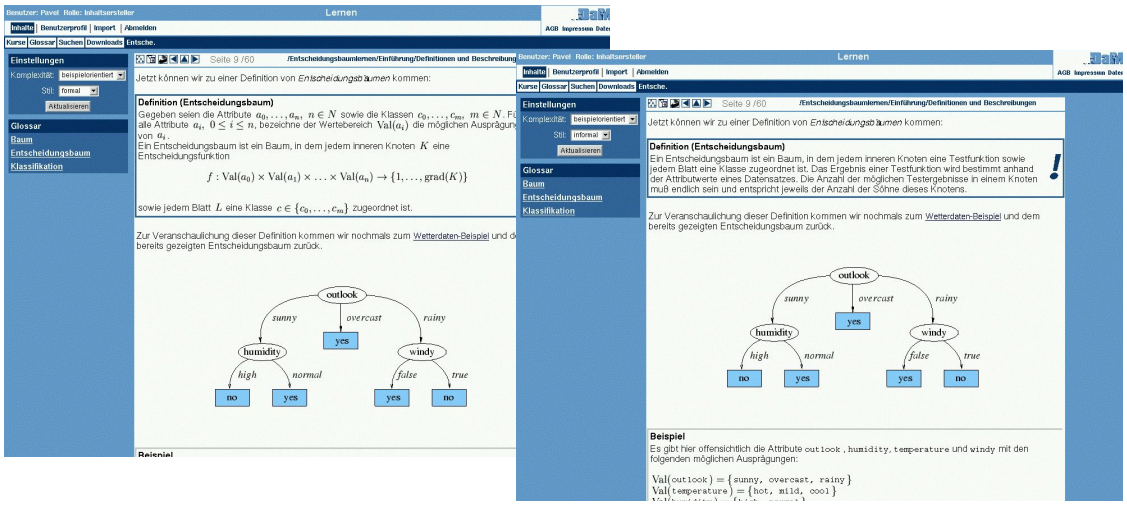

Figure 8: Two screenshots of pages with different presentation styles in DaMiT. Each page consists of four different units: A motivation, a definition, an illustration and an example

Containers

As the name already implies, this type of learning object is aggregated from one ore more units or containers. As Figure 9 illustrates, the required metadata are similar to a unit, except for two main differences: First, there is a slot to define various types of containers (in case of the DaMiT system, the slot ″damitspec″ allows lectures and courses). The definition of these types depends on the current system and user requirements and the intended use of the containers. Second, containers can also be defined in different dimensions, but these dimensions refer to the ″what″-adaptations, i.e., they have an impact on which elements are shown in which order. In DaMiT, containers could be defined in different difficulty levels (basic and advanced). For example, the variety in difficulty levels can be realized by including more examples and illustrations and less

complex proofs or the other way around. In contrast to a unit, a container has to clarify what a different dimension of a container means. This can be done in various ways; for example, one can demand that both containers have to meet certain conditions regarding their corresponding learning goals.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topic>
<general name="dt_description" title="Definitionen und Beschreibungen" language="de" description="Eine formale
Problembeschreibung zum Thema Entscheidungsbäume"/>
<lifecycle version="1.0" status="final">
<contribute author="KL_MM" date="2002-12-5-11:20:6.000000"/>
</lifecycle>
<educational difficulty="basic"/>
<rights cost="0"/>
…
<relation kind="has_part" newpage="true"><resource name="dt_description-motivation-1"/></relation>
<relation kind="has_part" newpage="false"><resource name="dt_definition"/></relation>
<relation kind="has_part" newpage="false"><resource name="weather_data_dec_tree"/></relation>
<relation kind="has_part" newpage="false"><resource name="dt_definition_example"/></relation>
…
<damitspec isLecture="false" isCourse="false"/>
</topic>
```

Figure 9: A sample container

In this approach, the content of a page does not necessarily correspond to a single learning object. It can be composed from several units, e.g., a passage, a definition, an illustration, and an example as shown in Figure 8. The pagebreaks can be defined on the level of a container (by using the ″newpage″-attribute) or automatically be calculated by the system according to the estimated length of the corresponding units (the exact length of a unit sometimes depends on various client settings and thus cannot always be exactly calculated).

Relations Between Units and Containers

Numerous relations among units and containers are possible. Some content is prerequisite to understanding other material; examples are useful for different purposes; keywords may be explained somewhere else and so on. The relation-tag offers the possibility of annotating arbitrary types of relations that can be exploited by a system in a preferred way. In DaMiT, two types of relations are allowed:

- ″has-part″ is used to declare subelements (e.g., it contains the ids of the assets a unit is assembled from). This information can be used to refer to other learning material in which a currently displayed unit occurs.
- ″requires″ is used to declare prerequisites on the level of units. Containers automatically inherit the prerequisites of the units they are composed of. This facilitates automatic course generation by specifying the exact content that is required to understand various units selected by the user or the system.

Software Engineering for Instructional Design

Especially in industrial training settings, learning objectives correspond to concrete, well-defined job-related skills, specific tasks to be done, or problems to be solved. Hence, the delivered learning material and learning approach must suit the current situation in which the learner finds himself. The situation may change over time while the learner is performing his work. Nevertheless, conventional learning systems leave no space for dynamic selection and sequencing of learning

material (Brusilovsky, Vassileva 2003) and hence, offer little support for adapting the instructional design. Therefore, new types of learning services and mechanisms need to be developed that can account for this change of work situation.

Learning tasks and activities are an important characteristic of good instructional design. Engaging learners and actively involving them in the learning process often increases motivation and learning gain. However, the learning objects are usually not designed for interactivity. To be successfully reused for learning, these learning objects and materials need to be embedded in interactive learning activities (Yacci 2005). In addition, the quality of personalization and reuse of learning could be systemized by means of instructional interaction.

The requirement for more interactivity, more personalization through adaptation of delivered learning content, and more reuse of existing learning objects will lead to a higher relevance of sound Software Engineering (SE) principles, methods, and techniques: Learning objects should be considered as components instead of objects and, in some cases, even as independent deployable *software* components due to their increasing complexity in terms of interactivity with the learner and the system, different media used, increased set of metadata, and the demand for adaptability and higher reusability.

This section explains why Software Engineering (SE) could contribute especially to the understanding of learning objects as components and how SE approaches could help to develop learning material from learning objects following the component concept. In order to do so, we first elaborate on the skepticism regarding SE for e-Learning by clarifying the misunderstandings between learning objects and object-oriented programming objects, and we introduce components as a suitable concept for learning objects. Second, we introduce reuse-based software development by stating validated hypotheses and laws and by presenting the main reuse concepts, such as components, design patterns, and frameworks. Third, we list some implications on instruction design from an SE perspective and present a component-based approach for the instructional design of learning material from learning objects.

Learning Objects and Object-oriented Programming Objects

As explained in the previous sections, the main reason why learning objects have been invented is *reuse* and the desire for more flexible, adaptive learning systems. Current development efforts with learning objects are mainly concerned with metadata and content packaging aspects. Current object metadata says little about how to combine learning objects with others, and this will limit the success of the numerous repositories of learning objects that are being developed. In addition, putting together learning objects still happens without any deep knowledge about instructional design concepts. Another general problem is that the reuse process and its underlying principles are still not fully understood.

Before we elaborate more on reuse, we have to clarify why instructional designers and educators are skeptical when learning objects are put into relation with object-oriented programming objects and why their skepticism is justified.

Some authors toss around theoretical connections to object-oriented theory that stem from computer science. One reason why many of us attempt to connect learning objects to code objects is that there is a grammatical affinity between ‘object’ as used in ‘learning object’ and object-oriented programming theory. Another reason is that object-orientation is still understood as the silver bullet to reuse in general, because of the embedded concepts such as inheritance, encapsulation, and polymorphism. Robson defines learning resources as objects in an object-oriented model. He compared learning objects with code objects by saying that they both possess methods and

properties; methods include rendering and assessment methods, and properties contain content and relationships to other resources (Robson 1999). In fact, it is not wrong to refer to these concepts from object-oriented theory in order to increase our understanding of learning objects and our belief in successful reuse. Friesen states that there is not only a conceptual confusion in the literature between software objects and learning objects, it also seems that object-orientated programming objects and learning objects do not fit together at all (Friesen 2001).

As stated above, learning object complexity and hence their development will become more challenging based on several emerging issues such as the aim for more instructional interaction and repurposing. This will involve other concepts and development methods than those currently used. Mohan & Brooks (2003) claim that 'Object-oriented technology could be one technology to take learning objects out of their current static form and imbue them with behaviors that allow them to contribute more meaningfully to an instructional situation'. However, the next section will explain why object-orientation failed concerning reuse and why it should be considered mainly as enabling technology instead of the 'silver bullet' approach for reuse.

Reuse in Software Engineering

Software Engineering is concerned with the design and implementation of large scale, complex information processing systems that are robust, maintainable, modularly reusable, scalable, and extensible (Pfleeger 2001). Pfleeger defines object-orientation as an approach to software development that organizes both the problem and the solution as a connection of discrete 'objects'.

The success of a software company depends on the ability to react rapidly to changing market conditions and user requirements. There are various ways of addressing the demand for change, but one of the most effective methods is software reuse. The roots of software reuse come from an ITT workshop led by Ted Biggerstaff and Alan Perlis (Biggerstaff, Perlis 1991). Numerous approaches have been developed and tried over the years. Hence, much experience has been gathered by using these approaches. Endres and Rombach state that

'Software reuse reduces cycle time and increases productivity and quality' (Endres, Rombach 2003).

In the early 1980s, object-orientation was predicted to be the 'silver bullet' for reuse. However, about ten years later, Udell stated that:

'Object orientation has failed but component software is succeeding' (Udell 1994).

As Szyperski describes in his book on component software, objects are almost never sold, bought, or deployed. Instead, software reuse can take on the form of generators, frameworks, and components. *Component-based development* has become the most frequent reuse approach. The unit of deployment is something more static, such as a class or even a set of classes. 'Objects that logically form parts of component 'instances' are instantiated as needed, based on the classes that have been deployed with a component' (Szyperski 1998). Does this mean that software objects are not reusable assets? Yes, because objects, from a Software Engineering point of view, are purely technical – in other words encapsulation, polymorphism, and inheritance. When we reflect on the statements made above, reusing learning objects that are based on object-oriented technology is not useful. Are learning objects more like components? Before we come back to this question in the next section, we would like to refer to some additional Software Engineering laws that have had a big impact on software development and software reuse in general.

Reuse will be successful only if components show low *coupling* (i.e., the term for intra-component communication) and high *cohesion* (i.e., inter-component interaction), and if the details of the implementation are properly encapsulated.

'A structure is stable if cohesion is strong and coupling low' (Myer 1975).

The application of this law leads to more composite and structured design methods and forms the basis for object-oriented design. A common characteristic of the principles of cohesion and coupling, which are described next, is that systems are broken down into software units – this facilitates maintenance, enables reuse of software units, and supports their repurposing (i.e., using a software unit for a different application/purpose). According to Boyle, weak coupling and high cohesion are also essential for learning objects (Boyle 2003). He states that from an instructional point of view, 'each learning object should be based on one learning objective or learning goal'. This allows better sequencing compared to learning objects with more than one learning objective. In addition, low coupling means that a learning unit should have minimal bindings to other objects, i.e., the content should not refer to and use material in another object in such a way as to create necessary dependencies.

Another related law refers to the concept of *encapsulation*:

'Only what is hidden can be changed without risk' (Parnas 72).

The description of what a software unit does is separated from the description of how it does it. Encapsulating and hiding the details of how a unit works facilitates a 'divide and conquer' approach in which a software unit can be developed independently from its clients.

Several interrelated technologies have emerged that support reuse in Software Engineering: component-based development, architecture styles and design patterns, and product lines (Atkinson et al. 2002). These technologies emphasize that reuse can only be successful if we understand the distinction between 'developing for use' (developing a component for a specific application) or 'developing for reuse' (developing a learning object for numerous applications).

Reuse most often starts with code reuse. This distinction is true for all assets, not just components. When reuse is done in a systematic and disciplined way, reuse can also be applied to work products such as requirement specifications, design specifications, test cases, user documentation, screen layout, or project plans. Since learning objects are predicted to get more complex, these approaches could help to make the development of learning objects *for* reuse following certain instructional design rules more transparent and systematic. Therefore, several technologies will be explained in more detail in the following sections before we come back to LOs and instructional design. They support development for reuse on three different levels of abstraction. Learning objects should be considered as components instead of objects and in some cases even as independently deployable *software* components due to their increasing complexity in terms of interactivity with the learner and the system, different media used, increased set of metadata, and the demand for adaptability and higher reusability.

Components and Interfaces

'A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties.' (Szyperski, Pfister 1997)

The component paradigm can be seen as an extension of the object paradigm. According to the definition above, a software component has to be well separated from its environment and other

components. In order to be composable with other components, it needs to be sufficiently self-contained and has to provide a clear specification of what it requires and provides. Hence, a component will interact with its environment by means of interfaces. A component has no state, which means that a component cannot be loaded into a particular and activated like an object system. It exists only once in a system. A component normally consists of several classes and comes into life by instantiating these. Using components in the development of a new system is therefore more an assembly process than a programming process.

An interface defines the component's access points. A component can have multiple interfaces, each representing a service that the component offers. Keeping in mind that components and their clients are developed in mutual ignorance, interface specifications are the *contracts* that form a common ground for successful interaction between service providers and their clients. The contract states what the client needs to do to use the interface and describes what the provider has to implement to meet the services promised by the interface. A popular possibility to define contracts is to specify pre- and postconditions for each operation. The client has to fulfill the precondition before calling the operation, and the provider has to establish the postcondition before returning to the client.

The above definition also requires components to specify their needs. This means that so-called *context dependencies* have to be specified, which define what the deployment environment will need to provide so that the components can function (i.e., the component world is has been prepared for). These dependencies describe the context of composition and deployment.

The main goal is to maximize reuse. However, this has one substantial disadvantage: the explosion of context dependencies (Szyperski 1998). Since, components and the environments they are used in are evolving over time, a large number of context dependencies will decrease the number of possible clients that satisfy the requirements. Hence, the component designer has to decide whether a component will be prepared for environmental changes by increasing the degree of self-containment (i.e., by reducing context dependencies) or increasing the context dependencies in order to provide leaner components by means of reuse.

Components are one of the leading reuse technologies in software development. They embody accumulated knowledge about what good reusable components are and how they should be implemented. Nevertheless, additional knowledge about assembling components is needed. Architectures, frameworks, and patterns have become effective technologies for defining and creating assemblies of such components.

Architecture Styles, Design Patterns, and Frameworks

Architecture styles and design patterns address reuse on the next level of granularity. They support the reuse of strategies for assembling basic building blocks (i.e., components) on a higher level of abstraction. They abstract from the details of a certain application and capture the main aspects of a solution concept in the form of a model – mostly language independent. Architectures are assumed to be of a larger design scale, and styles can be viewed as more oriented towards automated application than patterns. Despite their differences, both generic assembly concepts have proven their success in solving recurring software problems.

The idea of pattern was imposed by Alexander. He worked out a pattern language for describing workable solutions to recurring problems in the design of buildings (Alexander 1979). In the software development domain, the 'Gang of Four' proposed the notion of design patterns as descriptions of communicating objects and classes that are customized to solve a general software design problem. A design pattern is a schematic description of a possible solution and a design

problem. It provides a template showing how elements of the solution (e.g., objects, classes) should be arranged.

Gamma et al. claim that:

'Reusing design through pattern yields faster and better maintenance' (Gamma et al. 1995).

In fact, design patterns for different purposes can be found: analysis patterns, architectural patterns, design patterns, and code idioms (Atkinson et al. 2002). Another way of reusing designs is through frameworks. They provide a skeleton of an application where details have to be filled in. Hence, they are almost finished systems that can be instantiated. The disadvantage is that they are limited to a specific domain and type of application. One of the most important roles of a framework is its regulation of the interactions that the components of the framework can engage in. Most frameworks apply patterns in their design. Hence, their design can be bottom up and pattern driven, or top down and target driven. The next section presents one of the most popular and systematic approaches for supporting the development of frameworks.

Product Lines

This approach supports reuse at the highest level of granularity. A product line relies on the fact that a family of similar products has several overlapping parts and variable parts within a single core. A product line approach provides support for the consolidation of common parts (i.e., *commonalities*) and support for the selection and tailoring of the variable parts (i.e., *variabilities*). Product line engineering profits from the concept of components since components allow a rapid and efficient assembly into new applications. Variability and commonality in product line engineering are captured via *genericity*. A generic artifact is an artifact that holds all possible variants of the family, but provides some possibilities to select between them (IEEE Software 2002).

The development cycle can be split up into two parts: one dealing with the development of the framework (i.e., the core of a product line), that addresses mainly the development of the generic artifacts capturing the feature sets that are characteristic of all members of a product line (i.e., framework engineering); and another related to the development of an application – a concrete instance of the framework, adapted and extended to the needs of a customer (i.e., application engineering).

One goal of product line engineering is to control the variabilities among applications in a family by planning in advance for future changes and requirements. All variabilities can be described by means of alternatives. However, as stated by Muthig, modeling the alternatives does not define which characteristics are associated to which products, nor which dependencies and interrelationships exist among variabilities. This is captured in so-called *decision models*. Decisions are variation points that play a special role during application engineering (Muthig 2002).

Implications on Instructional Design from the Software Engineering Perspective

After introducing approaches that support systematic reuse in software development, this section describes the implications on instructional design from the Software Engineering perspective. We believe that:

- Design patterns are a good means for making design ideas more explicit and hence applicable

- Learning objects should be considered as (software) components instead of object-oriented programming objects
- Developing learning objects *for* reuse by following the component paradigm will increase their reusability
- Separation of concerns by applying different SE technologies will decrease the complexity of instructional design and its embodiment (i.e., reusing concrete learning objects).

In the context of this paper, instructional design aims at the production of learning material by means of learning objects. This task relies mostly on the experience of the instructional designer. This is also true in the domain of software development: instead of being based on a sufficiently strong theory and instead of 'calculating' software design systematically, Software engineers rather combine a little theory with a lot of experience. Nevertheless, compared to Software Engineering, many instructional design theories exist – even if they are difficult to be applied to the learning object domain. On reason is that no explicit rules are available on how learning objects, in general, should be selected and sequenced to make instructional sense (Knolmayer 2003). Goodyear states that the means by which instructional design experience is shared – mainly by text – needs improvement (Goodyear 2005). The last point makes it extremely difficult, especially for non-instructional designers, to create their learning material, since good instructional design still requires much professional experience. In order to make instructional design more applicable, initial design patterns have been developed. Goodyear presents design patterns and pattern languages for networked learning (Goodyear 2005). His patterns use a similar template as the one used by Alexander in the architectural domain (Alexander 1979) and also in the software development domain (Gamma et al. 1995). He motivates design patterns as a good approach for 'providing a comprehensive set of design ideas in a structured way so that relations between design patterns are easy to understand, for clearly articulating the design problem and solution, and for encoding this knowledge in such a way that it supports an iterative, fluid process of design, extending over hours and days'.

As already stated in a previous section, learning objects are more comparable to the characteristics of components and, therefore, we should give more attention to approaches from the component-based development domain than we have done it in the past – if there has been any consideration at all. Boyle introduced compound objects to resolve the conflict between cohesive decoupled learning objects and pedagogically rich content. 'A compound object consists of two or more learning objects that are linked to create the compound'. They intend to be more pedagogically rich and suitable for repurposing by deleting or adding learning objects to the compound object (Boyle 2003). As detailed later, components could also contain other components. The relationships within such a component hierarchy are defined by a *containment tree*.

As Longmire says, developing learning objects requires two perspectives: on the one side, we need a global understanding or curricula to conceive a learning object as part of a whole, and on the other side, a detailed vision is needed to create content as standalone information for it to function as reusable object (Longmire 2000). This echoes our earlier statement that described the distinction between *instructional development* and *instructional design*. Software engineers have to deal with the same problem when they develop reusable components as parts of a complete system – they have to know the future environments where the component will be reused and they have to design the component in such a way that it can be deployed independently. Despite many new descriptions of what learning objects should look like, most recent work focuses on characteristics such as high cohesion, decoupling, etc. – the development methods with respect to

instructional design are still under consideration. It is exactly here where the development of learning objects can profit from Software Engineering approaches.

The last part of this section will explain how we can improve the development *for* reuse of learning objects by presenting a component-based approach that has been adapted and extended from the component-based product line engineering method KobrA (Atkinson et al. 2002). The approach uses the previously introduced technologies: components, frameworks, and product lines. The method is organized in terms of three orthogonal dimensions:

- development that deals with the *genericity* dimension refers to *product line engineering*,
- development that deals with the *composition* dimension refers to *learning space engineering*, and
- development that deals with the *abstraction* dimension refers to *component embodiment*.

The relationships between these dimensions are shown in Figure 10. These three approaches need not be performed in a certain sequence; they can be performed in various orders or even used in an interwoven manner.

In the following, we explain how we can use Software Engineering technologies for developing component-oriented learning material based on explicit models of instructional design. We use the term *learning component* to emphasize that a learning object is close to the definition of a component that was given in the previous section. Figure 10 shows the path from an abstract, generic, and coarse grained view of a conceptual component model to concrete, specific, and fine-grained learning material composed of learning components that can be used in a specific learning context. Our explanations focus more on the process steps of the method by explaining the Software Engineering technologies used and less on the artifacts that are created, adapted, or used during the process. A detailed description of these artifacts would be out of the scope of this chapter.

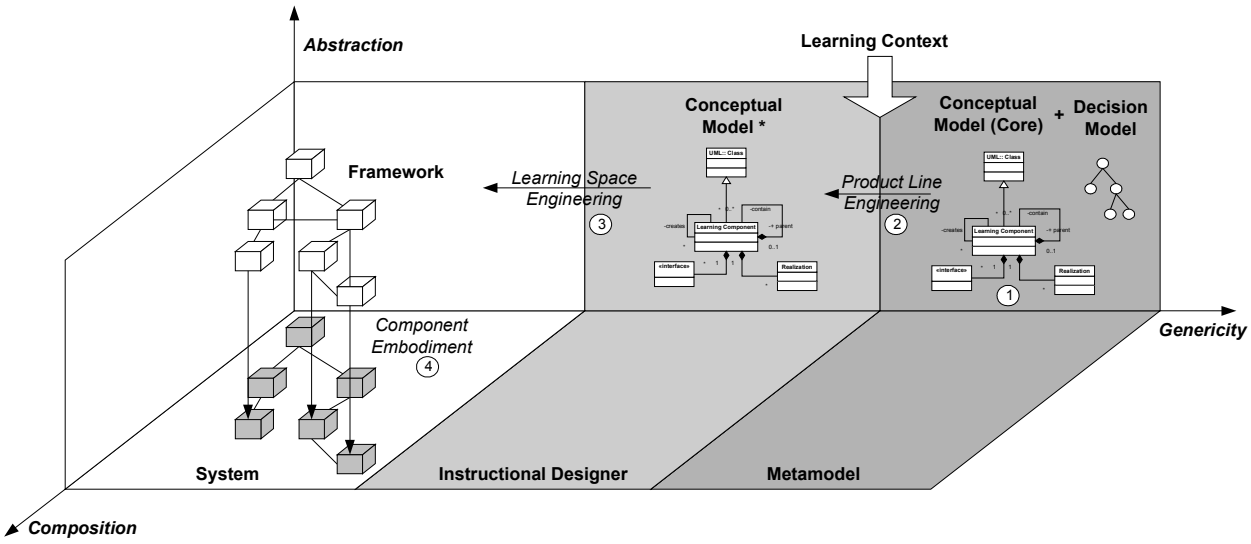

Figure 10: Development dimensions (according to the KobrA method (Atkinson et al. 2002))

First of all, the metamodel describes a learning component from a logical point of view on the highest level of abstraction (1). The metamodel describes the basic structure of a component and

a decision model. The metamodel can be described, for example, by means of UML class diagrams (UML stands for Unified Modeling Language and is used for object-oriented modeling). The conceptual model covers aspects such as the specification of metadata (e.g., by using LOM or parts of the standard), containment rules that specify the parent-child relationships between learning components within a containment tree (e.g., for modeling the location of learning components), specialization rules that define the categories of learning components and their specialization (e.g., definitions-, examples-, table of content-, overview-, and summary-components). Furthermore, this model contains elements that specify the kind of interaction between the system and the user, and adaptation rules for adapting the learning components to learner types or context aspects. These rules are very similar to the contracts used as specifications for interfaces. Beside the conceptual model, a decision model exists that enables the instructional designer to adapt the model (marked as conceptual model* in Figure 10) to a certain learning context.

Second, the next process step is based on product line engineering (2). The decision model contains so-called variation points, their resolution space, and their effects on the conceptual model. The variation points are resolved by questions. The instructional designer uses these questions in order to change the conceptual model in a systematic way. The questions refer to the categorization of learning objects (i.e., the instructional designer adapts the categories or the specialization structure), instructional design strategies (i.e., the instructional designer adapts the containment rules, for example, to an *experiential learning* strategy), or the questions consider adaptation aspects (i.e., the instructional designer changes the variable parts of the logical learning component). The answers to the questions include solution packages, so that the instructional designer gets support on how to adapt the model. One possibility for defining such solution packages is to use design patterns. They are very useful for describing instructional design strategies in a comprehensive manner. A design pattern can be understood as a transformation process from one conceptual model state to a new state. Each transformation step relates to specific parts of the model and tells the instructional designer how to change those parts. The next two steps are mainly performed by the system.

Third, the conceptual model, which defines a learning component on a logical level for a certain learning context, is instantiated to create a learning component framework (3). This step is called Learning Space Engineering. Such a framework is very close to the implementation level and can be understood as a network of learning components instances that have no content yet. This means that the conceptual model is instantiated for each component in the framework. The relations between the learning component instances are derived from the rules of the conceptual model (e.g., containment rules, adaptation rules, interaction rules, etc.).

The last step is called component embodiment. This step fills the learning components with physical data (e.g., text, pictures, etc.). Compared to conventional learning objects, learning components communicate through their interfaces with other components by following the previously specified contracts. Relations that reflect the learning strategy (e.g., experiential learning) connect the components of different types within the learning space. It depends on the framework whether these relations are implemented, for example, by hyperlinks or in some other way.

Conclusion

Flechsig uses an illustrative metaphor when discussing what instructional design–he might prefer the word didactic design–is about (Flechsig 1996). Imagine you find a tourist map where some path has been marked. Is this the a posteriori description of a path someone has been walking? Or is this the a priori indication of a way someone had been planning to walk? You can’t tell

from the path alone. It is just a path. It is neither descriptive nor prescriptive by its very nature. It has its constituents, its stations, its decision points and the like.

Instructional design deals with the set up of spaces in which (1) a designer has pre-selected a learning path, or in which (2) learners can create their own paths and consequently create their own learning experience, or in which (3) a pre-selected path can vary based upon adaptive decisions. It prepares learning paths such that learners with different prerequisites, with different needs and desires, in varying moods and under widely unforeseeable circumstances can find their way. Learning objects and learning components are building blocks for spaces of learning experience, and they are constituents of a variety of learning paths – both dynamic and static. When developing experience spaces, we should strive to move beyond text-centric building blocks and concentrate on designing interactive dynamic learning content that supports the construction of knowledge

From the perspective of building spaces of a particular type, instructional design is an engineering discipline, like software engineering and civil engineering (Heinich 1984). It could be thought of as a discipline of social engineering. All engineering disciplines have their scientific roots. Building bridges is partially based on calculus; IT security engineering is partially based on number theory and on cryptography, for instance. Instructional design has both scientific and practical roots. Its theory is derived from psychology, sociology, and organizational development, while its practical roots come from education.

The relationship between engineering disciplines and their underlying sciences is known to be delicate. So it is with instructional design. There are scientific results that tell us on a high level about the peculiarities of humans–especially of human teachers–in learning (Bransford et al. 2003), (Damasio 1999), (Davis et al. 2000), and others tell us on a very fine-grained level what does attract the learners' attention and what does not (Downing et al. 2004; Fleming & Levie 1978). Some tell us very explicitly how expertise shows internally (Gauthier et al. 1999), (Gauthier et al. 2000), but do not give any hint how to arrive at the states they are reporting about, thus exemplifying the distinction between a descriptive and a prescriptive field.

It is another peculiarity of didactic design in Europe that there are so many different approaches. After a divergence from the roots (Comenius 1638), several attempts in the second half of the 20th century to unify the perspectives made it even more cumbersome (Habermas 1985), because clarity got lost. The field of education, in the United States, has suffered from a similar lack of clarity; however, instructional design in the US has often been reductionist in its attempt to create cost-effective design templates and design models. This differs sharply from the US approach to education, which is still largely unstructured and teacher-centered.

Therefore, it seems to be reasonable–for the present book chapter, at least–to choose a pragmatic approach focusing the engineer's building blocks of spaces and paths: learning objects and components.

However, approaches developed so far are still limited. There is an urgent need to unite the terms and approaches of instructional design with a structured, systematic process approach offered, e.g., by Software Engineering. This will bring us towards an engineering approach that supports software design and development (Jantke, Knauf 2005). The present contribution takes some preliminary steps forward to propose learning objects as (software) components, focused on practical instructional design concepts. Thus, instructional design meets Software Engineering.

References

Alexander, C. (1979). The Timeless Way of Building. Oxford University Press.

Atkinson, C. et al. (2002). Component-based Product Line Engineering with UML. Addison-Wesley.

Biggerstaff, T.J., Perlis A.J. (eds) (1991). Software Reusability. Vol I, Concepts and Models. Addison-Wesley.

Boyle, T. (2003). Design Principles for Authoring Dynamic, Reusable Learning Objects. Australian Journal of Educational Technology, 19 (1), 46-58.

Bransford, J. D., Brown, A. L., Cocking, R. R. (2003). How People Learn. Brain, Mind, Experience, and School. National Academy Press.

Brusilovsky, P., Vassileva, J. (2003). Course sequencing techniques for large-scale web-based education. Int. J. Continuing Engineering Education and Lifelong Learning, 13, 75-94.

Comenius, J.A. (1638). Didactica magna.

Damasio, A. (1999). The Feeling of What Happens. Body and Emotion in the Making of Consciousness. Hartcourt Inc.

Davis, B., Sumara, D., Luce-Kapler, R. (2000). Engaging Minds. Learning and Teaching in a Complex World. Lawrence Erlbaum Associates.

Dewey, J. (1929) Experience and Nature, New York Dover. Enlarged and revised edition of the Paul Carus lectures first delivered in 1925. Examines experience and philosophical method; experience; nature; and experience, nature and art.

Downing, P. E., Bray, D., Rogers, J., Childs, C. (2004). Bodies capture attention when nothing is expected. Cognition 93 (2004), B27-B38.

Endres, A., Rombach, H.D. (2003). A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories, Fraunhofer IESE / Pearson Addison-Wesley, 327 p., ISBN 0 321 154207, 2003.

Flechsig, K.-H. (1996). Kleines Handbuch didaktischer Modelle. Neuland.

Flechsig, P. (1920). Anatomie des menschlichen Gehirns und Rückenmarks auf myelogenetischer Grundlage. Leipzig.

Fleming, M. & Levie, W. H. (1978) *Instructional Message Design*, Englewood Cliffs, NJ: Educational Technology Publication.

Forsha, H. I. (1994). The Complete Guide to Storyboarding and Problem Solving. ASQ Quality Press.

Friesen, N. (2001). What are Learning Objects? Interactive Learning Environments. 9, (3).

Friesen, N. (2004). Three Objections to Learning Objects and E-learning Standards. In R. McGreal (Ed.), Online Education Using Learning Objects, London: Routledge, 59-70.

Gagne, R.M. & Briggs, L.J. (1979). *Principles of Instructional Design: Second Edition*. New York: Holt, Rinehart, and Winston.

Gamma, E., Helm, R., Johnson , R., Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-oriented Software. Addison Wesley, Reading, MA.

Gauthier, I., Skudlarski, P., Gore, J. C., Anderson, A. W. (2000). Expertise for cars and birds recruits brain areas involved in face recognition. nature neuroscience 3 (2000) 2, 191-197.

Gauthier, I., Tarr, M. J., Anderson, A. W., Skudlarski, P., Gore, J. C. (1999). Activation of the middle fusiform "face area" increases with expertise in recognizing novel objects. nature neuroscience 2 (1999) 6, 568-573.

Goodyear, P. (2005). Educational Design and Networked Learning: Patterns, Pattern Languages and Design Practice. Australasian Journal of Educational Technology, 21(1), 82-101.

Habermas, J. (1985). Die neue Unübersichtlichkeit. edition suhrkamp.

Hagebölling, H. (2004). Interactive Dramaturgies. New Approaches in Multimedia Content and Design. Springer Verlag.

Heinich, R. (1984). The proper study of instructional technology. Educational Communications and Technology Journal, 33(1), 9-15.

IEEE Software (2002). Special Issue on Software Product Lines, July/August 2002, Vol. 19 (4).

IMC GmbH (2001). Corporate Learning and Information Exchange, Technical Whitepaper, Freiburg, Germany.

Jank, W., Meyer, H. (2002). Didaktische Modelle. Cornelsen.

Jantke, K. P., Degel, G., Grieser, G., Memmel, M., Rostanin, O., Tschiedel, B. (2004a). Technology Enhanced Dimensions in e-Learning. In M. E. Auer, U. Auer (eds.), International Conference on Interactive Computer Aided Learning, ICL 2004, Sept. 29 - Oct.1, 2004, Villach, Austria (CD-ROM).

Jantke, K. P., Memmel, M., Rostanin, O., Rudolf, B. (2004b). Media and Service Integration for Professional E-Learning. In E-Learn 2004, World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education, November 1-5, 2004, Washington D.C., USA, 725-731.

Jantke, K. P., Igel, C., Sturm, R. (2005). Learner Modeling and Adaptive Behavior Toward a Paradigmatic Shift from e-Learning Tools to Assistants. In R. Kaschek (ed.), First International Workshop on Perspectives of Intelligent Systems' Assistance, PISA 2005, Palmerston North, New Zealand, March 3-5, 2005. Massey University. Dept. of Information Systems, 60-71.

Jantke, K. P., Knauf, R. (2005). Didactic Design through Storyboarding: Standard Concepts for Standard Tools. In B. R. Baltes et al. (Eds.), First International Workshop on Dissemination of E-Learning Technologies and Applications, DELTA 2005, in: Proceedings of the 4th International Symposium on Information and Communication Technologies, Cape Town, South Africa, January 3-6, 2005, 20-25.

Keller, J. M. (1983). Motivational Design of Instruction. In C.M. Reigeluth (Ed.), Instructional-design theories and models: An overview of their current status. Hillsdale, NJ: Lawrence Erlbaum Associates.

Knolmayer, G.F. (2003). Decision support models for composing and navigating through e-learning objects. 36th IEEE Annual Hawaii International Conference on System Sciences, Hawaii, USA.

Koster, R. (2005). A Theory of Fun for Game Design. Paraglyph Press, Inc.

Longmire, W. (2000). Content and Context: Designing and Developing Learning Objects. Learning without Limits, 3, Informania.

Memmel, M. (2005). Adaptivity with Multidimensional Learning Objects. In E-Learn 2005, World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education, October 24-28, 2005, Vancouver, Canada.

Merrill, M.D. (1983). Component display theory. In C.M. Reigeluth (Ed.), Instructional-design theories and models: An overview of their current status. Hillsdale, NJ: Lawrence Erlbaum Associates.

Mohan, P., Brooks, C. (2003). Engineering a Future for Web-based Learning Objects. ICWE 2003, 120-123.

Mohan, P., Daniel, B. (2004). The Learning Objects' Approach: Challenges and Opportunities. In E-Learn 2004, World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education, November 1-5, 2004, Washington D.C., USA, 2512-2519.

Muthig, D. (2002). A Lightweight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. In Series of PhD Theses in Experimental Software Engineering, vol 11, Fraunhofer IRB Verlag

Myer, G. J. (1975). Reliable Software Through Composite Design. New York: Petrocelli.

Newport, E. L. (2002). Critical Periods in Language Development. In L. Nadel (Ed.), Encyclopedia of Cognitive Science, London: Macmillan Publishers Ltd., 737-740.

Parnas, D.L. (1972). On the Critera to Be Used in Decomposing Systems into Modules. Comm. ACM 15, 12, 1053-1058.

Pfleeger, S.L. (2001). Software Engineering Theory and Practice. Prentice Hall. 2nd ed.

Posner, M. I., Raichle, M. E. (1998). The neuroimaging of human brain function. Proc. Nat. Acad. Sci. USA 95 (1998), 763-764.

Rabenalt, P. (2004). Filmdramaturgie. VISTAS media production.

Ras, E., Memmel, M., Weibelzahl, S. (2005). Integration of E-Learning and Knowledge Management - Barriers, Solutions and Future Issues. In Professional Knowledge Management (Berlin, 2005), K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, and T. Roth-Berghofer, Eds., Lecture Notes in Artificial Intelligence LNAI, Vol. 3782, Springer Verlag.

Reigeluth, C. M. (1983). Instructional-design theories and models: An overview of their current status. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.

Reigeluth, C. M., (1999). Instructional Design Theories and Models: A New Paradigm of Instructional Theory, Volume II, Mahwah NJ. Lawrence Erlbaum Associates, Inc.

Robson, R. (1999). Object-oriented Instructional Design and Web-based Authoring. World Conference on Educational Multimedia, Hypermedia and Telecommunication, Seattle, Washington, USA, pp. 698-702.

Rogers jr., H. (1967). Theory of Recursive Functions and Effective Computability, McGraw-Hill.

Rostanin, O. (2004). An Alternative Approach to Building Web-Applications. Sixth International Conference on Enterprise Information Systems ICEIS 2004, Porto, Portugal, Proceedings, Vol. 1, 119-124.

Rymer, R. (1994). Genie: A Scientific Tragedy. Harper Collins.

Self, J. (1992). Computational Mathetics: the missing link in Intelligent Tutoring Systems research? Directions in Intelligent Tutoring Systems, NATO ASI Series F, Vol. 91, 36-56.

Smith, P.L & Ragan, T.J. (2005). Instructional Design, 3rd Ed. Wiley & Son, Inc.

Spitzer, M. (2002). Lernen. Gehirnforschung und die Schule des Lebens. Spektrum Akademischer Verlag.

Szyperski, C., Pfister, C. (1997). Workshop on Component-Oriented Programming, Summary. In Mühlhäuser, M. (ed), Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader. dpunkt Verlag, Heidelberg.

Szyperski, C., (1998). Component Software – beyond Object-Oriented Programming. Addison-Wesley, England.

Udell, J., (1994). ComponentWare. BYTE Magazine, 19(5), 46-56.

Wiley, D. A. (2000). Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. In D. A. Wiley (Ed.), The Instructional Use of Learning Objects: Online Version. Retrieved April 20, 2005, from the World Wide Web: http://reusability.org/read/chapters/wiley.

Wittgenstein, L. (1953). Philosophische Untersuchungen. Philosophical Investigations. G.E.M. Anscombe, R. Rhees (eds.), Oxford.

Yacci, M. (1999). The Knowledge Warehouse: Reusing Knowledge Components. *Performance Improvement Quarterly*. Vol 12, Number 3. Pps. 132-140.

Yacci, M., Haake, A. & Rozanski, E.P. (2004). Eye-Tracking Edutainment Interfaces: Operations and Strategy Learning. Proceedings of AACE World Conference on Educational Multimedia, Hypermedia, and Telecommunications. Lugano, Switzerland.

Yacci, M. (2004). Game Based Learning: Structures and Outcomes. Proceedings of the Society for Information Technology and Teacher Education (SITE) 15th International Conference, Atlanta, Georgia.

Yacci, M. (2005). The Promise of Automated Interactivity. In Professional Knowledge Management (Berlin, 2005), K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, and T. Roth-Berghofer, Eds., Lecture Notes in Artificial Intelligence LNAI, Springer Verlag.

Biography

Martin Memmel studied mathematics and computer science at the University of Kaiserslautern and received the diploma in 2001. From 2001 to 2003 he was working as a scientific assistant in the research and development project DaMiT sponsored by the German Federal Ministry for Education and Research (BMBF) at the University of Kaiserslautern. The practical goal of the project was to provide an e-learning system for the domain of knowledge discovery & data mining. In DaMiT, he was responsible for the architecture and the metadata specification of the learning object repository and the conception and implementation of the authoring tool.

Since 2003, he has been working as a research scientist at German Research Center for Artificial Intelligence (DFKI) in Kaiserslautern in the Knowledge Management research department, working on knowledge objects and metadata specification for e-learning and Knowledge Management systems. He is involved in the NoE Prolearn (WP1 and WP4) and co-organizer of the

workshop LOKMOL (Learner-Oriented Knowledge Management & KM-Oriented e-Learning) and currently working in the BMBF-funded project BibTutor on the development of an electronic tutoring system assisting users in literature research.

Eric Ras earned his diploma in computer science at the Technical University of Kaiserslautern in 2000. From that time, he has worked as a scientist on different public and industrial projects in the domain of Knowledge Management, agent-based e-learning, and Document Engineering at the Fraunhofer Institute for Experimental Software Engineering. He has gathered experience in reuse-based learning material production, work-process oriented vocational training methods, and has worked intensively with current e-learning standards and tools. His PhD topic focuses on pedagogical information agents, which play a key role in personalization of learning and pro-active information delivery at the workplace, experiential learning for software engineers, and the application of Software Engineering principles and approaches to e-learning.

He has been chair of the Workshop on Learning-oriented Knowledge Management and KM-oriented E-Learning 2005 (LOKMOL '05 takes place in conjunction with the Wissensmanagement 2005 Conference). In addition he is a PC member of the Workshop on Human and Social Factors of Software Engineering'05 (HSSE '05 takes place in conjunction with the international conference on Software Engineering) and Integrated Working and Learning'05 Track (IWL'05 takes place in conjunction with the i-Know Conference that focuses especially on the integration of KM and e-learning and put a strong emphasize on knowledge and learning objects).

Klaus P. Jantke studied mathematics at Humboldt University Berlin and earned his diploma in 1975 at Humboldt. He has a Ph.D. in computer science (1979) and a habilitation in computer science (1984), both at Humboldt. He was winning the Karl Weierstrass award in 1977 and the Humboldt award in 1981. Klaus P. Jantke has a supplementary education in university level teaching completed with the Facultas Docendi in 1984. He has another degree in quality management and a qualification in IT security evaluation.

Over the years, his main research areas are algorithmic learning theory, abstract data types and formal semantics, planning in dynamic environments and process control, meme media technology, and technology enhanced learning. He put most emphasis on the learnability of computer systems, even in areas like ADT, planning, meme media, and e-learning.

Klaus P. Jantke became a full professor of computer science at Leipzig University of Technology, Leipzig, Germany, in 1987. After German unification in 1990, he served as the first freely elected head of the department and as the first dean of the Faculty for Informatics, Mathematics and Natural Sciences. He worked as a research fellow at Stirling University (S.E.R.C. Research Fellowship), at ICSI Berkeley, at Fujitsu Research Labs in Numazu and in several other places, and he worked as a professor at Kuwait University, Kuwait, and at Hokkaido University Sapporo, Japan. For about seven years, he has been a principal researcher at the German Research Center for Artificial Intelligence (DFKI) and the first head of the DFKI's Competence Center for e-Learning (CCeL). Since 2005 he is the Chief Executive Officer of FIT Leipzig, the Research Institute for Information Technologies in Leipzig, Germany. He is regularly working as a professor at the Meme Media Laboratory of Hokkaido University, Sapporo, Japan.

Klaus P. Jantke's current focus of research is on intelligent systems' assistance and knowledge evolution with particular applications to information extraction, meme media technologies and technology enhanced learning.

Michael Yacci received his B.S. degree in Communications (Ithaca College), M.S. degree in Instrucional Technoloy (Rochester Institute of Technology) and PhD. In Instructional Design, De-

velopment, and Evaluation (Syracuse University). He has done consulting design work for corporate clients such as Kodak, Xerox, and IBM as well as the US Government's Center for Excellence in Government. He has received research and project funding from the National Science Foundation. Professor Yacci has taught at Rochester Institute of Technology since 1986, winning the prestigious Eisenhart Award for excellence in teaching in 2000. He has been instrumental in developing several new degree programs at RIT including two MS degrees, and a PhD program. His research interests range from computer based training, instructional design, learning theory, performance support, knowledge management, and evaluation.