Chapter 1
Introduction

Abstract This chapter introduces an educational software oriented perspective of the Technology Enhanced Learning field, this book's motivations and objectives, and a first general overview of its content and structure.

This book is about educational software, i.e., software designed as a means to implement computer-based pedagogical settings and contribute to addressing some pedagogical objectives.

Design of educational software consists of imagining, thinking, elaborating and describing a computer-based system with respect to some pedagogical objectives and to the different educational constraints to be taken into account in relation to the setting within which the software will be used. The output is an understanding of the software to be implemented, i.e., what is to be made operational on the computer by programming. This programming technical phase may be conducted from scratch or build on existing software components.

Design of educational software may correspond to very different realities such as inventing and implementing new software or analyzing how to aggregate and/or adapt existing components according to objectives and constraints, as when using Information and Communication Technologies (ICT) to build ICT-based systems (see Sect. 3). It may be conducted in relation to different rationales, such as the implementation of a particular pedagogical setting (which consists of considering different issues from which software) or the elaboration of software whose existence will allow implementation of some given type of settings (e.g., software meant to support on-line collaborative problem solving). It may arise in different contexts such as a development project and/or a research project,[1] as a means for studying some research question or exploring some innovation. It may be based on

[1]We will use the term *development project* for projects whose only goal is to produce an outcome (a system, a software component) to be used in effective settings, by contrast to *research projects* which are projects seeking to produce research advances. As we will argue in Chap. 6, development and research projects are not mutually exclusive.

different entry points such as a learning theory or the specific features of an emergent technology.

In this book, we explore a transversal issue: providing computer scientists and educationalists with means to address together the relationships between pedagogical settings and software, in particular when considering the design or adaptation of software for some targeted pedagogical settings.

As a way to provide a background for understanding this book's rationale, content and organization (described in Sect. 5), this introductory chapter's structure is as follows. First, we propose a general picture of the Technology Enhanced Learning field or, more precisely, an educational software oriented perspective of this field. In particular, we introduce the core notions of educational software and computer-based pedagogical setting (Sect. 1) and illustrate them by examples (Sect. 2). We then highlight that designing software with respect to educational concerns may correspond to different realities and different education/Computer Science[2] interplays (Sect. 3). This puts to the fore a central issue, that of making explicit matters of concerns and perspectives (Sect. 4). The different elements addressed in these different sub-sections will be further defined and explored in the following chapters.

1 General Picture

1.1 Technology Enhanced Learning

Educational software concerns take place within a more general field that we will call Technology Enhanced Learning (TEL). Other terms convey similar meanings such as E-learning, Learning Technology, Computer Assisted Instruction, On-line Learning, Computer-Based Learning or Computer-Based Teaching. Using one or another of these terms may contextually denote a particular perspective such as when emphasizing the *on-line* or the *teaching* dimensions.

TEL is an arena where different disciplines such as Computer Science (CS), education, psychology, philosophy, communication or sociology intersect. We will refer to actors as *computer scientists* and (when unambiguous) *educationalists* as a generic term for Human and Social Sciences actors. An actor may act as both a computer scientist and an educationalist when competent in both areas.

Considering educational software design corresponds to a specific matter of concern in TEL, and not to a subfield. As a complex field within which different types of issues are to be considered, TEL is to be addressed within different

[2]In recent years, the term *Computer Science* has in some places been supplanted by the term *Informatics*, a term giving more importance to the human and social aspect of computer systems design, usage and evaluation. However, we will stick to the term Computer Science, which is widely accepted (the term Informatics is subject to different interpretations whose discussion is not a matter of concern in this book). In this book, Computer Science refers to software design and implementation issues, and not to mathematical foundations of computing.

perspectives. These perspectives are not only related to disciplinary issues and interplay of disciplines: they also lie in the adopted focus or matters of interest. For instance, how learning mechanisms may be enhanced by technology, how basic educational practices may be changed, or how to design educational software for some given pedagogical setting, correspond to different issues. They require consideration of different types of objects and features, but also lead to consideration of common objects addressed within different perspectives and different matters of concern (e.g., the notion of computer-based pedagogical setting or the analysis of software usage).

Considering notions such as *pedagogical setting* or *educational software* requires introducing precise definitions. However, definitions encompassing multiple perspectives are usually rather general. As a consequence, they provide little guidance and, more importantly, may create confusions or misunderstandings when one skips from a general discourse to precise considerations. In order to deal with this issue we will introduce working definitions, i.e., definitions making salient dimensions that are of interest given the considered topics and intentions. Such definitions introduce a conceptualization[3] of the field that is adapted to the considered matters of concern. It corresponds to just one view, to be complemented by others.

1.2 TEL and CS Technical Artifacts

With respect to computer-based technology, TEL systems may be based on software and/or hardware. As examples of hardware applications: gestural knowledge may be addressed with virtual reality systems embedding haptic devices; mobile technologies may be used for implementing in-the-wild learning settings; etc.

In this book, we consider software design issues. To a large extent, software and hardware raise similar design issues, and thus this book's content is also of interest when considering hardware. Hardware-specific issues, however, will not be addressed here.

Implementing software (we will also use the terms "computer-based systems" – systems for short – or "programs") is the technical task of making software operational. This is not to be confused with implementing a pedagogical setting, which consists of considering different issues (the context, the actors, the timing, etc.) from which software to use.

The use of software in TEL is strongly connected to both the evolution of technology and the evolution of learning and teaching theories. Seen from the point of view of the technology push, the advancement of CS continuously opens up new possibilities, providing ideas and means for innovative software and innovative pedagogical settings. The history of educational systems (and current trends) can be correlated to the history of CS concepts and techniques: algorithmic, hypertext and hypermedia, Artificial Intelligence, network and communication

[3]A *conceptualization* is a differential system of notions.

means or, more recently, mobile technologies. Seen from the point of view of learning sciences, the advancement of technology allows implementation of teaching or learning theories (and raises new questions). For instance inquiry learning, as such, has nothing to do with computers, and may be implemented in classrooms with no technology. However, the advance of CS technologies and the way they allow one to build simulations, to offer learners dedicated cognitive tools or to create means for collaboration allow changes in the way inquiry learning (on-line and in-presence) can be addressed.

1.3 Educational Software

Software used in TEL may be existing software incidentally used in a pedagogical context or software *designed for* educational purposes.

We will refer to software used in a pedagogical context whilst not having been designed according to any educational considerations as *basic software*. As examples: a basic chat tool may be used to allow a group of learners to engage in a collaborative task; a spreadsheet may allow implementation of a powerful simulation if its properties (symbolic formulas, automated calculus, graphical representations) appear pertinent for the considered pedagogical objectives; etc. "Basic" software may of course be complex, e.g., a virtual reality environment.

In contrast with basic software, we will define educational software as software *designed for* educational purposes (a more precise definition will be introduced in Chap. 2). The "design for" dimension puts to the fore the existence of relationships (intentions, expectations, hypotheses, etc.) between pedagogical objectives (and, more generally, pedagogical considerations), design decisions, software properties and ways learners use software.

Designing software for educational purposes leads to taking into account, at design time, educational considerations. This may consist in basing software design on specific educational conceptualizations or models. For instance, let us consider a piece of software meant to implement some given type of calculus that learners need to use while solving some type of problem: the way software is thought of and, in particular, the definition of the software role, may be addressed within a more general analysis of the setting and a model or theory of how learners develop knowledge when solving such problems. As a result, the software specification will be influenced by pedagogical considerations. In addition, software may also present specific educational-related functions or properties. For instance, software may embed knowledge related to the learning domain, implement teaching mechanisms or support learners in a way that is supposed to contribute to the addressing of pedagogical objectives.[4]

[4]To further examine the point that pedagogical principles may be embedded in software or only impact software design see Baker, M. (2000). The roles of models in Artificial Intelligence and Education research: a prospective view. *International Journal of Artificial Intelligence in Education*, 11(2), 122–143.

When considering the implementation of a given pedagogical setting or type of setting, if some more or less adequate basic software exists, designing educational software is just one option. When emphasis is on addressing the considered pedagogical objectives, whether the software used has been designed according to pedagogical considerations is not a matter of concern. Basic software may be appropriate to a given setting, or even if not particularly appropriate, may correspond to the most efficient solution for some other reasons such as its freeness, its availability or the fact learners are used to it.

Designing educational software is a requirement when no satisfactory basic software is available. This may be related to the fact specific functions (e.g., tutoring capabilities) or properties (e.g., the way a simulation may act as a cognitive tool or the way communication tools may support learners' collaboration) are required to allow the implementation of the considered pedagogical setting, are expected to increase learning outcomes and/or to allow the addressing of some other objectives.

The dichotomy basic/educational software is based on whether design takes into account some pedagogical dimensions, and thus puts to the fore the issues such a design may raise. It does not implicitly suggest basic software should not be used to implement pedagogical settings or that settings implemented with software designed according to educational concerns necessarily lead to better learning outcomes than if implemented with basic software (as will be discussed, software properties are just one dimension of why and how software is used).

1.4 Computer-Based Pedagogical Settings (CBPSs)

Nothing is "pedagogical" as such: almost any artifact may present some pedagogical interest and virtues in some given context, and software that has demonstrated virtues in some context may be completely inadequate in another one, or under different conditions. The core issue is not that of the software properties but what happens, i.e., the way settings unfold and, with respect to software, the way learners use it, and how this relates to the pedagogical objectives.

Design of educational software or analysis of educational/basic software pedagogical interests or impacts must therefore be addressed with respect to more or less precise settings or types of settings.

When designing educational software, the prototypical case is that of institutional settings taking place within some curriculum. Designing software for informal learning, however, also requires some definition of the type of setting considered (e.g., designing software that contextually helps users to draw connections between an experience they face in the context of their professional practice and some knowledge repository). Of course, the settings within which software is effectively used, and the way software is used, may differ from what it was defined for (and the implications of this may be part of the software specification, for instance targeting tailorable[5] software). However, design is never addressed without some anticipated uses and contexts of use in mind.

Considering educational software puts to the fore the notion of *Computer-Based Pedagogical Setting* (CBPS), which may be defined as a pedagogical setting involving some software that is meant to play a role in relation to the considered pedagogical objectives. When designing educational software, the scope to be considered is not restricted to that of the software but, more generally, the context within which the software is supposed to be used. The CBPS is an essential dimension of this context. In formal settings, another important dimension is the teaching setting, i.e., the different institutional, human and material features forming the context within which the CBPS takes place. More generally, different types of dimension may require consideration (e.g., sociological or cultural) and, of course, in some projects, actors' individual characteristics.

When designing or analyzing CBPSs matters of concern and, in particular, considered pedagogical objectives, may correspond to very different realities. Accordingly, software role may relate to very different features. As examples of the variety of possible concerns, software design may be addressed in relation to whether:

- Learners are provided with resources (e.g., content resources or communication means).
- Learners become familiar with an issue or a type of problem.
- Learners engage in some activity (e.g., problem solving or argumentation).
- Learners practice some domain-specific skills (e.g., addition of fractions) or meta-level competences (e.g., synthesis or learning-to-learn).
- Learners challenge their current knowledge.
- Learners acquire some target knowledge or develop some skills.
- Teachers[6] are supported (e.g., provision of automatic tutoring means that help teachers in supervising learners' actions).
- A setting can be organized in a way that fits with the institutional constraints.
- Etc.

This list (far from being exhaustive) illustrates that concerns may be very different, may relate to different notions and levels of granularity, and are generally intertwined. For instance, considering whether software provides access to content resources is different from considering whether learners will learn from these resources, although this is probably the rationale for providing this access. Considering whether learners *practice* some skills is different from considering whether they *develop* their skills, etc. We will come back on the importance of disentangling objectives in Chap. 2, Sect. 3.1, in particular for evaluation issues.

[5]A computer-based system is said to be *tailorable* if it provides its users with integrated support for modifying it in the context of its use. For an analysis of different tailoring techniques see for example: Morch, A. (1997). Three Levels of End-user Tailoring: Customization, Integration, and Extension. In: Kyng, M. & Mathiassen L. (Eds.) Computers and Design in Context (pp 51–76). Cambridge, the MIT Press.

[6]When unambiguous, the term *teacher* will be used in a generic way for *teacher*, *tutor*, *facilitator*, *pedagogical engineer*, etc.

1.5 Non-definitional Character of Software

Independently of whether software has been designed for a pedagogical setting, software is not *definitional* of this setting.

Pedagogical settings are socio-technical systems, they involve humans and artifacts (the setting as such, the more general institutional, human and material features within which this setting takes place, the resources offered to learners and teachers, etc.). Software may be meant to play (or appear to play) a more or less structuring role according to the setting and/or the type of considered software. However, software does not determine what will happen.

As an example, let us consider the way the role of (1) an Intelligent Tutoring System (ITS) and (2) a chat or a forum meant to be part of a set of basic tools offered in a network learning context may be thought of.

The basic principle of ITSs is to introduce a task to be addressed (e.g., a mathematics problem to be solved), a set of specific tools for achieving the task (e.g., an editor allowing learners to emphasize the different problem-solving steps and logical connections) and individualized tutoring based on the analysis and interpretation of learners' actions and output (correction, help, explanations, adaptation of the task to be addressed, etc.).

Networked learning settings are settings in which network technology is "... used to promote connections between one learner and other learners, between learners and tutors; between a learning community and its learning resources".[7] Such settings may for example be implemented with Learning Management Systems (LMSs), i.e., Web-based platforms offering general means such as generic communication tools (chat, forum) or file exchange zones.

When learners solve a problem with an ITS, the system makes salient some notions, imposes some actions, provides some hints, etc. The system plays a pro-active role and, in some sense, orchestrates the setting. Its properties are designed to (and are likely to) have an important impact on what happens and how.

Differently, let us consider a group of on-line learners asked to collectively write a document via an LMS. The major definitional features are likely to be the task at hand and the way it is introduced by the teacher, the individuals' and the group's experiences of on-line collaboration or the different roles learners' adopt. The specific features of the LMS communication tools, if any, are not necessarily important matters of concern.

The way these two types of systems are thought of, their expected role or the expected impact of the software properties are thus very different. However, in both cases, it would be misleading to think that the software used defines the setting and defines what happens. Fundamentally, software is only a mediator (and just one mediator) of learners' activity, and learners' activity is something that will

[7]Jones, C., Asensio, M., & Goodyear, P. (2000). Networked learning in higher education: practitioners' perspectives. *ALT-J, The Association for Learning Technology Journal*, 8(2), 18–28.

contextually emerge and dynamically evolve in relation with many dimensions of which software is only one factor.

As an element of the setting, software has some impact, in any case ("neutral software" in an oxymoron). An ITS's and an LMS's expected roles and structuring impact are very different. However, whether the software has features that have been designed to structure the setting and learners' activity in one way or another does not guarantee this will happen. The setting may unfold very differently from what was expected. We will come back to this "impact" notion in Chap. 3.

1.6 Summary

Within an educational software design perspective, TEL may be defined as the scientific field addressing CBPSs and the software used for implementing these settings. This view conveys the following ideas:

1. What is considered is the use of a specific technology (software, i.e., programs to be run) as a means to contribute to the addressing of some pedagogical objectives and, as an overall objective, to enhance learning.
2. The educational considerations are defined with respect to CBPSs or types of CBPSs.
3. Within a CBPS, the software role may correspond to different realities, and may relate to a variety of pedagogical objectives.
4. When addressing the TEL field in general, the considered software may be educational software (i.e. software designed for educational purposes) or basic software (i.e., software not designed for educational purposes but used in educational settings).
5. Software may participate in the definition and structure of the setting to different extents, and in different ways.
6. Software is not to be thought of as prescriptive of how the setting will unfold.

At the very core of the educational software notion is the consideration of how to *design* software that presents functions or properties studied according to educational considerations.

2 Examples

2.1 Examples of Computer-Based Pedagogical Settings (CBPSs)

We introduce hereafter three different examples of CBPSs, based on different types of software and different ways of considering software: the *Java programming*, the *biology inquiry learning* and the *learning theories forum* CBPSs. These examples are only superficially sketched to introduce the concept; they will be reused later on. An effective description would require going into details and addressing many other issues (teaching setting, etc.).

2.1.1 The Java Programming CBPS

The *Java programming* CBPS is typical of a tutoring setting, i.e., a setting within which software analyzes learners' actions and provides tutoring feedback.

The addressed pedagogical objective is that learners should develop some knowledge related to object-oriented design and programming, using as a means the Java programming language.[8]

The setting is individual. Learners are asked to consider a task that consists of a set of exercises, each exercise corresponding to the creation of a Java program according to some given specification.

Within this context, software is thought of as a means to (1) allow learners to build and edit their solution and (2) support them by providing individualized hints.

2.1.2 The Biology Inquiry Learning CBPS

The *biology inquiry learning* CBPS is an inquiry learning setting, i.e., a setting based on "(. . .) an approach to learning that involves a process of exploring the natural or material world, and that leads to asking questions, making discoveries, and rigorously testing those discoveries in the search for new understanding".[9]

The addressed pedagogical objective is that learners should develop skills related to experimental methods and become able to engage themselves in processes within which they identify interesting questions, define different hypotheses, test these hypotheses and draw rational conclusions. For this purpose, a given application domain in biology is used.

The setting is collective: learners collaborate in front of the software (i.e., two learners use the same computer) and/or via the software (i.e., an individual or a group of two learners sharing a computer may collaborate with another group via communication tools and shared access to some of the software features). Learners are asked to consider a task that consists of building and describing a model (and underlying theoretical elements) to explain the way cells react when immerged in a given salty liquid. It is expected that working collaboratively helps them elaborate and discuss different hypotheses.

[8]*Object-oriented* is an approach to design and implementation of software within which focus is on defining *classes* (which allow creating *objects*) and the relationships between these classes (e.g., *inheritance*). *Java* is an object-oriented language.

[9]de Jong, T. (2006). Computer simulations – Technological advances in inquiry learning. *Science*, 312, 532–533.

Within this context, software is thought of as a means to (1) offer a computer-based simulation that simulates the cell behavior and allows testing of hypotheses, and (2) offer collaboration tools.

2.1.3 The Learning Theories Forum CBPS

The *learning theories forum* CBPS is a networked learning setting promoting connections between learners on the basis of an LMS.

The addressed pedagogical objective is that learners should develop some general understanding of a set of learning theories from the behaviorist and cognitivist paradigms, and practice argumentation skills.

The setting is collective. Learners are asked to consider a task that consists of developing a scientific discussion related to the differences and similarities of explanations these theories provide for a topic (e.g., “how learners learn” or “how skills developed in formal teaching may transfer to real-life problem-solving”). The task is broken down into different subtasks (e.g., gathering and structuring some data related to one or the other theory, identifying key concepts, identifying controversial issues or making explicit the underlying hypotheses) and learners are given different roles (e.g., an option is that each learner addresses all the subtasks for a given theory; another option is that each learner addresses the same subtask for all the theories). Learners are supposed to deliver a final common document.

Within this context, software (the LMS) is thought of as a means to (1) deliver the instructions (what learners are supposed to do) and some material (documents related to the theories) and (2) allow learners to upload their final document. Implicitly, it is supposed that the general means provided by the LMS (e.g., the communication tools) may also be of interest.

2.1.4 Discussion

These examples illustrate different issues (some of them already mentioned) that will be addressed in more details in the following chapters:

1. Software may be more or less central in the definition of the setting. For instance, in the case of the *Java programming* and *biology inquiry learning* settings, the software role is different but in both cases of major importance, while this is less the case for the *learning theories forum* setting.
2. Software may be introduced as a central feature of the pedagogical setting or as only a possible means. For instance, in the *learning theories forum* case, it is not imposed that the different tasks should be addressed via the LMS. The way the CBPS is defined implicitly acknowledges the fact that learners may achieve these tasks in a variety of ways and may use whatever means they find to be convenient.

3. Pedagogical objectives (and considerations) are usually multiple and mixed. For instance, in a setting such as the *biology inquiry learning* CBPS it is likely that meta-level dimensions and skills (related to inquiry processes) and domain-related ones (related to biology) coexist.

Finally, these examples illustrate the fact that, in some cases, both basic and educational software may be used. Learners may address the *learning theories forum* CBPS with basic communication software. Similarly, the *biology inquiry learning* CBPS may be implemented with a simulation dedicated to education or not. However, the contextual and individualized hints required for the *Java programming* setting requires specific software.

2.2 Examples of Educational Software

We sketch hereafter four theoretical examples, i.e. imagined systems that do not exist, but are inspired by the field literature: *JavIT* the ITS, *Bio-sim* the simulation-based learning environment, *GeLMS-1* the generic LMS (a counter-example) and *Argue-chat* the graphical argumentation chat tool. More detailed presentations (and other examples) are presented in Chap. 4. These examples will be used throughout this book to illustrate different ideas.[10]

2.2.1 *JavIT*, the ITS

JavIT is an ITS designed to implement the *Java programming* CBPS.

Consistently with the targeted setting, *JavIT*'s principle is to introduce programming exercises and to ask learners to enter a certain number of constructions (the different Java classes, their relationships, the way notions such as inheritance or polymorphism are used, etc.), which constitute different steps towards the solution they propose.

JavIT's specificity lies in (1) the interfaces it provides for editing the solving steps, which are defined to make learners consider some particular concepts in a precise way, make explicit their constructions and justify their propositions and (2) the diagnosis and feedback capabilities, i.e., the system's ability to analyze learners' output and provide individualized epistemic feedback.

JavIT presents both of the characteristics of educational software we have raised in Sect. 1: (1) its design is based on specific educational conceptualizations and models, and (2) it presents educational-related specific functions and properties.

[10]The purpose of using theoretical examples is to highlight issues contrasting differences in perspectives while avoiding describing the details of actual systems.

2.2.2 *Bio-sim*, the Simulation-Based Learning Environment

Bio-sim is a simulation-based learning environment designed to implement the *biology inquiry learning* CBPS.

Consistently with the targeted setting, *Bio-sim* offers a computer-based simulation of cell behavior. When using the simulation, learners can make different variables vary and observe what happens.

Bio-sim's main role is to provide a simulation that makes salient features identified as pertinent for the CBPS (e.g., cell nucleus, cell membrane or liquids) and hides or gives less importance to some others. When using the simulation to test hypotheses (e.g., what happens if the salt concentration is changed from value v_1 to value v_2), only some notions may be used as variables. Moreover, the simulation is complemented by additional data visualized via different indicators (e.g., electric charge) which, here again, are meant to make learners consider some particular notions in some particular way, using different representations (e.g., numerical or graphical). These affordances[11] have been defined on the basis of a specific study of learners' cognitive processes and difficulties in such settings. The environment also presents learners with specific editors that help them to model the phenomenon at stake or to relate hypotheses to experimental data. Different communication tools allow distant learners to collaborate. Finally, different methodological hints are provided in order to help them to conduct the inquiry process, e.g., to properly separate variables.

Here again, *Bio-sim* presents both of the characteristics of educational software: its design is based on specific educational conceptualizations and models; it offers functions presenting educational-related specific properties (e.g., the simulation affordances) and specific functions (e.g., the additional cognitive tools, the hypothesis management tools or the model editors).

2.2.3 *GeLMS-1*, the Generic LMS

GeLMS-1 is a generic LMS, i.e., a Web platform managing learners' access to pedagogical resources such as documents or videos and to basic communication tools such as chat rooms, forums, whiteboards or file exchange zones.

GeLMS-1 is, in fact, an information system used (after some cosmetic customization) in the context of education: it manages how users (in this case, learners) access resources (in this case, pedagogical material), and it offers basic communication tools.

GeLMS-1 does not meet any of the characteristics raised in Sect. 1: its design is not based on specific educational conceptualizations or models, and it does not present any educational-related specific functions or properties. The only aspect related to education is the nature of the documents and resources managed by the

[11]*Affordances*: here, aspects suggesting how (in this case) artifacts may be used.

system. However, this dimension, or the expected educational use of the communication tools, does not have any influence on the design. *GeLMS-1* is basic software (see discussion).

2.2.4 *Argue-chat*, the Graphical Argumentation Chat Tool

Argue-chat is a graphical argumentation chat tool designed to support learners in conducting argumentative interactions.

Argue-chat is not dedicated to the implementation of a precise CBPS but, rather, to CBPSs within which whether learners develop argumentation is of importance. Its rationale is that, in mediated collaborative learning settings, it is a classical strategy to make learners learn one from another (and/or make them develop interactions on the basis of which teachers will build) by making them argue about the domain or task at hand. The *learning theories forum* is an example of such a setting although, given the way it is defined, the argumentation dimension is not put to the fore.

Argue-chat's role is to address different difficulties that have been identified when learners use basic chat tools to argue, which often lead them not to develop very productive interactions. For instance, learners often build sentences whose intention is unclear, use complex sentences mixing different ideas and arguments, or have difficulties in relating ideas and arguments to one another. For this purpose, *Argue-chat* provides a set of graphical shapes (boxes and connections) denoting specific argumentation notions such as *statement*, *argument*, *counter-argument*, *approval*, *question* or *answer*, and allows their connection.

Argue-chat meets the first characteristic: its design is based on specific educational conceptualizations and models. Whether it meets the second one may be argued (see next section).

2.2.5 Discussion

Bio-sim illustrates that CBPSs may be implemented with both basic and educational software. In this case, the rationale for developing a specific simulation and environment is to support learners by providing cognitively-studied affordances and tools. *Bio-sim* also illustrates the variety of concerns that may be considered: whether the environment may be built (i.e., the conceptual and technical issues addressed); whether it supports learners in an effective way; the impact on learning outcomes; whether it fits teaching settings or not or economical considerations; etc. All these different dimensions may be regarded as such and in relation to one another (the other examples could be used to illustrate this point in the same way).

GeLMS-1 highlights the fact that presenting software as "educational" requires emphasizing what educational dimensions are taken into account, how this is done, and what the results are. In this case, the system is used in education (and might be perfectly satisfactory for some settings), but no educational considerations impacted design.

We have introduced *GeLMS-1* as an information system incidentally used in an education context. Now, if such a system were originally constructed for education purposes, should it be considered as educational software? It may be argued that the first characteristic (design is based on specific educational conceptualizations or models) is met, as designing a platform such as *GeLMS-1* for educational purposes is not free of educational perspective, but, on the contrary, denotes a particular (and very poor) perspective of on-line teaching: e-learning is e-commerce whose product is data (text, video, etc.) considered to be pedagogical. Moreover, it may be argued that the second characteristic (software presents educational-related specific functions or properties) is also met as effective systems of this kind usually present specific functions related to the back-office dimension, e.g., managing learners' registration or marks. Such "educational" considerations are qualitatively different from the ones put to the fore in the *JavIT* or *Bio-sim* cases.

The educational software definition we have introduced is not meant to contrast effective and ineffective systems, or what may be "legitimately" considered as educational software, but design considerations. As a matter of fact, most existing LMSs can hardly be considered as educational software as they provide very few properties dedicated to educational issues (however, this is not intrinsic to the notion of LMS, and counter-examples do exist, see Chap. 4). Anyway, LMSs, because of their generic character and the fact they consider back-office dimensions more than educational considerations, are not very good examples for exploring educational software issues.

Argue-chat highlights the proximity that may exist between educational software and, more generally, software specifically designed to influence users' processes.

Finally, we have introduced *Argue-chat* as educational software because its specificities have been studied and designed according to difficulties that have been identified when learners use basic chat tools to argue. *Argue-chat* is explicitly designed to help learners develop productive interactions. However, addressing the argumentation difficulty may be useful in other contexts (e.g., coordination in collaborative work) and for other non-pedagogical objectives (e.g., collective problem-solving efficiency or collective-design-rationale documentation). If *Argue-chat* had been designed within such a context, this would not change its interest for education.

This example highlights that it would be very unproductive and misleading to separate efforts and work related to educational software from other efforts related to, for instance, Computer-Supported Collaborative Work and interaction (if considering *Argue-chat*-like systems) or, as another example, problem-solving (if considering ITS-like systems). More generally, designing software to influence learners' activity or knowledge is a particular case of designing software to influence users' processes, and shares the more general objective of understanding what *designing software to support human activity* means and how it may be addressed (see Chap. 6, Sect. 3.2). For instance, educational software greatly benefits from work studying how to design software to support interactions (and how such software is used) because such work addresses the relation software/(interaction)-objectives in an explicit way (and, of course, because interaction is closely related to learning).

3 Design of Educational Software: Different Realities

Designing software with respect to educational concerns may correspond to very different realities. We contrast and exemplify here two prototypical cases, which illustrate different prototypical education/CS interplays, and discuss them in Sect. 3.3:

1. Designing and implementing new software, i.e., defining the software specifications from the analysis of the pedagogical setting and implementing them. Technically, this implementation may be addressed in various ways such as programming from scratch or interoperating preexisting software components.
2. Articulating and/or customizing already existing software components to educational needs.

Other options are possible, e.g., instantiating generic software (e.g., an ITS framework) or a basic high-level framework. For instance, a system such as *Argue-chat* may be implemented by instantiating a generic graph-tool.

3.1 Designing and Implementing New Software

Bio-sim is a prototypical case of educational software that is likely to be designed as new software, i.e.: specification of the software is established from the targeted CBPSs (and, possibly, teaching setting) analysis, implemented, tested and iteratively refined. More generally, if one considers complex systems such as ITSs or simulations, building new software is prototypical.

At the basis of design are a given learning theory and an identified set of issues to be addressed. In *Bio-sim*'s case, the rationale[12] is that inquiry learning appears to be a promising approach to make learners learn how to regulate their own learning, gain new knowledge and update their existing knowledge. However, the positive impact of inquiry learning settings may be hindered by the difficulties learners often encounter when facing inquiry processes such as choosing the right variables to work with or implementing the experimental processes. The problem to be studied is less to allow learners to solve the task than to lead them to do so in a knowledge-productive way. The targeted system is meant to play an important and precise role, providing specific affordances and cognitive support. The exact software properties

[12]See de Jong, T. (2006). Computer simulations – Technological advances in inquiry learning. *Science*, 312, 532–533.

are important issues, and the requested properties are defined from the pedagogical needs.

It may be noticed that such software is “new” in the sense that, fundamentally, educational specifications are identified, and software is constructed so as to match these specifications. However, from a technical point of view, *Bio-sim* may be implemented from scratch or by reusing and adapting different existing software components (a simulation, a generic editor, etc.). Pushing forward this idea, *Bio-sim* may be just one of a set of inquiry learning environments to be constructed and, from a technical point of view, the strategy may consist of building technical frameworks (pieces of software dedicated to a general set of related features) from which different variations may be constructed according to the precise specifications of one or another system.

3.2 Articulating and/or Customizing Software Components

ICT has considerably changed the TEL field and, if one considers systems used in effective settings, educational software is often a particular selection, adaptation and/or inter-connection of already existing ICT-based software components. Typically, within on-line learning centers, so called “learning technologists” are in charge of understanding, with the involved teachers, what already existing technology may be adapted to their needs, and customize it as necessary.

Let us consider the *learning theories forum* CBPS. A basic LMS offers little support for learners engaged in such a setting. An option may be to build a specific environment from scratch, but this is rather expensive. A more rational process is to study how some of the technical possibilities of the LMS, together with additional software components, may be locally aggregated, customized and interoperated so as to build a kind of task-related activity framework (let us call it *Colab-edit*). Such an environment may for instance be created by adapting and gluing together an editor, a versioning system and a forum as a way to offer learners a collaborative editor that allows them to edit text, track other learners’ modifications, submit and comment on work-in-progress reports and launch mediated discussions associated with particular pieces of text. Such an approach may involve both basic tools and specific tools such as *Argue-chat*.

A “mash-up” such as *Colab-edit* is an example of educational software built by selecting and adapting existing components and articulating them in a particular way: it is software that has been designed with respect to educational purposes, and which is meant to influence learners’ processes. A different but related example of this type of process is the way a basic information system (named *GeLMS-1* for commercial reasons) may be re-engineered into *GeLMS-2*, technically a new version but, now, a system that takes into account educational considerations and addresses drawbacks highlighted by the practical use of the previous version.

3.3 Education and CS Interplays

Let us consider in more detail the education and CS interplay in the *Bio-sim* and *Colab-edit* cases:

- In the *Bio-sim* case, the educational software is imagined and created from the pedagogic idea. *Bio-sim* is designed as new software. Technically, it may be implemented upon existing software, but the existence of the software used to build *Bio-sim* is not a matter of concern for educationalists.
- In the *Colab-edit* case, the educational software is imagined and created from (1) the pedagogic idea and (2) the existing technology or, more precisely, the way educationalists and computer scientists interpret the existing software affordances with respect to the pedagogical needs. The existence of already existing software is at the basis of the design process (and, also, of the implementation process).

Colab-edit exemplifies the fact that building educational software by articulation and/or adaptation of components can be seen as a movement from two existing constructions (the initial pedagogical idea and the existing software or components) to a third (the designed system). Such a movement is submitted to multiple different influences such as the individual educationalists' and computer scientists' views, their mutual understanding, the common conceptual constructions they may build together (or fail to build), their perception of the existing technologies, the general teaching context, the general technological context (e.g., the technology availability and scalability or economical constraints) or the project's lifespan. The process is usually not linear but iterative, ideas and software being iteratively tested and improved.

However, in fact, such a process occurs in all cases, including when building software from scratch. It is difficult to imagine educationalists (and computer scientists of course) imagining and designing software without any influence from already known systems or, at the least, a certain perception of technology and of what can and cannot be done.

It may also be noticed that a technological framework designed to fit some given pedagogy, once deployed, often leads one to find some of its aspects or assumptions were not noticed, and requires reconsideration of the design.

Therefore, as a general principle, highest importance should be given to the educational dimensions and not to the technical dimensions, and the start point of any project involving the design of educational software should be the pedagogical setting (which is itself to be considered within the more general teaching setting), and not the technical system. Technology should adapt to pedagogical needs and not the contrary. However, what happens is usually a bit more complicated and balanced. In all cases (inventing new software on the basis of a pedagogical idea or a technical opportunity, adapting existing software, gluing together existing components, building CBPSs upon the limited means offered by a platform such as *GeLMS-1*, etc.), there is not a pedagogical idea on the one side and CS work on the other, but co-constructions, adaptations, compromises and/or taken opportunities.

Designing software with respect to educational concerns may thus correspond to very different realities, with significantly different education and CS interplays. This is part of the reason why, if general methodological considerations may be highlighted (CBPSs and teaching levels must be considered; CBPSs are socio-technical contexts; technology is not prescriptive of what will happen; individual characteristics of learners, but also groups' phenomena, institutional or sociological dimensions are of major influence; iterative or participative approaches are useful; effective use of systems is not necessarily related to their properties and impacting effective practices requires considering other factors; etc.), listing precise guidelines or managing projects requires going into specifics of the setting.

4 Addressing Educational Software Design

4.1 Considering Software Properties

It has been raised in Sect. 1 that how learners use software is subject to many influences and that the software does not define what will happen. This may lead one to consider that whatever the software properties are, their effects may be rendered anecdotic by other dimensions, and thus they may be considered contingent.

We argue that addressing the level of software properties is necessary and of interest as these properties have an influence on what is going to happen or not, and how.

First, whatever software is, software properties do impact users' behavior. Software is in no way neutral. Software properties (whatever their design rationale is) define part of the socio-technical system with which users or, in this case, learners, are engaged. They play a role in the development of users' perception and representations of the task and/or of how software may be used as a means, and of course in what actions are possible. Studying how software properties influence or not (and how, when, why, etc.) the unfolding of pedagogical settings is necessary to, at the minimum, avoid raising pedagogically unproductive representations or constraints.

Second, in formal learning settings, the fact that learners are asked to use a given system to achieve a given task in the context of a given institutional setting creates a very specific context. This is the case for still rather open settings such as the *learning theories forum* and, more importantly, for more constrained settings such as the *biology inquiry learning* CBPS implemented with *Bio-sim* or tutoring contexts implemented with *JavIT*. In such cases, as reported in the literature, learners' behavioral and cognitive processes are influenced by software properties.

It makes sense to consider the objective of influencing by the software design what is going to happen.

However, software influence is to be considered in the light of the different points raised in Sect. 1, and with respect to precise contexts. For instance, the objective of influencing by the software design what is going to happen corresponds to very different realities in the *learning theories forum* and the *Java programming* CBPS implemented with *JavIT*. This gives particular importance to the clarification of the setting, the software role, and what the expectations are.

4.2 Making Explicit Matters of Concern and Perspectives

Considering educational software design requires drawing relations between pedagogical considerations, software properties, software usage and the outcomes of this usage. This is the case in both development and research projects studying how to base design on theoretical elements or attempting to capitalize knowledge. With respect to design, a central issue is that of thinking, problematizing and making explicit the relationships between (1) the discourse that is used to denote the considered pedagogical objectives and settings, and the underlying assumptions, and (2) the design and implementation decisions, and the corresponding software properties.

In the preceding sections, we have introduced the *Bio-sim*, *JavIT*, *Argue-chat* and *Colab-edit* examples by making salient (1) the description of the considered CBPSs and (2) the description of the systems' features. Such a presentation allows the general idea to be grasped, but drawing precise relations between pedagogical considerations, software properties and usage requires both broadening the scope and going into other details.

As examples, we highlight hereafter a few dimensions whose clarification is required.[13]

4.2.1 The Way the Pedagogical Setting and Software Role Are Thought of

An important dimension underlying educational software design projects is whether the targeted computer-based setting is expected to present additional value in terms of learning or it is thought of as an alternative to a non-computer-based setting. For instance, a system (and the underlying constructions) such as *Bio-sim* must be considered and evaluated very differently according to whether it is designed to produce better learning outcomes than inquiry learning not based on simulations, as an alternative to in-classroom settings (the notion of "better learning outcomes" is not central anymore), or as an additional means to be used with some others

[13]The examples of analysis axes introduced here are part of the list developed in Chap. 7.

(which raises concerns such as the coherence of these different means). Similarly, *Colab-edit* will be considered very differently if it is thought of as a response to a limitation (the fact distant learners cannot attend classrooms) or as a means for a different pedagogy. Such considerations may impact analysis and design decisions and, if not clarified, may be at the origin of misunderstandings and confusions between the project actors.

As a second example, another important dimension is whether software is thought of as something that will impact the way learners conceptualize the domain or it is thought of as merely a resource. For instance, the importance of the fact that learners use *JavIT* editors, and the importance of the way they use them, is very different depending on whether these editors are supposed to impact the way learners will conceptualize object-oriented notions such as inheritance or polymorphism (i.e., act as cognitive tools) or not.

4.2.2 The Considerations That Have Been Taken into Account at Design Time

An important dimension underlying educational software design is whether the project is based on a reference to a non-computer-based setting or not. If the analysis is conducted in reference to such a setting, what is the level of granularity of this reference? Which objects are considered? For instance, an important dimension for analyzing *Bio-sim* is whether it is meant to mimic authentic inquiry (and to what extent, for what notions, etc.) or not.

As a second example, another important dimension is whether the learning domain specificities (notions, objects, assumptions) are taken in account in the analysis and, in this case, how. For instance, some biological knowledge is necessary for designing *Bio-sim* simulation, but the way the system scaffolds learners' processes may have been addressed in reference to generic inquiry principles or in a way that mixes generic principles and domain issues. Addressing *JavIT* feedback on the basis of generic principles or on the basis of a domain-specific analysis of how learners may develop knowledge related to object-oriented design leads to very different issues. Etc.

As a third example, design may or may not be based on, or use, theoretical bases. For instance, *JavIT* feedback properties may have foundations in some given cognitive theory; *Bio-sim* simulation may have foundations in theories relative to knowledge representation, cognitive load or how skills may transfer from one domain to another; *Argue-chat* may have foundations in interaction theories; etc.

As a final example, another important dimension is how the different considered features (theoretical context, domain specificities, etc.) impact the analysis. Do they provide a general way of thinking? Do they provide guidelines? Do they provide concepts? For instance, *JavIT* design may be influenced by theories explaining how learners learn by solving problems in different ways, from very general concerns (e.g., the importance of learning-by-doing) to very precise ones (e.g., the fact *JavIT* should consider learners' zone of proximal development[14]).

4.2.3 The Impact of Pedagogical Considerations on Design

An important dimension is to clarify which of the objects considered at the level of the pedagogical setting are taken into account in the software design. For instance, *Colab-edit* rationale is related to the fact that learners may learn from one another by arguing and co-constructing knowledge. Considering this objective, part of the issues related to making learners *argue* and *co-construct knowledge* may be addressed by the structure of the CBPS (e.g., grouping learners whose background is different as a way to increase the chances they have different views and, thus, have to argue) and another part taken into account in the software design, for instance offering a collaborative editor that allows versioning pieces of text. In other words, design is only impacted by some of the pedagogical considerations, and clarifying these is of crucial importance.

As a second example, another important dimension is whether the considered objects impact design only and/or they also are reified in one way or another in the system. For instance, considering *JavIT*, the fact that learners are expected to make explicit their different problem-solving steps may impact the way the system introduces the exercise (the instructions) and/or the design of the editor which is to be used by learners to edit their solution (the editor may for instance impose the use of constructions such as "I create class *C* because . . ." or "I use inheritance as a way to . . ."). Going a step further, the notion of "explicit solution" may also be reified in the system as a specific software component capable of evaluating the extent to which a solution has been made explicit and is properly justified.

As a third example, another important dimension is how features or properties thought of as dedicated to educational dimensions relate to the pedagogical objectives and constraints. For instance, which *Bio-sim* properties are meant to provide specific affordances for learning (and which others are just introduced for the environment to be usable)? How does the use of *Argue-chat* and properties such as offering *sentence openers*, considering *balanced interactions* or imposing *turn-taking rules* relate to the CBPS it is used within? etc.

As a final example, another dimension is the precise processes of the system (data acquisition, data analysis, accessibility management, etc.) and how they relate to the pedagogical objectives. For instance, when *Bio-sim* provides learners with hints, does it analyze learners' hypotheses in terms of biological knowledge or does it only react to general features such as the number of variables involved? If

[14]The *zone of proximal development* is "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance, or in collaboration with more capable peers". Vygotsky, L.S. (1978). Mind and society: The development of higher psychological processes. Cambridge: Harvard University Press.

building a system enhancing *Argue-chat* by a component that analyzes learners' interactions, what type of process is implemented? Are sentences' semantics analyzed using a Natural Language analyzer? Are they categorized according to the sentence openers used by learners (independently from the effective text, which may not correspond)? Does the system just sum up the number of sentences?

These different examples highlight that the description of the considered CBPSs and of the system features is far from capturing all the different dimensions underlying design of educational software. As a consequence, if analysis is limited to these considerations, many misunderstandings and confusions may develop between actors from different disciplines and/or with different matters of concern.

4.3 Importance of Explicitness

Making explicit the details of work related to the design and implementation of educational software is important for different reasons, in particular:

1. It increases the chances of the actors engaged in design and analysis to develop a shared understanding. This is an issue of particular importance and difficulty in TEL due to the intrinsic complexity and multidisciplinarity of the field, and to the variety of possible matters of concern, perspectives or approaches.
2. It increases the chances to benefit from and/or reutilize constructions (e.g., models, processes, software components or lessons learned) from other projects, and the chances to get the constructions elaborated in one's projects to be reusable for other projects, by other persons. In a less positive wording, it helps in avoiding wasting time due to the erroneous idea that some constructions are innovative and avoiding replicating mistakes already made with the previous wave of technology, which is indeed a recurrent pattern of TEL history.
3. It provides a basis for the definition, the evaluation, the criticism and the dissemination of scientific results.

Making explicit design details is a *sine qua non* condition for knowledge capitalization, which may take different forms: understanding of an issue; statement or lesson learned; model (for thinking and analysis, for prediction, for run-time control, etc.); process (general approach, engineering or re-engineering process, benchmark); conceptualization (i.e., a set of concepts proposed as a substratum for some given work); software component; etc. (see Chap. 7).

As an example, one of *Bio-sim*'s specific designed features is to support learners in making connections between the simulation, the data and learners' hypotheses. From such a project may be capitalized lessons learned related to the impact, influence, usage (etc.) of such support, and of how learners may develop and manage hypotheses in such a setting; lessons learned related to the issues raised within such a project; the first steps of an approach or an engineering process related to inquiry learning environment design or to cognitive scaffolding, to be tested in other projects; software components as construction bases for inquiry learning settings; design patterns; etc. Considering *Argue-chat*, which imposes the use of a certain number of graphical constructions related to argumentation, similar knowledge may be capitalized, and *Argue-chat* features (issues, impact, usage) may be compared to those of other approaches to interaction structuring (e.g., using sentence openers), etc.

As will be emphasized in Chap. 6, CBPS and educational software are artificial objects, and knowledge develops via the design and analysis of systems: advances are derived via engineering projects involving, within a larger pedagogical study, the design and implementation of software. As a consequence, both research and development projects may be productive of knowledge. In research-oriented projects, the objective is usually not to build some given software to be used in some effective setting, but to identify and understand issues to be considered, phenomena to be understood, or possible approaches. The constructed software is both the resource and the objective of scientific work. In development projects, whose objective is to build systems to be used in effective settings, knowledge or lessons learned may be elaborated from the analysis of design decisions and their impact. In fact, as discussed in Chap. 6, research and engineering dimensions are to a large extent intertwined.

4.4 Difficulty and Limits of Explicitness

The way the examples of CBPSs and educational software have been introduced in Sect. 2 is typical: researchers or engineers engaged in the design of educational software usually present their projects by focusing on the way they imagine the setting and the innovative features they attempt to introduce, which is related to their perception of the field, the spectrum of the issues they consider, etc.

However, this conceptual context is often largely idiosyncratic, and difficult to share.

The *raison d'être* of this book is the fact that there is an intrinsic difficulty in making explicit (in detail) work related to the design and implementation of educational software. This book introduces notions, analyzes, examples, characterization items and methodological considerations as means to support such a process.

In this book, we refer to "clarifying" or "making explicit" (design) matters of concern as the efforts made by an actor or a set of actors to render these features intelligible by some other actors (and, by the way, improve their own analysis and understanding).

"Explicitation" is a goal that, however, raises fundamental problems. At a theoretical level, the notion of explicitation leads to considering notions such as accountability (see ethno-methodology studies) or shared understanding.

In particular, considering shared understanding, intelligibility is not only related to language issues (i.e., sharing a common interpretation of language items), but also to many other material, social, historical or cultural dimensions. The documentary artifacts that may be considered by an actor (or a set of actors) as an explicitation of his/her concerns are the result of an encoding by a sender, and will in turn be decoded by a receiver. These encoding and decoding processes are of course context-dependent, and related to many dimensions. In other words, what has been considered as comprehensive and unambiguous by a given actor in a given context may be understood differently by another actor who has a different history, a different culture or a different view of the field and what the important notions or matters of concern of this field are, and is presently facing some given tasks within some given context.

When arguing for the interest of explicitation and attempting to support such processes, these fundamental issues are not to be ignored. They do not mean efforts for explicitation are meaningless, but that they must be considered with these difficulties and intrinsic limitations in mind.

In particular, highlighting and analyzing issues, approaches or characteristics only makes sense, and is only intelligible, if the considered notions are properly defined. In other words, it is not possible to say anything more or less precise, consistent and, above all, shareable, if there has not been proposed some prior clarification of the conceptualization that underlies the proposed constructions.

A conceptualization is a differential system of notions. A given conceptualization emphasizes some aspects of a field (some notions, some dimensions, some properties, etc.) and not others. Because a conceptualization is never neutral, the underlying perspective on the domain must be clarified: the fact that it is useful to denote a given notion or to dissociate two notions that appear similar, or the reasons why a given definition draws attention to particular aspects, only makes sense within a general view of the considered domain.

When addressing complex domains, using differently-oriented conceptualizations allows precision, when attempting to address different issues within a single common view (a kind of "all-in-one conceptualization") often leads one to be rather abstract and general. TEL poses different types of issues and thus requires different views, drawing attention to different dimensions and providing different conceptual means.

In this book, the focus is on the relationships between the discourse that is used to denote pedagogical issues (settings, considerations, objectives, etc.) on the one side and, on the other side, the elaborated software. This is addressed within a software-design-oriented conceptualization that has been sketched in this introductory chapter and is described in detail in Chap. 2. This conceptualization is to be seen as a tool dedicated to conducting multidisciplinary work related to educational software design in TEL.

As a way to emphasize that a given conceptualization draws attention to some dimensions and not to others, let us consider notions such as *learning* or *educational practices*. When focusing on educational software design, these dimensions are not put to the fore. This is not to say learning is not important or that design issues are disentangled from educational practices. Of course, for any TEL project, at a general level, the issue is to address the fact that some targeted CBPSs will allow a better (or a different type of) learning, and will make sense with respect to teachers' practices and institutions. However, if one considers these objectives as such, different analyses on different plans are required. For instance, considering whether a pedagogical setting allows *better learning*, or a *different type of learning*, requires means to describe and analyze TEL settings in terms of learning and properties of learning (e.g., to define what is meant by "a better learning"). In other words, this requires a conceptualization of the field that allows these issues to be addressed. Such a conceptualization may, however, make it hard to grasp some other dimensions, for instance, *the way the pedagogical intention is reified within the system* or *the way the computer-based artifact properties impact learners' activity*, which are not first-class matters of concern and may be completely contingent when addressing the project within a learning-focused view. On the contrary, these dimensions are at the very core of an analysis considering CS-oriented design issues, and the conceptualization underlying such an analysis must allow these dimensions to be grasped. Viewing the domain from the point of view of teachers' practices is again a different view, and leads to the consideration of other notions (and/or addressing notions in a different way).

Coming back to the need for explicitation efforts, from a long perspective, putting into evidence the pragmatic interest of explicitation, and proposing constructions (such as conceptualizations or lists of characteristics) as examples of explicitation tools, may lead communities (researchers or communities of practice) to collaboratively develop and adopt shared conceptual tools.

As a pragmatic feature, efforts at explicitation may finally be related to the following point: designers may associate some logos to the designed software or may not ("technology" in the etymological sense) but, in any case, other actors and, in particular, users, will do. Associating some logos to the designed artifact from the start could be seen as a means to "impose" the designers' view but, anyway, users will take possession of the artifact according to their purposes and needs. Rather, this may help in tracing and understanding the way, in some sense, "design is continued in usage".

5 Content and Structure of the Book

5.1 Objective

TEL projects considering the design and implementation of educational software are intrinsically multidisciplinary and complex. As such, they present the difficulty of conducting detailed analyses allowing the different involved actors to understand respective matters of concern and build common constructions. Another difficulty, related to the previous one, is to benefit from lessons learned in the literature, and to contribute to this capitalization of knowledge.

Within this context, the general objective of this book is to highlight the importance of making explicit the details of work related to the design and implementation of educational software, and to propose a substratum to do so.

More precisely, this book concentrates on the very core difficulty of providing a framework to think about and make explicit the relationships between (1) the discourse that is used to denote the considered pedagogical objectives and settings, and the underlying assumptions, and (2) the elaborated models and software components.

5.2 Content Synthesis

The overall content[15] of this book is:

1. A highlight of the fact that what is referred to by the "design of educational software" may be subject to very different perspectives, and the importance of making explicit matters of concern.
2. A general conceptualization that helps in disentangling issues and clarifying matters of concern with respect to educational software design and implementation.
3. A set of items that may be used to characterize (1) the way the pedagogical setting is considered and (2) the software properties and construction processes.
4. A review of some methodological dimensions.
5. A perspective on the field anchored in an engineering approach, and propositions on how to push the field forward.
6. Different examples used to raise issues and illustrate propositions.

Although this book addresses the field in a transversal way and does not describe a particular methodology, its content (the description of issues, concepts, approaches, examples, methodological considerations) has heuristic value for conducting projects.

5.3 Rationale for the Organization

This book presents two dimensions: analyzing issues and proposing means. These dimensions are intertwined (as opposed to an analysis and then proposals), the general structure being:

- Chapter 1. Introduction
- Chapter 2. A general conceptualization for educational software
- Chapter 3. Understanding differences in perspectives
- Chapter 4. Review of prototypical examples

[15]We will come back to this list of contributions and its rationale in Chap. 6, in the context of the analysis of educational software engineering developed in that chapter.

- Chapter 5. CS perspectives and TEL
- Chapter 6. Educational software engineering
- Chapter 7. Characterizing the design context and the software artifact
- Chapter 8. Methodological considerations
- Chapter 9. Conclusions

5.3.1 Analysis Dimension

Analyzing perspectives within which educational software notions and issues may be considered is specifically addressed by Chap. 3, which provides different examples of how a given project and some important notions may be thought of and managed. In Chap. 4, several examples of systems are studied, highlighting differences with respect to these notions in particular. Chapter 5 also contributes by analyzing the different ways in which CS contributes to the TEL field, and the different ways in which computer scientists may conceptualize their involvement.

This analysis dimension is also more or less indirectly addressed in all the other chapters, as for example in the preceding sections of this introductory chapter and in the concluding chapter, which analyzes some issues for TEL development.

5.3.2 Propositions Dimension

Contributions consist of means for thinking, problematizing and describing educational software design and implementation work and outcomes.

The first proposition is a software-design-oriented conceptualization of the TEL field. This dimension is specifically addressed by Chap. 2, the general perspective (and preliminary definitions) having been introduced in this introductory chapter, and being completed by all the other chapters.

The second proposition is a list of items for characterizing, within a given project, the way the pedagogical setting is considered, and the software properties. This dimension is specifically addressed by Chap. 7. Listing such characteristics is useful because it ensures that specific attention is paid to a certain number of important dimensions. This helps to clarify projects in several ways. First, the listed characteristics that appear pertinent given the considered project can be used as an analysis grid: the list provides a resource for describing projects with respect to an external reference. Second, checking whether the proposed characteristics are pertinent and whether or not they apply leads to identification of variants or other characteristics that best denote some dimensions of the project, which here again helps in clarifying projects. The proposed list is based on the general conceptualization introduced in Chap. 2 and the CS and engineering dimensions analyses developed in Chaps. 5 and 6.

The third proposition is a review of methodological dimensions, consideration of which is of particular importance. This dimension is specifically addressed by Chap. 8, and is present in some way or another in most of the others.

The fourth proposition is an argument for addressing TEL as an engineering field, which defines the rationale for making specific efforts in making explicit matters of concern and elaborating multiple descriptions of projects. This dimension has been sketched in this introductory chapter, and is specifically addressed by Chap. 7. This is completed by a perspective on how to push forward the educational software engineering field (Chap. 9).

5.4 General Comments

5.4.1 Keeping Context in Mind

Writing a text addressing a complex multidisciplinary domain for a multidisciplinary audience opens many opportunities for misunderstandings. This is particularly the case when the entry point is software, the domain is education, and the text is long. Sentences or paragraphs picked up in the text may be misunderstood if considered out of their context. As restating this context in every section would be difficult, it is assumed the reader keeps this context in mind.

As an example, we use the notion of *impact of software properties* to denote the fact that software properties have an influence on what happens or does not. This is to be understood in the context of other considerations introduced before such as the facts that software is not definitional, software is only a mediator of learners' activity, and learners' activity is something that contextually emerges and evolves in relation to many dimensions of which software is just one factor. Not keeping this in mind may result in over-interpreting or misunderstanding the "impact" notion.

As another important dimension to be kept in mind, we have emphasized that a conceptualization is never neutral. In a field such as TEL, a conceptualization emphasizes notions that are of importance given an objective. The proposed conceptualization is related to the adopted orientation, that of the design of CS artifacts (software design; not to be confused with a technical point of view, e.g., addressing languages or platform architectures, or with a techno-centered approach). This is only one of the different possible perspectives that can be developed, and must be developed, when considering the TEL field.

5.4.2 What This Book Is Not

First, this book does not attempt to propose a unified view of notions, theories or any other specific constructions related to TEL. For instance, it does not present a list and a particular definition of notions frequently used in TEL such as *feedback*, *learner model*, *cognitive conflict* or *interaction analysis*. Similarly, it does not propose a unified process for building or evaluating educational software.

The reason for this is that TEL projects may be of very different natures, consider very different types of objectives and/or address them at different levels of granularity, use different theoretical or empirical foundations, etc. It is within a given project that notions such as *feedback*, *learner model*, *cognitive conflict* or *interaction analysis* will find a particular precise definition, will correspond to precise challenges, will be subject to particular attentions and treatments, and will lead to particular processes or methodologies. Attempting to build a kind of unification is useless and meaningless. It may be done, but at a level of granularity and abstraction that will render it of little interest for effective use. Moreover, this could easily fall into an attempt at normalization, which has indeed no sense, for ethical and, if nothing else, practical reasons given the heterogeneity of perspectives, settings and systems.

Second, this book does not attempt to explain how to build some given software or component or how to conduct research, although its content is useful for such tasks.

Here again, given the diversity of TEL projects, addressing such issues "in general" would lead to listing trivial generalities or, here again, to a kind of normalizing view. How to conduct TEL projects and build educational software cannot be reduced to a set of recipes. Every project (which may involve design of one or several systems) requires a detailed analysis in order to understand matters of concern, to adapt to context and to build on lessons learned from the literature.

In this book we address a conceptual level in the sense that emphasis is on proposing means for thinking and problematizing, which may be used for conducting different types of work. This may be inspiring for inducing a methodology or a set of guidelines for a given project, but this dimension is not addressed *per se*. Other books address contextually defined issues such as user-centered software engineering techniques, research methodologies, evaluation procedures or how a given type of problem may be addressed (e.g., building a specific type of system such as ITS).