# Chapter 6
# Educational Software Engineering

**Abstract** This chapter clarifies the concept of engineering as a vector for addressing (and elaborating knowledge related to) complex artifacts design, and how this may be applied to educational software. In the conclusion, the relationships between TEL and CS research are questioned again, in the light of this analysis.

We have mentioned in Chap. 1 that capitalizing knowledge related to educational software design issues may be powerfully addressed via the design and analysis of systems if, however, considerations, issues, hypotheses, alternatives, means (etc.) are made explicit. In this chapter, we come back to this point and study it in more detail in the light of the *Sciences of the Artificial* way of considering engineering. We first recall the notion of engineering (Sect. 1), and define educational software engineering within this perspective (Sect. 2). Analyzing the implications of such a view will allow us to make more precise the rationale for the issues and contributions considered in this book. In Sect. 3 we reconsider the CS-TEL relation (first addressed in Chap. 5) with respect to this engineering approach to the field.

## 1 Engineering and Research

Engineering is generally referred to as what is required for designing and implementing artifacts that meet a desired objective. As a consequence, engineering is often contrasted with research: research considers the understanding of phenomena, while engineering considers the use of this understanding and the organizing of know-how in order to tackle development projects.

As an example of the research/engineering dichotomy in TEL, part of research in educational psychology considers the elaboration of knowledge related to human learning. It considers an external problem (how subjects learn) and attempts to provide answers. It is clearly different from "engineering" in the sense of "elaborating solutions", e.g., elaborating a solution to the problem of teaching a given topic to a given set of learners in a given setting. In such a context, engineering consists of searching in knowledge elaborated by science for solutions to the considered problem (or, rather, elements of solutions that can be pragmatically articulated).

Work in line with Simon's perspective[1] have, however, popularized the notion of Sciences of the Artificial. Within this perspective, designing complex artificial objects cannot be considered as the simple application of knowledge defined elsewhere. While natural sciences consider things as they are, within the Sciences of the Artificial one considers things as they may be, possible objectives and means to address them. In other words, while basic sciences are driven by the objective of understanding things that already exist (e.g., understanding how people learn), design-oriented sciences are driven by the objective of solving problems through building artifacts (e.g., software). Researchers and practitioners concerned with such areas consider as scientific objects, and within scientific processes, both the objects they build and the design processes they elaborate to build these objects.

Designing complex artifacts that meet their requirements cannot be achieved by applying a collection of lessons learned: it is not intuitive and it is not easy. It not only requires theoretical work and knowledge elaborated in "academic sciences", but also proper knowledge. Such a view thus promotes transdisciplinary approaches and fruitful synergies of theoretical and technical work.

## 2 Educational Software Engineering as a Scientific Field

### *2.1 Educational Software as Complex Artificial Objects*

CBPSs and educational software are artificial objects. They are created by humans, for humans: they are not "natural". They create settings that do not exist prior to the creation of the artifact. They are contingent on the purposes and context for which they have been created.

CBPSs and educational software are complex due, in particular, to their intrinsic nature: they are socio-technical objects. Design of educational software cannot be thought of without considering a wider context including the CBPS and teaching setting levels. It requires cycles consistently articulating *a priori* analyses (pedagogical dimensions, epistemological dimensions, expected usage, expected impact, etc.), technical implementation, usage analysis and evaluation.

Problems related to educational software design are thus not limited to engineering issues in the traditional sense, i.e., building a solution from preexisting knowledge from one or another discipline. Learning, cognitive, teaching or software engineering theories only provide more or less relevant and useful perspectives, analysis

[1]Simon, H. (1969 and following editions). The Sciences of the Artificial. Cambridge: MIT Press.

frameworks or notions. There is no comprehensive set of ready-to-use notions or techniques to be used, and the history of the field has shown that building systems on the simple basis of knowledge and processes elaborated in other contexts (e.g., psychology on the one hand, CS on the other) rarely leads to satisfactory results.

As mentioned in Chap. 5, considering TEL as a field that poses complex issues, for which specific TEL constructions might be necessary, leads to consideration of TEL as a proper field, to be addressed within a transdisciplinary approach.

Both (1) work related to complex systems in general and (2) Software Engineering advances[2] suggest addressing complexity by building multiple viewpoints, i.e., interpretations from different perspectives that allow different dimensions to be captured. Adopting a given viewpoint does not consist of breaking a complex problem into separate simple sub-problems and, in particular, into separate disciplinary views. Rather, it consists of disentangling what may be disentangled with no loss as a means to make a given dimension addressable. While some of these viewpoints may be separated from one another, the essence of complexity is that some others are not independent. Considering the overall problem from the different viewpoints is a means to address complexity in a tractable way.

For instance, designing an ITS such as *JavIT* requires consideration of different viewpoints related to different dimensions: the domain (object-oriented design and programming); the way the domain may be conceptualized for teaching issues; the issue of interpreting learners' outputs; relationships with the CBPS and the teaching setting; the socio-technical dimensions related to these artifacts; etc. Most of these views and underlying models are interrelated with others. Difficulties lie in identifying what models are necessary to capture what is needed, elaborating these models while not falling into simplification, managing the different models' articulation and coherency, and implementing them in a way that allows their traceability.

## 2.2 *Definition and Matters of Concern*

Taking an engineering perspective (in the sense defined in Sect. 1), educational software engineering may be defined as:

*Educational software engineering* is the scientific field the objective of which is to study the issues related to educational software design and implementation. It is concerned with the notions, methods, theories, techniques, technologies or lessons learned that may facilitate the design, implementation, evaluation or diffusion of CBPSs, educational software and pedagogical-setting support software in a way that allows more than just *ad hoc* treatment of issues and problems.

[2]See for example approaches such as UML (Unified Modeling Language; http://www.uml.org; retrieved November 11, 2010) or MDA (Model Driven Architecture; http://www.omg.org/mda/; retrieved November 11, 2010).

Educational software engineering requires articulation of the CS dimensions of design and implementation (concepts, principles, processes, methods, techniques, technologies) and the Human and Social Sciences dimensions that must be considered at design time due to the pedagogical purpose of the constructed artifact and to the fact that it is used by humans (specific learning, teaching or usage phenomena; relationships between educational software and CBPSs or, when pertinent, teaching settings; relationships between design processes and usage or impact dimensions; etc.). The way these dimensions are articulated defines the nature of X when stating that design is an X-disciplinary work (see Chap. 3, Sect. 5).

Efforts dedicated to educational software engineering may correspond to:

1. Transversal efforts, whose objective is to clarify issues, define concepts, contrast possible approaches, etc. This is the perspective adopted in this book.
2. Specific efforts, whose objective is to build engineering methodologies.

We more precisely define these two options (which are disentangled here, but which may cohabit) and the conditions for effective projects to allow knowledge development in the following sections.

## *2.3 Transversal Efforts to Clarify Issues*

In the preceding chapters, we have highlighted that the educational discourse in the light of which software dimensions are to be addressed often leaves open many options. This may be related to an array of reasons: differences in matters of concern; differences of perspective, for instance with respect to the importance of software properties as influencing factors; unawareness of the level of precision required by software design; unawareness of difficulties; etc.

This issue may be thought of as something educationalists should solve, i.e., one may consider that educationalists should be helped to produce sufficiently precise specifications for computer scientists to act as computer scientists only, i.e., elaborate the required CS abstractions and/or enable the specified models and processes to be run on computers (see Chap. 5, Sect. 1). This is consistent with the view that a kind of line separates educational work and CS work.

In direct contrast, taking an engineering perspective (in the sense defined in Sect. 1) leads one to consider that an intrinsic dimension of educational software design is the existence of a zone within which occurs an education/CS interplay, and this zone must be addressed within a transdisciplinary approach.

Conducting such work requires the different actors to share some understanding of the *raison d'être*, rationale, objectives, specifications, models, descriptions, analyses (etc.) underlying the notions and artifacts at stake. As illustrated by the preceding chapters, this is of particular difficulty in the TEL area given the variety of perspectives that may co-exist with respect to what "educational software" is, what "designing educational software" may refer to (both in terms of design and technical dimensions), what important notions such as "impact" or "activity" may refer to, the different roles of CS-dimensions in the process, etc.

Within this perspective, a central issue of educational software engineering is to identify these difficulties and provide conceptual means to support their addressing. This is what this book is about, and the rationale for its contributions:

1. A general conceptualization oriented towards educational software engineering, introducing definitions that are of interest for disentangling issues and specific notions such as software pedagogical rationale (Chap. 2).
2. A highlight of the importance of understanding differences of perspective and an analysis of a set of issues that may lead to misunderstandings (Chap. 3).
3. Examples which may provide insights (Chap. 4).
4. An analysis of CS roles – and computer scientists' possible views – whose explicitation may, here again, contribute to understanding engagements and actions (Chap. 5).
5. An analysis of educational software engineering (this chapter).
6. A list of analysis axes and characterizing dimensions for educational software engineering work (Chap. 7).
7. General methodological considerations (Chap. 8).
8. A perspective on the way to push the field forward (Chap. 9).

Such transversal efforts serve both conceptual and pragmatic goals as they tend in the direction of:

1. Contributing to a general understanding of the field. TEL, as a complex field, requires description of the complexity and elaboration of different conceptualizations denoting different perspectives. Educational-software-engineering is one such perspective.
2. Defining intermediation objects (or elements to build intermediation objects) for actors acting from different perspectives, in both research and development projects.
3. Providing a consistent basis for elaborating knowledge (see Sect. 2.5).

## 2.4 Specific Efforts to Build Engineering Methodologies

Elaborating an educational software engineering methodology consists of describing a more or less precise process of how to build such software, i.e., (1) a sequence of actions described at a given level of detail and, possibly, (2) tools facilitating the implementation of this process (e.g., a modeling language or a dedicated software engineering framework). Compiling lessons learned serves similar goals.

Educational software denotes too large a variety of artifacts, considered settings or matters of concern to allow a useful engineering methodology to be elaborated. Indeed, this would result in considering issues at a level that is common to all settings, which would lead to only very general principles.

By contrast, it makes sense to consider engineering methodologies related to a precise type of object $x$, taking into account what this $x$ is and what considered matters of concern, notions or issues are. For instance:

- For $x =$ "on-line learning setting defined as a setting within which learners are provided with a set of on-line documents and a scenario stating how to go through these resources", what can be considered is, for example, a list of tasks to be tackled in order to identify and elaborate the resources: what are the pedagogical objectives? How may these pedagogical objectives be broken down? How do the pieces of knowledge relate to each other (prerequisites, etc.)? What resources and tasks may be related to these pieces of knowledge? How should these tasks be scheduled? etc. This may be based on, for example, instructional design approaches.
- For $x =$ "collaborative learning setting", it is possible, for instance, to list notions of importance when attempting to build scenarios and technical settings that enhance the chances that learners collaborate, such as the definition of roles, the way learners should be grouped or the issues to be considered.[3]
- For $x =$ "computer-based simulation", what may be considered is concepts, issues or lessons learned related to such simulations: how learners usually behave when confronted with a simulation, the issues that learners encounter to transfer what has been practiced within a simulation to an effective setting, etc. The addressed level leads to consideration of general notions that take specific importance and meaning in this context (e.g., the notion of *representation*). The analysis is implicitly positioned at a level that is independent of the taught domain.
- For $x =$ "simulation-based inquiry learning", what may be considered is the learning mechanisms related to inquiry and the simulation dimensions as related to these learning mechanisms: the positioning and, thus, the considered concepts and issues are different.
- For $x =$ "dynamic geometry", what may be considered is related to the specific studied domain (geometry), its properties, specific knowledge related to learners' issues with respect to geometry, how geometrical notions may be directly manipulated on the screen or the synergies that may be gained from using different representation registers. For a more precise $x$, e.g., $x =$ "dynamic geometry seen within the perspective of Brousseau's theory of didactical situations",[4] an engineering methodology may build on the basis of the notions introduced by the theory such as *situation*, *didactical transposition* or *didactical contract* (these different terms having precise meanings defined by the theory).

If engineering approaches may build from theories such as instructional design, domain-specific teaching theories or cognitive theories, the level of the considered

[3]See for example Tchounikine, P. (2008). Operationalizing macro-scripts in CSCL technological settings. *International Journal of Computer-Supported Collaborative Learning*, 3(2), 193–233.

[4]Brousseau, G. (1997). Theory of Didactical Situations in Mathematics. Dordrecht: Kluwer.

teaching setting and its organization may also be a matter of concern; see the way institutions such as Universities may divide up tasks between different services.

Efforts may also address specific types of systems (e.g., a methodology that allows building of ITSs), addressing the ITS notion in general or within a particular perspective and specific matters of concern (see how very different concerns may be considered with the presentation of *JavIT* in Chap. 4).

## *2.5 Conducting Projects as Vectors for Knowledge Development*

Following the view of engineering introduced in Sect. 1, building systems is a core vector for elaborating knowledge. When addressing design and implementation of educational software, there is not, on the one hand, a "scientific" activity consisting of the understanding of some "fundamental" issues and, on the other hand, a "technical" activity consisting of building systems: both dimensions are intrinsically meshed. Knowledge (theories if any, models, techniques, methods or lessons learned) may be addressed via the efforts to identify, understand and describe issues, to elaborate and analyze possible solutions, to capitalize lessons learned or to consider generalization issues.

In the case of research projects, cases studies can be defined according to the considered issue (a tailorability mechanism, an algorithm to build learners' models, a strategy to make learners argue, an innovative type of interface, etc.). Typically, this may consist of identifying the system or series of systems whose construction requires the research question to be addressed, and conducting the project in a way that considers abstract constructions and knowledge (as opposed to an *ad hoc* solution), the spectrum of validity or interest of these constructions, etc. Within such a process, the constructed system is both the resource and the objective of the scientific work. If the constructed system turns out to be pedagogically useful one may of course consider dissemination.

In the context of development projects, knowledge may be elaborated by making specific explicitation and capitalization efforts, in addition to the objective of implementing the system (see also Chap. 9). Analyzing how basic software is used by learners and teachers is also a resource.

As we have mentioned it in Chap. 3, addressing design of educational software as an important dimension is related to the hypothesis that the fact that a system presents some features and/or some properties rather than others may make – or contribute to – a difference (notion of *impact of a given system*, to be considered within the general perspective explained in the preceding chapters and not repeated here). In order to be productive of knowledge, work conducted in the context of particular projects must thus address the level of the general software functions and that of the detailed properties, which is one of the levels where design alternatives are to be considered, and decisions made. Building on the example of communication tools, considering in the same way chat tools presenting different properties (e.g. chronological presentation of freely typed sentences, presence of sentence-openers or graphical representation) is very unproductive.

As we have mentioned it, many dimensions related to software usage are subject to complexity and uncertainties. Considerations related to software properties must be examined at the level where they make sense. For instance, the impact of *Bio-sim*'s cognitive tools' properties on learners' inquiry processes may be rendered contingent by other factors (e.g., the way this CBPS is introduced to learners, pre- and post-sequences or institutional dimensions). It is thus of core importance to disentangle issues and matters of concern. Considering how software properties may be used to influence learners' activity is a precise and limited topic, which addresses only one dimension of a more general issue but, nevertheless, may be regarded as such. This may be contrasted with approaches whose explicit objective is to define methodologies aiming to improve practices (and develop knowledge)[5]: although, when studying software properties, improving practices may be an underlying objective, entry points, matters of concern, objects to be considered or methodologies are different.

## 3 Reconsidering the CS-TEL Relationship

### *3.1 Educational Software Engineering and Research*

In this chapter, we have said that engineering and research should not be treated as contradictory. Although the motivations, control, validation or diffusion processes are different than when just considering the objective of obtaining a usable system, differentiating engineering and research work is difficult since different dimensions and matters of concern are often entangled. A research project may require one to engage in purely programming work as a requirement to experiment with ideas. Conversely, the basic objective of obtaining a usable system may require one to produce innovative techniques and interesting lessons learned: design and implementation of systems often acts as a generator of problems, which may be disciplinary problems (e.g., psychology, pedagogy or CS) or TEL problems (e.g., perception and understanding of learners' activity or elaboration of learners' models). However, researchers should be able, at any time, to explain the research dimension.

Part of the misunderstanding related to the relationship between engineering and research stems from the fact that many of the disciplines addressing the TEL field have developed more common approaches to research and knowledge elaboration. For instance, when using empirical evaluation and statistical analysis in

[5]See for example Wang, F. & Hannafin, M.J. (2005). Design-based research and Technology-Enhanced Learning Environments. *Educational Technology Research and Development*, 53(4), 5–23.

psychology, the methodology and how a hypothesis may be turned into knowledge is crystal clear. In CS, the evaluation issue varies considerably depending on the type of work. For instance, a project related to the building of an algorithm may be examined by means of both qualitative and quantitative evaluations: the algorithm works or does not (given some input, it provides the expected output or does not); a proof can be exhibited (formal properties of an algorithm are demonstrated, using mathematical or logical foundations); the algorithm can be analyzed with respect to some metrics (e.g., given a benchmark, it produces the output in less time, or with more precision, than a comparative algorithm). However, benchmarks or proofs are rather rare in the TEL field. Evaluating a model, a process or a method is much more difficult. For instance, given a model or a method that is supposed to help in the design of computer-based systems, it is just not possible to create experimental conditions such as a set of modeling teams (not to say companies) that solve the same problem and for which the support provided by the model or method is the only impacting factor.

Within research communities, discussions related to methodologies are also often more or less implicitly underlined by the "research" status of the considered work. For instance, in the educational field, many discussions address differences separating empirical approaches ("code and count"), ethno-methodological approaches ("explore and understand"), design-based research or, now, neuroimaging. In CS, many misunderstandings separate people focusing on mathematical proofs of the properties of a given algorithm from the ones focusing on (for example) participative software-engineering methodologies. The educational software engineering field is not defined by a given methodology, but by the questions and issues it raises.

## *3.2 Educational Software Engineering and CS Research*

Considering educational software engineering as a scientific field makes it possible to re-question the relationships between TEL and CS and, in particular, CS research. We explore hereafter the relationship of TEL with three different visions of CS: (1) a vision focusing on the theoretical foundations of data processing, (2) a technical or technological vision focusing on artifacts and (3) a socio-technical vision, and come back to computer scientists' engagement and specific skills.

### 3.2.1 TEL and the Theoretical Dimensions of CS

When considering CS as the field studying the theoretical foundations of data processing by computers, TEL is basically a potential application domain for the technological applications of CS.

### 3.2.2 TEL and the Technological Dimensions of CS

When considering a technical or technological vision of CS, focusing on artifacts and their design processes, TEL is, like various other fields (e.g., information systems, HCI or knowledge engineering), a domain that allows the study of a variety of general CS issues: model-driven engineering, knowledge modeling, problem solving, interaction analysis, software adaptability, etc. Designing and implementing educational software leads to the addressing of issues that may, at a certain degree of granularity, be disentangled from their initial context and addressed as such.

Within this view, the general interest of TEL is (similarly to some other application fields) to create a context in which techniques and methods (in AI, software engineering, HCI, networking, etc.) may be analyzed and improved with respect to externally defined explicit constraints. This makes it possible not to consider these issues within "toy" artificial problems. Such work may lead to advances related to the TEL field and/or which find a wider application scope.

As an example of how TEL creates interesting instances of general CS issues and provides an interesting context in which to address them, let us consider tailorability, i.e., the fact that a system may provide its users with integrated support for modifying it in the context of its use. For educational software, tailorability appears to be a potential means to combine the two objectives of (1) providing learners with software designed to guide and support them, this design being based on pedagogical decisions, while (2) providing learners with built-in flexibility features to adapt software to their emergent activity and needs, in context. Introduction of tailorability in educational software, however, raises major issues: tailorability for learners ought to be studied with respect to the intention underlying the system, and if the system is designed to reify some pedagogic principles, its potential flexibility must be studied with respect to these principles and constraints. Furthermore, tailoring must be technically easy to carry out. Finally, tailoring is, with respect to the learners' activity as related to the learning scenario, another activity; therefore, the risk of generating a breakdown in the activity flow should be taken into account. TEL defines a context where a classical CS issue is a core issue, has a particular instantiation, and is subject to specific constraints.

When considering this technical or technological vision of CS, the specific interest of TEL is the importance of human and social dimensions, the multiplicity and complexity of the levels to be taken into account (sociological and psychological; individual and collective; etc.) and the evolutionary character of users.[6] The major drawback may be the difficulty of evaluation.

[6]By definition, educational software's objective is that users' (learners') activities and knowledge evolve.

### 3.2.3 TEL and the Socio-technical Dimensions of CS

Within a socio-technical vision of CS, CS is concerned with the functional dimension of software and with all the dimensions that may influence its effective usage.

Works related to TEL are of particular interest for CS in this respect. In addition to the technical dimensions (see *supra*), it provides a particularly productive field for studying socio-technical issues such as understanding the features that may influence stakeholders' activities and how to take them into account at the design stage and/or by providing software with run-time adaptation features as a way to contextually adapt to actors' effective activity. In other words, TEL is a means of addressing one of the fundamental issues of CS: understanding what *designing software to support human activity* means, and how to address this issue.[7] Here again, TEL may be used to elaborate knowledge that has a wider application scope.

### 3.2.4 Engagement of Computer Scientists

Putting emphasis on the engineering dimension raises the particular importance of elaborating pertinent abstractions in a way that transcend the boundaries of conventional academic disciplines, i.e., in a transdisciplinary way. This allows to be made more precise the different types of actions that computer scientists engaged in educational software design may be concerned with:

- Participating in the understanding of TEL issues, the elaboration of TEL-specific notions (e.g., learners' models) or processes (e.g., specific methodology) and, more generally, in the overall definition and understanding of the field.
- Elaborating and defining the CS dimensions of TEL projects.
- Solving the CS issues of TEL projects, building CS abstractions and implementing them.
- And, on another dimension: finding and exploring clever educational applications of CS technical advancements.

As mentioned in Chap. 5, Sect. 2.3, the specific skill of TEL-oriented computer scientists (or, in other words, TEL specialists with a background in CS) is the capacity to pertinently address the conceptual zone within which occur the education/CS interplays. TEL requires actors (researchers, engineers) with computational and TEL expertise allowing them to understand TEL concerns when managing the initial design, going from general issues to design decisions and participating in the interpretation of usage with respect to design issues. Methodological skills (not specific to TEL) such as technology-driven design, participatory design (involving future users in design), empirically-based design (prototyping with user testing), theory-based design (basing a design on a theory or conceptual framework originating outside of the technology domain) or evolutionary (incremental) design

[7]A provocative way to highlight this dimension is to suggest that some dimensions of CS may be viewed as part of the Human and Social Sciences.

are also of interest for conducting projects, as well as the CS skills allowing the management of the CS issues disentangled from their educational context.

## 4 Conclusions

In this chapter, we have re-examined a certain number of considerations related to educational software design issues that were first addressed in Chap. 1, and anchored them in a theoretical perspective. These considerations define the rationale for this book: addressing the problem of making explicit matters of concern and work, within the intrinsic difficulties and limits of this exercise. Without going again through the argumentation developed in Chap. 1, in summary, drawing attention to this issue and proposing means to address it serves objectives such as: facing the issues related to developing some shared understanding in a multidisciplinary team; reutilizing or adapting preceding work; refining or reconsidering design decisions; enhancing the chances that work is understood; identifying results and how they should be evaluated; avoiding re-inventing or re-building things that already exist, wasting time in work that does not produce any progress or falling into well-known methodological traps; capitalizing lessons learned or developing methodological guidelines; contributing to whether educational software is based on scientific knowledge and is not only subject to inventiveness or, less positively, technological push and the latest trends.

Addressing issues within a transdisciplinary perspective does not contradict the fact that some issues are disciplinary from the start, and that some others may become disciplinary at some level of granularity. Similarly, for computer scientists, TEL may be addressed both as a transdisciplinary field, within which they act as TEL specialists, and as an application field for different CS domains (e.g., Software Engineering, HCI or AI).

Finally, we have disentangled the objectives of understanding the issues related to educational software engineering and that of building an engineering methodology. This is a classic approach in software engineering. For instance, in object-oriented design, emphasizing the importance of disentangling static and dynamic views of classes or exploring the issues posed by identifying classes (etc.) is different from listing phases to be addressed when designing software and, for each of these phases, the tasks to be addressed and how they should be realized.