Computer Support for Constructionism in Context

Meurig Beynon, Chris Roe
Department of Computer Science, University of Warwick, Coventry, CV4 7AL.
{wmb,croe}@dcs.warwick.ac.uk

Abstract

The benefits of constructionism as a learning paradigm are widely recognised. Though the constructionist philosophy can be seen as applying to activities that are not necessarily computer-based (such as bricolage and concept mapping), its modern application in educational technology has been closely linked with computer use. In particular, Papert's work on LOGO programming in schools has both informed the original concept of constructionism and been a major influence over subsequent computer-based constructionist developments. This paper questions whether – despite these precedents – traditional computer programming is well-suited for the constructionist educational agenda. It argues that other approaches to computer model-building, such as those based on spreadsheet principles, are in fact much better aligned to the objectives of constructionism. Building on this basis, it proposes that more effective computer support for the constructionist perspective is offered by Empirical Modelling (EM) within a conceptual experiential framework for learning (the EFL).

1 Introduction

Throughout its history, educational software has progressed through many phases, mirroring the development of cognitive theories of learning. Early instructional software was dominated by programs that reflected a behaviourist outlook – the inspiration for much drill-and-kill software – where the computer acts as a replacement teacher, simply asking questions and gauging learning from the pupil’s responses. Over recent years, educational software has tended to reflect a constructionist approach, where learners are given an environment to explore, make hypotheses and guide their own learning.

Of particular significance in the constructionist tradition are environments that support ‘active learning’, in which learners are actively involved in building their own public artefacts. The emphasis in active learning is on the mental processes that occur during the construction of the artefact, not on the quality of the final product. The situated and public nature of the construction activity is also identified as important. For instance, in developing his vision for constructionism, Seymour Papert stresses that the active building of knowledge structures in the head often happens especially felicitously when it is supported by construction of a more public sort ‘in the world’ [15]. There are many reasons why active learning is seen as particularly beneficial: learners can pursue their particular interests, can see a tangible result with potential application and relevance, and are motivated to communicate their understanding to others.

The advent of computer technology for learning has opened up new avenues for developing concrete models in the form of interactive computer-based artefacts. To meet the requirements for (computer-mediated) active learning, it must be possible for ordinary computer users to construct such artefacts, so that meaningful learning of a domain can proceed in tandem with the construction of the interactive artefact. In her study of end-user programming [12], Bonnie Nardi claims that those who are not computer specialists can create personally meaningful computer models if the programming environment eliminates much of the accidental computational complexity. By way of example, she cites end-users creating computer-aided design models, LOGO programs and spreadsheet models.

In our view, the issues surrounding computer support for active learning have yet to be adequately addressed. Ever since Papert first developed the LOGO environment, there has been some ambiguity about the relationship between computer programming and the educational objectives of constructionism. Is computer programming to be viewed as an activity that – of itself – serves the educational objectives of the constructionist agenda, as the LOGO environment might suggest? Or is computer programming simply the means to set up environments for model making using techniques that are not – or at any rate are not perceived as – computer programming? In practice, the distinction between ‘learning about computer programming’ and ‘learning about a domain independent of computer programming’ is not always clearly respected in computer-based environments that support active learning. What is more, educationalists and computer scientists alike seem relatively insensitive to the potential implications of adopting different perspectives and approaches to constructing computer models.

In this paper, we argue that there are highly significant distinctions to be made between the different perspectives we can adopt on providing computer support for active learning. In particular, there is a fundamental conceptual distinction to be made between using spreadsheet principles and other programming paradigms (both procedural and declarative) that focus on programs as recipes for performing goal-directed transformations. Our thesis is that programming paradigms rooted in the classical view of computation are not well-suited to providing support for the constructionist learning agenda. On this basis, we propose an alternative framework that builds on the principles already represented in spreadsheet software (cf. [9]).

The paper is in two principal sections: the first discusses the relationship between classical computer programming and constructionism; the second briefly introduces and illustrates our alternative perspective on computer-based model building ("Empirical Modelling").

2 Programming and constructionism

The relationship between computer programming and constructionism is conceptually complex. Papert's aspiration for the use of LOGO is that constructing a program should be a valuable learning experience in which a pupil becomes familiar with geometric concepts and with strategies for problem solving and design [14]. There is an implicit assumption that the process of program construction is well-aligned to useful domain learning and to constructionist principles, but there are potentially problematic issues to be considered:

- *extraneous activity* – Much of the learning associated with model-building is computer-programming specific: it is concerned with manipulating programming language commands, procedures and parameters rather than with developing knowledge of geometric concepts or abstract thinking strategies;
- *planning rather than exploration* – Classical programming is not conceived as an iterative experimental process: programmers are encouraged to plan and preconceive their application rather than to develop a model in an open-ended fashion where its significance can emerge during the development.

Extraneous activity in computer supported domain learning is a problem for which many different remedies have been proposed. In [19], Soloway raises "the heretical question: *Should* all students learn to program?", and advocates the use of domain-specific, scaffolded, computer-aided design environments as an appropriate substitute for "those pesky semi-colons". The educational experts who respond to his question are positive about the importance of learning to program, and about the valuable – if not essential – contribution it can make to broader domain learning. The diversity of opinions about how to teach programming so that it does not obstruct domain-learning highlights such issues as: how best to provide programming interfaces for end-users; whether or not to use graphical front-ends; whether to use object-oriented principles or recursion, linked lists and trees.

The significance of being able to treat computer programming as an exploratory activity, rather than a planned activity, is likewise well-recognised. In [3], Ben-Ari discusses how computer programming, as practised, contains the element of re-design in response to interaction with the partially-developed program that is characteristic of bricolage (as originally characterised by Levi-Strauss [11]). This is endorsed by Fred Brooks's observation [5] that programmers see their work as a craft where they wrestle with incompletely understood meaning, and by proposals for software development based on techniques such as 'extreme programming' [2].

The thesis of this paper is that a proper appreciation of the problem of providing computer support for constructionism can only be gained through looking at a deeper issue than the flavour of programming paradigm, the interfaces for the end-user, or the method of software development. There is a profound ontological distinction between an artefact that is developed in active learning and a computer program. To interpret computer support for constructionism effectively it is necessary to shift attention from the concept of computer program that is endorsed by the classical theory of computation, and focus instead upon the way in which the programmed computer itself serves as a physical artefact. This is best appreciated by comparing the thought processes that accompany contemplation of the artefact in active learning with those associated with developing a computer program.

In active learning, the artefact under development is a source of experience. Throughout its development, the learner is invited to project possible interpretations and applications on to the artefact as it evolves. The learner asks such questions as "what can I do with this now?" and "how can this particular kind of interaction with the artefact now be interpreted?". In so far as some reliable interactions with the artefact are familiar to the learner, it implicitly embodies knowledge. At the same time, since many of the plausible interactions contemplated may be as yet unexplored, the artefact in some respects embodies the learner's ignorance. The educational qualities of interaction with the artefact mirror those exhibited in an informal exposition of a proof. Such an exposition is mediated by artefacts, so that the reader can be invited to anticipate the next step, and introduced to the situations in which false inferences can be drawn or unsuccessful strategies adopted.

By contrast, developing a program is understood (from the perspective of the classical theory of computation) with reference to assertions of the form "this is what the program is intended for; these are the kinds of interaction that it admits; these are the ways in which responses to this interaction are to be interpreted". It is of course the case that, in any complex programming task, essential knowledge of the domain is developed through experimental activity involving artefacts (as represented by use cases, UML diagrams [10], and prototypes of various kinds). But while this domain knowledge plays a fundamental role in programming, it is primarily directed at the intended functionality and interpretation of the program. For this reason, the artefacts developed in framing requirements serve only for reference purposes once the program implementation begins. A computer program resembles a formal proof in that it follows an abstract pattern of steps whose meaning is entirely contingent upon adhering to a preconceived recipe that is invoked in the correct – fastidiously crafted – context.

The above discussion suggests that the conventional perspective on computer programming is unhelpful in understanding how to give computer support to constructionism. This is not to deny the practical value of computer-based environments that have already been developed for active learning, but to observe that they ideally demand a conceptual framework quite different from that offered by classical computer science. With the possible exception of domains in which learning is primarily concerned with understanding processes, it is in general inappropriate to think of a learning artefact as a computer *program*. For reasons to be briefly explained and illustrated in the following sections, we prefer to characterise computer-based artefacts for active learning as “construals”.

Our proposal to discard the notion of program in favour of ‘construal’ is in the first instance significant only as a meta-level shift in perspective. In practice, spreadsheets already provide examples of such construals. It is also likely that, in asserting that “we need to fundamentally rethink how we introduce programming to students”, “we require new types of programming tools”, and “we need new programming paradigms”, Resnick and Papert [19] have in mind a much broader notion of ‘program’ than the classical view of computation supports. Nevertheless, making the explicit distinction between programs and construals liberates a radically different view of what computer support for constructionism entails, and lays the foundation for a better understanding with implications for theory and practice. For instance, it can help to identify more effective principles and tools for building learning artefacts, and may help to explain practical developments, such as the success of spreadsheets and the relative lack of popularity of programming as a learning tool for the non-specialist (cf. [12]), and the emergence and subsequent disappearance of Logo from the UK National Curriculum (cf. [13]).

3 Empirical Modelling

Our description of a learning artefact as a ' construal' borrows from the work of David Gooding, a philosopher of science. Gooding [8] used the term to describe the physical artefacts and procedures for interaction, observation and interpretation that Faraday developed to embody his understanding of electromagnetic phenomena, as it evolved through practical experiment and communication with other experimental scientists. In that context, experiment has a significance beyond the popular understanding of the scientific method (as in [18]: "One develops a theory that explains some aspect of reality, and then conducts experiments in order to provide evidence that the theory is right or demonstrate that it is wrong."). Though Faraday' s experiments did eventually underpin Maxwell' s mathematical theory, they initially had a far more primitive role. For instance, they served to distinguish transient effects from significant observables, and to relate Faraday' s personal construals of a phenomenon to those of others who had typically employed different modes of observation and identified different concepts and terminology. Such experiments were not conducted post-theory to ‘explain some aspect of reality’, but rather to establish *pre-theory* what should be deemed to *be* an aspect of reality.

A construal is typically much more primitive than a program. It is built with a referent in mind. The conventions for interacting with it and interpreting these interactions are quite informal and fluid. In general, whether a particular interaction has an interpretation can only be appreciated by consulting the immediate experience it offers and recognising this as corresponding to an experience of the referent. A possible construal for the electromagnetic phenomenon associated with a wire coil might be a depiction (e.g. by means of a diagram on a computer screen) of the direction and strength of the electric current, and the disposition and density of the lines of the magnetic field. A primitive interaction with such a construal would involve observing the impact of changing the current on the strength of the magnetic field in both the computer model and its referent. The relationship between current and field would be perceived as a direct correspondence between dependencies in the model and its referent. In this context, the counterpart of a program would be a much more sophisticated construction, such as a model of an electric motor, that has some autonomous reliable behaviour that cannot be experienced through being present in just one situation.

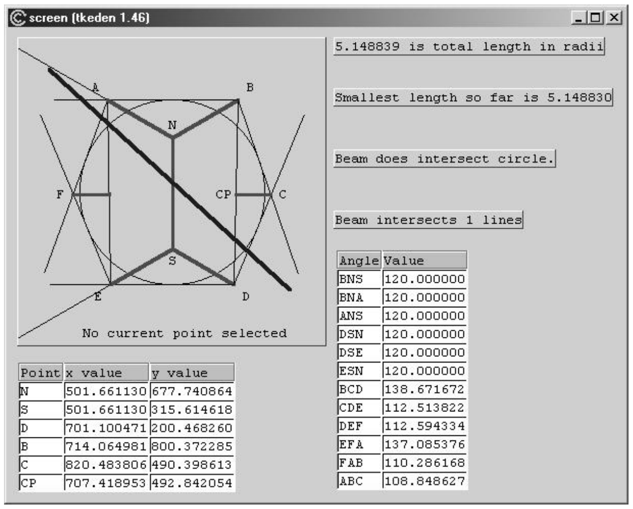
Empirical Modelling (EM) describes the characteristics of a construal (cf. a spreadsheet) with reference to three key concepts: observables, dependencies and agency. An *observable* is a feature of the situation or domain that we are modelling to which we can attach an identity (cf. a spreadsheet cell). The main requirement of an observable is that it has a current value or status (cf. the value of a spreadsheet cell). A *dependency* is a relationship amongst observables that expresses how they are indivisibly linked in change (cf. the definition of a cell). Unlike constraints, which express persistent relationships between values in a closed world, dependencies express the modeller's current expectation about how a change in one variable will affect the value of another in an open-ended exploratory environment. Observables and dependencies together determine the current state of an EM model. An *agent* is an entity in the domain being modelled that is perceived as capable of initiating state-change. In developing an EM model, our perspective on agency within the domain evolves with our construal.

Developing a construal in EM is a voyage of discovery, a creative activity that is quite unlike conventional programming, where the emphasis is on representing well-understood behaviours. An EM model is empirically established (informed by experience and subject to modification in the light of future experience) and experimentally mediated (our experience with it guides its evolution). A construal must be testable beyond the limits of the expected range of interactions with it. In specifying a conventional program, the modeller has to preconceive its behaviour, thereby restricting the exploratory interactions that can be undertaken. In contrast, EM model construction privileges experimental interaction. Interactions can take account of the changing real-world situation; can probe unknown aspects of a referent; and may even be nonsensical in the world.

The potential implications of adopting an EM perspective on computer support for constructionism will be briefly illustrated with reference to a simple example. A *beam detector* for the unit circle is a set of points that intercepts all lines crossing that circle. Eppstein [7] describes a beam detector constructed by taking a regular hexagon ABCDEF that circumscribes the unit circle, joining the points ABDE using a Steiner tree, and dropping line segments from the two vertices C and F on to the nearest side of the quadrilateral ABDE. The length of such a detector is $2/\sqrt{3} + 4 = 5.1547$. Eppstein observes that this is non-optimal and conjectures that non-regular hexagons can be used to reduce this length.

A teacher wishing to exploit Eppstein's beam detector as an aid to active learning might consider many issues:

- *motivating the search for a detector of optimal length.* To this end, Ian Stewart [20] devises a detective story, recasting the problem as digging trenches of minimal size that are guaranteed to detect a drainage pipe in the neighbourhood of a statue. To exploit this interpretation, it might be helpful to construct a virtual reality model.
- *situating the problem within computational geometry.* Eppstein's construction is an application for Steiner trees. This motivates making a model that incorporates and builds on a method of Steiner tree construction. For further investigation, this model could be extended to display critical lines that pass through just one of the five straight-line segments of the given beam detector.
- *using the beam detector to illustrate school geometry.* Modelling the detector is an exercise in geometric construction that helps students to learn about tangency, trigonometric relationships, perpendicular lines etc.
- *using the detector as a case study for modelling tools.* Students could make a geometric model of the detector using a special-purpose tool such as CABRI [6], or study it as an optimisation problem using a spreadsheet.



No current point selected

5.148839 is total length in radii

Smallest length so far is 5.148830

Beam does intersect circle.

Beam intersects 1 lines

Point	x value	y value
N	501.661130	677.740864
S	501.661130	315.614618
D	701.100471	200.468260
B	714.064981	800.372285
C	820.483806	490.398613
CP	707.418953	492.842054

Angle	Value
BNS	120.000000
BNA	120.000000
ANS	120.000000
DSN	120.000000
DSE	120.000000
ESN	120.000000
BCD	138.671672
CDE	112.513822
DEF	112.594334
EFA	137.085376
FAB	110.286168
ABC	108.848627

Figure 1: An EM model of a beam detector

Issues of presentation are also relevant. The teacher might wish to present the construction of the beam detector, as described by Eppstein, using an interactive whiteboard, to distribute instances of the construction to the pupils for them to experiment and compete to find the best solution, and to monitor and to display the details of the detector of smallest total length encountered to date concurrently in real-time (e.g. as might be done in a sporting event).

If we regard these potential applications as specifications for independent programming exercises to be addressed, there is a prohibitive overhead. Model-building directed at capturing the different functional requirements involved in developing a VR environment, setting up a spreadsheet, or emulating CABRI, cannot exploit abstraction above the level of a general-purpose programming language. By building a construal, on the other hand, it is possible to build an integrated family of models adapted for each of these different purposes.

Figure 1 shows a version of the EM beam detector model that has been developed by the authors with items from the specific agenda above in mind. Its development illustrates adaptation and extension of an initial basic model that is typical of EM. The grid displays are drawn from an EM spreadsheet model developed by the second author [21, spreadsheetRoe2002]. The line segments in the beam detector have been highlighted, and a beam has been added for experimental validation of the detector. Redefining any observable is a legitimate interaction for the modeller, but a typical 'user' interaction with the model involves selecting and positioning the points B, N, S or D. The equality of the six 120° angles tabulated – as determined by dependency – identify the main component of the detector as a Steiner tree. Figure 1 also validates Eppstein' s conjecture. Further details and examples of other adaptations are beyond the scope of this short paper, but can be found in [21, beamdetectorRoe2002].

4 Concluding remarks

The application of EM principles to educational technology has been further discussed in previous papers [1,4,17]. The significance of construals can be best appreciated in relation to an experiential framework for learning (the EFL) [16] which spans learning activities ranging from private interaction with concrete artefacts to the manipulation of formalisms and theories subject to public conventions for interpretation. As classically conceived, computer programming and programs are most closely related to public-theoretical-formal activities in the EFL, and constructionism and construals to private-empirical-concrete activities. EM is well suited to supporting learning activities at the concrete end of the EFL because of its primary emphasis on experimental interactions with artefacts and the representation of state-as-experienced. It can also support the fluid movement between learning activities in the EFL through integrating the abstract and the concrete within a single modelling environment.

The simple illustration in this paper has been chosen to give a general impression of the character of EM, and to highlight its links with the spreadsheet concept. Applications for pre-theory modelling covering a wide variety of domains can be consulted in the EM web-archive [21]. The concept of developing an EM construal from which a diverse suite of programs can be generated has been most effectively exploited in the work of Richard Cartwright and his colleagues at the BBC R&D laboratories. Their research exploits tools developed by Cartwright as part of his doctoral research with the EM group at Warwick, and shows the feasibility of applying EM principles – in particular dependency - to cross-platform publishing of interactive television applications.

References

[1] A.Bhalerao, W.M.Beynon, C.Roe, A.Ward. A computer-based environment for the study of relational query languages. In *Proceedings of the Teaching, Learning and Assessment in Databases workshop*, Coventry, United Kingdom, 14th July, pages 104-108, 2003.

[2] K.Beck. *eXtreme Programming explained*. Addison Wesley, 2000.

[3] M.Ben-Ari. Constructivism in Computer Science Education. In *Journal of Computers in Mathematics and Science Teaching*, 20 (1), pages 45-73, 2001.

[4] W.M.Beynon. Empirical Modelling for Educational Technology. In *Proceedings of Cognitive Technology 1997*, pages 54-68, University of Aizu, Japan, IEEE, 1997.

[5] F.P.Brooks Jr. *The Mythical Man-Month: Essays on software engineering*. Addison-Wesley, 1995.

[6] CABRI geometry. http://www-cabri.imag.fr

[7] D.Eppstein. http://www.ics.uci.edu/~eppstein/junkyard/beam

[8] D.Gooding. Experiment and the making of meaning. *Kluwer Academic Publishers*, 1990.

[9] T.A.Grossman. Spreadsheet Engineering: A research framework. In *Proceedings EUSPRIG 2002*, pages 23-34, 18th-19th July, 2002.

[10] I.Jacobson, M.Christeron, P.Jonson, G.Overgaard. Object-oriented *Software Engineering: A use-case driven approach*. Addison-Wesley, 1992.

[11] C.Levi-Strauss. *The savage mind*. University of Chicago Press, 1968.

[12] B.Nardi. *A small matter of programming: Perspectives on End User computing*. MIT Press, 1993.

[13] R.Noss, C.Hoyles. *Windows on mathematical meanings: Learning cultures and computers*. Dordrecht: Kluwer, 1996.

[14] S.Papert. *The children's machine*. New York: Basic Books, 1993.

[15] S.Papert, I.Harel. Situating constructionism. In S.Papert, I.Harel (Eds). *Constructionism: Research reports and essays*, Ablex Publishing, pages 1-11, 1991.

[16] C.Roe. *Computers for learning: An Empirical Modelling perspective*. PhD Thesis, Department of Computer Science, University of Warwick, November 2003.

[17] C.Roe, W.M.Beynon. Empirical Modelling principles for learning in a cultural context. In *Proceedings 1st International Conference on Educational Technology in Cultural Context*, University of Joensuu, Finland, pages 151-172, September 2002.

[18] D.Sannella. What does the future hold for theoretical computer science? In *Proc. 7th Intl. Conf. On Theory and Practice of Software Development (TAPSOFT'97)*, LNCS Vol.1214, pages 15-19, Springer, 1997.

[19] E.Soloway. Should we teach students to program? *Log On Education*, CACM, 36 (1), October 1993, 21-24.

[20] I.Stewart, The great drain robbery, in *Math Hysteria – Fun and games with mathematics*, OUP, 2004.

[21] The Empirical Modelling Web Repository at the URL: http://empublic.dcs.warwick.ac.uk/projects (typically referenced in conjunction with a project name from the repository).