Enriching Computer Support for Constructionism

Meurig Beynon

Empirical Modelling Research Group,
Department of Computer Science,
University of Warwick,
Coventry,
CV4 7AL.
+44 2476 523089
{wmb@dcs.warwick.ac.uk}

Chris Roe

Centre for New Technologies Research in Education,
Institute of Education,
University of Warwick,
Coventry,
CV4 7AL.
+44 2476 523855
{Christopher.Roe@warwick.ac.uk}

Enriching Computer Support for Constructionism

ABSTRACT

The dominant emphasis in current e-learning practice is instructionist in character. This is surprising when we consider that the benefits of constructionism as a learning paradigm are so widely recognised. Moreover, 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 mathematical education through LOGO programming has both informed the original concept of constructionism and been a major influence over subsequent computer-based constructionist developments. This chapter questions whether – despite these precedents – traditional computer programming paradigms are 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 can potentially be offered by Empirical Modelling (EM). Adopting this approach demands a reappraisal of the relationship between the formal and the informal with relevance for education, mathematics and computing.

KEYWORDS – Technology Enhanced Learning, Educational Technology, Constructivism, Active Learning, Electronic Spreadsheets, Modelling Languages, Electronic Learning (e-Learning).

INTRODUCTION

The development of e-learning environments has been driven by the needs of universities, where the lecture is the dominant teaching method. The Internet can serve as an efficient mode of effecting lecture delivery. In some situations, the delivery of factual information is entirely appropriate. On the other hand, educationalists recognise the importance of interaction, and constructionists (Harel & Papert, 1991) go further to propose principles that facilitate active learning. In this chapter, we reflect on two complementary experiences: of developing microworlds as part of an e-museum; and of experimenting with a novel approach to building computer-based models for educational applications - that of Empirical Modelling (EM). We draw on these experiences to raise fundamental questions about how computer support for constructionism can best be provided. We identify respects in which conventional thinking about general-purpose computer programming conflicts with the aspirations of constructionism. This leads us to propose EM as a promising alternative foundation for constructionist practices. Though we introduce a simple illustrative example to inform the discussion, a full demonstration of the practical usefulness of EM is outside the scope of this chapter. In any case, without a broader infrastructure for our principles and tools, a proper empirical evaluation of EM in an educational context would be premature at this stage. And though there are indications that EM potentially has useful educational application across a wide range of subjects (see (Roe, 2003). and (McCarty, Beynon and Russ, 2005) for instance) our attention in this chapter is confined to mathematical education. This is in keeping with Papert's original motivation for introducing constructionism, and informs a brief discussion of the deeper foundational issues that are topical in understanding mathematical concepts.

PRINCIPLES OF CONSTRUCTIONISM

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 constructivist approach that acknowledges the active role played by the learner in the making of meaning.

A more specific form of constructivism began to emerge in the 1960s when a team, headed by Papert and Feurzeig, was developing the Logo programming language at MIT. This early work was primarily concerned with programming and problem-solving in the context of mathematical education (see Papert, Watt, diSessa and Weir, 1979; Watt, 1979). In particular, it advanced the radical notion that children need to play with and use mathematical concepts within a supportive computer-based environment before being introduced to formal work with those concepts (Papert, 1972, p.18):

When mathematizing familiar processes is a fluent, natural and enjoyable activity, then is the time to talk about mathematizing mathematical structures, as in a good pure course on modern algebra.

These initial ideas led Papert (1980) to a vision for mathematical education that was subsequently elaborated into a new paradigm for teaching and learning mathematics – the *constructionist* approach (Harel & Papert, 1991). Following (Roe, Pratt and Jones, 2005),

we shall discuss model-building with reference to six underlying features of constructionist learning distilled from the literature:

1) Quasi-Concrete Objects – Turkle and Papert (1991): The computer allows formal ideas to be accessed in a concrete way, through developing iconic representation of abstract mathematical ideas that can be manipulated directly by the user;
2) Integrating the Informal and the Formal – diSessa (1988): Incorporating formal representations of mathematical objects in models in different ways may help a child to make connections between the various formalisations and their informal use;
3) Using Before Knowing – the *Power Principle*, Papert, 1996: In everyday life, we typically use tools for particular purposes. Through such use, we learn about the effectiveness of a tool, its limitations, how well it serves its purpose, and may sometimes gain some insight into how it works. In schools, mathematics is a subject where, in general, you learn how to generate the object before you use it. In practice, the task of object generation proves too difficult, especially when disconnected from purpose. The computer makes it possible to invert the activity of learning mathematics so that use precedes generation;
4) Dynamic Expression – When Papert proposed the turtle as a tool for constructing a dynamic notion of angle (and of course much else), he acknowledged that the computer offers a medium which – unlike paper and pencil – can incorporate dynamic representations of the world. He suggests that systems which are expressive of dynamic and interactive aspects of the world are more engaging to learn than static and abstract formalisms;

5) Building – Constructionism is based on the tenet that encouraging the building of artefacts is a particularly felicitous way of teaching mathematics. Pratt (2000) has demonstrated how this approach generalises to related activities such as *mending*;
6) Purpose and Utility – The microworld approach can encourage purposeful activity through the building and modification of artefacts. Through such interaction, the usefulness of the relevant abstractions is appreciated and their limitations are gradually discriminated.

The e-Muse project illustrates an application of constructionist principles. The broad aim of this project was to develop an Internet museum. The output of the project comprised exhibits relating to the ancient Olympic Games, explanatory material related to the exhibits, and interactive activities to engage visitors. These resources were developed with both museum and schools environments in mind.

From the first, the project engaged with two tensions. In developing a virtual museum that bridged museum and school environments, there was an underlying cultural conflict. Whereas museologists were concerned primarily with accuracy and appropriate presentation, classroom practitioners' foremost concern was promoting interaction and engagement. The second tension stemmed from the first. The primary interest of the museologists – in common with that of the designers of so-called e-learning environments – was in the efficient delivery of accurate materials, whereas that of the educationalists – in common with that of classroom practitioners – was on stimulating learning and engagement.

In e-Muse, two microworlds were developed. These provided an interactive experience for e-museum visitors. They were based on the throwing events of the Olympics and targeted at children of 10 years old and upwards. Their development reflected a

constructionist perspective; it made use of the Imagine Logo programming environment (Kalas & Blaho 2000), and was guided by observation and feedback obtained at a local school, where prototypes were trialled with schoolchildren.

One of the primary findings of the e-Muse project was that the realisation of constructionist ideals for learning was obstructed to some considerable degree by the lack of a facility for children to build or modify models themselves (Roe, Pratt & Jones, 2005). Though the Imagine Logo environment offered many features to assist the developers in empowering the user to interact with models in imaginative ways, the task of adapting models in response to children's perceived and expressed needs was typically too technical to be undertaken by any participant other than the primary developer. For this reason, the e-Muse environments afforded less flexibility and openness in interaction than was ideally envisaged.

The predominant delivery model for e-learning exhibits similar restrictions and lack of flexibility in interaction, perhaps to an even more marked degree. As Bannan-Ritland *et al* (2002) observe, designers of e-learning environments typically structure content in a particular sequence for delivery to the learner. This leads them to remark (p.12) that:

... there are alternative theoretical foundations other than a traditional instructional system design perspective that can be applied to learning object systems based on constructivist philosophy of learning. To the best of our knowledge, a learning object system based in theoretical approaches steeped in constructivism has not yet been developed.

Of course, it is not self-evident that the level of interaction advocated by a constructivist philosophy is achievable. Indeed, Ehrmann (2000) has argued that the idea

of attaining interactive courseware that can give full support to constructionist principles is a mirage. He claims that this is due to the high human costs needed to achieve appropriate levels of interactivity. We shall argue that, whatever ultimate limits may have to be set on the aspirations of computer support for constructionism, the adoption of an alternative approach to computer-based modelling offers the potential for far greater levels of interactivity in e-learning environments. In the next section, we provide the background and motivation for this argument by considering the relationship between conventional programming and constructionism, both in its historical context, and in relation to the findings of the e-Muse project.

CONSTRUCTIONISM AND PROGRAMMING

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' (Papert & Harel, 1991). 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, Bonnie Nardi (1993) 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 chapter, 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 that focus on programs as

recipes for performing goal-directed transformations. Our thesis is that programming paradigms rooted in the classical view of computation (which embrace the full range of traditional programming idioms including procedural, object-oriented and declarative approaches) 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 for spreadsheet engineering identified by Grossman (2002). The remainder of the chapter is in two principal sections: the first discusses the relationship between classical computer programming and constructionism; the second briefly introduces and illustrates the alternative perspective afforded by Empirical Modelling.

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 (Papert, 1993). 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. Soloway (1993) 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 responded to his question were 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. Ben-Ari (2001) 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 (Levi-Strauss, 1968). This is endorsed by Fred Brooks's observation (1995) 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' (Beck, 2000).

The thesis of this chapter 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 (Jacobson *et al*, 1992), 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 (Soloway, 1993) 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. Nardi, 1993), and the emergence and subsequent disappearance of Logo from the UK National Curriculum (cf. Noss & Hoyles, 1996).

CONSTRUCTIONISM AND EMPIRICAL MODELLING

Our description of a learning artefact as a construal borrows from the work of David Gooding, a philosopher of science. Gooding (1990) 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 Sannella (1997): "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."). Although 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 in an open-ended fashion with a situation rather than a specific purpose 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, such as Faraday himself first developed as a physical artefact, and we might now realise on a computer, would depict the relationship between the direction and strength of the electric current and the disposition and density of the associated 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 using a simple example. A *beam detector* for the unit circle is a set of points that intercepts all lines crossing that circle. Eppstein (1998) 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 (2004) 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 Geometry, or study it as an optimisation problem using a spreadsheet.

Issues of presentation are also relevant. The teacher might wish to present Eppstein's construction of the beam detector 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.

```
%donald
viewport Figure
line SSE, NNE, NS
point SE, NE, S, N
real lenS, lenN, len
SSE = [S, SE]
NNE = [N, NE]
NS = [N, S]
S = O - {0, 0.6 * rad}
N = O + {0, 0.6 * rad}
lenS = 0.75 * rad
lenN = 1.01 * rad
SE = S + {lenS @ (-pi div 6)}
NE = N + {lenN @ (pi div 6)}
label lSE, lNE, lS, lN
lSE = label("L", SE + {offset, -offset})
lNE = label("U", NE + {offset, offset})
lS = label("S", S - {0, offset})
lN = label("N", N + {0, offset})
point extNW
line constNW, constSW
point extSW
extNW = N + {len @ (5 * (pi div 6))}
constNW = [N, extNW]
constSW = [S, extSW]
extSW = S + {len @ (7 * (pi div 6))}
```

Listing 2(a) – Some preliminary definitions

```
%donald
real angleNNE, angleSSE
angleNNE = acos((N.1 - S.1) div dist(N, S))-(pi div 3)
angleSSE = acos((N.1 - S.1) div dist(N, S))-(2*pi div 3)
SE = S + {lenS @ angleSSE}
NE = N + {lenN @ angleNNE}
extNW = N + {len @ (angleNNE + (2 * pi div 3))}
extSW = S + {len @ (angleSSE - (2 * pi div 3))}
```

Listing 2(b) – Definitions for extension and revision

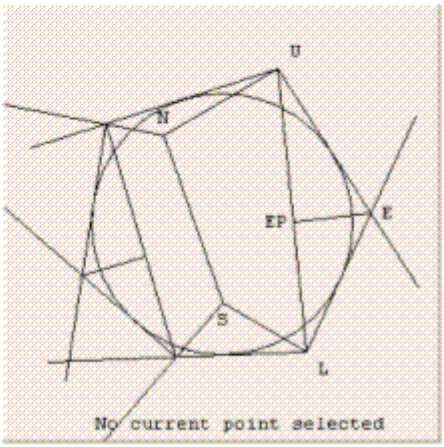

Diagram 2(c) – The basic beam detector

Figure 2 – The original Beam Detector model and some of its script definitions

Screenshots and extracts from variants of an EM model of Eppstein's beam detector are shown in Figures 2 and 3. A full account of the principles behind the construction of the model and its variants is beyond the scope of this chapter. The details of the models can be inspected and exercised more closely by accessing the `beamdetectorRoe2004` directory of the EM repository at `http://empublic.dcs.warwick.ac.uk/projects/`.

Other models from the repository illustrate that the features needed to make the extensions of the Beam Detector model envisaged above are broadly within the scope of the current EM tools. The following brief discussion will highlight some of the most salient points about the development of the Beam Detector models.

The original source for the Beam Detector model was developed by the first author at the suggestion of a colleague who was studying beam detection as an abstract optimisation problem in computational geometry. The model was constructed over a period of two or three days, and involved some three or four hours development. The listings in Figure 2 illustrate the kind of activity that was involved – the creation of a script of definitions to record the key observables (such as points, lines and labels) and dependencies (such as relationships of incidence and perpendicularity) in Eppstein's construction. The script creation was an incremental process, so that the definitions in Listing 2(a) were devised first, and those in Listing 2(b) were developed subsequently (see `beamdetectorRoe2004` for all the sub-scripts involved in building up the entire model stage by stage, from which the listings in Figure 2 have been extracted).

Figure 2 illustrates some general characteristics of EM. The script of definitions for the model evolves in conjunction with the visual artefact as the modeller's understanding is clarified. At any stage of the development, there is a script and an associated visualisation that records such observables and dependencies as the model-builder has so far explored. Some experimental interaction, guided by the visualisation, is typically involved in each step of the incremental construction. In introducing the definitions in Listing 2(b), for instance, some experimentation was used to establish and confirm how the principal value of the angle denoted by the `acos` function was being selected. The overall character of EM in this context is consonant with the way in which Edwards and Hansom (1989) identify modelling as an iterative process comprising understanding the particular

phenomena to be modelled, identifying the key variables and explicitly defining the relationships amongst the variables.

As Figure 2 also illustrates, a script of definitions serves as both a description of the model in its current state of development, and a record of the interaction that has led to this current state. This dual role lends to the script the rather loose and messy quality that is characteristic of much of learning activity. The evolution of the models reflects different stages in the understanding, for which different configurations and visualisations of observables are generally appropriate. In Listing 2(b), for instance, the observable SE in the model, which refers to the geometric location of the point labelled "L" in the diagram, has been redefined. This redefinition has been made in order to ensure that whenever the points N and S are relocated, the angle NSL is 120^{o}, as is appropriate in a Steiner tree.

The attentive reader will note that in Figure 2(c), which depicts one of the possible states of the basic Beam Detector model in (EMRepository: `beamdetectorRoe2004`), the angle NSL is no longer 120^{o}. This is because, at a subsequent stage in the model-building, the definition of SE was restored to that in Listing 2(a). The explanation for this is that – without loss of generality – any location of the points N and S leads to a beam detector configuration that is congruent to one in which the line NS is vertical. On this basis, a full exploration of the design space of optimal beam detectors on Eppstein's pattern can be carried out without needing to re-orient the line NS at any stage. With this fact in mind, there is a useful educational purpose in neither constraining angle NSL as in Listing 2(b), nor preventing NS from taking up a non-vertical orientation. Allowing a student to displace N and S arbitrarily then supplies experimental evidence that deviating from a Steiner tree can only make the length of the beam detector sub-optimal.

The above discussion illustrates the complexity of the issues involved in building models to support learning, and in particular the subtle role played by placing constraints on interaction. The virtue of the basic Beam Detector model as an interactive artefact for experiential learning is that, unlike a conventional program, it is not developed with closed learning objectives and ease-of-use in mind. However, we may extend the basic model with a view to making it less open-ended and more self-explanatory, so as to give greater prominence to specific targets for the learning. Figure 3 is an extension of the Beam Detector model, carried out by the second author at a much later date, that incorporates features to assist the learner. In this model, points and lines can be manipulated dynamically, rather than merely relocated in a discrete fashion, so that experience of a different quality is brought to bear in trying to minimise the length of the beam detector. The geometric components of the beam detector have been highlighted and a representation of the beam itself has also been added, as have spreadsheets and textboxes to display significant values. This process of extension has precisely the same character as the creation of the original model, and exploits re-use of other EM models.

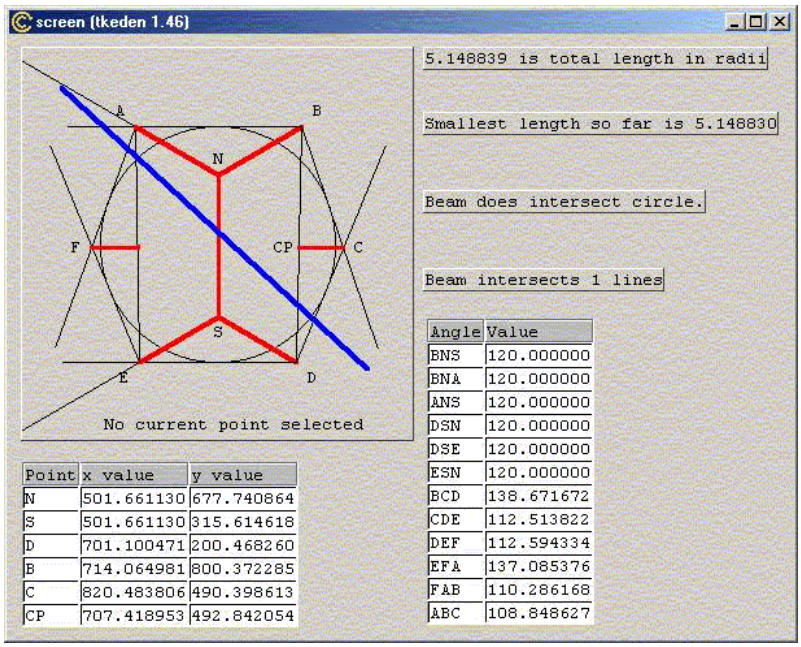

Figure 3: An extension of the basic EM model of a Beam Detector

PERSPECTIVES ON CONSTRUCTIONISM

In the previous section, we have compared and contrasted the support for constructionist principles afforded by EM and by conventional programming. Our illustrative example, the Beam Detector model, like the e-Muse microworlds, relates to implementing a constructionist approach to mathematics education. In this section, we review our findings from a broader perspective, briefly discussing how they relate to topical perceptions of education, mathematics and computing. In this way, it becomes apparent that some of the specific tensions between learner, teacher and developer perspectives alluded to in the previous section are symptomatic of more profound conflicts in thinking about mathematics, education and computing, both interdisciplinary and intradisciplinary. When trying to bring these disciplines together, these conflicts are not merely unresolved – they are to a large extent unacknowledged.

From an educational perspective, model-building by computer is an activity that superficially appears best aligned to teaching mathematics, or a mathematical science. There are a number of plausible reasons for this. Computer support for constructionism has its historical roots in mathematics education. Programming computers is perceived as primarily a logical exercise in framing sequences of action that, like formal mathematics, requires great precision and abstract thought. The kind of model-building with computers that has most educational credibility is model-building that is based on mathematical theory, as when Newton's Laws of Motion are implemented in an e-Muse microworld.

Orthodox computer science thinking endorses this attitude to computer use in education only partially. Having regard to the still unresolved problems of the 'software crisis', building software from a theory is perceived by many as the only way forward for

computing (see for example Turski and Maibaum, 1987). In this context, the issue of which computer programming techniques are to be commended, and which deprecated, is a matter of great controversy. It is widely recognised that what is actually involved in instructing the computer by way of programming is far less significant than how these instructions are linked to the key observables of the domain in which the program operates.

The difference in viewpoint between educationalist and computer scientist can call the educational value of computer-supported constructionism into question. For instance, in rule-based programming in the context of a microworld (Goldstein *et al*, 2001), the educationalist sees value in engaging a bright pupil in discussion of whether a particular rule should be attached to one object or another. The computer scientist, by contrast, recognises the kind of uncomfortable pragmatic decision that is typically encountered in thinking about applying programming paradigms; decisions for which the lack of principled grounds for judgement underline the very disconnection of program from domain understanding that computer science seeks to avoid.

In fact, the formal computer scientist's dream of building software from theories is far from being realised in practical computing. On the contrary, as critics such as Brian Cantwell-Smith (2002) have argued, theoretical computer science is most ill-suited to accounting for contemporary computing practice. What is even more discomforting for computer science as it is presently understood is that – whilst much practice remains unconvincing and incoherent – some aspects of practice deliver results unanticipated and unexplained by computational theory. In particular, classical thinking about computation has little relevance for one of the most widely used and powerful techniques for computer based model-building – modelling with spreadsheets. It is indicative of how far practical experience has outstripped theoretical explanation in computing that Baker and Sugden

(2003) conclude their extensive review of the applications of spreadsheets in education by observing: "There is no longer a need to question the potential for spreadsheets to enhance the quality and experience of learning that is offered to students."

It is against the background of such highly confused and contradictory visions for making sense of the relationships between mathematics, computing and education that EM has been conceived. In EM, the aspiration is to develop principles and tools that can support computer-based model-building that is intimately connected with domain learning. The precedents for EM are drawn not from traditional computer programming or formal mathematics, but from other disciplines where practical activities have a well-developed role, such as laboratory sciences, engineering design, the humanities and the arts. In these domains, practice also takes mature forms, more difficult to formalise than mathematical model-building, but established on far sounder conceptual foundations than computer programming. Consider for instance, the 'scientific method', architectural design and musical analysis.

A key observation is that, though the association of mathematics education with computer programming and with constructionism is very natural, it is also potentially misleading. Because both mathematics and conventional computer programming operate with precise and abstract concepts, traditional computer programming can offer good support for mathematical model-building in some respects. In the Beam Detector model, many of the functional relationships that feature in definitions use simple mathematical operators whose implementation requires relatively straightforward procedural code. However, constructionism is not essentially about precise and abstract concepts; on the contrary, it is motivated by the desire to engage with pre-articulate experience and tacit knowledge that is made accessible only through exposure to situations. Whilst the educator is able to envisage imaginative ways of introducing mathematics into the world

of experience (cf. the e-Muse microworlds), classical computer programming – with its roots in logic and abstract computation – is a reluctant fellow-traveller.

In keeping with the features of constructionism identified earlier in the chapter, EM typically entails blending the formal and informal. As the discussion of the Beam Detector has shown, EM can support this integration of the formal and the informal without in any way compromising its own integrity. In this respect, it resembles the 'scientific method', which is fundamentally concerned with interaction in the world, yet (in the context of the school science laboratory, if not necessarily in its more authentic setting of the research laboratory) is typically exercised in conjunction with abstract theoretical understanding. EM is also distinguished from mathematics and from computer programming as they are conventionally conceived, in that its characteristic movement is from the informal to the formal (cf. Beynon *et al*, 2000), rather than from the formal to the informal.

The adoption of an EM perspective on computer-based model-building involves a switch of priorities where pre- and post-theory understanding is concerned. In the context of mathematics education, this is consistent with the "**re-evaluation** of the concrete" to which Turkle and Papert (1991) refer in their discussion of constructionist practices. Such a shift in perspective also has a philosophical aspect concerned with whether we take a Platonistic or intuitionist view of the foundations of mathematics (Goodman, 1994). Where the Platonist is merely setting formal ideas in the context of concrete experiences in order to make them more accessible (cf. the discussion of characteristic features of constructionism earlier in the chapter), the intuitionist regards their very meaningfulness as contingent at some level upon experience. Of the "two versions of constructivism" in the foundations of mathematics alluded to by Goodman (1994), EM seems best aligned with what Goodman describes as "Among philosophers the most influential

contemporary version of constructivism ... the intuitionism of Michael Dummett (1977)". Dummett's intuitionism follows "an essentially Wittgensteinian philosophy of language: to understand mathematics is primarily to understand mathematical speech, the meaning of which must be constituted by its use". More specifically, Dummett advocates a version of phenomenalism with respect to which: "The phenomenalist ... must interpret the sentence 'the book is on the table' by explaining what sense experiences would justify the assertion of the sentence". Also relevant in this context is Goodman's observation (1994) that, according to the mathematical empiricism of Lakatos: "... mathematics is expounded in an order almost the reverse of that in which it is discovered".

CONCLUDING REMARKS

This chapter has argued that established thinking about the nature of computer programming and its relation to formal mathematics is obstructive where enhancing computer support for constructionism is concerned. In our view, the use of EM to build construals can better approach the ideal of integrating the roles of learner, teacher and developer to which constructionism aspires. To fully understand the prospects and implications for EM in this respect requires a more mature and coherent understanding of the relationship between mathematics, education and computing than we have at present.

Shifting the emphasis away from mathematical model-building based on pre-existing theory echoes the philosophy of engineering developed by Vincenti (1990). Vincenti characterises engineering as a species distinct from applied science, where there is a role for *blind variation* – interaction 'without complete or adequate guidance' potentially leading to discovery. When seeking to support personal engagement and creativity in learning, the motivation for a perspective of model-building in which there is no

preconceived and fixed framework for interpretation is clear. Modelling activity that enables us to manage cognitive conflict and construct new meanings to resolve such tensions is an essential foundation of constructionist learning. It will not prove easy to gain full acceptance for such an approach to modelling, as it superficially appears to encourage just such practices – experiment without abstract specification, exploration without preconception – as are deprecated in conventional software development. Helpful precedents are to be found in existing modelling tools that exploit dependency, like spreadsheets and engineering design packages. Effective model-building to support learning demands something conceptually much more radical than merely adding dependency to the arsenal of conventional programming techniques however – in this connection, EM principles are vital in helping to discriminate between emerging understanding and incoherence in interaction.

Where e-learning is concerned, the general application of constructionist principles will require a model-building approach that can be adapted to a much broader range of disciplines. The range of topics addressed in the EM repository already indicates that EM has much wider potential application than has been illustrated in this chapter. Some preliminary thinking about how EM might be applied in modelling for the humanities is further described in McCarty, Beynon and Russ (2005).

An interesting development in the use of the web for learning is that adopted by the WebLab project (Weblabs). The WebLab portal has been designed to encourage children to share their projects, written in ToonTalk (Kahn 1996), with other children both local and remote. Such sharing involves posting a project onto the website, commenting directly on other peoples' projects, running projects directly on the web, and downloading them to allow re-programming in ToonTalk.

The Weblab project clearly highlights the enormous potential for collaborative e-learning within a constructionist framework, but the arguments advanced in this chapter suggest that it will be exceptionally difficult to deliver to this potential with the chosen programming paradigm. In the longer term, we believe that EM will prove far more effective at meeting the challenge of implementing the kind of interaction envisaged in the WebLabs project. For this purpose, we would favour interaction through multi-user – potentially concurrent – redefinition in scripts, such as has been illustrated in distributed EM models, and is commended for collaborative Web based modelling in Cartwright *et al* (2005).

ACKNOWLEDGEMENTS

We are indebted to Dave Pratt and Ian Jones for their reflections on the e-Muse project and to three anonymous referees for their helpful comments.

REFERENCES

Baker, J., Sugden, S. (2003). Spreadsheets in Education: The first 25 years. In Baker, J., Sugden, S. *Spreadsheets in Education e-Journal*, Vol. 1, No. 1, 18-43.

Bannan-Ritland, B., Dabbagh, N., Murphy, K. (2002). Learning Object Systems as Constructivist Learning Environments: Related Assumptions, Theories and Applications. In D.A.Wiley (Ed.) *The Instructional Use of Learning Objects: Online Version.* Available at http://reusability.org/read/chapters/bannan-ritland.doc

Beck, K. (2000). eXtreme *Programming explained.* Addison Wesley.

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

Beynon, W.M. (1997). Empirical Modelling for Educational Technology. In *Proceedings of Cognitive Technology 1997*, 54-68, University of Aizu, Japan, IEEE.

Beynon, W.M., Rungrattanaubol, J., Sinclair, J.E. (2000). Formal specification from an observation-oriented perspective, *Journal of Universal Computer Science*, Vol. 6 (4), 407-421.

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

CABRI geometry. http://www.cabri.com/web/nsite/html/home.html.

Cantwell-Smith, B. (2002). The Foundations of Computing. In Scheutz, M., *Computationalism: New Directions*, MIT Press.

Cartwright, R., Adzhiev, V., Pasko, A., Goto, Yuichiro., Kunii, Tosiyasu.L. (2005, to appear) Web-based shape modelling with HyperFun. *IEEE Computer Graphics and Applications*.

diSessa, A. (1988). Knowledge in pieces. In G. Forman, P. Pufall (Eds.), *Constructivism in the Computer Age*, Hillsdale, NJ: Lawrence Erlbaum Assoc., 49-70.

Dummett, M.E. (1977). *Elements of Intuitionism*, Oxford: Clarendon Press.

Edwards, D., Hansom, M. (1989). *Guide to Mathematical Modelling*, Boca Raton, FL: CRC Press.

Ehrmann, S. (2000). Technology & Revolution in Education: Ending the Cycle of Failure. *Liberal Education*, Fall, 40-49.

Empirical Modelling: Model Repository. http://empublic.dcs.warwick.ac.uk/projects

Empirical Modelling Website. http://www.dcs.warwick.ac.uk/modelling

e-Muse Project: e-learning for museum and schools environments, http://emuse.cti.gr. EC e-learning initiative: 2002-4084/001-001 EDU-ELEARN.

Eppstein, D. (1998). The Geometry Junkyard: http://www.ics.uci.edu/~eppstein/junkyard/beam.

Goldstein, R., Noss, R., Kalas, I & Pratt, D. (2001). Building Rules, in Beynon, W.M., Nehaniv, C.L., & Dautenhahn, K. (Eds.), *Proceedings of the 4th International Conference of Cognitive Technology CT2001*, 267-281. University of Warwick, Coventry: UK.

Gooding, D. (1990). *Experiment and the making of meaning*, Kluwer Academic Publishers.

Goodman, N.D. (1994) Some current positions in the philosophy of mathematics, in Grattan-Guinness, I. *A Companion Encyclopedia of the History and Philosophy of the Mathematical Sciences*, Routledge, London

Grossman, T.A. (2002) Spreadsheet Engineering: A research framework. In *Proceedings EUSPRIG 2002*, 23-34, 18th-19th July.

Harel, I., Papert, S. (Eds.) (1991), *Constructionism*, Ablex, Norwood, New Jersey.

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

Kalas, I., Blaho, A. (2000). Imagine… New Generation of Logo: Programmable pictures. In *Proceedings of WCC2000*, Beijing, 427-430.

Kahn, K. (1996). ToonTalk™ – An animated programming environment for children. *Journal of Visual Languages and Computing* 7, 197-217.

Lakatos, I. (1976). Proofs and Refutations. In Worrall, J., Zahar, E. *The Logic of Mathematical Discovery*, Cambridge University Press.

Levi-Strauss, C. (1968). *The savage mind*. University of Chicago Press.

McCarty, W., Beynon, W.M., Russ, S.B. (2005). Human Computing: Modelling with Meaning, in *Proc. ACH/ALLC Conference 2005*, Vancouver (to appear)

Nardi, B. (1993). *A small matter of programming: Perspectives on End User computing*. MIT Press.

Noss, R. Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers.* Dordrecht: Kluwer.

Papert, S. (1972). XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*, New York: Basic Books.

Papert, S. (1993). *The children's machine*. New York: Basic Books.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations, *International Journal of Computers For Mathematical Learning*, 1(1), 95-123.

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

Papert, S., Watt, D., di Sessa, A., Weir, S. (1979). *Final report of the Brookline Logo Project: Parts 1 and 11* (Logo Memos Nos. 53 and 54). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Pratt, D. (2000), Making sense of the total of two dice, *Journal for Research in Mathematics Education*, 31 (5), 602-625.

Roe, C. (2003). *Computers for learning: An Empirical Modelling perspective*. PhD Thesis, Department of Computer Science, University of Warwick.

Roe, C., Pratt, D., Jones, I. (2005, to appear) Putting the *learning* back into e-learning. *Proceedings of CERME4*, Spain, February 2005.

Sannella, D. (1997). 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, 15-19, Springer.

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

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

Turkle, S., Papert, S. (1991). Epistemological Pluralism: Styles and Voices within the Computer Culture. In Harel, I., Papert, S. (Eds.) *Constructionism*. Norwood, N.J. Ablex Publishing Corp, 161-191.

Turski, W.M., Maibaum, T.S.E. (1987). *The Specification of Computer Programs*, International Computer Science Series, Addison-Wesley.

Vincenti, W. (1990). *What engineers know and how they know it: analytical studies from aeronautical history*, Johns Hopkins Studies in the History of Technology, Johns Hopkins University Press.

Watt, D. (1979). Final Report of the Brookline Logo Project Part III: Profiles of Individual Student's Work, *Logo Memo No. 54*, MIT, 4.10 - 4.17.

Weblabs project website: http://www.weblabs.eu.com/