An Introduction to CS405 “Introduction to Empirical Modelling”

Meurig Beynon
Reader Emeritus in CS

Theme for 2013-14

Towards a new conceptual framework for software development

Orientation

To what extent have the problems that underlie the 'software crisis' been resolved? And can current principles and tools for software development meet the additional challenges that contemporary use of computers present?

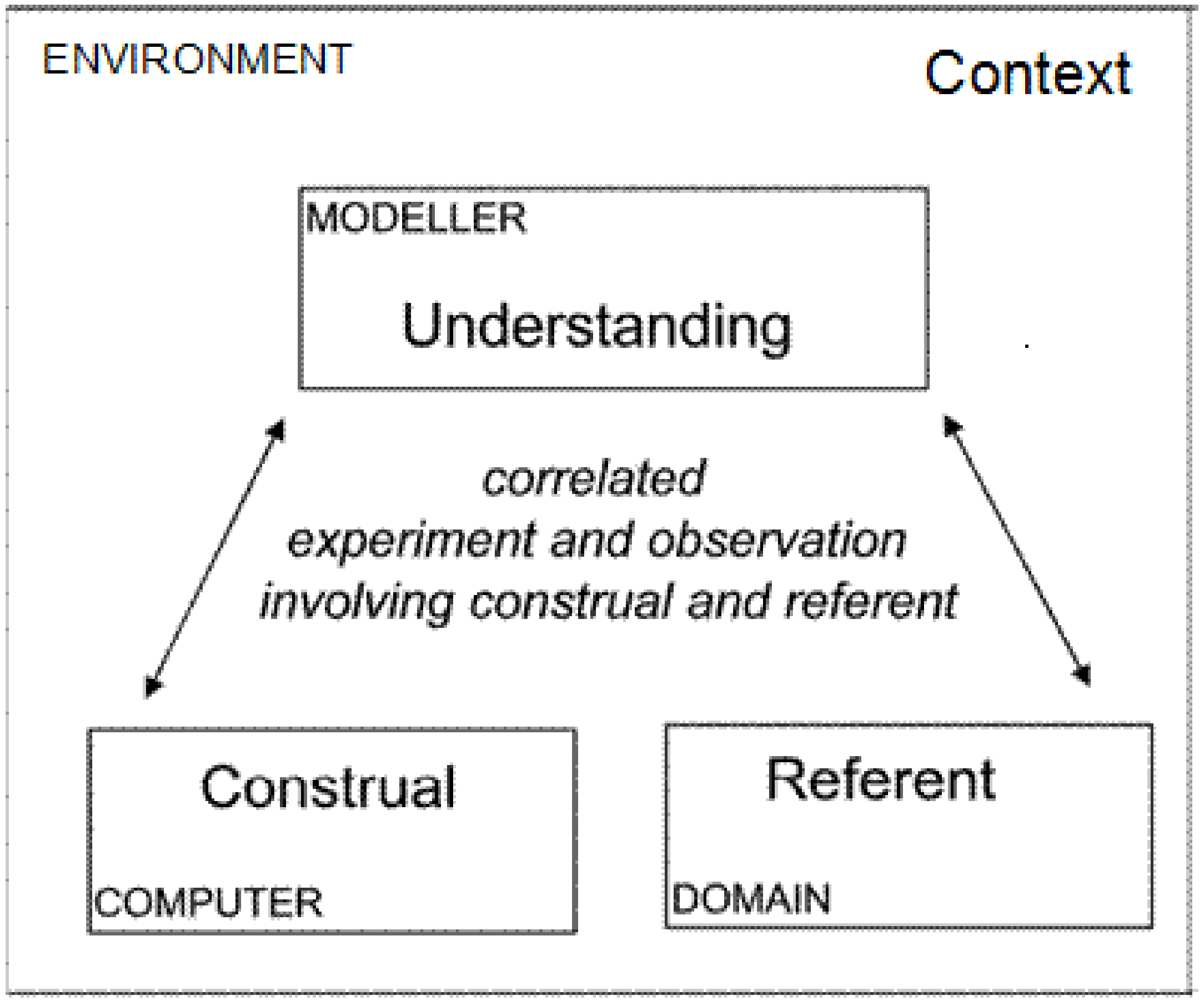
The problems underlying the software crisis are unresolved. Current principles and tools for software development are inadequate to meet the challenges of contemporary use.

Construals

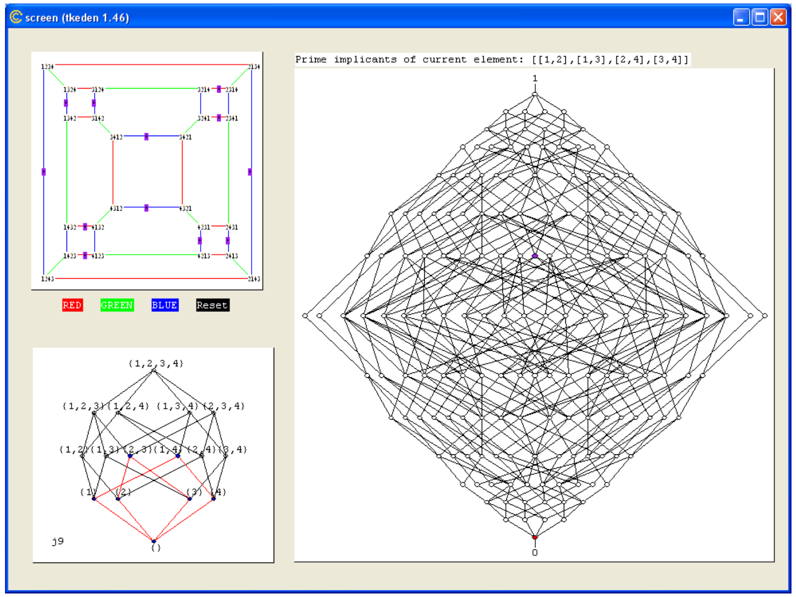
Do we / why do we need a science of computing that is broader than 'computational thinking' supports? And what alternative core focus is there for computer science other than programming, logic and algorithms?

We need a science of computing that is broader than ‘computational thinking’ that supplies a better account of the experiential dimension and sense-making aspect of computing

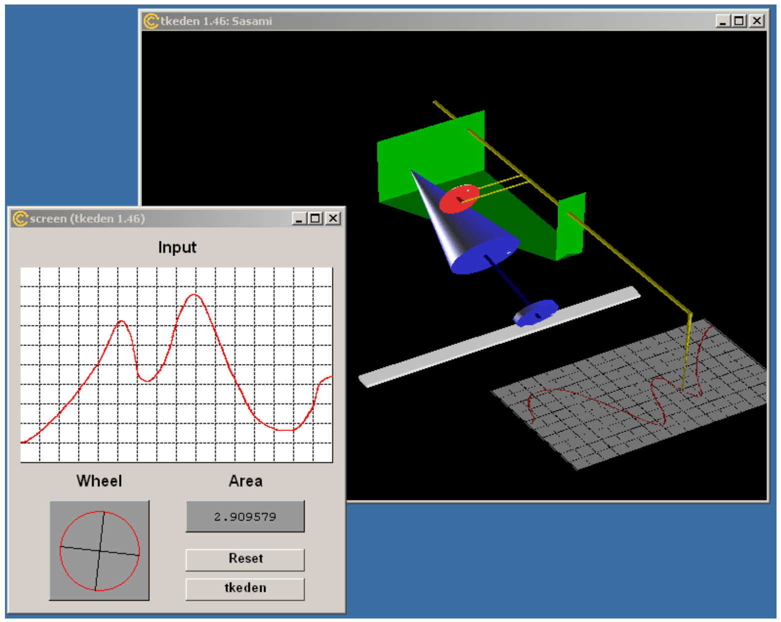
Empirical Modelling as *Construction*

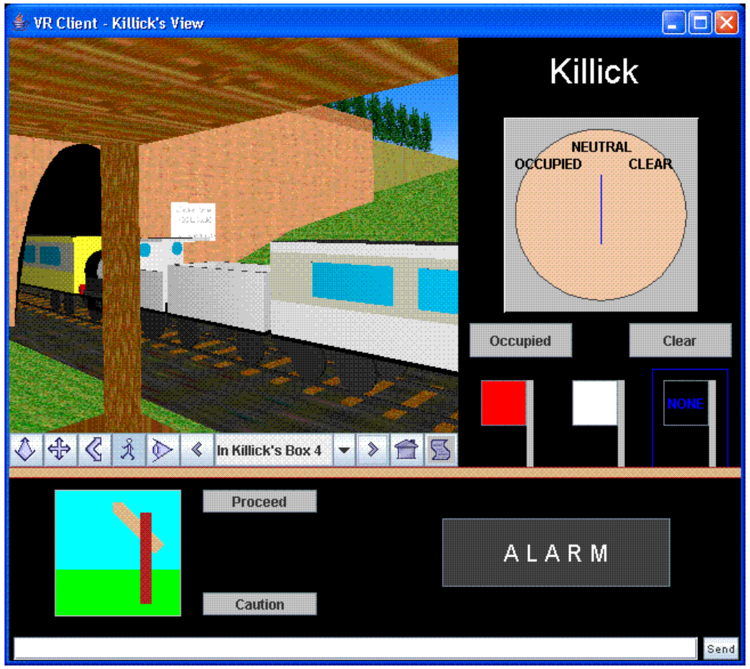


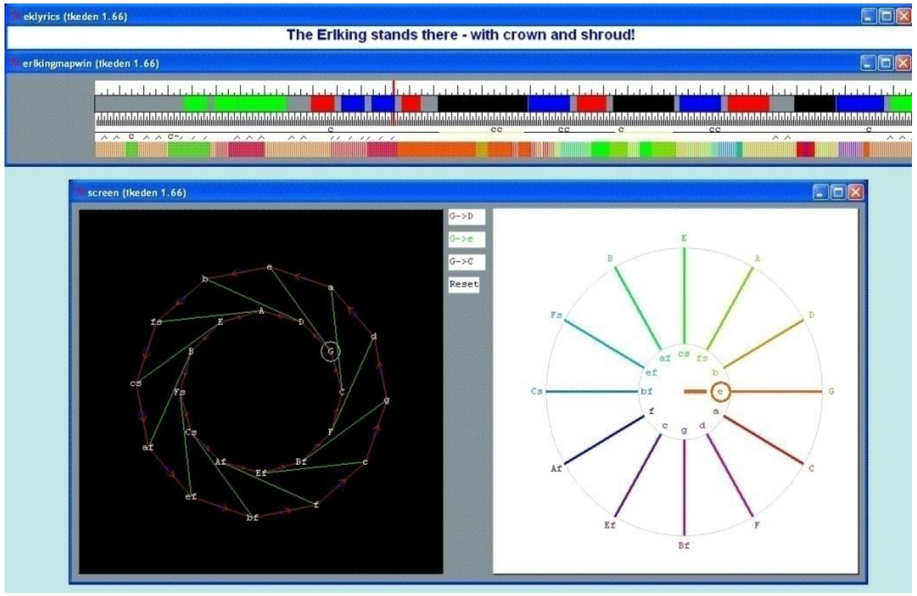
Sense-making in mathematics, in the physical world, social interactions and music ...



Prime implicants of current element: [[1,2],[1,3],[2,4],[3,4]]







The Erlking stands there - with crown and shroud!

Making construals

What principles and tools are best suited to making construals? What role can the computer play in their construction?

Construals are interactive digital artefacts that embody configurations of *observables*, *dependencies* and *agency* encountered in the situations to which they refer (cf. spreadsheets). Computer technology enables the essential visualisation / perceptualisation and interaction.

Programming paradigms

Is every principled way to use the computer just a different style of programming a Turing machine? Is it possible to resolve the problems of software development by integrating different styles of programming, and why has this proved to be so difficult?

There is more to computing than programming Turing machines. Programming paradigms fail to account for how computing is experienced.

Formalism

What is the role and potential for formal specification and verification in software development? Is there any alternative basis on which the quality of software can be assured?

The role of formal methods cannot be dissociated from the identification of machine-like environments for computation. Quality assurance for software must in general rely on the practices observed in engineering.

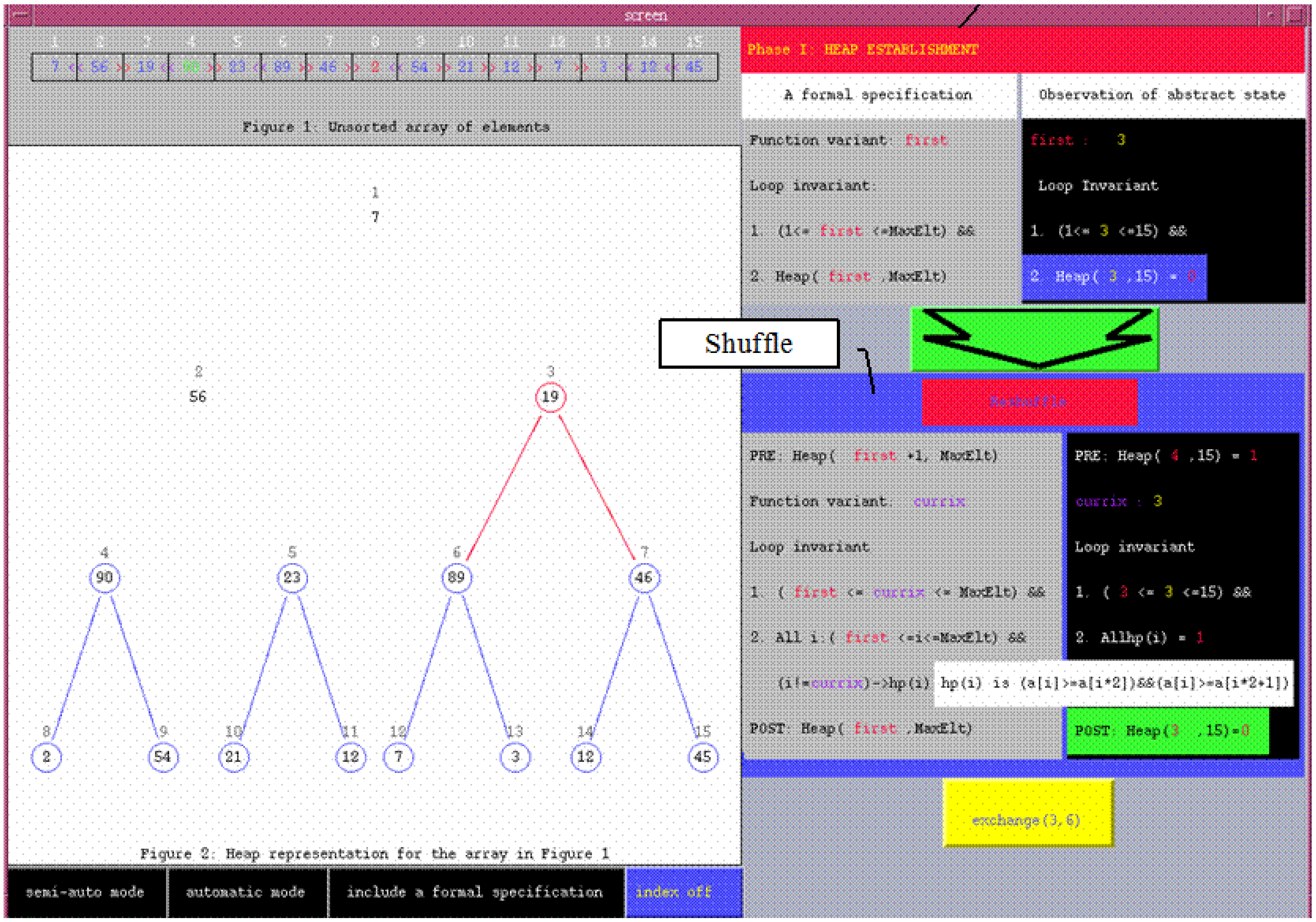


Figure 1: Unsorted array of elements

Figure 2: Heap representation for the array in Figure 1

Phase I: HEAP ESTABLISHMENT

A formal specification	Observation of abstract state
Function variant: first	first : 3
Loop invariant:	Loop Invariant
1. (1<= first <=MaxElt) &&	1. (1<= 3 <=15) &&
2. Heap(first ,MaxElt)	2. Heap(3 ,15) = 0

PRE: Heap(first +1, MaxElt)	PRE: Heap(4 ,15) = 1
Function variant: currix	currix : 3
Loop invariant	Loop invariant
1. (first <= currix <= MaxElt) &&	1. (3 <= 3 <=15) &&
2. All i:(first <=i<=MaxElt) &&	2. Allhp(i) = 1
(i!=currix)->hp(i)	hp(i) is (a[i]>=a[i*2])&&(a[i]>=a[i*2+1])
POST: Heap(first ,MaxElt)	POST: Heap(3 ,15)=0

Formal specification from an observation-oriented perspective

Development process

To what extent can techniques such as agile development and scripting resolve the problems of meeting users' evolving requirements? Can the demands of users of the web, mobile devices and embedded systems for ways to customise applications flexibly in the stream-of-thought be addressed within the conceptual framework of software development as currently understood?

The key issue is: how can we take account of the immediate experience of developers and users?

Traditional programming

Requirements capture and specification	specification	Program design implementation maintenance	user interface	**Use** affordances interface culture
Identifying agency in the machine-like components and in the human context for use		*constructing and programming the machine-like components*		*human factors study*
				interface design
Framing goals for the design protocols for interaction and interpretation		*designing program by identifying objects and functions*		*empirical studies of use*
				prototyping
		technical interface development		*e.g. goals, operators, methods (GOMS) evaluation*
e.g. devise UML		*e.g. writing Java code*		

Empirical Modelling

Requirements capture and specification	Program design implementation maintenance	**Use** affordances interface culture
develop scripts in isolation as "furry blobs" that represent the observables and dependencies associated with putative machine-like components and human interactions and interpretations	*identify and document reliably reproducible sequences of redefinition / chains of "furry blobs" that correspond to programmable automatable machine behaviours and ritualisable human behaviours and interfaces*	*exercise, explore, customise, revise and adapt sequences of redefinition and interpretation to reflect emerging and evolving patterns of interaction and interpretation; extend and augment observables to support additional functionalities combining scripts*

Learning

How well is current thinking about software development suited to the constructionist goal of establishing an intimate link between development and domain learning? What can be learned from parallel research in educational technology?

Procedural thinking within a computational framework is ill-suited to a constructionist stance, as is corroborated by problems encountered in developing effective educational technology.

private experience / empirical / concrete

interaction with artefacts: identification of persistent features and contexts

practical knowledge: correlations between artefacts, acquisition of skills

identification of dependencies and postulation of independent agency

identification of generic patterns of interaction and stimulus-response mechanisms

non-verbal communication through interaction in a common environment

directly situated uses of language

identification of common experience and objective knowledge

symbolic representations and formal languages: public conventions for interpretation

public knowledge / theoretical / formal

An Experiential Framework for Learning (EFL)

Concurrency

How well do current software development techniques address the need to represent and reconcile the perspectives of all the state-changing agents in a complex system?

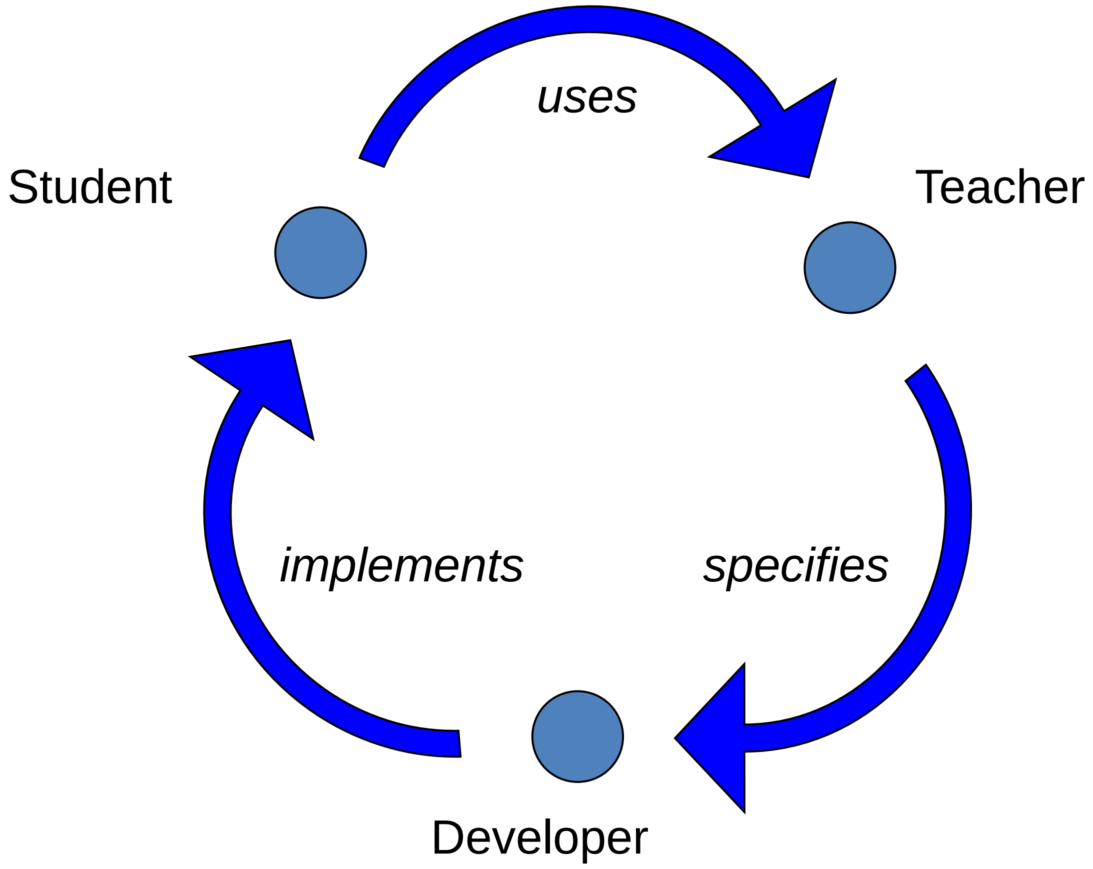
Computational thinking is predicated on objective rational perspectives that are not in general appropriate. Dealing with subjectivity and inter-subjectivity demands radically different ways to conceive and represent state.

Multi-agent development

How can we integrate the design activities of the many different participants in complex software system development? How can we achieve conceptual integrity? How can we best support distributed participatory design?

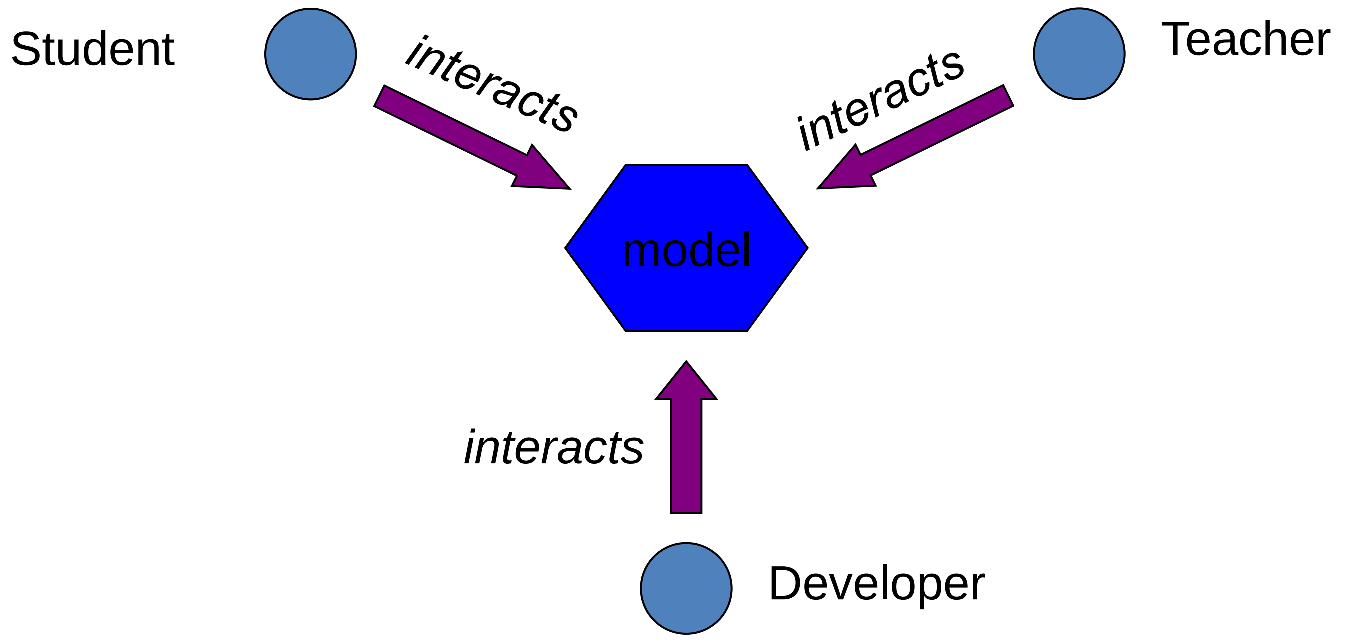
We need to dissolve the dualities between designing, implementing and running software so that all the participants interact in what is conceptually the same environment.

Developing educational software



Empirical Modelling (EM)

- Offers a set of principles for model building in any of the student, teacher and developer roles:



Retrospect and prospect

How far does the reconceptualisation of computing that making construals enables address the challenges of software development? What further development of the principles and tools is needed?

In principle, *making construals* has great promise and potential to change software development in a radical way. In practice, it needs much further exploration and investment building on proof-of-concept tools.