Notional Machines in Computing Education: The Education of Attention

Sally Fincher*
School of Computing
University of Kent
Canterbury, Kent, UK
S.A.Fincher@kent.ac.uk

Johan Jeuring*
Department of ICS
Utrecht University
Utrecht, The Netherlands
J.T.Jeuring@uu.nl

Craig S. Miller*
School of Computing
DePaul University
Chicago, IL, USA
cmiller@cs.depaul.edu

Peter Donaldson
Computing Science
University of Glasgow
Glasgow, Scotland, UK
peter.donaldson.2@glasgow.ac.uk

Benedict du Boulay
Department of Informatics
University of Sussex
Brighton, Sussex, UK
b.du-boulay@sussex.ac.uk

Matthias Hauswirth
Software Institute
Università della Svizzera italiana (USI)
Lugano, Switzerland
matthias.hauswirth@usi.ch

Arto Hellas
Department of Computer Science
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Felienne Hermans
LIACS
Leiden University
Leiden, The Netherlands
f.f.j.hermans@liacs.leidenuniv.nl

Colleen Lewis
Department of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL, USA
colleenl@illinois.edu

Andreas Mühling
Institute for Computer Science
Kiel University
Kiel, Germany
andreas.muehling@infomatik.uni-kiel.de

Janice L Pearce
Department of Computer Science
Berea College
Berea, KY, USA
pearcej@berea.edu

Andrew Petersen
MCS Department
University of Toronto Mississauga
Mississauga, Canada
andrew.petersen@utoronto.ca

ABSTRACT

This report defines notional machines (NMs), and provides a series of definitional characteristics by which they may be identified. Over several sections, it includes a first-hand report of the origin of NMs, reports a systematic literature review to track the use and development of the concept, and presents a small collection of examples collected through interviews with experienced teachers. Additionally, the report presents NMs in a common format, and makes some preliminary explorations of their use in practice, including examples of instructors using multiple NMs in sequence. Approach and method are fully detailed in evidential appendices, to support replication of results and adoption/adaptation of practice.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

Notional Machines; Computing education

ACM Reference Format:
Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L Pearce, and Andrew Petersen. 2020. Notional Machines in Computing Education: The Education of Attention. In *2020 ITiCSE Working Group Reports (ITiCSE-WGR '20), June 17–18, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 30 pages. https://doi.org/10.1145/3437800.3439202

*Working group co-leaders

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ITiCSE-WGR '20, June 17–18, 2020, Trondheim, Norway
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8293-9/20/06…$15.00
https://doi.org/10.1145/3437800.3439202

1 INTRODUCTION

Originating in the work of perceptual psychologist Eleanor J. Gibson, there is a thread of work examining the phenomenological nature of teaching and learning, not always in formal situations. So the parent points to the place on the paper where a child must write the answer to a sum [25]; surgeons gesture to indicate important anatomical areas to medical students [40]; archaeologists kneel together at the site of an excavation to feel and discuss the nature of soil [24]; a young hunter is taken to the forest, told what to look out for, and shown "subtle clues he may otherwise fail to notice" [36, p. 37].

How does a novice programmer (or an increasingly expert programmer) know "where to look?"

In the teaching and learning of programming, the artefacts (code) are symbolic and difficult to apprehend. It is by no means obvious

when looking at some code what the code actually does or even, indeed, what it is meant to do. Many things that are important (language, compiler, libraries) are not included with it, are not visually represented; things that a programmer just has to "know."

So, for computing education, the process of drawing a novice's attention to the things that matter is complex. One of the ways that educators have found to address this complexity is in the creation of notional machines (NMs). NMs are representations or analogies that put a spotlight on those things that are important to look at, or that do something that makes apparent otherwise invisible behaviour which, if un-noticed or misunderstood, would cause the learner to go hopelessly wrong.

The key here is Gibson's identification of one of the problems of learning: "Not all of the available information is relevant for the task at hand … The key to perceptual learning is the *education of attention*—learning which variables to attend to and which to ignore." [1] (emphasis added).

Gibson worked only at the level of perceptual learning, but she suggested (and others have generalised) that the concept applies to education more widely [50, p.81].

For the domain of computing education, we claim NMs as a primary engine for the education of attention.

What is a notional machine?

A notional machine (NM) is a pedagogic device to assist the understanding of some aspect of programs or programming.

An NM has a *pedagogical purpose*, its generic function is to *draw attention to, or make salient, some hidden aspect* of programs or computing. It will have a *specific focus* within programs or computing, and will adopt a *particular representation* that highlights specific aspects of the focus.

Pedagogical Purpose: The purpose of an NM is for use in teaching to support student learning of computational concepts. A crucial aspect of an NM is that it should simplify an actual concept or skill as an aid to understanding.

Function: The generic function of an NM is to uncover something about programming, computers or computation, or to draw attention to something, that is not obviously apparent in the artefact the student is using.

Focus: An NM typically focuses on a particular aspect of programs and their behaviour. As well as programs, an NM's focus can also be concerned with computers as places where programs can be built, run, and stored.

Representation: An NM will have a representation and this representation will draw attention to certain aspects of the focus and possibly ignore others.

The aim of this working group was to explore the history, meanings, uses and value of NMs as aids to teach about programming and computers. There were four objectives:

- To conduct a literature review, to ground and inform the work;
- To capture examples of NMs in use;
- To catalogue them in a common scheme; and,
- To arrange them in clusters or sequences.

At this time we solely considered NM that teachers use. We did not consider NMs that students may construct to explain things to themselves (or each other); neither did we look at teaching strategies designed to support students' metacognitive skills in developing and representing their own NM. These, and other promising avenues of investigation, remain for future work.

This report is divided into six further sections. Section 2 outlines the origin of the term notional machine within its historical context and disentangles the meanings of the terms notional machine, conceptual model and mental model. Section 3 describes a systematic literature review that we undertook on the topic of NMs. Section 4 draws on the literature review to clarify the current meaning and characterisation of NMs. Section 5 describes an illustrative subset of NMs that we collected via interviews, reflective practice and from the literature. Section 6 describes ways in which NMs are used in instruction. Most often, NMs are used as explanatory devices to accommodate the learner's current level of knowledge and avoid unnecessary cognitive load. Section 7 summarises and concludes. There are four appendices providing: (A) a list of the NMs we identified, (B) the systematic literature review process, (C) the literature review extraction spreadsheet, and (D) the interview protocol.

2 SITUATING NOTIONAL MACHINES: ORIGINS AND THEORY

This section looks at the origin of the term notional machine in its context of teaching Logo to novice programmers in Edinburgh in the late 1970s. Some of the influences on that work are provided. The section moves on to distinguish the terms notional machine, mental model and conceptual model, as they have been used somewhat interchangeably in the literature.

2.1 The emergence of the idea of notional machines

The specific term notional machine arose in the 1970s following an increasing interest in the psychology of programming, both among novices and experts [see, e.g. 51, 70, 71]. This period also saw the development of high-level languages specifically for teaching novices programming, notably Basic and Logo. Allied to the interest in the psychology of programming, work started to emerge on the pedagogy of teaching programming [see, e.g. 11, 66].

An issue that rapidly developed involved how to make the largely hidden actions of a program understandable and perhaps also visible. The Basic Instructional Program (BIP), for example, provided a visualisation of the execution of a Basic program by highlighting each command as it occurred [7]. Mayer [46] described a Basic program to learners as if it was a sequence of "transactions" involving an "object" and a "location", and Carroll and Thomas [12] argued that an effective way of teaching programming involved providing

learners with carefully chosen metaphors for different aspects of programs and programming.[1]

Building on the work of Mayer [46], du Boulay et al. [17] coined the term notional machine to describe their strategy for teaching Logo to children and teachers. Their historical definition of an NM is: "A notional machine is the idealised model of the computer implied by the constructs of the programming language." This idealised model should be simple enough to learn, but comprehensive enough to build programs to solve problems of interest:

> "Functional simplicity can be achieved by limiting the set of transactions and by ensuring that each instruction does not need too many transactions to describe its action. This aspect of the simplicity of the notional machine must be distinguished from two other aspects of simplicity in a first programming language, namely logical simplicity and syntactic simplicity. Logical simplicity implies that problems of interest to the novice can be tackled by simple programs, i.e. the tools are suited to the job. Syntactic simplicity is achieved by ensuring that the rules for writing instructions are uniform, with few special cases to remember, and have well chosen names." [17, page 238]

Their initial attempt at creating NMs was embodied in a manual for learning Logo aimed largely at children but also primary school teachers [16] (see Figure 1). This used a variety of hand-drawn representations and analogies, including that of a "worker" for commands and functions whose ears "heard" parameter values, whose mouth "spoke" outputs, and whose hands carried out actions. This representation was gradually built up to explain built-in commands, user-defined procedures and functions, sub-procedure calls and recursion.

At that time, the children were working on very noisy Model 33 Teletypes, with what they typed printed on a roll of paper and turtle movements drawn by either a mechanical turtle on the floor, a graph-plotter or a visual display. Note that there were no screens for the users with a desktop or windows or icons to represent the computer. The actual computing system consisted of a laboratory containing the input and output devices, a nearby room containing a mini-computer to control those devices and a mainframe in the basement to run Logo. So there was a need to provide some simplified sense of the computer itself, and this consisted of a hand-drawn representation in the manual of where their user-defined procedures were "inside" the computer via the differences between "working memory" and "long-term storage". The former was the place where newly built user-defined procedures were stored and could be run; the latter was where user-defined procedures were stored in between sessions so long as they had been explicitly "saved". Given that the researchers had some control over the naming of primitives in the version of Logo that they used, they were able to choose what they hoped were more understandable names than the standard ones at the time in order to create a sense of unity between the manual, the names of the primitives in Logo and the terminology that they used in teaching.

[1] For a much more detailed account of the early research into learning and teaching programming, see Guzdial and du Boulay [28].

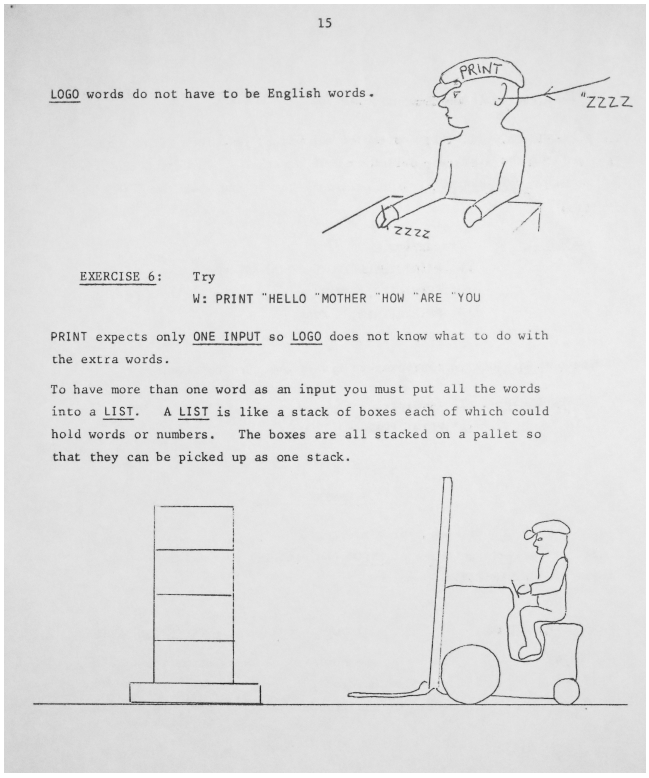


Figure 1: A page from the 1976 LOGO manual

Under the guidance of Jim Howe at Edinburgh, du Boulay worked with trainee primary school teachers who needed to have a better understanding of mathematics, and O'Shea worked with children from a local school. Their aim was to teach mathematics through Logo and evaluate any increases in mathematical understanding and motivation. They did not set out to evaluate how far the pedagogy based on NMs was effective as compared to some other pedagogy. They had some success in the main goal:

> "For example, we have worked with trainee teachers weak in mathematics, and shown that writing LOGO programs to explore troublesome topics can promote understanding of the underlying mathematics. However, the intrusiveness of the programming activity could frequently distract the teachers' attention from mathematical issues. . . other studies [with children] at Edinburgh suggest that computer modelling can improve both the maths performance of some underachievers, and their ability to talk about mathematics." [35, page 244]

What they did find was that, despite their best efforts, Logo was harder to learn for some trainee teachers and some children than they had expected.

2.2 Mental models, conceptual models and notional machines

With the rapid development of cognitive science and mental models in the 1980s, the more general notion of a "conceptual model" evolved. For instance, Greca and Moreira [26] contrast the classic theoretical literature on mental models and knowledge-representation in general [e.g. 37] with the educational literature on the mental models of learners and teachers [e.g. 6].

In the case of education, Greco and Moreira distinguish the mental models of learners from the scientifically informed understandings of teachers' conceptual models. They characterise the difference between conceptual models and mental models as follows:

> ". . . conceptual models are precise and complete representations that are coherent with scientifically accepted knowledge. That is, whereas mental models are internal, personal, idiosyncratic, incomplete, unstable and essentially functional, conceptual models are external representations that are shared by a given community, and have their coherence with the scientific knowledge of that community. These external representations can materialize as mathematical formulations, analogies, or as material artifacts." [26, page 5]

They quote Barquero [6] who characterises learners' mental models as

> "a type of knowledge representation which is implicit, incomplete, imprecise, incoherent with normative knowledge in various domains, but it is a useful one, since it results in a powerful explicative and predictive tool for the interaction of subjects with the world, and a dependable source of knowledge, for it comes from the subjects' own perceptive and manipulative experience with this world."

Much more recently, Seel [59] goes further in that he offers a theoretical account of using models (in general) in teaching and describes a number of pedagogic strategies for accomplishing this kind of teaching. He also points out that models and conceptual models can model processes and procedures as well as static relations, so that, for example, the teacher can answer questions such as "what would happen to inflation if the Bank of England prints lots of money?" or "what would happen if you pressed the accelerator and the brake on a car at the same time?" Ideally, the learner's mental model will develop to also be able to answer these 'what if' questions by mentally simulating the processes or mechanisms being mastered.

So where do NMs fit into this picture? It seems that conceptual models are one kind of model and that an NM is effectively a special kind of conceptual model. A characteristic of NMs is that they represent something that can be interacted with, even if just mentally, in other words a machine. So although the term conceptual model often implies a declarative model, this is not a necessary feature. They are created in the context of teaching computing (in general) by teachers as pedagogic devices to help learners understand a simplified version of a conceptual model (see Figure 2). An NM may

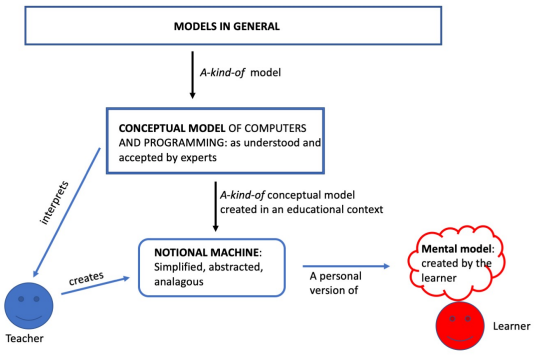


Figure 2: The relationships between models in general, conceptual models, NMs and mental models. Models in General refers to the universe of models of all kinds. The black arrows indicate 'a-kind-of relationship'. The blue arrows indicate a timeline of development. The leftmost blue arrow indicates the teacher's interpretation of the expert conceptual model.

remove unnecessary detail, may abstract details into broader concepts, and will often make use of analogies that provide scaffolding to help understanding.

The learner's (personal) mental model will initially be 'incomplete, imprecise, incoherent,' as indicated above and may or may not cohere, first towards the NM offered by the teacher and perhaps later towards the more complex, generally accepted technical conceptual understanding of the computing phenomenon 'caricatured' in the NM.

Since their original identification and definition, the concept of NMs has been used and refined by many other researchers and practitioners. For example, Krishnamurthi and Fisler [41] describe the interaction of notional machines and programming paradigms, suggesting that "While notional machines are usually viewed as a tool for learning to write and trace programs, they are also a useful way for us to think about language classification: essentially, the similarity between two languages is the extent to which a notional machine for one gives an accurate account of the behaviour of the the other." In the remainder of this document we trace their appearance in the literature. We also capture and present rich examples of their use in practice.

3 NOTIONAL MACHINES IN THE LITERATURE

The notional machine idea has been adopted, refined, and in some cases re-developed in a number of different areas of computing education over the past four decades. We conducted a systematic literature review to explore how the term notional machine has evolved and to determine how and where it has been used. Our focus is on instructor-defined NMs – not the NMs and mental models generated by students.

Appendix B provides details on the methods of the systematic literature review, namely identifying, filtering, and processing relevant literature. At a high level, we searched for instances of

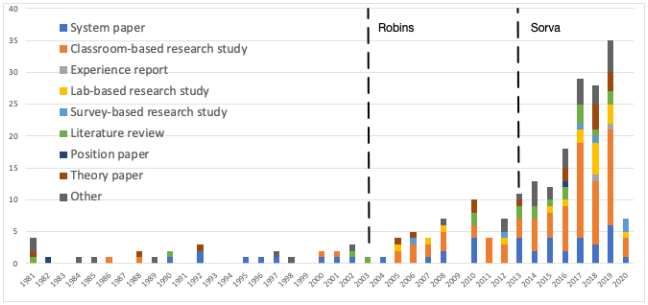


Figure 3: The type of published papers that cite or reference NMs by year. The dotted lines indicate publication of highly-cited reviews that featured NMs prominently.

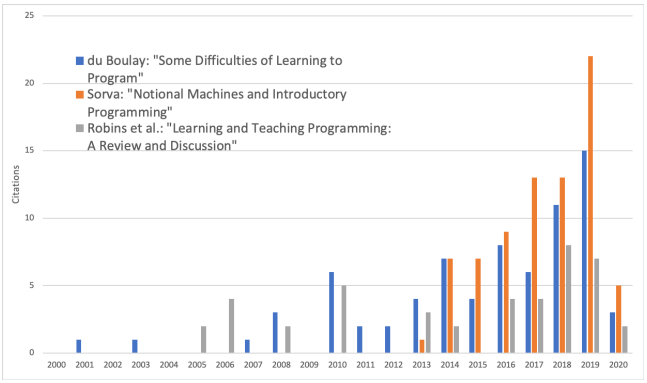


Figure 4: Timeline describing the influence of du Boulay, Sorva, and Robins et al.'s work on the papers being reviewed.

the strings "notional machine" and "conceptual machine" in three databases (the ACM digital library, IEEE Xplore, and Scopus), further including expert-identified articles and articles that were often cited by the identified literature. (The term "conceptual machine" was suggested by a domain expert. See Appendix B for details.) Pairs of reviewers ruled out papers that were unlikely to be relevant to programming using the title and abstract, and then individual reviewers extracted information from each paper using an extraction template that was iteratively refined during the systematic literature review process.

The extraction template is described in Appendix C and includes fields for entering the type of paper (e.g. literature review, research study, etc.), explicit research question, definition of notional machines used, NMs identified, and evaluation performed. A copy of the paper itself was also collected for analysis of the abstract and references. Data from a total of 226 papers was extracted.

In the next subsection, we look at the set of papers, as a whole, to see when papers referring to NMs were published and what might have driven the spread of the term across the discipline. Then, in "How and where are notional machines used?" we consider what role the notional machine concept plays in papers. In "Topics of papers that refer to notional machines", we look at the topics addressed by the papers in our set and identify theories and areas that are well-connected to the NM literature. Finally, we examine how the term notional machine is defined and how the theory is evaluated.

3.1 Timeline of notional machine research

During the extraction process, we classified the papers by type (e.g. literature review, experience report, or system paper). The goal of identifying the type is to qualify where the notional machine concept is being used. Some reviewed papers could have been classified in more than one type, and the reviewer identified the primary goal of the paper. Figure 3 maps the papers that were extracted over a timeline. While the term notional machine was introduced in the early 1980's, with few exceptions, the term didn't start to be broadly incorporated into studies until the mid-2000's. We suggest that the prominence of NMs in recent (2016-2019) works is the product of two catalyzing publications. First, in the early 2000's, Robins, Rountree, and Rountree [56] featured the concept of virtual machines prominently in what became a foundational (and oft-cited) perspective on computing education. They informally define an NM to be a "model of the computer as it relates to executing programs" while also, later, quoting du Boulay's 1986 definition [15]. Second, and even more significantly, in 2013, Sorva et al. published a review of the role of NMs in computing education that argued for greater awareness and adoption of NMs in research and in practice [63]. Sorva et al. [63] defines an NM to be "an abstraction of the computer in the role of executor of programs of a particular kind" while also citing du Boulay's 1986 paper. They also note that several NMs, at different levels of abstraction, may describe the execution of a single program [63]. To get a sense of the influence of these papers, consider Table 1, which lists the ten papers cited most frequently by the works we reviewed. (Note: Not all of these papers focus on instructor-defined NMs; they are just the most cited references from within our set.) Appendix B describes the process used to extract these citations. In brief, the process was lossy and only draws from 165 papers published since 2000, so these counts should be treated as both a lower bound and as generating only a relative ordering. It's notable that Sorva's 2013 work [63] is cited as frequently as one of du Boulay's works [15] and more than twice as often as the original paper on the topic [17]. Robins, Rountree, and Rountree's work [56] is the next most cited paper that focuses on NMs.

Furthermore, it's possible that many recent publications derive their understanding of NMs from Sorva's review, rather than from du Boulay's original work. Only 29 papers cited both du Boulay's "Some Difficulties of Learning to Program" and Sorva's "Notional Machines and Introductory Programming Education", leaving 49 citing only one or the other of these authors. Figure 4 provides further evidence, showing that, since its publication, Sorva's paper has been cited more frequently than the original work introducing NMs.

3.2 How and where are notional machines used?

Next, we studied how and when in the identified research literature NMs are used. Figure 5 presents how the concept of notional machines is used in these 226 papers. Some used NMs for more than one purpose (for motivation and in an intervention, for example),

Table 1: Frequency of citation for the papers most commonly referenced from within our review set. The original papers are highlighted in bold, and the papers we propose contributed to its popularity are italicized.

Paper	Citation Frequency	Year of Publication	Included in Review?
Some Difficulties of Learning to Program [15]	78	1986	Yes
Notional Machines and Introductory Programming Education [63]	78	2013	Yes
A Multi-National Study of Reading and Tracing Skills in Novice Programmers [43]	45	2004	No
Learning and Teaching Programming: A Review and Discussion [56]	39	2003	Yes
A Review of Generic Program Visualization Systems for Introductory Programming Education [65]	37	2013	Yes
The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices [17]	36	1981	Yes
A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students [48]	30	2001	No
Constructivism in Computer Science Education [8]	26	1998	Yes
Fragile Knowledge and Neglected Strategies in Novice Programmers [54]	21	1986	No
Exploring the Role of Visualization and Engagement in Computer Science Education [53]	20	2002	Yes

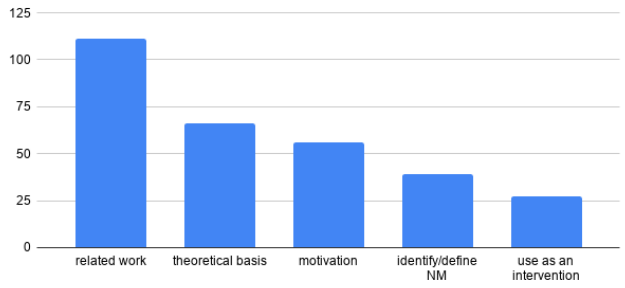


Figure 5: Apparent purpose of referencing the notional machine concept.

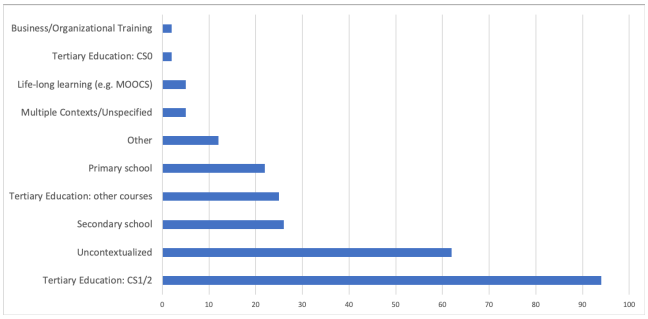


Figure 6: Educational context featured in the reviewed papers.

so the columns in the figure total to more than the number of papers. Most (72%) cited NMs either as related work, theoretical basis for the work, or motivation. Very few (27%) directly engaged with the concept to identify an NM or to use an NM in an intervention. This issue is a theme in our analyses: in later sections, we see more evidence that NMs do not feature in research questions but are frequently found in abstracts, which suggests that the idea of NMs are assumed to be present and important but are not investigated. As is typical for computing education, much of the work that refers to NMs (42%) is situated in the context of the first year of a university education (CS1/2), and another 12% is based on other courses at the tertiary level. Another 26% is uncontextualized (e.g. literature reviews, some theoretical work, and some position papers), leaving only 21% in the primary or secondary educational contexts. Figure 6 illustrates the breakdown of contexts in more detail.

3.3 Topics of papers that refer to notional machines

Based on Figure 5, we can infer that the majority of the papers are focused on another topic; they do not evaluate or develop NMs directly. To explore where these papers are situated – what research topics they explore – we performed two independent analyses. For the first analysis, we coded and categorized the explicit research questions stated in each paper. For the second analysis, we extracted common words from the abstracts of the papers we reviewed and then matched the most common words with areas of focus in the discipline. The results of those two analyses are presented in the next two subsections.

3.3.1 Topics defined by research questions. Out of the 226 reviewed articles, 74 articles had explicitly stated research questions (33%), while the rest of the articles either had no research questions or stated goals, objectives, or purpose without explicitly framing them as research questions. This is in line with previous literature reviews conducted within computer science education research – for example, a recent ITiCSE working group that reviewed literature on predicting academic performance [33] observed that approximately one third of the identified articles had explicitly stated research questions. A single researcher coded each of the 74 stated research questions. They used a grounded approach and identified codes as they emerged. After coding all 74 questions, similar codes were grouped together and then a second pass was performed to assign codes that were missed in the initial pass. Overall, the research

topics in the articles with explicit research questions varied considerably, but a few recurring and overlapping topics occurred. The most common topics were related to understanding and/or comprehending concepts and misconceptions (22 articles), tools and/or IDEs and/or programming environments (12 articles), visualizations and/or animations and/or program state representations (12 articles), tracing and/or debugging code (8 articles), teacher and/or lecturer perspectives (7 articles), and syntax and/or logical errors (5 articles). These topics could also appear together, where researchers could, for example, consider the lecturer perspectives on the usefulness of a novel visualization. Including the term notional machine in the explicit research question was rare – out of the 74 articles with explicitly stated research questions, 3 explicitly considered an/the NM, two of them being a part of the same research topic. This highlights an important issue that we discuss later in a broader extent. That is, research on NMs is mostly lacking – or at least it is not phrased using the term notional machine.

3.3.2 Topics defined in the abstract. Since research questions are often missing or not clearly stated, we also analyzed the paper abstracts to further identify the topics being explored in the papers we were reviewing. Content analysis of abstracts is a method for creating a high-level view of thematic areas and assumptions held by the research community and has been used in computing [64]. While content analysis often relies on human tagging, here we take an NLP approach and identify the most common sequences and single words. We hoped that this analysis would reveal what concepts and research areas are most closely linked to the notional machine concept. In the previous subsection we described a separate but supporting analysis. There we described our process for analyzing the research questions as stated and described by researchers, and in that subsection, we used human tagging.

Table 2 contains the results of our NLP analysis of the abstracts. We counted the number of instances of each unique word found in the papers we reviewed, and then we selected the nouns or noun phrases for further analysis. The top twenty bigrams (pairs of words) and singletons (single words) are presented in the leftmost columns. In addition, we removed singleton words that are found in the most common bigrams; the remaining singletons, which are not part of a common bigram, are found in the rightmost columns. The three most common bigrams observed – by a considerable margin – are computer science, NMs, and introductory programming. This contrasts starkly with our findings from the analysis of the explicit RQs, where we observed that NMs are very rarely present in explicit research questions, while simultaneously aligning with the observation in Figure 5 that the notional machine concept is more frequently used as the basis or motivation for work, than being the focus of the work itself. The presence of the term notional machine in the abstract suggests that it is seen as important to contextualizing the work: use of NMs is common in computer science education, and researchers do not see a need to question it. This explanation is supported by, for example, the observations of the ITiCSE 2002 working group on the role of visualization and engagement in computer science education [53], which highlighted that over 75% of survey respondents used dynamic program visualizations, which seek to present an abstracted model of execution (that is, an NM), as a part of their teaching. If this is the case, then this data is one

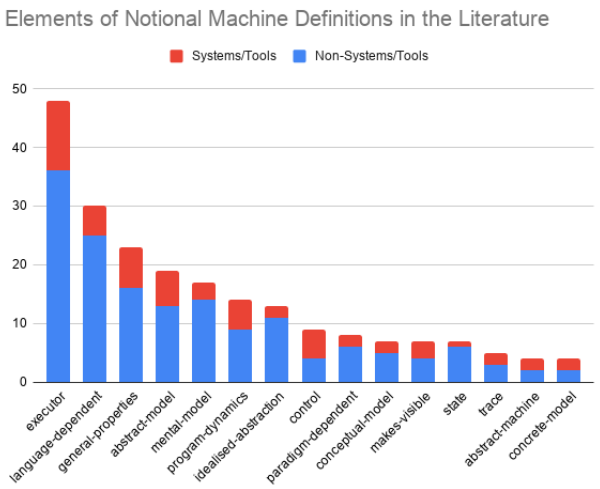


Figure 7: Key elements of NMs in the literature. Papers describing systems or tools (usually visualizations) have been indicated.

piece of evidence that NMs may be a signature pedagogy. Reviewing the entries in Table 2, we see a number of areas of research interest in computing education. Most of the common bigrams – introductory programming, computational thinking, etc. – are at a higher level than the research topics identified in the analysis of research questions. However, the words misconceptions (76) and errors (65) show up frequently, as do tools (65) and programming environments (13) and, to a lesser extent, program visualization (18).

3.4 Definitions of notional machine used

Only 116 (just under 50%) of the papers defined what the authors meant by the term notional machine. By identifying common concepts in these definitions, we find the ideas of the NM are that: it acts as an executor of code, that it is language-dependent, that it expresses general properties of the abstract machine, that it is an abstract model, etc. This analysis was carried out via concept-mapping, not simply words utilised. For example, when an article described tracing the execution of a program, this was tagged with the concept "trace" regardless of whether the word "trace" was used. We also categorized these papers into systems/tools papers when a major portion of the paper described a system or a tool and its use. (see Figure 7).

Many of these notions quote or draw from du Boulay, O'Shea, and Monk's 1981 definition, "A notional machine is the idealised model of the computer implied by the constructs of the programming language." [17] and from here one can see the origins of idealised model as well as language dependency. Many others quote or paraphrase Sorva [63], who drew from du Boulay. Many other articles conflate NMs which are constructed or utilized by instructors for pedagogical purposes with mental models which are models constructed by learners. Some articles also simply referred to the term without citing a source or defining the concept.

Table 2: Most frequent nouns and noun phrases found in the abstracts of reviewed papers. Words related to particular areas of study or theories in computing education are in bold.

Bigrams	Count	Singletons	Count	Singletons Not in a Bigram	Count
computer science	68	student(s)	456	student(s)	456
notional machine	63	programming	445	learning	232
introductory programming	54	program(s)	250	paper	126
mental model(s)	34	learning	232	research	108
computational thinking	32	course	147	system(s)	95
programming course(s)	49	computer	133	study	93
novice programmers	32	code	131	understanding	91
programming language	29	paper	126	results	84
computing education	25	language(s)	119	design	78
programming concepts	24	research	108	**misconceptions**	76
computer programming	21	concepts	105	knowledge	68
source code	21	machine	102	**tools**	65
science education	18	model	101	approach	60
program visualization	18	education	101	problems	58
object-oriented programming	17	system(s)	95	**errors**	54
program execution	17	study	93	analysis	46
programming education	15	understanding	91	development	45
problem solving	14	results	84	process	45
programming environments	13	**teacher(s)**	82	literature	45
spatial skills	13	science	81	instructional	45

3.5 Measurement and evaluation

Relatively few papers performed an evaluation on an NM or its impact on students. As discussed in Section 3.2, only 27% of the papers we reviewed identified an NM or used one in an intervention, and most of these performed an evaluation. 42 (19%) performed a qualitative analysis of some form, and 38 (17%) performed a quantitative analysis. 18 (8%) performed both.

The evaluations we saw focused on individual mental models or their impact on students. For example, in "Investigating and improving the models of programming concepts held by novice programmers" [44], the researchers identified that students held "'non-viable' mental models" and proposed and evaluated a method that used program visualization (reflecting an NM) and cognitive conflict to repair these models. None focused their evaluation on the idea of NMs themselves, to test the theory.

3.6 Limitations and threats to validity

When conducting systematic literature reviews, one of the core challenges is finding and identifying the relevant literature. In our case, we used multiple approaches to reduce the number of relevant articles missed: we conducted keyword-based searches on three article indexes (ACM Digital Library, IEEE Xplore, and Scopus), augmented the literature based on expert recommendations, and finally analyzed the bibliographies of extracted articles and included references that were considered to be in scope. We did not, however, create a list of known relevant key papers at the beginning of the systematic literature review process. We were uncertain if the term notional machine had been widely adopted, so we did not feel we could a priori identify a reference list of papers that would represent the breadth of research that needed to be found. Instead, we crafted our search terms to be as broad as possible: we included all articles from the literature databases that contained the term notional machine as well as close variants (notational machine, conceptual machine). Nevertheless, we know, unfortunately, that some papers were missed. Figure 3 presented a timeline of the papers found in our searches, and there's a noticeable gap in 2009. A search on Google Scholar using the string "notional machine" identifies several potentially relevant papers from 2009 that we did not cite as they were not indexed in the databases we searched, including [57]. Our decision to not create a list of additional key papers means that we do not have data on how well our searches identified those key papers. That is, we do not have data on the completeness or accuracy of our searches (when compared to a key reference list).

As observed during the literature review process, the computing education research community consists of multiple subcommunities. Adoption of theories across subcommunities can take some time, and in the interim, related theories may arise, and each community may adopt its own terminology. As a result, it is likely that we could have uncovered even more relevant literature had we included search terms such as "program visualization," "mental model," and "conceptual model." We believe, however, that the current set of papers includes papers that represent research in these areas and is sufficiently large to create a representative picture of the topic. The decision of whether to include or exclude an identified paper is also of importance for building a representative picture. In our case, when deciding whether an article should be included in the study or not, two researchers assessed each article independently. In case of disagreement, the article was included into the study. As discussed in Appendix B, although the interrater reliability was

only moderate, we believe that the inclusive approach avoided premature exclusion of articles. The main threat to inclusivity is our decision to focus on programming. We recognize that ideas related to NMs may be found outside of programming education (e.g. in networks [3] or training for business software [67]), but we ruled them out of scope of this effort. The extraction process also involves a number of decisions that influence the picture that developed from the review. First, the literature review was focused around three research questions (see Appendix B) that directed the construction of the extraction sheet (see Appendix C). If the research questions had been different, it is likely that the extracted data would also differ. Second, it is possible that there may be inconsistencies in how the extraction form was interpreted. The group of researchers extracted data from the papers individually due to the large number of reviews required, so inter-rater reliability cannot be calculated. To mitigate the issue of inconsistency and to maintain a focus on the research goals during paper extraction, the extraction process ramped up slowly (the first few weeks were a training period), and the group met weekly to discuss the process and to identify inconsistencies.

3.7 The role of notional machines in the literature

To synthesize our findings from the systematic literature review, we note that while NMs were introduced in the late 1970's and early 1980's, our analysis suggests that its role in computing education is still emerging. It is possible that this could be due to an early divide between research communities, where the community centered around the psychology of programming did not initially succeed in disseminating their ideas to other scientific communities such as the one focusing on program visualizations, or, for example, due to the concept being abstract and difficult to grasp. While the notional machine idea has been naturally present in computing education, as evidenced in longstanding efforts on building program visualizations for teaching purposes, it has taken more than one prolific review to bring the notional machine term and associated principles to the mainstream computing education research vocabulary. Furthermore, as evidenced by recent work using the related term "conceptual machine" and the lack of a coherent and consistent definition across papers, the term notional machine has not yet been adopted and integrated across the entire discipline. Despite the increased visibility of the term, the notional machine concept is most often used in the introduction and related works sections of articles. NMs are rarely explicitly used – or evaluated – in interventions in the identified articles. This is especially evident in our analysis of explicit research questions extracted from the literature. Out of the 74 articles with explicit research questions, only 3 incorporated the term notional machines into the research questions. Evidence that the term is becoming a part of mainstream computing education research terminology may also be found in the contexts of the articles that use the term. While the term was originally coined in the context of teaching children (and teachers) programming (as discussed in 2.1), much computing education research focuses on CS1/CS2 courses in higher education, and the use of the term has transferred directly to that new context. When considering the foci of the identified articles, despite the rare occurrence of the term in research questions, we observe that NMs are positioned as a central concept in the work as evidenced by the abstracts. This suggests that NMs are a (still rarely acknowledged) signature pedagogy in computing education. As such, studying the educational effectiveness of (various types) of NMs is called for. We next provide an overview of educational effectiveness of NMs, linking it to relevant learning theories. Then, in Section 4, we respond to the issue that no single, standard definition of an NM has been adopted by outlining the definitional characteristics of NMs.

3.8 Educational effectiveness of notional machines

There is limited literature looking at the educational effectiveness of NMs, although there are papers that track changes in student attitude or performance following the introduction of an IDE of some kind [see, e.g. 9]. There are also papers that observe individual students as they learn programming, noting their difficulties, impasses and successes along the way [see, e.g. 13]. There are fewer papers that compare the relative educational effectiveness of one NM against another or, for example, compare one modality of NM *vs.* another modality.

The closest relevant research comes from the pedagogy of other STEM subjects. Here the use of simulations to draw attention to processes that are hard to see or, indeed, impossible to see is large [see, e.g. 68]. Moreover, general theories of pedagogy have tended to address teaching issues at a level that does not particularly identify programming and the pedagogy of using NMs. Some theories focus on limitations in human mental and perceptual processing as they apply to learning, e.g. Cognitive Load Theory [14, 39, 58] and Multimedia Information Theory [47]. Other theories look at learning largely independently of low-level human information processing issues, focusing on the kinds of representation and the sequencing of representations that best promote learning [21], the coordination of multiple representations in learning [2], or examination of the role that modelling and analogy play in teaching and learning [23, 59]. Other work takes a developmental approach to learning, arguing that the evolution of coming to understand an abstract concept follows much the same Piagetian sequential pattern as children go through developmentally [42], in contrast to a phenomenographic view of learners and learning programming [10].

In terms of the pedagogy of NMs, these theories all point in broadly the same direction. These include (i) early concrete examples and analogies assist the understanding of abstract concepts, (ii) simplicity of the presentation of the NM with its important aspects made salient reduces the cognitive load on the learner, (iii) in the presentation of an NM aims, as far as possible, to keep from overloading the learner's perceptual or mental processing.

The above theories only partially account for the way that in technical subjects, such as programming, learners often have to move between highly abstract concepts and practical hands-on activity in order then to fully understand the abstract concepts.

These ideas are explored in semantic wave theory, which is a pedagogic theory that considers timing, sequencing and contextualising central components [see e.g. 69].

4 NOTIONAL MACHINES: DEFINITIONAL CHARACTERISTICS

The historic description of a notional machine referred to "the idealised model of the computer implied by the constructs of the programming language" du Boulay et al. [17] and some of its properties were expanded in du Boulay [15]. The much cited review by Sorva [63] built largely on du Boulay et al.'s descriptions, although it focused only on program execution. However, there has never been a fully precise definition of the concept. This has supported the development of a rich literature, but also a confused literature, since there is no agreed definition and no consistent use of terms. As one outcome of this working group (drawing on both our theoretical and empirical work), we offer these defining characteristics based on the purpose, function, focus and representation of NMs.

4.1 A notional machine has a pedagogical purpose

The general purpose of an NM is for use by a teacher to support student learning of computational concepts. Therefore, a crucial aspect of any NM is that it should simplify an actual concept or skill as an aid to understanding. Thus programming language semantics by themselves are not an NM when they are used without a specific pedagogical intent. It is called "notional" in the sense that what is being described is a simplified, partially true, version of the truth. The "truth" about what happens when a program runs is complex and multi-layered, from quantum mechanics, via electronics, via machine code and upwards through various intermediate representations to the program as written. To say that an NM is a partial truth is to say both that it is concerned only with some layers of description and at any layer with only some details. It may be partial in various ways. Sometimes an NM omits details that are not of concern for the learner at that point in their learning and would confuse or mislead them. Sometimes an NM makes salient details that the learner needs and that they might otherwise overlook. Sometimes an NM reveals an aspect or connection that is not clear from surface features. It is called a "machine" because it makes a direct and explicit analogy to a mechanism with parts that interact to produce behaviour, like a clock with its gear wheels or a food-blender with its motor and blades. A piece of code always has an action, always does something. Drawing attention to the place of action is part of the work of an NM. For example, a variable is like a box with a label, and assignment copies/moves a value into that box. The machine idea can be applied both to the computer itself and to describe the behaviour of programs. So one can think of a computer as a machine that enables a programmer to create, edit and run other machines. Ultimately "machine" relates to the specific content area of programming, typically done for and within a, maybe abstract, machine (see the subsection on Focus below).

4.2 A notional machine's function is to draw attention to something

The generic function of an NM is to uncover something about programming, computers or computation, or to draw attention to something, that is not obviously apparent in the artefact the student is using. There are different reasons as to why the focus of interest might not be apparent. It might be because there is a lot of functionality and the student is lost in a "fog of relevance" not knowing which parts are salient for the task in hand [34]. It might be because the focus of interest isn't there at all; a compiler is invisible in the code. It might be because the subject of focus is a cognitively demanding concept (like references) and the attention that an NM affords makes the concept more accessible.

4.3 A notional machine has a focus

Typically, NMs focus on programs and their behaviour to explain various facets of execution. Some might be quite broad in focus, others narrower. For example, an NM might elucidate the mechanism of sub-procedure calls. Within that focus, a particular aspect might be emphasised, e.g. parameter passing. So an "aspect" is treated as a part or a property of the focus. The distinction between a focus and its aspects is useful, rather than definitional, see the next subsection.

In contrast to [63], we argue that as well as programs and execution, an NM's focus can also be concerned with computers as places where programs can be built, run, and stored. For example, the location of the program inside the computer, and the difference between the locations of (say) the executable version of the program and the file that contains the editable version are not obvious, nor is it obvious why these should be important.

An NM therefore may focus also on the associated epiphenomena of programs and programming, such as the names of program constructs, the causes and content of error messages, the names on buttons in the environment (e.g. "save", "file" etc.), the locations of files containing programs, the difference between editing and running a program and so on.

4.4 A notional machine has a representation

An NM will have a representation and this representation will draw attention to certain aspects of the focus and possibly ignore others: for example, elucidating the aspect of parameter passing within a focus of procedure calls. A representation may simply be verbal, such as saying "a variable is like a box", and does not necessarily need to be visible. In our definition of representation we are much more concerned with *what* is being represented than with *how* it is being represented. Of course, from a pedagogical point of view the "how" can be very important.

Thus, two notional machines can draw students' attention to the same thing and yet be different notional machines in terms of their representation. For example, one teacher might use arrows to address objects (NM1) another might use IDs to address objects (NM2), or the same teacher might use them on separate occasions. Both NM1 and NM2 focus attention on object identity/aliasing, but they are different NMs because their representations draw attention to different aspects of the focus. In one the ID aspect is implicit, in the other the ID aspect is explicit.

By contrast, two hand-drawn or machine-drawn representations that draw attention to the same thing may essentially be the same NM because, although the surface modes of the representation may be different, the aspects of the representation are the same. For example, a teacher may introduce a variable assignment NM with two representations: a variable table on a whiteboard and the variable values in a debugger.

4.5 Designing notional machines

Based on the preceding, defining characteristics, an NM draws attention to some specific aspect of programming, computers or computation with a pedagogic intent. While it is possible to transfer the concept of a notional machine to areas of computer science unrelated to programming, historically - and for the purpose of this report - we only consider applications that are, at least indirectly, related to the task of programming. Then, there are three areas of interest that an NM may place its focus on (see Figure 8): 1) the programming language itself, 2) the "machine" that one is trying to control, as in du Boulay's original definition [15], and 3) the interaction (or "overlap") between these two.

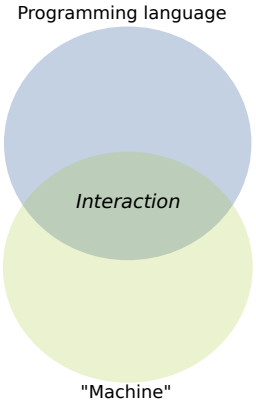


Figure 8: Areas relevant to notional machines

For example, placing a focus on the fact that every value in Python is an object, is only concerned with the programming language. In contrast, looking at how a micro-controller board has pins that can act as inputs when connected to a switch in the right way, is only concerned with the "machine", in this case the actual hardware that a program will run on. Depending on the context, this "machine" can also be a programming environment (e.g. for systems like Scratch), an operating system, a virtual machine, or even something completely detached from actual computing hardware and software, for example a set of algebraic rules that fully define the semantics of some language. Finally, looking at how settings for virtual memory might differ for JVMs depending on the operating system or implementation is something that is concerned with the interplay between programming language and "machine". In contexts where the "machine" is closely related to the programming language, e.g. because it is the interpreter of a language, the overlap will have a much larger extent than in other scenarios.

Following its pedagogical purpose, an NM is an idealized (partially true) model that guides students' attention to aspects related

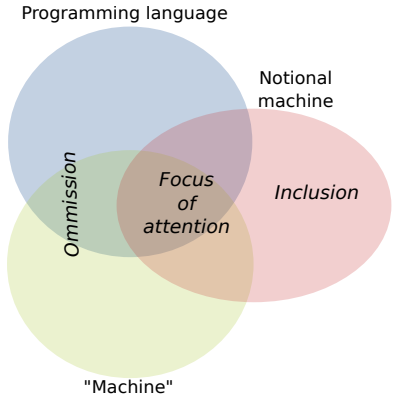


Figure 9: An NM as a pedagogic model.

to some or all of the three areas of Figure 8. Idealization as a general process of creating a model from a system can happen both by omitting aspects of the system not relevant for the model and by including something that is not part of the system at all (see Figure 9). For example, characterizing a variable as a location in memory that is addressed by its label is an omission of details whereas characterizing a variable as a box is an inclusion of something new.

In natural sciences, these two ways of arriving at a model are known as Aristotelian and Galilean idealizations [20] with the main difference being that only an Aristotelian model is still a true, albeit simplified, representation of the original system whereas a Galilean model typically is a more distorted representation of the system. The purpose of idealization in NMs is to focus attention and to aid understanding. Omission helps focussing and inclusion helps making something comprehensible, for example by forming analogies. Both processes are driven by the intent of the educator when designing an NM and, together with the decision of what (not) to include (i.e. represent) from either of the three areas of interest, determines the characteristic of the NM. Another way of looking at the process of designing an NM is that it maps specific aspects of a real-world system into the (pedagogic) domain of the NM, while other aspects are not mapped. This mapping defines the form of the NM, first and foremost, but in doing so also defines the pedagogic augmentation.

For example, the NM presented in the next section (see Figure 19) that explains variables as an analogy to parking spaces places a very specific focus on a small aspect of the programming language but includes the notion of parking spaces from the everyday life of students. This also introduces the Galilean distortions described above, as a variable in many aspects is clearly not like a parking space. An NM focusing on presenting stack diagrams (see Figure 17) for a specific point in program execution, on the other hand, draws attention to aspects related to the interplay of programming language and machine and does not add anything extra (see Figure 10).

In general, NMs geared towards novices may add more "extra" than NMs geared towards advanced learners, or even experts. Transitioning from a first high-level programming language like Python to a programming language closer to the system, like C++, may require a shift in NMs towards those that place more focus on aspects

related to the machine but, as basics might have been mastered by the students already, may not include as much extras (see Figure 10).

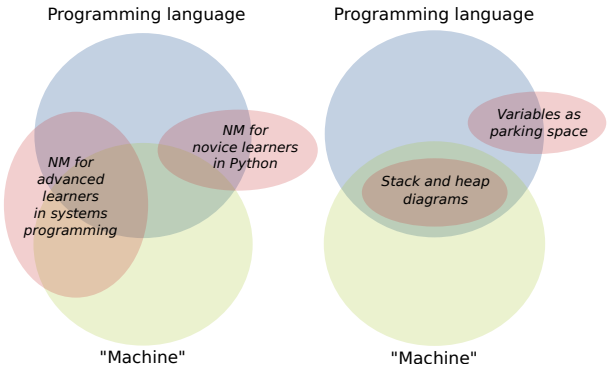


Figure 10: Specific NMs cover distinct "terrains" in this space

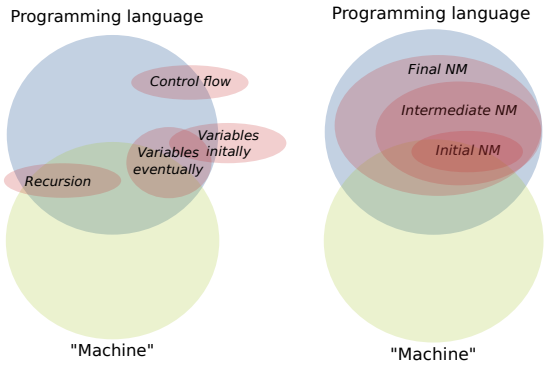


Figure 11: NMs may be arranged as repertoires and sequences as teaching progresses.

As described later in this report, teachers will often not just use *one* NM. Instead, they may draw on a *diverse repertoire* of NMs that are drawing students' attention to various different aspects of programming (see Figure 11, left): One might start with explaining variables as boxes and then, when introducing references, switch to an NM built on tables and IDs. Control flow might be introduced by a worker who follows a given set of rules to determine the line to be executed whereas recursion might be explained later in the course without analogies simply by referring to the semantic rules of function calls.

Alternatively, when following a bottom-up approach of gradually introducing new concepts along a programming course, the same NM might be expanded as well, as shown in Figure 11 on the right and exemplified in our examples by the Python computer (Figure 26).

5 CAPTURING NOTIONAL MACHINES: EMPIRICAL WORK

As well as situating the work of notional machines theoretically and historically, we also undertook empirical work to investigate current usage. We worked to identify the use of NMs in the practice of teaching, whether in the classroom, as instructional tools, or in textbooks. While a teacher's choice to use an NM provides some estimation of its value, our immediate aim in capturing and presenting NMs is not to validate their effectiveness. Rather, we seek to explore common properties that characterise NMs and offer candidate relationships of how NMs may be connected to each other, such as in a sequence of use to explain an increasingly complex set of concepts. In this section, we present our empirical work for capturing NMs, followed by presentations of examples. These presentations set the stage for the next section where we further explore their use.

5.1 Methods

We worked to identify NMs from several sources: our own practice, the instantiated practice of others (by interview), from published papers and from software (visualisations and IDEs). Each collection method had its own benefit and advantage, which we summarise here.

(1) **Interviews.** By interviewing other educators, we broadened the pool of possible examples beyond our own practice. In doing this we could identify exceptional educators, with rich experience. We were careful to ensure that we focused on the interviewee's teaching practice, and asked for instantiated examples (materials, photographs of whiteboards, screenshots etc.) where possible. The protocol we used, together with details of its development and refinement, is contained in Appendix D.
(2) **Our own practice.** We were interested to document our own practice. It is not an easy task to "interview yourself"; much of our knowledge is tacit and our classroom skills taken-for-granted. It is also hard to identify what is important for someone else to know about what we do, or what may be safely left to shared expertise. We devised a structured form to aid the capture of our own practice, to allow us to surface and externalise appropriate details of the NMs we use. Learning from previous projects [18] we were careful to avoid second-guessing the context of other classrooms, where we imagined these NMs might be used.
(3) **Software.** NMs may be represented in software in two ways. Educators may write their own programs that embody NMs: we captured these when we encountered them. Mass-market, sometimes commercial, tools and environments may also embody an NM. These include program visualization tools such as Python Tutor [27].
(4) **Not found in Literature.** In our systematic literature review (reported above and detailed in Appendix A, below) one of the questions we asked was "if an NM is described, what is it". We were surprised to discover that there were relatively few actual descriptions of NMs. It seems that most of these are not described or reported in the mainstream research literature, but rather reside in close-to-practice publications, like "teachers' tips" sites, blogs and twitter. At the end of Section 5.2, we provide an example from twitter (Figure 23).
(5) **Other.** Although they are occasionally represented in our collection, we did not seek out examples from textbooks or

other instructional materials, such as online learning repositories, or online videos.

Our use of these methods was opportunistic, not systematic, so there should be no expectation that we evenly covered all curriculum areas (although, for organisational convenience, our NMs are arranged by topic in Appendix A). Our methods also biased us to collecting examples with an explicitly physical representation; "talking" examples, where a teacher gives a swift, clear analogy are largely absent. (For example, "Using a library is like using curry powder in a recipe, it means you can "buy" the flavour you want. You could make your own curry powder, and in certain circumstances it might be better to do so, but it's tricky and time-consuming. For the majority of the time, adding in a standard product that someone else has made is perfectly good enough.") Also using our methods another known, but hard-to-capture practice, is the circumstance where a teacher adopts an "approach", perhaps the repeated, systematic application of an analogy. Such as consistently using a mobile phone to illustrate a wide variety of computational concepts. For example the nature of classes and objects (distinguishing between text messages in general and the text message I sent to Helen this morning); last-in first-out lists (ordering incoming messages by placing the most recent as the first revealing the stack-based nature of the incoming message list) and exception handling (failure to send a text message results in an exception being thrown by the provider) [61]. Interviews may also fail to elicit how teachers verbalize programming constructs, which nevertheless convey some aspect of an NM. For example, an instructor may choose to pronounce a literal string as a sequence of characters, a practice that we only uncovered as a "self interview" and with the explicit goal of considering how programming elements are pronounced. Future development of the interview protocol may include prompts for eliciting these talking examples.

Our empirical work resulted in a collection of 43 NMs (see Appendix A). We have grouped these into three types, depending on the form they took – Machine-generated representations automatically generated by software or environments; Handmade representations created by teachers, often hand-drawn; and those NMs which are primarily analogies[2]. Of course, "analogy" is orthogonal to the form the representations take, and so the territory NMs inhabit may be better represented as a quadrant:

	Handmade	**Machine-generated**
Representation		
Analogy	A	B

It is, however, misleading to think of these boxes as discrete, with sharp-edged spaces. Rather the boundaries are blurred and in reality the columns (in particular) are a gradual spectrum. Nor is the space equally populated: we have many more Analogy examples in the Handmade quadrant (A) than in the Machine-generated quadrant (B).

[2]The "work" of an NM is always to assist the formation of student understanding, often as visualisation: "The action or fact of visualizing; the power or process of forming a mental picture or vision of something not actually present to the sight" [55]. However, the verb-form of visualisation is easily confused with the product of visualisation, a picture or model, so we avoid the confusion by avoiding the term.

Because we found machine-generated NMs using analogies to be empirically scarce (although there are some examples in the literature, e.g. [60]) in this paper we use the category Analogy solely in respect of handmade representations. Future work could explore machine-generated examples and, if warranted, establish distinguishing labels.

In this way we have identified three groups of NMs, but there are other candidates. For example, there are recognisable subsets of "handmade", there is a grouping of "concrete/tangible" NMs and another of "unplugged" representations which use students (bodies) as manipulative objects: these should perhaps be separate categories. And, of course, there may be others that we have not even considered (because we have not seen them, or no-one has drawn our attention to them: "Those phenomena with which we have no affinity and which we are not in some sense ready to see are often not seen at all" [49]).

As with all qualitative work, the raw data was unwieldy. We devised a compact form to present our NMs, initially to each other. In doing this, we worked to include sufficient information so that we might grasp what the NM was for, but in a succinct format. We considered, and discarded, many abstractions. We reached a stable state with a minimal collection of mandatory fields with some optional extras (such as "notes"). We found that it was important for us to have pedagogical rationale represented, hence the "conceptual advantage" and "draws attention to" fields. Also, where and how the NM was gathered provided a surprising amount of shorthand detail, so the inclusion of "origin" and "attribution" proved useful beyond simple acknowledgement. Our expectation is that we will continue to work on this form and that it will continue to evolve: the current form is represented in Figure 12.

5.2 A brief tour of the notional machines

In this section, we present examples of the NMs we discovered, in three thematic groups: Machine-generated representations, Handmade representations, and Analogy. We have also created a website on which we present many NMs: https://notionalmachines.github.io/notional-machines.html.

5.2.1 Machine-Generated Representations. Many NMs are embodied in program visualization tools. Such tools automatically generate representations of program executions and often provide means to step through the execution. They usually show the state of the execution at any given step. These tools often can visualize all program code written in a certain programming language, and they often represent many different aspects of program execution. As such, they do not represent one, but many different, intertwined MSs, for example they may represent control-flow and data, show the stack and the heap, and show intra-procedural as well as inter-procedural aspects of control-flow.

A well known example tool for imperative languages is Python Tutor [27] (see Figure 13). Python Tutor allows a student or teacher to enter arbitrary Python code and to step forward (or, more surprisingly, backward) through an execution of that code. At each step, Python Tutor visualizes the state of the computation: it shows the state of global variables, the call stack frames with the state of the local variables, and the heap with the objects and their fields, and the lists and their elements. Each variable is represented as a

NM Name

Image

Programming Paradigm:
e.g. imperative / object-oriented / functional
Programming Language:
e.g. Java / Python / C / any / ...

Conceptual Advantage
Why do you use this? What does it buy you? Why do this and not something else? Allows the reader to get at the value of the NM

Mapping

PL	NM
PL element (e.g. variable)	Notional machine element (e.g. parking space)
PL element (e.g. value)	Notional machine element (e.g. vehicle)

Form
Choose from: Machine-generated representation; Handmade representation; Analogy

Draws Attention To
What (of already visible things) does it focus on and/or what (of invisible things) does it make visible? What work does it do for the teacher/learner? What do they understand after using it that they didn't before?

Cost
Investment - time/cognitive

Origin/Source
Whether collected from own practice/by interview/from publication/software etc.

Attribution
Person who used it, person who ***collected it***

Note: The "name" is also a link to more detailed explanatory material—perhaps extracts from an interview, or a published paper—that someone would need if they were to actually instantiate the NM.

Figure 12: Template card with mandatory elements

box. Primitive values are drawn inside the box and reference values are represented as arrows from within a box to the heap object or array the reference identifies. In some cases, whether something is an NM or not depends on its use. Python Tutor, for example, is a visualisation tool. If used in a particular pedagogical context, however, it can embody NMs, or serve as one, such as the one presented in Figure 14.

For functional languages, an exemplar of machine-generated representations is the Dr. Racket Stepper [19] (see Figure 15). The Stepper shows one step in the evaluation (rewriting) of the functional program. On the left side, it shows the program before the step, and on the right side, it shows the program after the step. The term that is about to be rewritten is highlighted. Like Python Tutor, the Stepper is a tool, but not an NM in itself.

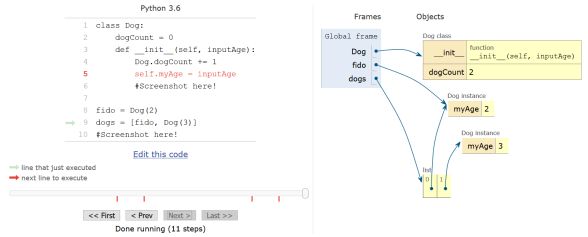


Figure 13: Python Tutor visualizing a moment in the execution of a Python program

Note that the NMs embodied in these tools can appear in different forms in other contexts. For example, the Python Tutor representation can be hand-drawn by a teacher on a whiteboard, it can be used in a textbook [5], it can be drawn by students in the Informa Stack & Heap tool [31], or it can be used in the form of diagnostic probes. Similarly, the "expression rewriting" representation produced by the DrRacket Stepper can also be hand-drawn by teachers to explain expression evaluation on a board, or it can be produced by students on paper as an exercise or an assessment. Thus, while our investigation of machine-generated representations served as a way to identify NMs, the identified NMs are independent of their embedding in the tool.

5.2.2 Handmade Representations. In addition to the machine generated NMs above, we also found many examples of handmade ones. These can be hand-drawn, like a manually created flowchart, as well as concrete or tangible objects to demonstrate concepts (in talking about "a variable is a box"). Hand-drawn NMs can be both drawn by students as well as teachers, and similarly, tangible devices can be used by teachers to draw attention to specific parts of the real machine, as well as by students to draw their attention to an aspect via their own practice. A third form of handmade NMs are activities, where students use their own bodies in unplugged activities so that they can "play out" concepts, such as recursive calls or packet switching.

Teachers often hand-draw NMs. Sometimes, these representations are created ad hoc, to serve an immediate need, and used only once in a specific situation. But they can also be elaborate representations, used on multiple occasions, and are the same every time they are used.

Hand-drawn NMs have impact on the instructional strategies in several ways. They require the teacher (or student) to "do the work" and really play out that they are the machine. In addition, when a teacher draws an NM, this can slow down their instruction, allowing for students to follow along better, and giving the teacher the opportunity to explain how the machine works. As such, hand-drawn forms can be an addition to machine-generated representations, which can be mechanically "clicked through" without resulting in a deeper understanding. In this context, it is worth noting the interviewee's concern about Python Tutor (see the quote in the Notes field of the Recursion Demo using Python Tutor NM, Figure 14).

Examples of hand-drawn visualizations are diverse and plentiful in CS education. Two examples are shown in Figures 16 and 17.

Recursion Demo with Python Tutor

Programming Paradigm: imperative
Programming Language: Python

Conceptual Advantage
Demonstrates what happens to variable values when they are passed to a function. Distinguishes frame elements among each recursive call. Shows separate stack frames, each with own parameters and variables, for each recursive call.

PL	NM
frame	group of elements with function name
parameter	labeled box inside frame
local variable	labeled box inside frame

Form
Machine-generated representation

Draws Attention To
Addition of each stack frame as each recursive call is made. Collapse of stack frame as function completes and returns value.

Cost
Small, if you use Python Tutor anyway

Notes/Other
Students "*tend to use it a little bit mechanically so they tend to just step through it and not really think about what's actually happening on the screen.*"

Origin/Source
Interview

Attribution
Used by Amber Settle, collected by ***Craig***

This NM involves the use of Python Tutor, which provides a machine-generated representation of memory in an executing Python program. In this case, an instructor limits the scope of the presentation by focusing on the runtime stack of recursive calls.

Figure 14: Recursion Demo with Python Tutor

NMs can also be handmade and tangible. An example of such an NM is using a paper version of a linked list with list nodes as shown in Figure 18.

5.2.3 Analogy. We identified a group of NMs whose primary form is analogical: that is, they draw students' attention to what is important by a process of analogical transfer, transporting structural

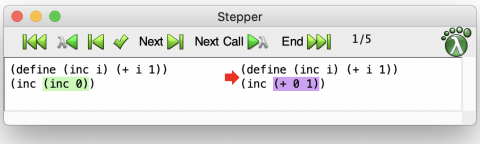
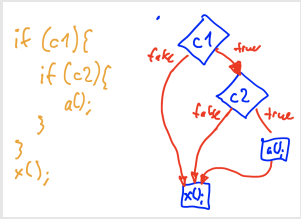


Figure 15: DrRacket Stepper visualizing a step in the evaluation of a program written in BSL

Control Flow as Graph

Programming Paradigm: imperative
Programming Language: Java



Conceptual Advantage
Make visible the flow of computation through instructions to introduce the structured programming ideas of sequence, selection, and repetition

Mapping

PL	NM
statement	rectangular node
condition	diamond-shaped node
control-flow between statements	arrow from node to node
condition outcomes	label on an arrow coming out of a condition (e.g., "true", "false")

Form
Handmade representation

Draws Attention To
The idea of a statement, a condition, and how they are used to build "sequence", "selection", and "repetition" constructs of structured programming

Cost
Small, but e.g., mismatch for short-circuit operators (&&, | |) or conditional expressions (c ? a : b), which don't map as naturally.

Notes/Other
Great to detect misconceptions (e.g., that loops do not loop).

Origin/Source
Own practice: compilers (control-flow graphs) and "flow charts"

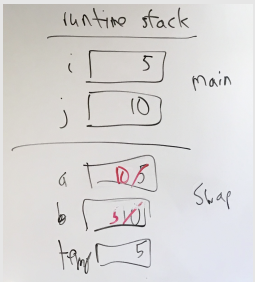
Attribution
Used by Matthias, collected by ***Matthias***

This NM highlights the semantic structure of nested conditionals. As a hand-drawn NM, it can be dynamically constructed calling attention to the semantic components of the code while omitting the syntactical details.

Figure 16: Control Flow as Graph

Hand-Drawn Runtime Stack

Programming Paradigm: imperative
Programming Language: Java



Conceptual Advantage
Distinguishes between variable values in the main function and the parameters and local variables in a called function

PL	NM
variable in main	box with label in a frame called main
parameter	box with label at the top of a frame labeled with function name
local variable	box with label at the bottom of a frame labeled with function name

Form
Handmade representation

Draws Attention To
Each function call/frame has its own set of stored values.

Cost
Low: Requires that the instructor writes out variable boxes for each function call.

Origin/Source
Interview

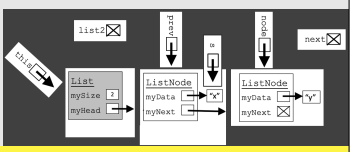
Attribution
Used by John Rogers, collected by ***Craig***

This hand-drawn NM demonstrates how variable and parameter values are stored separately in their own stack frame. The instructor can demonstrate how modifying the value for a parameter does not change the value of the corresponding variable in another frame. As a modifiable hand-drawn NM, the instructor can interactively make changes to values and call attention to what is changing and what is not.

Figure 17: Hand-drawn Runtime Stack

Paper List/ListNode (Video)

Programming Paradigm: imperative
Programming Language: Java



Conceptual Advantage
Helps students plan code; avoids confusion of crossed out arrows.

PL	NM
object	type written/underlined in a rectangle
variable	variable name followed by a small box
int	number written in a small box
reference	arrow originating from a small box
null	X written in a small box
local variable	free floating variable
instance variable	shown in an object

Form
Handmade representation

Draws Attention To
The effect of resetting references.

Cost
A pain to cut out all the papers. No increase in class time needed.

Origin/Source
Own practice

Attribution
Used by Colleen; collected by ***Colleen***

These paper versions of Java variables that hold a reference and List and ListNode Objects can be used to plan or trace code. A student can move the paper to illustrate resetting a reference and can fold the paper to cover an arrow with an X, which would represent a null reference. This NM draws attention to the constraint that a variable can only reference one thing at a time.

Figure 18: Tangible NM depicting linked list with paper sheets

features of a common, understandable situation into a new, unfamiliar one. The analogical NMs are often "small" and lightweight, possibly just an idea of an example or explanation, such as the Variable as Parking Space NM presented in Figure 19.

One aspect of NMs in this group relates to the concreteness (or tangibility) of the analogous object. Some NMs are extremely concrete. Teachers use physical boxes and post-it notes, and bring actual hanging folders into class (Figure 20).

NMs can also take the form of activities where a teacher represents parts of the machine that attention will be drawn to. An example of this kind of NM is Method-Call Dance (Figure 21), where a teacher executes a dance to a certain location and then comes back.

NMs where students represent parts of the machine (e.g. Figure 22) enable rich opportunities for analogical transfer, because such NMs map programming language ideas to the students' very own properties and behaviours.

While we were writing this report, a twitter thread was started with the question "Imagine you're talking to someone new to Computer Science. How would you describe a Variable?" [52]. There were many contributions, and most were extremely concrete examples: a glass containing some liquid, a container with a label, a visitor parking spot containing a vehicle, a house at an address containing

Variable as Parking Space

Programming Paradigm: imperative
Programming Language: statically-typed

Conceptual Advantage
Illustrated the concepts of ***type*** in programming with an associated warning that using the wrong type can cause problems.

PL	NM
variable	parking space
value	vehicle
type of variable	size/shape/constraint on parking space
type of value	kind of vehicle

Form
Analogy

Draws Attention To
The concept of type and the importance of getting it right. Use when first introducing variables in a statically-typed programming language.

Cost
Very low

Notes/Other
Different types of parking spaces are deigned for different types of vehicles. Bad things can happen if you put the wrong type of vehicle into a space that is not designed for it.

Origin/Source
Own practice

Attribution
Used by Jan, collected by ***Jan***

This NM works as an analogy in that it draws on student understanding of parking space limitations, where certain bays are designated for certain sorts (and sizes) of vehicle.

Just as a motorcycle may be restricted to parking in particular spaces, an integer value may only be assigned to a variable that is declared for integers. The effectiveness of the analogy depends on student knowledge of parking lots and how well that models the semantics of the programming language.

Figure 19: An example of an analogical NM

one thing, a locker, a bucket, a purse, a file folder, a post-it note, a pocket, a drawer, a shoebox, a jar containing tea/coffee/sugar, a pigeon hole, empty labels you can write on, an envelope, a suitcase … and so on. Interestingly, there was also a contribution (Figure 23) that suggests that these early analogies may be strongly retained.

Variables as Clothespins

Programming Paradigm: imperative
Programming Language: any

Conceptual Advantage
Makes variable tangible, helps prevent multiple values misconception

Mapping

PL	NM
variable	clothespin
variable name	label on the pin
value	paper sheet in the pin

Form
Handmade representation

Draws Attention To
Variables contain *one* value

Cost
Low

Origin/Source
Interview

Attribution
Used by Emily Bakker, collected by ***Felienne***

This NM presents a variable as a clothespin holding a value. This focuses students' attention on the value going into the variable, and fits vocabulary commonly used in programming, such as saying a variable holds a value.

Figure 20: Variables as Clothespins

6 USE OF NOTIONAL MACHINES: REPERTOIRE AND DIAGNOSIS

As we gathered examples of notional machines, we also gained insight into how they are used in instruction. In this section, we review a few recurring themes of their use. They include how NMs may be sequenced in a course, applied for a short-lived purpose, and used to diagnose student misconceptions.

Most often, NMs are used as explanatory devices to accommodate the learner's current level of knowledge and avoid unnecessary cognitive load. Frequently analogical, they also strengthen the "semantic gravity" of a concept, thus making it more accessible to a learner. Semantic gravity is the degree to which meaning relates to its context; the more meaning is dependent on context, the stronger the gravity [45].

Explanations that alter semantic gravity are said to take the form of 'semantic waves' (as in Figure 24), where knowledge is transformed between relatively decontextualized, condensed meanings

Method-Call dance

Programming Paradigm:
imperative / object-oriented l
Programming Language:
Java / C# / any / ...

Conceptual Advantage
To illustrate method calls, the teacher dances to another location, performs some actions there (changing the memory), and then returns to the previous location through the return address, where he continues with performing actions.

Mapping

PL	NM
method call	dance to another location
return address	dance back to the previous location

Form
Analogy

Draws Attention To
It illustrates that you 'go elsewhere in the memory' when you call a method, and that you need to remember where you came from (return address) when you go elsewhere.
A dancing teacher helps students to remember the concept.

Cost
Hardly any cost involved

Origin/Source
Interview

Attribution
Used by Jeroen Fokker; collected by ***Johan***

This NM uses a dance to illustrate the execution of a method. It is an embodied form of an NM, drawing the student's attention to the fact that you need to remember where you are coming from.

Figure 21: Embodied NM expressing methods with a dancer

and context-dependent, simplified meanings. As Maton observes, "... teaching often involves (to put it simply) a repeated pattern of exemplifying and 'unpacking' educational knowledge into context-dependent and simplified meanings." [45]. NMs (particularly analogical) are by their very nature designed to reduce semantic density, and so make a concept more *understandable*, at the same time they increase semantic gravity, so they make a concept more *familiar*.

6.1 Classroom use

Teachers who contributed NMs often said that they did not use them singly, but as part of a larger repertoire of examples so that a single course might include a set of NMs, each separately serving a specific learning objective in the whole course narrative. Others were based on an "approach", where NMs were related to each other.

Recursion Role Play

Programming Paradigm:
imperative
Programming Language:
any

How many students are in this row?

Conceptual Advantage
Makes explicit recursive calls as a form to "delegate" parts of work, base case as a situation where delegation stops. Allows unpacking many aspects of recursive computation (pairing of call/return, passing info down through params, up through return values, tail recursion).

Mapping

PL	NM
object	person
method call	verbal request (how long?) to next person in line
return value	verbal response from next person in line
base case	(action of) last person in line (answering "1")
recursive case	(action of) all other persons in line
linked list	line of persons (e.g., waiting in amusement part, or row in classroom)

Form
Analogy

Cost
Difficult for instructor to react to students' actions during role-play, and to catch (and exploit) all the teachable moments. May require prior introdution of role-play (e.g., with "Object as Student").

Draws Attention To
recursive calls and returns, base case & recursive case, tail recursion

Origin/Source
Based on an original conversation on how to teach recursion with Benedict Du Boulay.

Attribution/Origin/Source
Used by Ben and Matthias, Collected by ***Matthias***

This NM presents recursion by having students relay messages to each other. By playing a role in passing values, students will experience how a call stack passes messages.

This NM uses analogical transfer in multiple ways: (1) Students make verbal requests to their friends (method calls), and they await responses (method return values). The students know what that means already. (2) The fact that the last student in the row will not find someone else to talk to also is obvious to students, and they can transfer this understanding to the programming world (i.e., the idea of a base case of the recursion).

Figure 22: Recursion Role Play

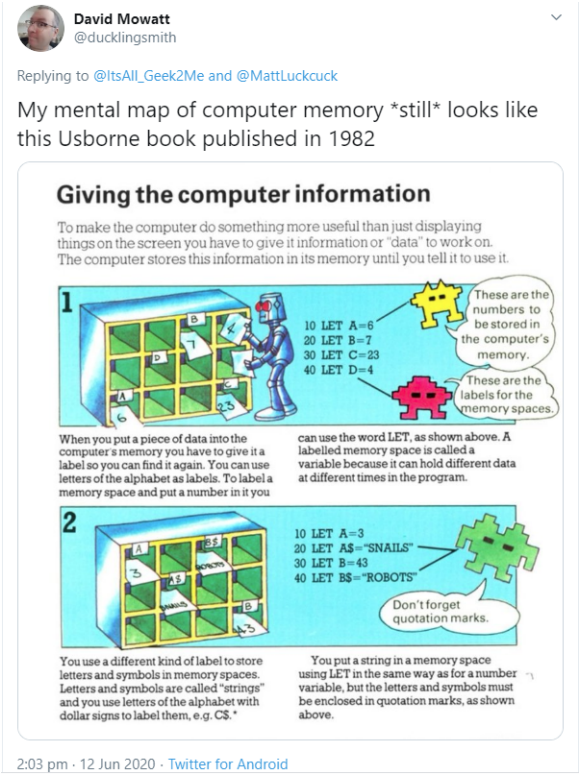


Figure 23: Tweet showing a lasting NM

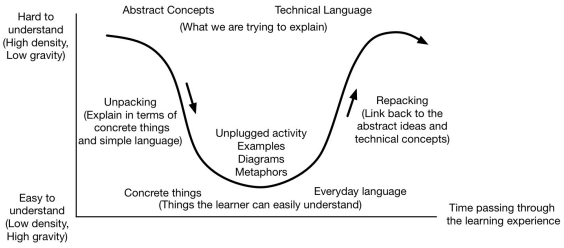


Figure 24: Semantic Wave [69]

Teachers' explanations naturally and necessarily build upon students' prior knowledge. So too, an NM may build upon a previous NM. For example, a "Variable as Box" NM can be extended to be "List as a Line of Boxes" or "Stack as a Stack of Boxes" for use later in the course. Similarly, an NM showing equality of references using arrows, may later be built upon to introduce the NM that every item has a unique ID.

An NM may be nested within another NM. This can be seen as a special case of building upon. For example, a "Variable as Parking Space" NM may be explicitly incorporated into an "Array as a Row of Parking Spaces in a Parking Lot," presented in Figure 25. NM. Another example, a "Variable as Box" NM may be explicitly incorporated into an "List is a Line of Boxes" NM. These are different from the way in which the NM using unique IDs builds upon an understanding of equality using an NM with equality represented with arrows, which does not explicitly incorporate that NM.

Array as Row of Spaces in Parking Lot

Programming Paradigm: imperative
Programming Language: any

Conceptual Advantage
Builds from the notion of a variable as a parking space and leverages the notion that spaces in larger parking lots are often numbered.

PL	NM
array	row of cars in a parking lot
array index	space number in lot
array element	car
array value	specific car in specific space

Form
Analogy

Draws Attention To
The use of indices in arrays as well as the array's construction from contiguous adjacent variables in memory.

Cost
Short time to introduce but learners need to be familiar with parking lots and how someone might locate a specific car in a row in a parking lot.

Notes/Other
If in a statically-typed language one can discuss rows that are for cars only, trucks only, motorcycles only, etc.

Origin/Source
Own practice

Attribution
Used by Jan, collected by ***Jan***

This NM works as an analogy in that it draws on student understanding of rows of numbered parking spaces in lots, where consecutive bays are numbered as an analogy of the index and represent consecutive variables in memory.
The effectiveness of the analogy depends on student knowledge of numbered parking spaces and how well that models the semantics of the programming language.

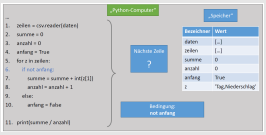
Figure 25: Array as Row of Spaces in Parking Lot

Sometimes NMs are very explicitly sequenced, for example in the consistent use of an environment to portray a series of NM. Figure 26 presents an example of the first in such a sequence.

Eventually, this series of NMs (used in a consistent environment) builds to include concepts such as lists, references, classes, objects and methods. It is interesting to speculate that this sort of approach serves a different purpose in the semantic wave framework, in that it is not to "ground" students' understanding (on the down side of

Python Computer V1

Programming Paradigm:
imperative / object-oriented
Programming Language:
Python (adaptable to others)



Conceptual Advantage
Expandable "offline" model of program execution. In its beginning stage (V1) it only covers control flow and variables (without references) of Python scripts.

Mapping

PL	NM
memory	a table of identifer and value
instruction	update of NMs state according to defined rules
variable	entry in the table
conditional	conditional evaluator (flag) that affects the "program counter"
control flow	program counter that identifies next line to get executed and is updated according to a set of rules.

Form
Handmade representation

Draws Attention To
Semantics of control flow in Python. Visible effects of each line on the "state" of the Python computer.

Use When
At the beginning of a typical "walk through" of the different programming constructs.

Cost
Time consuming to prepare. Easy to follow for learners, as it grows in complexity. Can be used for exercises (fill out sheets of the computer).

Notes/Other
Can be expanded (see V2-V4) and easily adpated to other (imperative) languages.

Origin/Source
Own practice

Attribution
Used by Andreas, collected by ***Andreas***

This NM is part of a sequence of four, increasingly complex, NMs. The increasing complexity follows the educational trajectory of introducing concepts of procedural and object-oriented programming in Python. At the first stage, only a few concepts are introduced and conversely a rather simple NM is offered to students as a means of explaining program behaviour based on semantic rules and a visualization of stepping through a program.
In subsequent stages, the model is expanded to allow for e.g. reference semantics to be incorporated. Each expansion follows the introduction of some concept or phenomenon that cannot be explained by the previous stage and each stage is a superset of the previous one, i.e. nothing is removed or replaced along the trajectory.

Figure 26: Python Computer V1

the wave) but to elevate it (on the up side), increasing the semantic density of their knowledge.

These examples of tightly integrated approaches may also suggest that NMs are not best sourced or presented singly - and also leads to Keith Johnstone's wry observation on the transfer of teaching knowledge "if you want to apply the methods I'm describing ... you may have to teach the way I teach." [38, p29].

6.2 NM lifecycles

Several of the NM examples we collected were very small, very specific, and deliberately constructed to have a limited lifespan.

> The number of times the same NM is used, or maybe used in a slightly different way, is also part of the story that we're telling. So we had the idea that there needed to be a very simple Prolog NM at the very beginning which you couldn't do without, but it didn't have a very long life. It was necessary, but not actually sufficient. I think that notion of the role of time in the exposure of the learners to the NM is also an important issue here. [BdB, meeting notes]

We also noticed that these "atomic" NMs aimed at beginning students, most often analogical, are most often based in domains external to the discipline, in the "real world" of fork-lift trucks and stacks of paper, whereas those aimed at more advanced students were more likely to be drawn from within computing using technical language and computational concepts:

> When I explain more advanced language features – let's say I explain lambdas or inner classes in Java, the way I explain that is not using metaphors or analogies outside the domain of programming. I'm basically rewriting code using these new features in code that is just using existing features that students already understand. So my NM [now] is just a subset of the programming language. [MH, meeting notes]

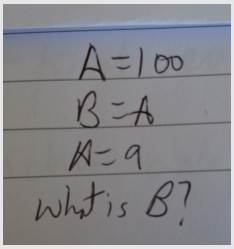
This is perhaps not surprising. As a student learns to program, they learn what to pay attention to and their focus becomes more refined, as Gibson and Radar put it, "There is increasing specificity of correspondence between what is perceived and recognition of its utility for performance of some task. The task itself becomes more specific, in the sense that the person progresses in ability to define it more precisely; and the recognition of a relation between information in an event and its utility for the task becomes more precise" [22].

6.3 Diagnosis

We also found evidence of a quite separate sort of use, of which we did not have as many examples, but they were distinct. Rather than using NMs to help students build knowledge, some teachers were using them to expose students' mental models, and so ensure they can make appropriate interventions. Figure 27 provides an example of one such diagnostic NM.

Greg Wilson devised a very simple test to ascertain whether his incoming students have a programming or spreadsheet background. The test reliably distinguishes whether students have a *sequential execution* or *constant evaluation* mental model.

How do you tell?



Programming Paradigm:
Imperative

Conceptual Advantage
Makes visible students' mental model of program execution, whether from *programming* (sequential execution) or *spreadsheet* (evaluates formulae and updates values on an ongoing basis).

Form
Handmade representation

Cost
Insignificant

Origin/Source
Interview

Attribution
Used by Greg Wilson; collected by ***Sally***

Greg Wilson devised a very simple test to ascertain whether his incoming students have a programming or spreadsheet background.

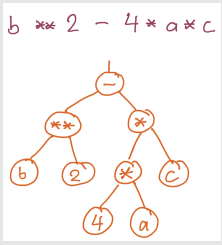
The test reliably distinguishes whether students have a ***sequential execution*** *or* ***constant evaluation*** *mental model.*

Figure 27: A diagnostic NM

The Informa Clicker system was developed with this diagnostic goal in mind. Besides the multiple-choice questions of traditional classroom response systems, Informa provides question types that embody various NMs [29, 30, 32], including those we captured here as "Stack and Heap Diagram", "Control Flow as Graph", and "Expression as Tree". Informa used these NMs to have students construct or complete a diagram of program state (for the "Stack and Heap Diagram") or of a program fragment (for the "Control Flow as Graph" or "Expression as Tree") during a lecture. This allowed the instructor to immediately detect and correct possible misconceptions. For example, "Stack and Heap Diagram" allowed diagnosing the understanding of concepts such as object allocation, references and aliasing, method calls, parameter passing, and polymorphism, "Control Flow as Graph" provided insight into misconceptions about the semantics of for loops or exception handling, and "Expression as Tree" helped to detect problems in the understanding of operator precedence and associativity (see Figure 28).

Other diagnostic uses we found involved presenting students with an NM (the diagram on the left in Figure 29) and requiring them to overlay the flowchart on their own code, (the tracing on the right in Figure 29). The goal of this particular exercise is to highlight

Expression as Tree



Programming Paradigm:
any
Programming Language:
any

Conceptual Advantage
Makes explicit the tree structure of expressions, which otherwise is quite hidden in many text-based languages

PL	NM
expression	tree
operator	parent node
operand	child node

Form
Handmade representation

Draws Attention To
Expression structure, how evaluation proceeds, how types are determined

Use When
Distinguishing between expressions and statements, how to determine the value of an expression (and explaining associativity, precedence)

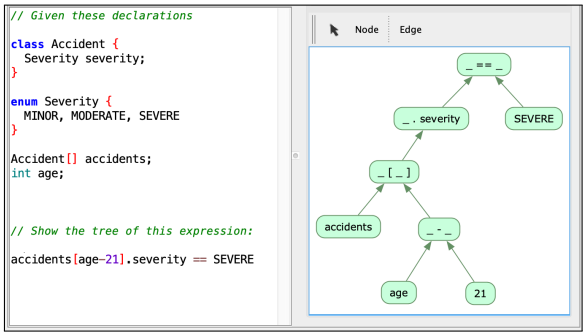
Cost
Learn a visual representation

Notes/Other
High school teachers were excited about this, mentioned this should also be used in math

Origin/Source
Own practice
(inspired by compiler ASTs)

Attribution
Used by Matthias; collected by ***Matthias***

This NM allows the diagnosis of misconceptions related to the structure and evaluation of expressions. The variant embedded in the Informa Clicker system provides students with an expression tree diagram editor:



The specific purpose of the above example task is to assess the students' understanding of array indexing and field accesses, in combination with previously covered arithmetic and relational operators.

Figure 28: Diagnostic NM for the structure and evaluation of expressions

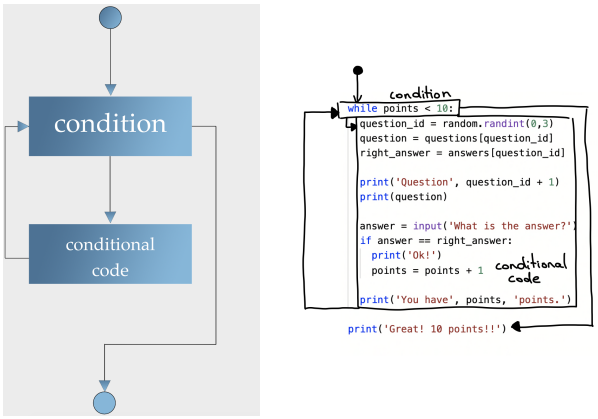


Figure 29: A flow chart teaching students how to execute code in a while loop, emphasizing checking the condition at each iteration

that after each execution of the body, the condition is examined again. This helps the teacher to understand the understanding of the student, which can be very hard to reach, and serves as a diagnostic tool to uncover their emerging mental model.

7 SUMMARY/CONCLUSION/FUTURE WORK

The aim of this working group was to explore the history, meanings, uses and value of notional machines as aids to teaching about programming and computers. There were four objectives:

- To conduct a literature review, to ground and inform the work;
- To capture examples of NMs in use;
- To catalogue them in a common scheme; and,
- To arrange them in clusters or sequences.

7.1 Summary and conclusions

The structure of this report has broadly followed the sequence of these original objectives.

The history section examined the initial development of NMs in the context of teaching programming in the late 1970s. It also distinguished NMs from mental models and conceptual models, as the terms have often been used as if they meant the same thing.

The literature review used a systematic review procedure to examine papers referring to NMs since the term was coined. The results of the review have been presented in a number of different ways that plot the development of the term, the kinds of educational research within which the term occurs, the degree to which there were references to specific NMs as opposed to some general mention of them as a "good idea", and via the range of terms used to describe them.

The main lessons from this review section have been (i) that the term itself has been used in many different ways, and (ii) that there were surprisingly few papers that examined the evidence in detail for their strengths and weaknesses in authentic educational settings. With regard to the first issue, we define *an NM as a pedagogic device to assist the understanding of some aspect of programs or programming*. With regard to the second issue, we noted that some of the educational literature that is relevant, for example cognitive load theory, would not have been found via the search terms in our review procedure that was focused on the computing literature. To compensate, we have included a brief section that points towards the educational and psychological literature, which may also help future researchers in designing studies on the effectiveness of NMs.

Next, in section 4 we characterise NMs using our new definition of the term (prefaced in the box in section 1). We have created ways of looking at NMs more holistically in terms of which aspects of programs and computers are most often referenced, as opposed to the more pedagogic focus of the previous section. This involves the generic purpose and generic function of NM, and the particular focus and particular representation of individual examples of NMs.

Finally, to gain a better sense of the current practice of using NMs in teaching, we conducted interviews with colleagues, examined our own teaching, and looked at the literature review to identify specific examples of NMs. These were found to fall into two main classes, namely software-generated representations, such as provided by IDEs, and handmade representations as drawn or created or described by teachers. In many cases, (particularly the handmade) NMs often relied on analogy to make salient and visible some aspect of the largely hidden underlying system. Because analogy plays such a strong role in pedagogy that uses NMs, we designated it as a third class, even though technically it is largely orthogonal to the two representations. We also looked at the temporal aspect of using NMs, i.e. how explanations at different levels of abstractness can be sequenced, referring to semantic wave theory in so doing.

A lesson from this section is that attempting to put NMs into neat watertight categories is hard and that there are grey areas between categories. As part of this work, we developed a visual notation for describing the NMs and have included many examples in the report.

There are four appendices: a list of NMs uncovered through our work, a description of the literature review methodology, the form we used to categorise papers in the literature review, and the protocol used in the interviews.

7.2 Future work

There are different challenges for the community depending on where one's main focus lies. As teachers of programming, much of actual practice is not documented or is scattered through the grey literature, so we hope that this report will provide a resource that will be helpful. But further work is needed to identify the strengths and weaknesses of particular NMs in specific authentic educational contexts. There is still much to understand about how to manage the problems that NMs raise. Two of these involve time. The first is how to build a coherent set of NMs that can function well together over a whole introductory course. Allied with this, is the issue of how best to orchestrate the learner's exposure to abstract ideas and hands-on practice in a way that maximises learning. There is also the ever-present possibility that incidental aspects of the NMs, especially those involving analogy, may lead the student astray to form unhelpful mental models.

From a researcher's point of view, we need to have a more clearly articulated pedagogic theory about learning and teaching with NMs that brings together the range of pedagogic and psychological theories that have been applied in other areas of STEM education. In particular, NMs as a candidate signature pedagogy has not received enough research attention. Lee Shulman proposes that disciplines often have a "signature pedagogy" which teachers and learners recognise as part of knowledge building in their discipline. He says, "To the extent that we identify signature pedagogies, we find modes of teaching and learning that are not unique to individual teachers, programs or institutions …" and that if a signature pedagogy does exist "… we should be able to find it replicated in nearly all the institutions that educate in that domain." [62]. Much more work would have to be done to affirm that NMs are computing's signature pedagogy, but based on this work we propose that they should be considered a good candidate. We call for studies that compare and contrast the effectiveness of various NMs in teaching and learning. We hope that our report invigorates new research on NMs – to name a few examples, significant research gaps exist in identifying appropriate abstraction levels of NMs, determining why certain analogies may be more useful than others, determining the effect of students' background characteristics such as age and previous programming experience of learning with NMs, determining the effect of interactivity and engagement of NMs, identifying appropriate sequencing of NMs and how this should be tied to the broader learning context, including the available learning materials.

ACKNOWLEDGEMENTS

This work builds on initial activity undertaken at Dagstuhl seminar 19281, Notional Machines and Programming Language Semantics in Education, 7th-12th July 2019. We would like to thank the interviewees who provided us with insight into their use of notional machines: Antonio Carzaniga, Luca Chiodini, Steve Engels, Jeroen Fokker, Scott Heggen, Steve Jost, Shriram Krishnamurthi, Michael Liut, John Lynch, James Riely, John Rogers, Dermot Shinners-Kennedy, Simon Thompson, Amber Settle, Juha Sorva, Greg Wilson, Daniel Zingaro.

REFERENCES

[1] Karen E. Adolph and Kari S. Kretch. 2015. Gibson's Theory of Perceptual Learning. In *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*, James D. Wright (Ed.). Elsevier, Oxford, 127–134. https://doi.org/10.1016/B978-0-08-097086-8.23096-1

[2] Shaaron Ainsworth. 2006. DeFT: A conceptual framework for learning with multiple representations. *Learning and Instruction* 16, 3 (2006), 183–198. Type: Journal Article.

[3] Sabah Al-Fedaghi and Hadeel Alnasser. 2018. Network architecture as a thinging machine. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, New York, NY, 884–889.

[4] Anon. [n.d.]. Scholarcy Reference Extraction API. https://ref.scholarcy.com/api/

[5] David John Barnes, Michael Kölling, and James Gosling. 2006. *Objects First with Java: A practical introduction using BlueJ*. Pearson/Prentice Hall.

[6] B. Barquero. 1995. *La representacion de estados mentales en la comprension de textos desde el enfoque teorico de los modelos mentales*. Thesis. Universidad Autónoma de Madrid, Madrid, Spain.

[7] A. Barr, M. Beard, and R.C. Atkinson. 1976. The computer as a tutorial laboratory: the Stanford BIP project. *International Journal of Man-Machine Studies* 8, 5 (1976), 567–582. https://doi.org/10.1016/S0020-7373(76)80021-1 Type: Journal Article.

[8] Mordechai Ben-Ari. 1998. Constructivism in computer science education. *Acm sigcse bulletin* 30, 1 (1998), 257–261. Publisher: ACM New York, NY, USA.

[9] Jens Bennedsen and Carsten Schulte. 2010. BlueJ Visual Debugger for Learning the Execution of Object-Oriented Programs? *ACM Trans. Comput. Educ.* 10, 2 (June 2010), 1–22. https://doi.org/10.1145/1789934.1789938 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[10] Shirley Booth. 1993. A Study of Learning to Program From an Experiential Perspective. *Computers in Human Behavior* 9, 2-3 (1993), 185–202. https://doi.org/10.1016/0747-5632(93)90006-E Type: Journal Article.

[11] A.B. Cannara. 1976. *Experiments In Teaching Children Computer Programming*. Report 271. Stanford University.

[12] John M. Carroll and John C. Thomas. 1982. Metaphor and the Cognitive Representation of Computing Systems. *IEEE Transactions On Systems, Man, and Cybernetics* 12, 2 (1982), 107–116. Type: Journal Article.

[13] Jie Chao, David F. Feldon, and James P. Cohoon. 2018. Dynamic Mental Model Construction: A Knowledge in Pieces-Based Explanation for Computing Students' Erratic Performance on Recursion. *Journal of the Learning Sciences* 27, 3 (2018), 431–473. https://doi.org/10.1080/10508406.2017.1392309 Type: Journal Article.

[14] Ton De Jong. 2010. Cognitive load theory, educational research, and instructional design: some food for thought. *Instructional Science* 38, 2 (2010), 105–134. Type: Journal Article.

[15] Benedict du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2, 1 (Feb. 1986), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

[16] B. du Boulay and T. O'Shea. 1976. *How to work the LOGO Machine: a primer for ELOGO*. University of Edinburgh, Department of Artificial Intelligence, Edinburgh, Scotland. Type: Book.

[17] Benedict du Boulay, Tim O'Shea, and John Monk. 1981. The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies* 14, 3 (1981), 237–249. http://www.sciencedirect.com/science/article/B6WGS-4T7YSPP-2/2/ea0352066f4fcc9d43d666d0c872090b Type: Journal Article.

[18] Sally Fincher, Marian Petre, and Martyn Clark (Eds.). 2001. *Computer Science Project Work: Principles and Pragmatics* (2001 edition ed.). Springer, London ; New York.

[19] Robert Bruce Findler. 2014. *DrRacket: The Racket Programming Environment*. https://docs.racket-lang.org/drracket/index.html.

[20] Roman Frigg and Stephan Hartmann. 2020. Models in Science. In *The Stanford Encyclopedia of Philosophy* (spring 2020 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[21] Emily R. Fyfe, Nicole M. McNeil, and Stephanie Borjas. 2015. Benefits of "concreteness fading" for children's mathematics understanding. *Learning and Instruction* 35 (2015), 104–120. Type: Journal Article.

[22] Eleanor Gibson and Nancy Rader. 1979. Attention: The Perceiver as Performer. In *Attention and cognitive development*. Springer, Boston, MA, 1 – 21. https://doi.org/10.1007/978-1-4613-2985-5_1

[23] John K. Gilbert and Rosária Justi. 2016. Analogies in Modelling-Based Teaching and Learning. In *Modelling-based Teaching in Science Education*. Springer, Cham, 149–169. https://doi.org/10.1007/978-3-319-29039-3_8 Type: Book Section.

[24] Charles Goodwin. 2000. Practices of Color Classification. *Mind, Culture, and Activity* 7, 1 (May 2000), 19–36. https://doi.org/10.1080/10749039.2000.9677646

[25] Charles Goodwin. 2007. Participation, stance and affect in the organization of activities. *Discourse & Society* 18, 1 (Jan. 2007), 53–73. https://doi.org/10.1177/0957926507069457

[26] Ileana Maria Greca and Marco Antonio Moreira. 2000. Mental models, conceptual models, and modelling. *International Journal of Science Education* 22, 1 (2000), 1–11. https://doi.org/10.1080/095006900289976 Type: Journal Article.

[27] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, New York, NY, 579–584.

[28] Mark Guzdial and Benedict du Boulay. 2019. The History of Computing Education Research. In *The Cambridge Handbook of Computing Education Research*, Sally Fincher and Anthony Robins (Eds.). Cambridge University Press, Cambridge, 11–39. Section: 1 Type: Book Section.

[29] Matthias Hauswirth. 2012. Moving from Visualization for Teaching to Visualization for Learning. In *Workshop on Visualization in University Level Computer Science Education @ CSERC'12*. Wroclav, Poland.

[30] Matthias Hauswirth and Andrea Adamoli. 2009. Solve & Evaluate with Informa: A Java-Based Classroom Response System for Teaching Java. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java (PPPJ '09)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/1596655.1596657

[31] Matthias Hauswirth and Andrea Adamoli. 2013. Teaching Java programming with the Informa clicker system. *Science of Computer Programming* 78, 5 (2013), 499–520.

[32] Matthias Hauswirth and Andrea Adamoli. 2013. Teaching Java Programming with the Informa Clicker System. *Sci. Comput. Program.* 78, 5 (May 2013), 499–520. https://doi.org/10.1016/j.scico.2011.06.006

[33] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V. Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Predicting academic performance: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion)*. Association

for Computing Machinery, Larnaca, Cyprus, 175–199. https://doi.org/10.1145/3293881.3295783
[34] Thomas T. Hewett. 2005. Cognitive factors in design: overview and some implications for design. In *Proceedings of the 5th conference on Creativity & cognition - C&C '05*. ACM Press, London, United Kingdom, 318. https://doi.org/10.1145/1056224.1056287
[35] James A.M. Howe and J. Benedict H. du Boulay. 1979. Microprocessor Assisted Learning: Turning the Clock Back? *Programmed Learning and Educational Technology* 16, 3 (1979), 240–246. https://doi.org/10.1080/0033039790160309 Type: Journal Article.
[36] Tim Ingold. 2000. *The Perception of the Environment: Essays on Livelihood, Dwelling and Skill*. Psychology Press, London, UK.
[37] Philip N. Johnson-Laird. 1983. *Mental Models: Towards a Cognitive Sceince of Language, Inference and Consciousness*. Harbard University Press, Cambridge, Massachusetts. Type: Book.
[38] Keith Johnstone. 2012. *Impro: Improvisation and the Theatre*. Routledge, Abingdon, England. Google-Books-ID: EVmminvaWDQC.
[39] Paul A. Kirschner, Paul Ayres, and Paul Chandler. 2011. Contemporary cognitive load theory research: The good, the bad and the ugly. *Computers in Human Behavior* 27, 1 (2011), 99–105. Type: Journal Article.
[40] Timothy Koschmann, Curtis LeBaron, Charles Goodwin, Alan Zemel, and Gary Dunnington. 2007. Formulating the Triangle of Doom. *Gesture* 7, 1 (2007), 97–118. https://core.ac.uk/display/60531167
[41] Shriram Krishnamurthi and Kathi Fisler. 2019. Programming Paradigms and Beyond. In *The Cambridge Handbook of Computing Education Research*, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge, 377–413. https://doi.org/10.1017/9781108654555.014
[42] Raymond Lister. 2011. Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer. In *13th Australasion Computer Education Conference (ACE 2011)*. Australian Computer Society, Darlinghurst, NSW, 9–18.
[43] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A multi-national study of reading and tracing skills in novice programmers. https://doi.org/10.1145/1041624.1041673
[44] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education* 21, 1 (2011), 57–80. Publisher: Taylor & Francis.
[45] Karl Maton. 2013. Making semantic waves: A key to cumulative knowledge-building. *Linguistics and Education* 24, 1 (April 2013), 8–22. https://doi.org/10.1016/j.linged.2012.11.005
[46] R.E. Mayer. 1979. A psychology of learning BASIC. *Commun. ACM* 22, 11 (1979), 589–593. https://doi.org/10.1145/359168.359171 Type: Journal Article.
[47] Richard E. Mayer. 2014. Cognitive Theory of Multimedia Learning. In *The Cambridge Handbook of Multimedia Learning* (2nd ed.), R.E. Mayer (Ed.). Cambridge University Press, Cambridge, England, 43–71. https://doi.org/10.1017/CBO9781139547369.005 Section: 3 Type: Book Section.
[48] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01)*. Association for Computing Machinery, Canterbury, UK, 125–180. https://doi.org/10.1145/572133.572137
[49] Iain McGilchrist. 2019. *The Master and His Emissary: The Divided Brain and the Making of the Western World* (2, new expanded edition ed.). Yale University Press, New Haven.
[50] Claire Michaels and Claudia Carello. 1981. Direct Perception. *New Jersey, Englewood Cliffs: Prentice-Hall. Moggridge, B. (1993). Design by storytelling. Applied Ergonomics* 24 (Jan. 1981).
[51] Lance A. Miller. 1974. Programming by non-programmers. *International Journal of Man-Machine Studies* 6, 2 (1974), 237–260. https://doi.org/10.1016/S0020-7373(74)80004-0 Type: Journal Article.
[52] David Mowatt. 2020. *Giving the computer information.* https://twitter.com/ItsAll_Geek2Me/status/1271380040043954176
[53] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and others. 2002. Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*. ACM, New York, NY, 131–152.
[54] David Perkins and Fay Martin. 1985. *Fragile Knowledge and Neglected Strategies in Novice Programmers*. Ablex, Norwood, NJ. https://eric.ed.gov/?id=ED295618
[55] Michael Proffitt et al. (Ed.). 2020. *Oxford English Dictionary*. Oxford University Press.
[56] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172. Publisher: Taylor & Francis.
[57] Pablo Romero, Benedict du Boulay, Judy Robertson, Judith Good, and Katherine Howland. 2009. Is Embodied Interaction Beneficial When Learning Programming?. In *Virtual and Mixed Reality (Lecture Notes in Computer Science)*, Randall Shumaker (Ed.). Springer, Berlin, Heidelberg, 97–105. https://doi.org/10.1007/978-3-642-02771-0_11
[58] Wolfgang Schnotz and Christian Kürschner. 2007. A Reconsideration of Cognitive Load Theory. *Educational Psychological Review* 19, 4 (2007), 469–508. https://doi.org/10.1007/s10648-007-9053-4 Type: Journal Article.
[59] Norbert M. Seel. 2017. Model-based learning: a synthesis of theory and research. *Educational Technology Research and Development* 65, 4 (2017), 931–966. https://doi.org/10.1007/s11423-016-9507-9 Type: Journal Article.
[60] Nianfeng Shi, Zhiyu Min, and Ping Zhang. 2017. Effects of visualizing roles of variables with animation and IDE in novice program construction. *Telematics and Informatics* 34, 5 (Aug. 2017), 743–754. https://doi.org/10.1016/j.tele.2017.02.005
[61] Dermot Shinners-Kennedy and David J. Barnes. 2011. The novice programmer's "device to think with". In *Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE '11)*. Association for Computing Machinery, Dallas, TX, USA, 511–516. https://doi.org/10.1145/1953163.1953310
[62] Lee S. Shulman. 2005. Signature pedagogies in the professions. *Daedalus* 134, 3 (June 2005), 52–59. https://doi.org/10.1162/0011526054622015
[63] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *ACM Trans. Comput. Educ.* 13, 2 (July 2013), 1–31. https://doi.org/10.1145/2483710.2483713 Place: New York, NY, USA Publisher: Association for Computing Machinery.
[64] Juha Sorva. 2019. Splashing the Surface of Research: A Study of Koli Abstracts. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research (Koli Calling '19)*. Association for Computing Machinery, Koli, Finland, 1–2. https://doi.org/10.1145/3364510.3366148
[65] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans. Comput. Educ.* 13, 4 (Nov. 2013), 1–64. https://doi.org/10.1145/2490822 Place: New York, NY, USA Publisher: Association for Computing Machinery.
[66] Joyce Ann Statz. 1973. *The Development Of Computer Programming Concepts And Problem-Solving Abilities Among Ten-Year-Olds Learning Logo*. Thesis. Syracuse University, Syracuse, NY.
[67] Gerrit C van der Veer and Robert Wijk. 1988. Teaching a spreadsheet application: visual-spatial metaphors in relation to spatial ability, and the effect on mental models. In *Interdisciplinary Workshop on Informatics and Psychology*. Springer, New York, NY, 194–208.
[68] Dimitrios Vlachopoulos and Agoritsa Makri. 2017. The effect of games and simulations on higher education: a systematic literature review. *International Journal of Educational Technology in Higher Education* 14, 22 (2017), 1–33. https://doi.org/10.1186/s41239-017-0062-1 Type: Journal Article.
[69] Jane Waite, Karl Maton, Paul Curzon, and Lucinda Tuttiett. 2019. Unplugged Computing and Semantic Waves: Analysing Crazy Characters. In *UKICER: Proceedings of the 1st UK & Ireland Computing Education Research Conference*. ACM, New York, NY, 1–7. https://doi.org/10.1145/3351287.3351291
[70] Gerald M. Weinberg. 1971. *The Psychology of Computer Programming*. Van Nostrand / Reinhold, New York. Type: Book.
[71] E.A. Youngs. 1974. Human errors in programming. *International Journal of Man-Machine Studies* 6, 3 (1974), 361–376. https://doi.org/10.1016/S0020-7373(74)80027-1 Type: Journal Article.

A LIST OF NOTIONAL MACHINES

We collected 43 notional machines. For each NM, we give its name, its form (analogy, handmade representation, machine-generated representation), the source from which we retrieved the NM (interview or own practice), and the person who retrieved it (one of the authors of this paper). We list the NMs under various categories, primarily describing the targeted programming constructs, such as data structures or functions. A more accessible version of this information is available online: https://notionalmachines.github.io (under construction).

State / variables

(1) Variables as Boxes
Form: Analogy
Source: own practice
Collected by: Felienne
(2) Arrays as Stacks of Boxes
Form: Analogy

Source: own practice
Collected by: Felienne

(3) Variables as Parking Spaces
Form: Analogy
Source: own practice
Collected by: Jan

(4) Variables as Pigeon Holes
Form: Analogy
Source: Introduction to Computer Programming, 1982
Collected by: Matthias

(5) Typed Boxes Memory
Form: handmade representation
Source: Interview with Jeroen Fokker
Collected by: Johan

(6) Variables as Clothes Pins
Form: handmade representation
Source: Interview with Emily Bakker
Collected by: Felienne

(7) Arrays as Clothes Line
Form: handmade representation
Source: Interview with Emily Bakker
Collected by: Felienne

(8) Stack & Heap Diagram
Form: handmade representation
Source: Interview with Luca Chiodini
Collected by: Matthias

(9) Python Computer V1
Form: handmade representation
Source: own practice
Collected by: Andreas

(10) Computer as Active Actor
Form: Machine-generated representation
Source: http://cs.joensuu.fi/jeliot/description.php
Collected by: *Unknown*

(11) Variable Trace Table
Form: handmade representation
Source: Interview with Steve Jost
Collected by: Craig

Behavior / Functions / Control-Flow
Data Flow / Evaluations / Expressions

(12) Simulating Software Behavior in Spreadsheet
Form: Machine-generated representation
Source: Interview with Frans Wiering
Collected by: Johan

(13) Operation as Domino Line
Form: Analogy
Source: own practice
Collected by: Matthias

(14) Expression as Tree
Form: handmade representation
Source: Matthias, based on standard abstract syntax tree representions
Collected by: Matthias

Control-Flow (intra-procedural)

(15) Control-Flow as Graph (Short Circuit)
Form: handmade representation
Source: own practice
Collected by: Matthias

(16) Manual while execution
Form: handmade representation
Source: Interview with Benjamin Turner
Collected by: Felienne

(17) TRACS Structured Code Annotation
Form: handmade representation
Source: own practice
Collected by: Peter

Control-Flow (inter-procedural)

(18) Calls as Sequence Diagram
Form: handmade representation
Source: own practice
Collected by: Matthias

(19) Hand-Drawn Runtime Stack
Form: handmade representation
Source: Interview with John Rogers
Collected by: Craig

(20) Python Computer V2
Form: handmade representation
Source: own practice
Collected by: Andreas

(21) The Shape of Computation
Form: Machine-generated representation
Source: Interview with Shriram Krishnamurthi
Collected by: Sally

(22) Recursion Demo with Python Tutor
Form: Machine-genetrated representation
Source: Interview with Amber Settle
Collected by: Craig

Data structures

(23) Function as Black Box
Form: Analogy
Source: Interview with Scott Heggen
Collected by: Jan

(24) Call Stack as Stack of Papers
Form: Analogy
Source: Interview with Antonio Carzaniga
Collected by: Matthias

(25) List as Stack of Boxes
Form: Analogy
Source: own practice
Collected by: Ben

(26) HashSet as Hanging Folders
Form: Analogy
Source: own practice
Collected by: Colleen

(27) Paper List/ListNode
Form: handmade representation
Source: own practice
Collected by: Colleen

(28) Linked-List Visualization
Form: Machine-generated representation
Source: Interview with James Riely
Collected by: Craig

Table 3: Results for search of "notional machine OR notational machine".

Library	Results
ACM Digital Library (ACM Guide to Computing Literature)	127
IEEE Xplore (Full Text & Metadata)	36
Scopus (All fields)	137
Total	300

(29) Array as Row of Spaces in Parking Lot
Form: Analogy
Source: own practice
Collected by: Jan

Analogies to Humans

(30) Object as Student
Form: Analogy
Source: own practice
Collected by: Matthias
(31) Recursion Role Play
Form: Analogy
Source: own practice
Collected by: Matthias
(32) Method-call Dance
Form: Analogy
Source: Interview with Jeroen Fokker
Collected by: Johan
(33) List Elements as Students
Form: Analogy
Source: Interview with John Lynch
Collected by: Craig
(34) Processor as File Clerk
Form: Analogy
Source: Interview with Antonio Carzaniga
Collected by: Matthias
(35) Blackboard Processing
Form: handmade representation
Source: Interview with (anonymous)
Collected by: Andreas
(36) Worker Analogy
Form: handmade representation
Source: own practice
Collected by: Ben

Other

(37) Grammar as Train Track
Form: Analogy
Source: own practice
Collected by: Matthias
(38) Verbal Expressions
Form: Analogy
Source: own practice
Collected by: Craig
(39) How do you tell?
Form: handmade representation
Source: Interview with Greg Wilson
Collected by: Sally
(40) The Role of Variables
Form: handmade representation
Source: Jorma Sajaniemi, http://saja.kapsi.fi/var_roles/
Collected by: Sally
(41) Real Machines: The Analogy in Your Pocket
Form: Analogy
Source: Interview with Dermot Shinners-Kennedy
Collected by: Sally
(42) Python Computer V3
Form: handmade representation
Source: own practice
Collected by: Andreas
(43) Python Computer V4
Form: handmade representation
Source: own practice
Collected by: Andreas

B SYSTEMATIC LITERATURE REVIEW PROCESS

This appendix outlines our systematic literature review process, outlining how the relevant literature was identified, augmented, filtered, and extracted. The overarching research questions that guided the review were as follows:

RQ1 How are notional machines defined?
RQ2 How are NMs discussed / used?
RQ3 In which contexts are NMs used?

B.1 Identification of relevant literature

The group used ACM Digital Library (ACM Guide to Computing Literature), IEEE Xplore (Full Text & Metadata), and Scopus (All fields) to conduct searches for relevant literature. Initially, the term "notional machine" was used as the query string. Rare occurrences where the term "notational machine" was used interchangeably with "notional machine" was observed, however. As a consequence, the query string was adjusted to "notional machine OR notational machine". The libraries and the results for searching for "notional machine OR notational machine" are shown in Table 3. The searches were conducted on 3.3.2020. As shown in Table 3, a total of 300 articles (including duplicates) were identified.

In parallel discussions with domain experts, the term "conceptual machine" was highlighted as one possible term used to represent NMs. As the group was already working on the dataset containing results from the query "notional machine OR notational machine", a separate effort to identify articles discussing conceptual machines was initiated. Searching for the term "conceptual machine" in the used libraries produced a total of 141 articles (including duplicates), as shown in Table 4.

Figure 3 presented a timeline of the papers found in our searches. There's a noticeable gap in 2009 where we found no papers. A search on Google Scholar using the string "notional machine" identifies several potentially relevant papers that we did not find in the three databases we searched. This highlights a limitation of systematic reviews: while they aim to generate reproducible results, they do not necessarily produce complete results. There will be works missed either because they were not indexed in the databases searched or because the search terms do not uncover them.

Table 4: Results for search of "conceptual machine".

Library	Results
ACM Digital Library (ACM Guide to Computing Literature)	36
IEEE Xplore (Full Text & Metadata)	356
Scopus (All fields)	49
Total	141

B.2 Filtering the identified literature

The results from the keyword-based searches were studied in two phases. In the first phase, the search results from the different digital libraries were merged and clear duplicates (based on title, DOI, and abstract) were excluded. This produced a list with 236 articles for the query "notional machine OR notational machine" and 112 articles for the query "conceptual machine".

In the second phase, four researchers went through the article titles and abstracts to identify out of scope articles: we excluded results that were not written in English, that were not related to programming, that indicated no content (removing e.g. proceedings forewords), or that were shorter than 2 pages. Each article was assessed independently by two researchers, who either voted for including or excluding the article. To avoid premature exclusion of potentially relevant papers, we used an inclusive approach and included all articles where the researchers were undecided (one voted for include and the other voted for exclude). Cohen's kappa between pair-wise reviewers varied between .46 and .78, indicating moderate to substantial agreement. Here, the seemingly low agreement is partially by imbalanced distribution of classes, i.e. low or high exclusion rate.

After the second phase, a total of 223 articles for the query "notional machine OR notational machine" and a total of 14 articles for the query "conceptual machine" were remaining. Merging the datasets produced a total of 236 articles after one article that was present in both datasets was removed.

B.3 Augmenting the keyword-based searches

In addition to the keyword-based searches on the digital libraries described above, the group constructed a list of expert-identified articles that could be of relevance to the topic but were missing from the search results. This list contained a total of 14 articles.

At this point, the dataset consisted of 250 articles. Whilst the group acknowledged that textbooks and instructional materials could be a good source for additional data, we opted to not include such sources to our analysis unless they explicitly were identified by the used search engines. This was decided as our main focus was on research literature and, in parallel, the group worked on identifying and characterizing NMs, as described in Section 5.

B.4 Development of the extraction sheet

Four researchers collaborated to develop an extraction sheet based on the overarching questions of the literature survey. The extraction sheet was iterated over several meetings, where the researchers used the preliminary versions of the extraction sheet to extract details from articles. Each iteration was followed by a joint discussion, where refinements to the extraction sheet were conducted. The extraction sheet shown in Appendix C was finalized in four iterations over a span of one month.

B.5 Data extraction

Data extraction was conducted by six researchers, including the four researchers responsible for the first pass of articles and the extraction sheet. Following the extraction sheet in Appendix C, the researchers independently extracted data from the identified literature. The data extraction was conducted over a time-period of three months, where a small set of articles were extracted each week. During data extraction, the researchers held weekly meetings to consider their findings, and to discuss potential concerns with the process.

When filtering the articles, as described earlier, the researchers studied the title and abstract of the article to determine whether the work was in scope. A scan of the body of 250 articles identified additional 11 articles that were considered to be out of scope when starting the extraction process, leaving a total of 239 articles that were extracted.

From these articles, during reading the articles and extracting information from the articles, a total of 34 articles were considered out of scope, and 205 in scope.

B.6 Further augmenting the dataset

To identify articles that were of relevance but did not appear in the digital library searches and that were not included by experts, we analyzed the references of the articles (n=205) that were found to be in scope during data extraction. The process used to extract the references is discussed in the next section.

Articles that were cited five or more times in the body of articles identified as being in-scope were manually analyzed for relevance by two independent researchers. A total of 91 new articles were identified, out of which 29 were considered potentially relevant after a first pass considering the title, abstract, and a high-level scan of the article. When extracting information from the 29 articles using the data extraction sheet, only 18 were considered to be in scope.

This brought the total number of extracted articles to 268, of which 45 were considered being out of scope during the extraction process. Finally, based on expert discussions, an additional 3 articles that were in scope were added to the body of articles. In total, the corpus consisted of 226 articles, which – after extraction – were further analyzed.

B.7 Extracting citations from the dataset

To explore how the development of NMs has evolved, we extracted the reference lists of the papers that we reviewed. Due to the variety of publishers hosting our papers, we determined that we could not easily extract references from web resources and, instead, extracted them directly from the PDF files of the papers under review. We used a service provided by Scholarcy [4] to extract and format the references.[3]

Due to the structure of PDF files, extraction of text is non-trivial, and transformation of the text into an analyzable format (such as a

[3]The service is available at https://ref.scholarcy.com/api/. We are grateful to Scholarcy for providing us with an API key at no charge.

bibtex entry) is also challenging due to the number of different reference formats in use. As a result, the process is lossy and imprecise: references cannot be reliably extracted from all articles (and cannot be extracted at all from PDFs that store their pages as images), and the references that are extracted may contain errors, such as incomplete titles or missing publication years or authors, which make matching citations difficult. In our experience, we found that papers were generally fully extractable or not at all: either most of the citations were largely correct or some feature of the PDF or the format of the paper led to many references in the paper being incorrect or incomplete.

To provide as accurate and representative a set of citations as possible, we limited extraction to more recent papers (published at or after 2000), since many earlier papers use an image format. After the bibliographies were extracted, an author reviewed each bibliography and discarded any bibliography in which paper titles were not being accurately extracted. 196 papers from the set were from 2000 or more recently, and of those, the bibliographies could be extracted from 165. Finally, the author reviewed all of the titles in the extracted bibliographies and corrected any remaining extraction errors identified.

C LITERATURE REVIEW EXTRACTION FORM

For the literature review extraction form, see Figure 30.

D INTERVIEW PROTOCOL

D.1 Capturing NMs with interviews: goals and principles

We sought to capture NMs as authentic practices in the classroom by interviewing instructors of computing courses. The goal was to obtain accounts of NMs that instructors have actually used in their teaching practice.

For all interviews, we provided guidance for helping instructors identify an NM used in their teaching. In particular, we stipulated that their NM must:

- Be something you've used with students
- Have a focus on developing conceptual understanding about program execution and/or program state

The interviewed instructors were encouraged to bring instructional materials to the interview and present them during the interview. Interviews were recorded and captured so that we could identify primary data as original quotes and any drawings or materials produced by the instructor.

Our initial interview protocol also prompted instructors to provide the following details:

- "What problem does this avoid / help students resolve?"
- "Do you use this idea regularly / frequently?"
- "Do you use this idea later – do you extend it, pick it up again in the course, … ?"
- "Does using that example create any issues later?"
- "Who do you use this with?"
- "Do you have an artifact that describes this example? (Could you share it.)"
- "Can you recall an inspiration for coming up with this example? When did you start using it and why?"
- "Have you used this with students in a room? How did it go?"

D.2 Additional development

We conducted 12 initial interviews. We then shared responses and organized them among ostensibly salient dimensions. We also identified additional details that came from some of the interviews that we thought would be of general interest or provide useful context. We then incorporated these details as prompts in a second version. Here are examples of additional prompts that were added:

- How did you first come up with this idea?
- Is it a recurring theme that you build upon in your [context / course]?
- Do you have a name for your notional machine?

The next section shows the resulting Recruitment Script and the Final Interview Protocol that resulted from these additions. We conducted 13 additional interviews with the added prompts. The chosen interviews were effectively a convenience sample targeting skilled teachers, although we obtained some diversity by asking each working group member to conduct at least one interview. In practice, working group members reached out to colleagues, all who agreed to be interviewed when asked. Appendix A presents which NMs were obtained by interviews and who conducted the interviews.

In addition to asking the interviewee's permission to share their NM to the public, we asked permission to attribute their identity to their NM. All interviewees affirmatively responded to this attribution. Ethics board reviews were passed with this form of consent and attribution.

D.3 Final interview script

Recruitment Mail

We are seeking examples of notional machines. **A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming.** We would like to interview you about a place in your teaching where you use a notional machine. It must:

- Be something you've used with students;
- Have a focus on developing conceptual understanding about program execution and/or program state.

Please bring any materials with you that would be helpful for explaining this teaching practice (e.g. a diagram, slides, a video, etc.).

[As appropriate, the interviewer may attach additional content for compliance with the interviewer's institutional ethics board.]

Interview Script

Field Name	Format and Options	Notes
Article Bibtex Entry	Short answer	For extraction of publication information such as year or authors
Article PDF	File chooser	For extraction of references
What type of an article is this?	Select-multiple from: Lab-based research study Classroom-based research study System paper (e.g. describing a tool) Methodological discussion (e.g. describing a method of how something should be done) Theory paper (e.g. forming a theory, perhaps based on evidence) Literature review Other (specify)	
In what educational context is the article set?	Select-multiple from: Uncontextualized Primary school Secondary school Tertiary education (e.g. college, university): CS1, CS2 Tertiary education (e.g. college, university): other courses Life-long learning (e.g. MOOCs) Other (specify)	
What are the article's explicit research questions?	Short answer	Copy-paste or "NA"
What is the purpose of referencing the concept of a notional machine in the article?	Select-multiple from: Motivation Related Work Theoretical basis Identifying or defining a notional machine Using a notional machine in an intervention Other (specify)	
How are notional machines explicitly defined in the article?	Short answer	Copy-paste, "implicit", or "NA"
If a notional machine is described in the article, what is it?	Short answer	Copy-paste, brief description, or "NA"
What is the educational or conceptual benefit of using a notional machine in this article?	Short answer	Copy-paste, short description, "unclear", or "NA"
What aspect of computing is being addressed by a notional machine?	Short answer	Copy-paste, short-description, "unclear", or "NA"
Does the article evaluate some aspect of a notional machine or its impact?	Select-multiple from: Quantitative Qualitative Not measured	
In scope?	Select-one from: Core paper for notional machines (include in second analyses) Adds breadth to notional machine discussion (include in second analyses) References notional machines (do not include in second analyses) Not at all related / exclude	Decision of whether to keep paper for further analysis
What is the contribution of this paper?	Short answer	Brief description
Additional Notes	Short answer	Frequently used to explain "in scope" decision
Additional Resources	File Chooser	

Figure 30: Questions on the literature review extraction form

Instructions to the interviewer:
Go through each numbered bullet in-order. Ask the main question for each numbered bullet. Only ask the lettered subquestions if the interviewee doesn't already answer them on their own. If the interviewee goes deeper at some point, that is fine. However, try to cover the points in this script if possible.

(1) Welcome!
(2) Before we begin, could you read the consent document and let me know if you have any questions. Do you consent to participate in the interview?

To go ahead and get us started, I have a series of questions I'm going to ask you. To make sure we address all of them, at times, I will return to them to ask the next question.

(3) What is your **background**?
(4) Let me restate the definition of "Notional Machine": *"A notional machine is a pedagogic device to assist the understanding of some aspect of programs or programming."*
(5) Can you tell me about a **Notional Machine** you use to help students understand program execution or program state?
If the interviewee seems confused about the definition of a Notional Machine or seems stuck, offer this additional prompt: You may want to consider diagrams, metaphors, phrases or demonstrations that you present to students to help them understand how a program works.
  (a) It must be something you've used with students.
  (b) It must focus on developing conceptual understanding about program execution and/or program state.

  [Note to interviewer: the interviewee might identify several notional machines with that answer. Identify only one of them for the following questions.]
(6) Do you have an **artifact/example/diagram** that describes this example?
  (a) Could you share it?
  (b) Could you draw something that illustrates this?
  (c) Could I see it?
(7) In what **context/course** and to what kind of **students** do you teach programming?
  (a) What **prerequisite knowledge** does it need?
(8) **How does this help** students understand the system/programming language/[..]?
(9) What works about this?
  (a) Which **concept** does this help you teach?
  (b) What **problem** does this help students resolve or avoid?
(10) **What does not work** about this?
  (a) What are the **limitations** of this notional machine?
  (b) Do you make **students aware** of these limitations?
  (c) Does this create **problems** later on in the [context / course]?
(11) **When/Where** do you typically use this idea?
  (a) How **frequently** do you use it?
  (b) How much **preparation time** do you need?
  (c) How much **time** do you spend **initially introducing** it?
(12) How did you first **come up** with this idea? / Can you recall an **inspiration** for coming up with this idea?
(13) Is it a **recurring theme** that you build upon in your [context / course]?
  (a) Do you **grow/extend** the notional machine throughout the course (e.g. introducing new features / increasingly powerful "language levels")?
(14) Do you have a **name** for your notional machine?
  (a) Do you communicate this name to your students?
  (b) Did you just come up with it?
(15) Now that we've gone through the interview, do you have **anything more** that you'd like to add?

  [As needed, interviewer may ask about the … how the notional machine is used]
  (a) **students**
  (b) **context**
  (c) **value**

  [As appropriate, the interviewer may seek additional permission to share interview materials and recording with colleagues]
(16) Thank you!