

Kent Academic Repository

Full text document (pdf)

Citation for published version

Dermot Shinnery-Kennedy (2012) Threshold concepts and teaching programming. Doctor of Philosophy (PhD) thesis, University of Kent.

DOI

uk.bl.ethos.652021

Link to record in KAR

<https://kar.kent.ac.uk/86522/>

Document Version

UNSPECIFIED

Copyright & reuse

Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (eg Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research

The version in the Kent Academic Repository may differ from the final published version.

Users are advised to check <http://kar.kent.ac.uk> for the status of the paper. **Users should always cite the published version of record.**

Enquiries

For any further enquiries regarding the licence status of this document, please contact:

researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at <http://kar.kent.ac.uk/contact.html>

Threshold Concepts and Teaching Programming

A thesis submitted to
the University of Kent at Canterbury
in the subject of Computer Science
for the degree
of Doctor of Philosophy

By

Dermot Shinnars-Kennedy

December 2012



IMAGING SERVICES NORTH

Boston Spa, Wetherby

West Yorkshire, LS23 7BQ

www.bl.uk

BEST COPY AVAILABLE.

VARIABLE PRINT QUALITY

Abstract

This thesis argues that the urge to build and the adoption of a technocratic disposition have influenced and affected the pursuit and development of a deeper understanding of the discipline of computing and its pedagogy. It proposes the introduction to the discipline of the threshold concept construct to improve both the understanding and the pedagogy.

The research examines the *threshold concept* construct using the theory of concepts. The examination establishes the conceptual coherence of the features attributed to threshold concepts and formalises the basis for threshold concept scholarship. It also provides a refutation for critiques of threshold concepts.

The examination reveals the inextricable links between threshold concepts and pedagogic content knowledge. Both rely on the expertise of reflective pedagogues and are situated at the site of student learning difficulties and their encounters with troublesome knowledge. Both have deep understanding of discipline content knowledge at their centre. The two ideas are mutually supportive.

A framework for identifying threshold concepts has been developed. The framework uses an elicitation instrument grounded in pedagogic content knowledge and an autoethnographic approach. The framework is used to identify *state* as a threshold concept in computing.

The significant results of the research are two-fold. First, the identification of *state* as a threshold concept provides an insight into the disparate difficulties that have been persistently reported in the computer science education literature as stumbling blocks for novice programmers and enhances and develops the move towards discipline understanding and teaching for understanding. Second, the embryonic research area of threshold concept scholarship has been provided with a theoretical framework that can act as an organising principle to explicate existing research and provide a coherent focus for further research.

Contents

Abstract	ii
Contents	iii
Overview	1
Hypotheses, Questions and Originality	3
Thesis Structure	5
Chapter 1 Introduction	7
1.1 Sources of the Technocratic Influence	8
1.2 Influence of the Technocratic Perspective	10
1.2.1 Professional Literature	10
1.2.2 Pedagogic Literature	12
1.2.3 Practitioners and Researchers	16
1.3 Threshold Concepts	18
1.3.1 What is a Threshold Concept?	18
1.3.2 Threshold Concepts and Pedagogues	22
1.3.3 Threshold Concepts and Pedagogic Content Knowledge	24
1.3.4 Threshold Concepts Scholarship	25
1.4 Summary	26
1.4.1 Thesis Structure	27
Chapter 2 On Concepts	28
2.1 Enumerative Reasoning and its Limitations	28
2.2 Exploiting Redundancy and Structure	30

2.2.1 Structure: Discovered and Imposed.....	34
2.3 Categorization.....	36
2.4 Purposeful Categorization.....	38
2.5 Concepts.....	42
2.6 Theories of Concepts.....	43
2.6.1 Classical View.....	44
2.6.1.2 Dominant Paradigm.....	45
2.6.1.3 The Illusion of Simplicity.....	46
2.6.1.4 Dominance Wanes.....	48
2.6.2 Family Resemblance View.....	49
2.6.2.1 Vagueness.....	49
2.6.2.2 Typicality.....	50
2.6.2.3 Basic Level.....	52
2.6.2.4 Difficulties Arise.....	52
2.6.3 The Theory Theory.....	53
2.6.3.1 Exposition of the Theory Theory.....	55
2.6.3.2 Status of the Theory Theory.....	56
2.7 Concepts, Threshold Concepts and Programming.....	57
2.8 Summary.....	59
Chapter 3 Threshold Concepts as Concepts.....	60
3.1 The Threshold Concept Metaphor.....	60
3.1.1 Threshold Concept Feature Types.....	62
3.1.2 Relationship Between the Features.....	63
3.2 Integration.....	66
3.2.1 Grand Linkages.....	67
3.2.2 The Scholarship of Integration.....	69
3.2.3 Integration and Liminality.....	71
3.2.4 Integration - A Summary.....	72
3.3 Troublesome Knowledge.....	73

3.3.1	Ritual Knowledge	74
3.3.1.1	Control Variables in Counting Loops	74
3.3.1.2	Java Method Header for <i>main</i>	75
3.3.2	Inert Knowledge	76
3.3.2.1	Pointers and References	76
3.3.3	Counter-intuitive Knowledge	78
3.3.3.1	Sequential Search	78
3.3.4	Foreign or Alien Knowledge	79
3.3.4.1	Number Systems and Representation	79
3.3.5	Conceptually Difficult Knowledge	80
3.3.6	Tacit Knowledge	81
3.3.6.1	Face Recognition	81
3.3.6.2	Specifying Algorithms	82
3.3.7	Troublesome Language	83
3.3.7.1	Parameter Passing	83
3.3.8	Troublesome Knowledge - A Summary	84
3.4	Transformation	84
3.4.1	Transformation - A Summary	87
3.5	Irreversibility	88
3.5.1	Irreversibility - A Summary	88
3.6	Threshold Concept Features - A Summary	88
3.7	Critiques of Threshold Concepts	89
3.7.1	Dependence on the Classical View of Concepts	89
3.7.2	Typicality	91
3.7.3	Threshold Concepts and the Theory Theory of Concepts	92
3.8	Summary	93
Chapter 4	Identifying Threshold Concepts	94
4.1	Approaches to the Accumulation of Evidence	95
4.1.1	Fundamental, Central and Difficult Concepts	95
4.1.2	The Threshold Concepts Group	97

4.2 Observations Regarding the Approaches Used	102
4.3 Critical Incident Interviews.....	103
4.3.1 Participant Profile and Interview Protocol.....	104
4.3.2 Interview Transcript Analysis	106
4.4 Retrospection and Hindsight	108
4.4.1 Hindsight Bias and The Curse of Knowledge.....	108
4.4.2 The Illusion of Memory	110
4.4.3 Expert Blind Spot.....	113
4.4.4 The Influence of Language.....	114
4.4.5 Emotion	115
4.4.6 Retrospection and Hindsight: A Summary	115
4.5 A Better Source of Identification.....	116
4.5.1 Teachers' Expert Knowledge	117
4.6 Pedagogic Content Knowledge	119
4.6.1 The Pedagogic Content Knowledge Model.....	120
4.6.2 Documenting Pedagogic Content Knowledge using a CoRe.....	121
4.7 Identifying Threshold Concepts.....	124
Chapter 5 State as a Threshold Concept.....	125
5.1 The Lure of the Big Concepts.....	125
5.2 Proposing State as a Threshold Concept.....	126
5.3 A Content Representation (CoRe) for State.....	127
5.3.1 Content.....	127
5.3.1.1 Whose Content?	128
5.3.1.2 An Autoethnographic Approach.....	129
5.3.1.3 Content Exposition	132
5.3.2 Representation.....	133
5.4 Notional Machine	137
5.4.1 Notional Machines Can Help Learners.....	138

5.4.2	Atomistic Thinking	139
5.4.3	The Turing Machine as a Notional Machine	140
5.4.4	Difficulties with Inspecting and Altering State	141
5.4.5	State as Troublesome Knowledge	143
5.5	State Space.....	144
5.5.1	Vastness of State Space	144
5.5.2	Useful Structure and Abstraction	145
5.5.3	Abstraction as Troublesome Knowledge.....	147
5.6	Variables and their Roles.....	148
5.6.1	Variables.....	148
5.6.2	Novice Programmer Difficulties with Variables.....	149
5.6.2.1	Problems with Assignment.....	150
5.6.2.2	Initialising and Using Variables	152
5.6.2.3	Using Variables in Plans	154
5.6.2.3	The Role Of Variables	156
5.6.3	Variables and Troublesome Knowledge.....	157
5.7	Methods and Parameter Passing	157
5.7.1	The Linguistics of Parameter Passing.....	158
5.7.2	The Linguistics of Parameter Receiving	160
5.7.3	Trouble with Parameter Passing.....	161
5.7.4	State and the Pedagogy of Parameter Receiving.....	162
5.8	Debugging	164
5.8.1	Close Tracking.....	164
5.8.2	A Good Understanding of State Makes Good Programmers.....	166
5.8.3	A Poor Understanding of State Makes Bad Programmers.....	167
5.9	Program Decomposition and Design.....	170
5.9.1	Control Abstraction	171
5.9.1.1	State and Control Abstraction.....	171
5.9.2	Procedural and Data Abstraction	172
5.9.2.1	State and Procedural and Data Abstraction.....	173

5.9.3	Trouble with Decomposition.....	174
5.10	Additional "Big-Ideas"	174
5.11	Summary	175
Chapter 6	Conclusions and Future Work.....	177
6.1	Contribution to Computer Science Education Research.....	178
6.1.1	First Threshold Concept Identified in Computer Science.....	179
6.1.2	Providing Insight into a Fifty Year Old Problem.....	179
6.1.3	Moving to Teaching for Understanding.....	179
6.2	Contribution to Threshold Concept Research.....	180
6.2.1	Embodying Threshold Concepts in the Theory of Concepts.....	180
6.2.2	Addressing the Criticisms of Threshold Concept Features	180
6.2.3	Clarifying the Nature of Troublesome Knowledge.....	180
6.2.4	Establishing the Primacy of Integration.....	181
6.2.5	Clarifying the Nature of Transformation	181
6.2.6	Innovative Research Approach	181
6.2.7	'Small' Concepts Are The Most Likely Candidates.....	182
6.3	Future Work.....	182
6.3.1	Are There More Threshold Concepts in Computer Science?	183
6.3.2	How Can Threshold Concepts Influence Our Teaching?	183
6.3.3	Identifying Pedagogic Knowledge Gaps.....	183
Appendix A	State - Sample Teaching Materials	184
A-1	Introduction.....	184
A-2	A Notional Machine.....	185
A-2.1	The Maze Machine	185
A-2.2	Light Bot Machine.....	191
A-2.3	Turing Machine	196
A-3	Introducing Inspections.....	198
A-4	Methods.....	201

A-5 Using Variables for Additional State Information	203
A-5.1 Initialisation	205
A-6 Summary.....	207
References.....	211

Dedication

To my parents **Phyllis (1930-1997)** and **Thomas (1930-2012) Kennedy**, who provided me with an excellent education, a small portion of which was supported by the formal education system.

You never failed to do your best
Your hearts were true and tender
You worked and toiled for those you loved
And those you loved remember
(Anon)

Acknowledgements

I will be the sole beneficiary of any credit that may accrue from this work but like all works of this type the author is just a front for a group of people who made it possible.

I sincerely thank

My family - Brenda, Eamonn, Caoilinn, Cianna and Mia - who continuously give me effective and affective support and encouragement for all my adventures and misadventures. You learned very quickly to avoid mentioning "the PhD" (affectionately code-named "that thing" by Mia) but you graciously accepted your boredom punishment on the occasions when you inadvertently did mention it. I love you.

Prof. Sally Fincher, supervisor and friend, whose wisdom and insightful observations never failed to keep me looking in the right direction even when I insisted on looking in any and every other direction.

Mr. Gearóid O Néill, my first computing tutor and very close friend and colleague who set my "programmer bit" and introduced me to a lifetime of enjoyment as a programmer.

Ms. Annette McElligott, friend and colleague, who made completion of this work possible when it shouldn't have been.

Overview

Computer programming is difficult to teach and learn. That seems ironic given that a large number of programming tasks involve the utilisation of everyday techniques and activities. For example, list manipulation operations such as searching, insertion and deletion are frequently programmed erroneously, even by experienced programmers, despite the fact that the casual utilisation of such operations is a routine part of a child's daily life. Such anomalies may explain why Shackelford and Badre were prompted to ask "Why can't smart students solve simple programming problems?" (Shackelford and Badre 1993). That question continues to confound computer science educators and researchers.

Difficulties with the acquisition of computing skills and abilities are not unique to novice programmers. Carroll and Rossen discuss "two empirical phenomena of computer use: (1) people have considerable trouble learning to use computers, and (2) their skill tends to asymptote at relative mediocrity" (Carroll and Rosson 1987). Novice programmer performance as assessed by educators, for example (McCracken, Almstrum et al. 2001), would certainly seem to be consistent with this evaluation.

The community of computer science education researchers has built an extensive literature documenting its theories and explanations, experiments and evaluations, interventions and strategies, tools and supports, all aimed at meeting the intellectual challenge that faces novice programmers. Several compilations and comprehensive reviews ((Mayer 1981); (Sheil 1981); (Soloway and Spohrer 1989); (Clancy, Stasko et al. 2001); (Robins, Rountree et al. 2003); (Clancy 2004)) trace and identify the various categories into which the accumulated work might be segmented. It is a matter of taste and preferred perspective as to which is the most appropriate taxonomy because, with some justification, Sheil has observed that with respect to

the study of programming “the claims that are made are all basically psychological; that is, that programming done in such and such a manner will be easier, faster, less prone to error, or whatever” (Sheil 1981).

Throughout the accumulated literature one pivotal, recurring theme has endured – knowledge integration. In one of the earliest contributions Mayer advises that “an important issue concerns how to foster meaningful learning of computer concepts by novices. Meaningful learning is viewed as a process in which the learner connects new material with knowledge that already exists in memory” (Mayer 1981). Almost a quarter of a century later Mike Clancy describes how students are “trying to build understanding of a concept...not just from instruction but from experimentation, analogies and other concepts, intuition and other knowledge” (Clancy 2004). He concludes “Particularly important are activities that aid the process of integrating and linking knowledge.”

As Ausubel explains linking is a very individual, personal activity that produces results unique to the learner

“whether learning material is potentially meaningful is a function of the learner’s cognitive structure rather than the learning material. The acquisition of meanings as a natural phenomenon occurs in particular human beings – not in mankind generally. Hence, for meaningful learning to occur in fact, it is not sufficient that the new material be non-arbitrarily and substantively relatable to corresponding relevant ideas...it is also necessary that such relevant ideational content be available in the cognitive structure of the particular learner” (Ausubel 1968).

If it is the case that novice programmers have the "relevant ideational content" available to them through their everyday experiences of lists, web surfing, mobile phones and so on, why are they unable to make the linkages and integrate the new and existing knowledge successfully?

Part of the answer may come from a proposal by David Perkins that much of our existing knowledge is "troublesome" because we find it difficult to use that knowledge when it would be beneficial. Perkins' list of the categories of knowledge that can be troublesome includes inert, tacit, ritual, counter-intuitive, linguistically nuanced, foreign (or alien) and conceptually difficult knowledge (Perkins 1999). If comprehension of material introduced in the programming context is dependent on knowledge falling into one of these categories it is very likely the learner will have difficulty acquiring it because they will be unable to deploy their existing knowledge and will fail to make the integrative connections required.

A second part of the answer may be associated with an idea developed by Erik Meyer and Ray Land. They have introduced the notion of a “threshold concept” to identify concepts that play an integrative role in the acquisition of a body of knowledge. Threshold concepts are distinguished from core or fundamental concepts because they are not merely concepts that must be acquired; they are constructors of the mechanisms that enable the knowledge to be unified. The acquisition of a threshold concept is said to have the potential to “transform” a student's learning and behaviour because it triggers a significant shift in the perception of a subject. The perceptive shift alters the way the student reasons about the subject and opens up a new and previously inaccessible way of thinking about it (Meyer and Land 2003).

Meyer and Land noted that their notion of threshold concept was linked to Perkins' idea of troublesome knowledge and that a threshold concept may lead to, or inherently represent, an instance of troublesome knowledge (Meyer and Land 2003).

Threshold concepts are distinguishable from other concept categories because they are integrative, transformative, irreversible and associated with troublesome knowledge. They represent an innovative pedagogic construct that provides a useful framework for identifying the knowledge that prevents novices from developing their expertise. They allow reflective pedagogues to refine their own understanding of content knowledge and its acquisition, to improve their own pedagogic content knowledge and to address the kind of complicated transitions learners undergo. Threshold concepts are both interesting and important because they are located at the site of the learning difficulties experienced by students and their investigation draws extensively on the pedagogues expertise in their own discipline.

Hypotheses, Questions and Originality

The central hypothesis of this research is that there are threshold concepts (i.e. concepts that integrate knowledge and provide learners with a transformed view of the material) and that there are programming concepts that fall into that category. The central question that inevitably follows is which programming concepts are threshold concepts? An important ancillary question relates to how we can identify a concept as being in the threshold category.

This research examines the role threshold concepts can play in our comprehension of the difficulties experienced by novice programmers. It seeks to make a contribution in two areas.

The first is in the development of our understanding of the nature and source of the difficulties that overwhelm novice programmers. The second is in the development of a theoretical and methodological framework for threshold concept research.

Although the idea of threshold concepts has been enthusiastically embraced by a large number of researchers in a wide variety of disciplines the idea has not received unanimous acceptance. By situating the theory of threshold concepts in a broader theory of concepts this research addresses the reservations articulated by threshold concept sceptics and proposes integration, and not transformation, as the central property of concepts categorised as threshold.

In this work the absence of an established mechanism for identifying threshold concepts imposed an operational requirement to propose a methodology for doing so. The proposed methodology draws on the pedagogic content knowledge of an experienced pedagogue using a form that situates it alongside the computer science education literature to identify potential threshold concepts. Threshold concepts and pedagogic content knowledge are interesting and important ideas with regard to disciplinary knowing and pedagogic skill because they are both located at the site of the learning difficulties experienced by students. The methodology, site of investigation and application of pedagogic content knowledge are important contributions to the ongoing development of threshold concept research and significant evidence of the originality of this work.

The examination of novice programmer difficulties is grounded in the hypothesis that most of the conceptual difficulties evident in introductory programming courses are representative of the types of troublesome knowledge described by Perkins. This immediately establishes a connection to threshold concepts. Using the concept of state as an exemplar the hypothesis has been tested using the methodology developed for identifying threshold concepts. One significant contribution this research hopes to make is to show that investigating problems like this does not necessitate building a software or hardware tool but instead requires us to comprehend the problem and its source. By approaching the problem from an epistemic rather than technocratic perspective we can deepen our understanding of not just of the pedagogic issues but also of the discipline more generally.

Thesis Structure

Chapter 1 sets out the background to the widespread difficulties reported for novice programmers and the mainly technocratic response by computer science educators. It introduces threshold concepts as an epistemic response that provides a mechanism for comprehending and addressing the problems documented in the computer science literature. The origins and properties of threshold concepts are outlined in preparation for a fuller exploration in later chapters.

Chapter 2 reviews the literature associated with the theory of concepts and traces the evolution of the theory from its Aristotelian origins to its current status, typically referred to as the theory theory of concepts. The fundamental roles that purpose, discovery, and invention play in categorisation are used to explicate common assumptions about the specificity of concepts; to clarify why vagueness does not undermine conceptual coherence; and to recognise the role inference and prior knowledge have on conceptual structure. The chapter finishes with a section that highlights the inextricable links between concepts (i.e. categorisation or classification) and the design and implementation of systems using object oriented programming languages.

Chapter 3 uses the concepts framework to examine the properties of a threshold concept. The centrality of integration is established and the other properties (i.e. transformation and irreversibility) are discussed in terms of their relationship to it. To prepare the reader for the examination of state as a threshold concept in Chapter 5, examples used to explore the different types of troublesome knowledge are drawn from programming scenarios. The elaborated basis for threshold concepts and their encapsulation in the general theory of concepts is used to address the critiques of threshold concepts presented in the literature. The dependence of the critiques on the Aristotelian view of concepts is used to expose their flaws and reaffirm the validity of the threshold concept approaches.

Chapter 4 reviews previous unsuccessful attempts at identifying threshold concepts in computer science and draws on the literature associated with retrospective recall to identify the inherent difficulties with the methods used. The methodological approach developed in this work is described. It is based on an autoethnographically oriented elicitation method that documents the researcher's pedagogic content knowledge relating to novice programmers'

acquisition of programming concepts. It uses a grid-style content representation structure (or CoRe) to record the data and augments it with evidence drawn from the computer science education literature. The CoRe is populated through an analytical process that obliges the researcher to reflect, question and justify their choice of "big ideas" and the pedagogic strategies that can be applied to realise the integrative and transformative outcomes of threshold concept acquisition. Thus the completed CoRe is a reasoned analysis and not an act of memory recall.

Chapter 5 proposes the concept of state as a threshold concept in computer science and uses the research instrument to set out the justification for that choice. The CoRe for state is presented at the beginning of the chapter but the substantive component of the chapter is devoted to the justification of the CoRe contents. Drawing heavily on the computer science education literature each chapter section documents and supports how state envelopes each individual concept and binds them into the coherent conceptual structure. The concepts documented are all related to the troublesome knowledge topics described earlier in Chapter 3. Chapter 6 documents conclusions and future work.

Appendix A provides supporting evidence adduced in classroom exercises to support some aspects of the pedagogic content knowledge set out in the CoRe grid included in Chapter 5.

Chapter 1

Introduction

In 1989 David Perkins and his colleagues at the Harvard Graduate School of Education evaluated the state of programming pedagogy and concluded it was "relatively underdeveloped." Their evaluation was moderated by an acknowledgement that programming was a "recent subject area." In those circumstances student achievement characterised by striking differences in competence and tutor statements to the effect that "Johnny can do anything, but Ralph just can't seem to get the hang of it" might be explainable, but it still begged the question "Why?" (Perkins, Hancock et al. 1989)

Almost twenty years later David Gries provided a sobering evaluation of the state of programming pedagogy in which he remarked

"For 50 years, we have been teaching programming.. And yet, teaching programming still seems to be a black art.... In some sense, we are still floundering, just as we were 50 years ago.... part of the problem may be that programming is indeed a difficult mixture of art and science—difficult to do and more difficult to teach. Yet another part of the problem may be that we have not discovered enough about programming and about teaching it. We need more research." (Gries 2008)

The difficulties associated with programming pedagogy are well-rehearsed in the computer science education literature. There is a substantial body of reports documenting the poor performance of first-year students with the multi-national, multi-institutional study by McCracken and others (McCracken, Almstrum et al. 2001) being viewed as a watershed. In

addition, attrition rates for computing-related courses are considered relatively high at around 30-50% (Denning and McGettrick 2005).

The discipline's youth may indeed help to explain why our understanding of the nuances of teaching programming may be deficient. For example, Almstrum *et al* described computer science education as "a young discipline still in search of its research framework" (Almstrum , Ginat et al. 2002, p.193). More recently, Fincher and Petre have described computer science education research as "an emergent area and still giving rise to a literature" which "does not represent a completely understood territory" and which we are "struggling to find the shape and culture of" (Fincher and Petre 2004).

Of greater influence has been the fact that throughout the last half century computing has evolved and developed largely on the basis of a technocratic perspective.

The perception that the *technocratic paradigm* "has come to dominate much of the discipline" (Eden 2007, p.135) has many followers. For example, Fred Brooks views computer science as being concerned with "making things, be they computers, algorithms, or software systems" and he categorises computer scientists as "toolsmiths" (Brooks 1996, p.62). For Loui "computer science is concerned with producing useful things such as microprocessors, software and databases" (Loui 1995, p.31). The consequences have been enormous for the discipline and its pedagogy.

How we perceive things affects how we act and learn (Brown, Collins et al. 1989, p.36). This is the basis of all philosophical discourse from epistemological, ontological and methodological perspectives (Eden 2007) and is not unique to the discipline of computing. Indeed, Stephen Covey, has suggested that how people see a problem *is* the problem (Covey 1989, p.40 emphasis in original).

1.1 Sources of the Technocratic Influence

The technocratic perspective is described by Edsgar Dijkstra as "more machine-oriented" and interested in specific devices and products, as well as being "more closely linked to application areas" and the development or production of tools (Dijkstra 1999). He challenged the perception of computing as a technocratic endeavour in his ACM Turing Award Lecture

"Automatic computers have now been with us for a quarter of a century. They have had a great impact on our society in their capacity of tools, but in that capacity their influence will be but a ripple on the surface of our culture compared with the much more profound influence they will have in their capacity of intellectual challenge which will be without precedent in the cultural history of mankind" (Dijkstra 1972, p.866).

The intellectual challenge had to be mastered and comprehended first and the understanding could then be used to underpin the intelligent creation of the tools required.

Nievergelt provided support for Dijkstra when he asked "How would computer science have developed if it had been influenced less by urgent practical demands and more by intrinsic intellectual interest" (Nievergelt 1980, p.928)

Sometime later Holloway noted that "The field of software engineering is in dire need of some philosophers and theologians...a lack of sufficient understanding of epistemology plagues the field" (Holloway 1995, p.20).

Comparing computing to the physical, chemical and biological sciences Milner noted "these sciences are driven by the urge to understand, and ours by the urge to build" (Milner 2006, p.385).

Carroll and Rosson identified a *production bias* which goes some way to explaining the technocratic disposition to build things. The production bias is characterised by a focus on throughput. Faced with a situation that could be more effectively handled by new procedures based on a better understanding we are unwilling to invest the time required to learn the new procedures and develop the better understanding. Instead we stick with what we know because we can maintain the throughput, even if the new understanding would ultimately be more efficient and provide long term benefits. The production bias gives rise to a *motivational paradox* that inhibits learning and promotes continued application of known strategies (Carroll and Rosson 1987).

The motivational paradox is consistent with the view articulated by Milner. Like Dijkstra, Milner was critical of the computing community for not investing enough effort in "understanding" the discipline. His justification identified the over-concern "with managing software production, at the expense of an intimate understanding of software itself" (Milner 2006, p.384). He argued that the extraordinary pace of technological advance had lured us to opportunistic design practices because theories take too long to gestate. This led to what Robinson described as a

“the triumph of speed over thoughtfulness, of the maverick shortcut over discipline, and the focus on the short term.” (G. Robinson cited in Milner 2006). Milner cited the Y2K problem noting “the appropriate theories were quite modest and already two decades old” but had not sufficiently informed software design and when retrofitted revealed weaknesses too late to mend.

Ultimately production, throughput and experience can result in, or lead to, understanding but the path is rarely well-organised and structured. Applying an appropriate theoretical framework can help order “experiential lessons into regular, memorable and understandable patterns” which assists in their explanation and comprehension (Padfield 1997, p91). However, building the theoretical framework first can be far more beneficial because it promotes understanding from the beginning and shows how the theory transcends particular technologies (Denning 2003, p.15).

1.2 Influence of the Technocratic Perspective

This technocratic perspective of the discipline is reflected in its published literature, both professional and pedagogic, and several reviews and analyses of published research confirm its dominance.

1.2.1 Professional Literature

Newman analysed the published proceedings of five sets of the Computer Human Interaction (CHI) and International Computer Human Interaction (InterCHI) conferences held between 1989 and 1993 (Newman 1994). His analysis, motivated by his belief that leading Human Computer Interaction (HCI) researchers had confirmed “the engineering design context is especially vital to HCI research” captures the essence of the technocratic perspective. He categorised approximately 70% of the papers as “what technological communities usually do.”

The remaining 30% presumably gave their allegiance to those things that, in Newman's view, technological communities usually do not do; these were congregated in a single category Newman labelled “Radical Solutions.” Of the members of this category Newman remarks that they “stand out as deliberate attempts to introduce new paradigms” and are characterised “by a need to reorganize the receiving environment.” Many of them, he concluded, “are explicitly

described...as new conceptual frameworks for design” which “bring about change, along many dimensions, often in unpredictable ways.”

Categorising papers that question and explore our understanding of a disciplinary topic as a “radical departure” highlights an ontological perspective that values building over understanding.

In a second published meta-analysis Mary Shaw reviewed the software engineering literature (Shaw 2003). Her analysis yielded a set of categorisations containing five types of research questions that would be posed by software engineering researchers. They were

- *Method or means of development* (e.g. How can we do/create/modify/evolve or automate doing X?)
- *Method for analysis or evaluation* (e.g. How can I evaluate the quality/correctness of X?)
- *Design, evaluation, or analysis of a particular instance* (e.g. What is a (better) design, implementation, maintenance, or adaptation for application X?)
- *Generalization or characterization* (e.g. What, exactly, do we mean by X? What are its important characteristics?)
- *Feasibility study or exploration* (e.g. Does X even exist, and if so what is it like?)

Shaw concluded that “Generally speaking, software engineering researchers seek better ways to develop and evaluate software” (Shaw 2003, p.727). Of her categorisations the first three are focused on operational issues associated with making things, in particular the “methods” and “practices” that facilitate construction. The two remaining headings deal with questions that lean towards understanding and embrace “characterization” and “exploration.” It is, of course, impossible to partition five items into two groups without creating what appears to be a 3:2 bias in favour of one group. In that context the presumed bias in favour of making things might be considered superficial and without implication. However, what we might perceive to be a superficial side-effect masks what is, in fact, a deeply dichotomous relationship. The five categories are not equally weighted and the first three attract significantly more weight than the last two. This is evident in the second part of Shaw's analysis.

As part of her review Shaw included an analysis of the submissions to the International Conference on Software Engineering (ICSE) for the year 2002. Using her categories Shaw tabulated the papers submitted and accepted and this resulted in the following distribution

ICSE 2002 Analysis

Type of question	Submitted	Accepted
Method or means of development	142(48%)	18 (42%)
Method for analysis or evaluation	95 (32%)	19 (44%)
Design, evaluation, or analysis of a particular instance	43 (14%)	5 (12%)
Generalization or characterization	18 (6%)	1 (2%)
Feasibility study or exploration	0 (0%)	0 (0 %)
TOTAL	298(100%)	43 (100%)

Reaffirming the toolsmith principle Shaw notes “The most common kind of ICSE paper reports an improved method or means of developing software – that is, of designing, implementing, evolving, maintaining, or otherwise operating on the software system itself” (Shaw 2003, p.727). Significantly only 6% of the submitted papers fell into the “Generalization or characterization” category and of the eighteen, only one was accepted. Not a single paper submitted could be categorised as ‘exploratory’. Thus, only one paper accepted for the conference *may* have addressed issues related to what might be broadly described as the nature or philosophy of the discipline in general, and software engineering in particular. With a 98% to 2% differential Shaw could not be accused of understating things when she observed that the “*tangible* contributions of software engineering research may be procedures or techniques for development or analysis; they may be models that generalize from specific examples, or they may be specific tools, solutions, or results about particular systems” (Shaw 2003, p.728 emphasis added).

1.2.2 Pedagogic Literature

Valentine undertook a similar, albeit more longitudinal, survey of the Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium Proceedings “to discover the current state of, and the longitudinal development of CS Educational Research” (Valentine

2004, p.255). His survey covers the two decades from 1984 to 2004 and focuses on first year topics (i.e. topics that would be taught to first years or novices) which accounted for approximately one-quarter of the papers published. Like both Newman and Shaw, Valentine developed a classification system which he used to analyse the surveyed papers. His analysis is tabulated in the table below.

Consistent with both Newman and Shaw, Valentine's findings are dominated by the technocratic perspective. Just 10% of the papers dealt with issues relating to the nature of computer science education. Whilst a bias in favour of "experience", "practice", "methods", "heuristics", "evaluation", "analysis" and "solutions" might be understandable, even justifiable, in application environments such as software engineering and HCI, the poor showing for 'philosophy' or 'theory' oriented papers is cause for concern in an educational context. In *The Reflective Practitioner* (Schön 1983) explains how reflecting on our actions is a crucial part of teaching and education. Devising and sharing schemes for presenting difficult concepts or building tools to assist students with comprehension are laudable activities but it is also important to stand back and look and ask what is happening and where it is leading. If we do not or cannot do that then we have no compass to help guide us to where we want to go.

Our belief system or theory of the world, or more formally our ontological perspective, guides and influences the things we do, the practices we have for doing them and the types of conclusions we reach. Proponents of an object-oriented approach to programming present material in a particular way and emphasise things in a style that is markedly different from proponents of other programming paradigms. Peter Denning has reviewed the "great principles of computing" and concluded that because the field of computing has been equated with programming many of the principles have "been interred beneath layers of technology in our understanding and our teaching." He believes the time has come to "set them free" (Denning 2003). From a teaching perspective Prosser and Trigwell explain how "university teachers who focus on their students and their students' learning tend to have students who focus on meaning and understanding in their studies, whilst university teachers who focus on themselves and what they are doing tend to have students who focus on reproduction" (Prosser and Trigwell 1999, p.142). The discipline's ontological perspective has been unequivocally characterised by a pervasive compulsion to build. The implications of this are visible in the types of activities that are valued in our practice and our pedagogy.

Category	Description	Total Papers	1984-2003	1984-1993	1994-2003
Experimental	Assessing something with some scientific analysis.	94	21.2%	19.6%	22.1%
Marco Polo	A description of trying something new.	117	26.4%	30.4%	23.9%
Philosophy	An attempt to generate debate of an issue, on philosophical grounds, among the broader community.	45	10.1%	11.9%	9.1%
Tools	Developed software for a course related purpose.	99	22.3%	18.5%	24.6%
Nifty ("most whimsical")	"Nifty assignments, projects, puzzles, games and paradigms."	78	17.6%	17.3%	17.8%
John Henry	Description of an outrageously silly intervention or course.	11	2.5%	2.4%	2.5%
TOTAL		444			

Valentine noted that the tools category was the fastest growing segment. When the twenty-year figures are partitioned by decade and we see the decade on decade changes it becomes apparent that the philosophy category has dropped by as much as a quarter. It would appear that even in computer science education we are unable to resist the temptation to build things.

Staying within the domain of pedagogy, a related pair of reviews analysed the research practices reported in two collections of papers published between 2000 and 2005. In the first study, by (Randolph, Bednarik et al. 2005), the collection consisted of 59 papers published between 2001 and 2004 in the proceedings of the Koli Calling¹ conference which features the

¹ The full title of the conference is Koli Calling: Finnish/Baltic Seas Conference on Computer Science Education.

work of computer science education researchers in the Nordic/Baltic countries, especially Finland. Only two of the papers came from outside of Europe and about 90% of the papers came from Finland. The analysis showed that most of the papers described an artefact or process, such as a "program, project, or intervention" created or developed by the author(s). Only one paper was categorised as "Theoretical, methodological or philosophical" (Randolph, Bednarik et al. 2005, p.107).

For the second review a stratified sample of 352 papers was selected from 1,306 computer science education papers that were published between 2000 and 2005 (Randolph 2007). Randolph sought to improve on the "breath, depth and reliability" of previous reviews and drew his sample from a wider range of sources. The 352 papers in his sample were taken from six computer science education publications, some of the oldest and most established in the field, including SIGCSE Bulletin, Computer Science Education, Journal of Computer Science Education Online, SIGCSE Technical Symposium Proceedings, Innovation and Technology in Computer Science Education (ITiCSE) Proceedings, Koli Conference Proceedings, ACE Conference Proceedings and The International Computing Education Research (ICER) Workshop Proceedings.

The analysis showed that groups in different regions around the world practice their research in different ways and there are distinct *islands of practice*. For example, papers originating in North America tend to use experimental or quasi-experimental approaches whilst papers originating in Europe or the Middle East do not. There was a higher proportion of North American papers using only anecdotal evidence, although this had decreased each year. A paper originating in the Middle East was more likely to use a qualitative approach whereas one originating in the Asian Pacific or Eurasia tended to use "stakeholder attitudes" as the "sole dependent measure" (Randolph 2007, p.94).

For the purposes of comparison Randolph tabulated the 352 papers using the categorisations developed by Valentine. The results, which are tabulated below, show some movement within the categories but the established and persistent trend visible in the other reviews is also evident here. Most of the papers describe physical or conceptual artefacts 'built' or 'made' by the author(s) and a small proportion, about one tenth, addressed more philosophical or theoretically oriented issues and topics.

Valentine's category	N	%
Experimental	144	40.9
Marco Polo	118	33.5
Tools	44	12.5
Philosophy	39	11.1
Nifty	7	2.0
John Henry	0	0.0
Total	352	100

Source : Table 24 from (Randolph, 2007, p76)

Randolph noted that "Both Valentine (2004) and Randolph, Bednarik and Myller (2005) converged on the finding that few computer science education research articles went beyond describing program activities" (Randolph 2007, p.11).

1.2.3 Practitioners and Researchers

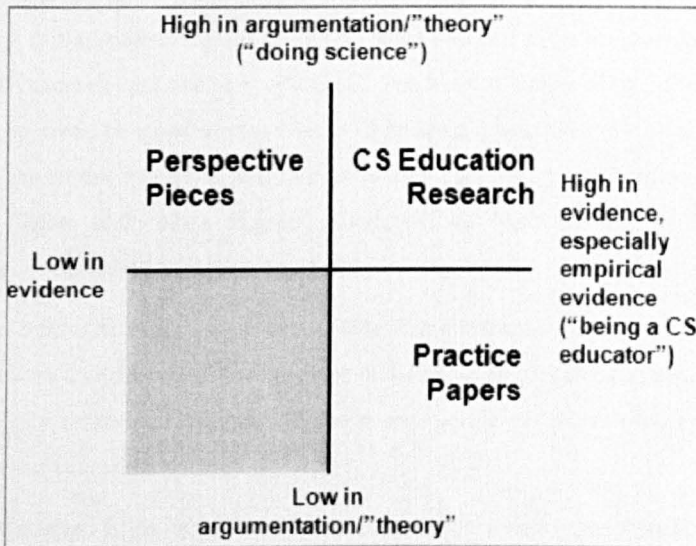
The literature of computing is dominated with publications allied to the technocratic paradigm and with building things.

Fincher and Petre have used a powerful diagrammatic representation to capture the recurring features highlighted by the literature reviews outlined above and specifically the computer science education literature. They note that publications have two essential components. The first they describe as the "rationale, argumentation or theory" and the second as "evidence - most often empirical" (Fincher and Petre 2004, p.2). They represented these components using the Cartesian plane segmented by mutually perpendicular axes and scaled the axes using the simple boundary conditions of high and low. The axes cut the plot space into four quadrants. The diagram is reproduced below. Describing it the authors write

On the top left, we have papers that have lots of argument, but little empirical evidence (although they may draw on other types of evidential material, similar to other disciplinary areas such as history). The bottom left quadrant *should* be empty; this is the home of papers with no evidence and no argument. The bottom right quadrant represent papers which are constructed around evidence - most often empirical - but are not strong on argumentation. Here is where descriptive, practice-based, "experience" papers are found, probably the most common type of paper in the area today. Finally, the top right quadrant represents papers that contain both evidence and arguments. This is where, we contend, most CS education research papers *should* be found. (Fincher and Petre 2004, p.2).

Fincher and Petre succinctly summarise the evidence adduced by the literature reviews when they write "The concentration of papers in the lower, right-hand quadrant is representative of the state and status of CS education research today" (Fincher and Petre 2004, p.2).

They are careful not to indicate a preference for one type of paper over another. Their caution may be motivated by an awareness of Garforth's advice some forty years earlier when he warned that "The study of education is prone to two dangers - an excessive preoccupation with empirical investigation and the isolation of educational theory" (Garforth 1965, p.vii).



They do, however, distinguish between the communities for which different types of papers can have relevance or import.

Practitioners, they note, are focused on "their students, in their classroom" and the context is an institutional one which, through professional meetings and conferences, expands into the community of practice. *Researchers*, in contrast, are interested in "ideas...theories and their exponents" that they can draw on to frame, inform and support their evidence and arguments (Fincher and Petre 2004, p.80).

The contrast is encapsulated in their descriptors *being a CS educator*, which is associated with high evidence, and *doing science*, which is associated with high theory.

Thomas Kuhn documented the characteristics of scientific development (i.e. doing science) and noted that in periods of acknowledged crisis "scientists have turned to philosophical analysis as

a device for unlocking the riddles of their field" (Kuhn 1996). Theory based understanding allows us to grasp what is happening around us, to explain it and to develop and improve it.

1.3 Threshold Concepts

Into this milieu the work described here introduces a theoretical and philosophically grounded analysis. The analysis maps out what is happening around those of us in the computer science education community who teach programming and are trying to unlock the riddle of why it is that Johnny appears to be able to do anything in his programming course and poor Ralph can't seem to get the hang of it at all. The analysis is developed from a researcher's perspective and draws on ideas and theories in computing, education and learning that provide evidence to support the argument. It is associated with high theory and resides in the top right quadrant of Fincher and Petre's diagram, categorised as doing science, where they argue computer science education research should be.

The analysis is a contribution to our understanding of the pedagogy of computer programming. It is a response to David Gries' call to discover more about the teaching of programming and Fincher and Petre's desire to improve the shape and culture of the territory associated with computer science education research.

Central to the analysis is the notion of a *threshold concept*, a novel idea introduced by Meyer and Land that suggests all disciplines have specific concepts which it is essential to master to develop expertise in the discipline (Meyer and Land 2003). Identifying the threshold concepts of computing, specifically programming, would enhance the discipline's understanding of the structure of programming knowledge as well as assisting pedagogues by providing a lens through which to examine the difficulties experienced by novice programmers attempting to acquire that knowledge.

1.3.1 What is a Threshold Concept?

The term *threshold concept* emerged from a national teaching and learning research project undertaken in the UK that sought to identify "factors leading to high quality learning environments." As part of his contribution to the project's strand on the discipline of economics, Meyer introduced the notion of a threshold concept, described as "akin to a portal, opening up a new and previously inaccessible way of thinking about something" (Meyer and

Land 2003, p.3). Development of Meyer's initial contribution led to work on threshold concepts becoming a project in its own right (Beaty 2006, p.xii).

The notion of a threshold concept resonated with the work of David Perkins, who was acting as a consultant on the teaching and learning project. For several years Perkins had been investigating the idea that knowledge should not be viewed as a bimodal resource that you either have or don't have, especially in the context of the difficulties facing novice programmers. He maintained that people don't just know or not know something. Instead they have what is described as *fragile knowledge*, knowledge that is "partial, hard to access and often misused" but which allows a person to sort of know, have a notion or some fragments "without being able to marshal enough knowledge with sufficient precision to carry a problem through to a clean solution" (Perkins and Martin 1986). As part of Perkins' on-going work fragile knowledge evolved into *troublesome knowledge* which he enumerated as a variety of problematic knowledge forms that includes *ritual knowledge*, *inert knowledge*, *counter-intuitive* or *alien knowledge*, *conceptually difficult* knowledge and *tacit* knowledge (Perkins 1999).

Perkins described how learners all over the world found these types of knowledge troublesome in systematic ways, particularly where "'emergent' effects occur as the collective consequence of many small interactions" (Perkins 2006, p.36). Perkins also noted that student encounters with troublesome knowledge often lead them to resorting to mimicry and developing "brittle understandings" because they only "learn enough about ideas, explanations, and alternative perspectives to pass the test without developing any real insider feel" (Perkins 2006, p.37). Marton and Säljö had previously described this dichotomy of student learning intentions as "surface" and "deep" learning (Marton and Säljö 1976).

In their seminal paper on threshold concepts Meyer and Land described threshold concepts and indicated the correspondences between the notion of a threshold concept and Perkins' idea of troublesome knowledge (Meyer and Land 2003). They enumerated five characteristics or features that could be used to distinguish threshold concepts from other types of concepts (e.g. core, fundamental) and listed them as

- transformative
- irreversible
- integrative
- troublesome
- bounded

In Chapter 3 we will consider these characteristics and threshold concepts in greater detail but a description is provided here to briefly elaborate the essence of each characteristic and offer simple illustrative examples. I have taken the liberty of substituting or supplementing the original examples provided by Meyer and Land with examples drawn from the discipline of computing.

The first feature is identified as the *transformative* effect that results from the acquisition of a threshold concept because "once understood, its potential effect on student learning and behaviour is to occasion a significant shift in the perception of a subject, or part thereof" (Meyer and Land 2003). The transformation is realised through a relocation of the learner's focus away from the surface knowledge associated with important but limited principles to the deeper knowledge required to sustain a conceptual model. For example, many novice programmers get bogged down with the detail of programming language syntax and the need for semi-colons, commas and brackets of various types (Jadud 2005). However, some make the connection to the pervasive use of dots, angled brackets, forward slashes, at (@) symbols and hashes they have encountered in email addresses, URLs, tweets, abbreviated codes for specific mobile phone operations, web-page tags and so on. More generally, they grasp the idea that the various delimiters they encounter in computing environments act as signposts that facilitate the correct interpretation of any operation they submit. Once acquired, the concept "delimiter" will facilitate learning of more advanced material, such as grammars, automatic translation and language processors.

The second feature (irreversibility) compounds the effect of the first (transformation) because "the change of perspective occasioned by acquisition of a threshold concept is unlikely to be forgotten, or will be unlearned only by considerable effort" (Meyer and Land 2003). Thus, transformation is (probably) *irreversible* because it is difficult to revert to a more primitive view once a sophisticated alternative has been acquired. As a consequence the knowledge acquired is permanent and unlikely to be forgotten. For example, many programmers equate recursion and iteration because they both support repetition (Kahney 1989). However, programmers who have mastered the concept of recursion are aware of the significant differences between repeating something in a recursive style and repeating something with a simple iteration construct. They still recognise that recursion can mimic a simple iteration construct but they now know the inverse is not true without the use of additional data structuring resources.

They could still use recursion for simple repetition but their transformed perspective discourages its deployment in such circumstances. More importantly, their newly acquired conceptual knowledge allows them to exploit the inherent additional functionality provided by the use of recursion for solving a variety of problems they previously solved more crudely and with greater effort.

The third effect of acquiring a threshold concept (integration) is that it "exposes the previously hidden interrelatedness of something" (Meyer and Land 2003). This ability to provide insight into previously unrecognised relationships between existing knowledge components makes the concept *integrative*. For example, the concept "class" integrates a wide variety of programming concepts including local and global data access, abstract data type behaviour and implementation, program design and code reuse. Understanding the class concept improves the learners conceptual binding of the individual knowledge components and makes them more coherent and robust. This strengthens the learners' knowledge base and acts as a tool to guide further transformations and development. The integrative effects of the class concept might further support the learners' realisation or deepened understanding of the concepts of encapsulation, component-based programming, inheritance and polymorphism.

Concepts that have integrative, transformative and irreversible effects are "potentially (though not necessarily) *troublesome*" (Meyer and Land 2003) in the sense that the trouble with acquiring them is associated with one or more of the troublesome knowledge categories identified by Perkins. Access to and utilisation of existing knowledge, that could trigger and sustain the effects attributed to a threshold concept, can be obscured by the inertness, ritual nature, inherent conceptual difficulty, alien traits or tacitness of the knowledge which makes deploying it troublesome. Many novice programmers have huge difficulties with array indexing mechanisms in which the counting starts at zero and as a consequence so called "off-by-one" errors are notoriously common. This occurs despite the fact that the novices are quite comfortable with the counting mechanisms used in buildings where the floors are numbered from zero and the floor immediately above the ground level is called the "first" floor. In lift systems the floor at ground level is invariably identified as floor zero. Their knowledge of the numbering mechanism used in lifts should be applicable in the array indexing scenario but it remains inert, difficult to access, and thus troublesome.

The scope of the effects of acquiring a threshold concept is "Possibly often (though not necessarily always) bounded in that any conceptual space will have terminal frontiers, bordering with thresholds into new conceptual areas" (Meyer and Land 2003) This characteristic acknowledges the reality that in any discipline the conceptual space is unlikely to be marked out by a single all-embracing threshold concept that engulfs all of the knowledge associated with the discipline. Where several threshold concepts are identified their individual boundaries may vary widely in extent but this does not necessarily imply that one (larger) threshold concept is superior or more valuable than another (smaller) one.

It is notable that four of the features are linked to the learner and their learning experience and one, the boundedness property, is uniquely concerned with the nature of conceptual frameworks. As a consequence, the bounded feature is often mentioned just for completeness and is frequently omitted altogether in cases where it is the consequences for learners that are the focus of attention.

1.3.2 Threshold Concepts and Pedagogues

The concept of a threshold concept enjoys a following among academics across a wide range of disciplines and is the subject of ongoing biennial conferences, websites and publicly funded projects (O'Donnell 2010, p.2). Material from the biennial conferences has been shaped into three volumes of submitted papers and invited contributions, which catalogue the miscellany of disciplines and the global reach of research activity associated with the idea ((Meyer and Land 2006); (Land, Meyer et al. 2008); (Meyer, Land et al. 2010)). A website² hosted by University College London is the most authoritative repository of references and sources on the topic.

Beatty has suggested that the threshold concept community has expanded and diversified because the idea "seems to fire the imagination of teachers and researchers" (Beatty 2006, p.xi). There is plenty of evidence to support this claim.

For example, introducing the threshold concept idea in a series of informal interviews with computer science educators, Boustedt and his colleagues report how the respondents found the idea "compelling: nearly everyone we spoke with was immediately interested" (Boustedt, Eckerdal et al. 2007, p.505). Osmond and her colleagues describe how they used threshold

² <http://www.ee.ucl.ac.uk/~mflanaga/thresholds.html> (last accessed 10 July 2013)

concepts "as a lens...to investigate perceptions of spatial understanding with both staff and students" (Osmond, Turner et al. 2008, p.244). They credit the strategy with enabling them to "open up a dialogue with the teaching staff in a discipline that appears, in the main, to be undertheorised. The usefulness of this dialogue was evidenced...by the enthusiasm of staff to participate" (ibid., p.256).

Entwistle has documented the effects of the threshold concept idea in its birth-place discipline of Economics, where he found that introducing the notion of threshold concepts to lecturers seemed to "open up their thinking about the nature of knowledge in Economics, and show how the subject can be represented to students in more interesting and effective ways" (Entwistle 2008, p.30). Perkins has highlighted what he considers the "entrancing" nature of threshold concepts which is derived from their ability to "offer something of a common language, provoke reflection on the structure of disciplinary knowledge, and inspire investigations of learners' typical hangups and ways to help" (Perkins 2010, p.xliii).

The widespread and discernible enthusiasm for threshold concepts is motivated and sustained by the fact that the idea is seen by tutors and subject specialists as "having immediate relevance to issues within their own practice" (Beaty 2006, p.xi). This allows them to "inform their pedagogy in ways that make sense within their own communities of practice, and for their own students" (ibid., p.xii).

Threshold concepts are close to the work of pedagogues and their students and are inextricably linked to the issues that make sense in their own disciplines. This is in contrast to the voluminous research literature on generic teaching and learning which Meyer has described a a challenge to "unpack and demystify" into a form that is "accessible to *all* university teachers" because there is a "big gap between reading about, being taught about, reflecting on, and discussing this knowledge" (Meyer 2010, p.196). Offering support for Meyer's view Cousin notes that the challenge he identifies is always an uphill struggle because "getting academics to underpin their reflections on their practice using educational theory" often reduces them to the status of "informed amateurs in another discipline" (Cousin 2010).

Writing about developments in higher education Lee Shulman noted "The field of teaching in higher education has been limited by the features of a generic or technical view of teaching" (Shulman 1999, p.x). He identified "the study of subject-matter and its interactions with pedagogy" as the missing paradigm in research on teaching (ibid., p.ix).

The accent on the relationship is important - the subject-matter and *its* interactions with pedagogy.

The threshold concepts idea has been enthusiastically embraced by pedagogues because it emphasises subject expertise (Cousin 2008, p.265). Indeed the significance of threshold concepts is that they attempt to locate troublesome aspects of disciplinary knowledge and not teaching knowledge (Land, Meyer et al. 2008, p.xi). This emphasis may be at variance with current education research discourse that is dominated by a student-centred focus (Prosser and Trigwell 1999). Cousin expresses her concern that the dominance of the student-centred approach may be causing the erasure of teacher expertise whereas the threshold concept approach requires an academic to go "more deeply into her own [discipline] for the purposes of formulating the best ways of teaching and learning it" (Cousin 2010).

1.3.3 Threshold Concepts and Pedagogic Content Knowledge

The benefits of placing the emphasis on the expertise of pedagogues is highlighted by Perkins when he notes "Seasoned teachers know what troubles are likely and draw on active, social, and creative learning to address them" (Perkins 2006, p.36). This is the type of expertise that Shulman was referring to when he coined the term *pedagogic content knowledge* (PCK) and which he described as a "veritable armamentarium" of "the most useful forms of representation... the most powerful analogies, illustrations, examples, explanations, and demonstrations - in a word, the ways of representing and formulating the subject that make it comprehensible to others" (Shulman 1986, p.9). Shulman argued that this type of knowledge is "uniquely the providence of teachers" and "most likely to distinguish the understanding of the content specialist from that of the pedagogue" (Shulman 1987, p.8). This type of knowledge is not representative of the generic or technical view of teaching and is firmly located in the discipline where it represents the "blending of content and pedagogy into an understanding of how particular topics, problems, or issues are organized, represented and adapted to diverse interests and abilities of learners, and presented for instruction" (Shulman 1987, p.8).

Gal-Ezer and Harel use a similar argument when they describe the difference between the backgrounds required to be an educator, practitioner or a researcher in the discipline of computing. All must have discipline expertise but the educator must assume the role of a "scientific intellectual" who is capable of conveying the discipline knowledge correctly and

reliably as well as infusing the students with "interest, curiosity, and enthusiasm" (Gal-Ezer and Harel 1998, p.77).

Threshold concepts and pedagogic content knowledge represent a synergy of ideas because they both have deep understanding of content knowledge at their centre. Both are student focused but not student centred. Both rely on the expertise of reflective pedagogues and are situated at what Cousin calls "the site of the subject" (Cousin 2010). They are distinguishable by the sharpness of the focus in their respective lenses. Pedagogic content knowledge has a wide view whereas threshold concepts zoom in on content knowledge that derails student learning. Awareness (albeit tacit) of threshold concepts and the difficulties they may cause is subsumed into pedagogic content knowledge. The two ideas are linked inextricably and mutually supportive.

1.3.4 Threshold Concepts Scholarship

The multi-disciplinary embrace of threshold concepts has produced an "established and expanding literature" (Meyer 2010, p.207) and the threshold concepts project has evolved from something that initially offered "a tentative conceptual framework and a lens through which to view the pedagogy of higher education anew" (Land, Meyer et al. 2010, p.x) into a "developing theoretical framework" (Meyer 2010) that "addresses the kind of complicated learner transitions learners undergo" (Land, Meyer et al. 2008, p.ix).

Schwartzman has observed that "Threshold concept scholarship is initially developing from a variety of sources and in a variety of what appear as not-necessarily connected directions; an eclectic mix of description, analogy and metaphor" (Schwartzman 2010, p.24). She compares the practice and research of the threshold concepts community with the type of activities typically pursued by a classic scientific community as described by (Kuhn 1996). She describes the threshold concept community as an "an assemblage of individuals" with a "concern for education and effective teaching" who "display receptivity to - even hunger for - any and all ideas about how to support student learning" (Schwartzman 2010, p.28).

Schwartzman has made a case for the development of a theoretical foundation for threshold concepts and concludes "An explanatory theory defined for threshold concepts is required for maturation of the field." The availability of such a theory would have value as a communal resource including making explicit any tacit presuppositions and placing them in their

meaningful context. She lists two roles such a theory could play. The first would be to illuminate the internal coherence of existing work and the second would be as an "organizing principle for future theoretical and application development" (Schwartzman 2010, p.25).

Contrary to Schwartzman's assertion threshold concept scholarship already has an explanatory theory that can illuminate the coherence of existing work and act as an organizing principle for future development. The *theory of concepts* dates back to the time of Aristotle and has an established, still evolving, literature that threshold concept scholarship implicitly draws on because concepts are at the heart of it.

No one in the field of threshold concepts scholarship has explicitly framed their contribution in a theory of concepts context. Every discipline has concepts. Whether an individual discipline, or every discipline, chooses to classify them as core, foundation, threshold or otherwise they are still subsumed by the theory of concepts. Thus, the theory of concepts is universally applicable across all disciplines.

This theoretical framework has been hidden in plain sight of the threshold concept community and appears to have assumed the status of inert or ritual troublesome knowledge for them. The theory of concepts is capable of addressing the perceived issues and problems Schwartzman believes are inhibiting maturation of threshold concepts' scholarship. It also provides the coherence required to articulate, explicate and support its continued development.

1.4 Summary

The urge to build and the adoption of a toolsmith disposition have influenced and affected the pursuit and development of a deeper understanding of the discipline of computing and its pedagogy. Developing such an understanding would provide a valuable asset with which to meet the challenges the discipline faces. Those challenges include the widespread and well-documented difficulties linked with learning to program.

Threshold concepts are an innovative pedagogic construct that provide a useful framework for identifying the knowledge that prevents novices from developing their expertise. They allow reflective, expert pedagogues to refine their understanding of content knowledge, and its acquisition, and to improve their pedagogic content knowledge.

1.4.1 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 reviews the concepts literature to facilitate situating threshold concepts in the theory of concepts framework. Chapter 3 uses the concepts framework to describe what a threshold concept is and addresses some of the issues that have arisen in the literature regarding the basis for threshold concepts. Chapter 4 reviews previous unsuccessful attempts at identifying threshold concepts in computer science and describes the research instrument, based on Shulman's pedagogic content knowledge, used in this work to identify threshold concepts in computing. Chapter 5 proposes "state" as a threshold concept in computer science and uses the research instrument to justify that choice. Chapter 6 documents conclusions and future work.

Chapter 2

On Concepts

Threshold concepts are a type of concept so we begin our consideration of them by examining the theory of concepts and setting out a foundation that will be used later to support our investigation of threshold concepts and computer programming.

2.1 Enumerative Reasoning and its Limitations

Our brains have vast but limited resources ((Medin 1989); (Dijkstra 1972); (Rosch 1978); (Komatsu 1992); (Jackendoff 1989)). Dijkstra emphasised this when he observed "The competent programmer is fully aware of the strictly limited size of his own skull" (Dijkstra 1972, p.864). This limited capacity has implications both for how our cognitive resources are deployed and for our ability to reason using those resources ((Lakoff 1987); (Medin 1989); (Dijkstra 1972); (Jackendoff 1989)). For example, in the context of sentence creation and comprehension Jackendoff explains why it is futile for us to attempt to depend on lists and to try to remember sentences in a list-like way

"a speaker's repertoire of syntactic structures cannot be characterized just as a finite list of sentences. Nor, of course, can it be characterized as an infinite set of possible sentences, because it must be instantiated in a finite (albeit large) brain" (Jackendoff 1989, p.70).

In addition, referring to the concept "dog" he explains,

"the concept expressed by the word dog... cannot be encoded as a list of dogs previously encountered; nor, because the brain is finite, can it be a list of all dogs there ever have been and will be, or of all possible dogs" (Jackendoff 1989, p.71).

Dijkstra described reasoning based on lists of specific values as "enumerative reasoning" (Dahl, Dijkstra et al. 1978, p.11). Enumerative reasoning establishes each entry in an enumeration as a unique entity. This is problematic because partial knowledge of an enumeration can inhibit its application. Dijkstra provides a simple example

when the notion of "number" dawned upon our ancestors, they invented individual names for each number they found they wanted to refer to; they must have had names for numbers in very much the same way as we have the names "one, two, three, four, etcetera."

They are truly "names" in the sense that by inspecting the sequence "one, two, three" no rule enables us to derive that the next one will be "four". You really must *know* that.

(At an age when I knew perfectly well how to count – in Dutch – I had to *learn* to count in English, and during a school test no clever inspection of the words "seven" and "nine" would enable me to derive how to spell "eight", let alone pronounce it!) (Dijkstra 1976, p.10)

We enter a state of ignorance when we find ourselves in a situation where we *really must know that* and we don't. Enumerative reasoning is useful for collections that are short, occur frequently or are fixed – the traffic light sequence (red, amber, green); the days of the week; months of the year; the numbers one to twenty; the names of the members of our family; and so on.

However, "The world is filled with an overwhelming variety of objects and living things" (Tversky and Hemenway 1984, p.170) and our daily interactions with our physical and social environment bring us in constant contact with a vast range of stimuli³. The combinatorial possibilities associated with these interactions and their stimuli are "limitless" (Medin 1989), "virtually infinite" (Rosch, Mervis et al. 1976) or "overwhelming" (Tversky and Hemenway 1984). Enumerative reasoning is not a suitable mechanism for dealing with these stimuli because "To remember and treat everything in one's environment as unique would require tremendous cognitive capacity" and humans don't have that capacity (Komatsu 1992, p.501). Articulating the limitations of enumerative reasoning Dijkstra warns

³ Influenced by the nomenclature of their disciplines and the context of their contribution to the discourse, many authors use generic referents such as "stimulus," "object," "artefact," "system," "thing," and so on, to refer to entities of interest. I have adopted "stimulus" throughout the text, except within quotations.

"Enumerative reasoning is alright as far as it goes, but as we are rather slow-witted it does not go very far. Enumerative reasoning is only an adequate mental tool under the severe boundary condition that we use it only very moderately" (Dahl, Dijkstra et al. 1978, p.11).

So how do we cope with the complexity and infinite variability of the physical and social environment we inhabit?

2.2 Exploiting Redundancy and Structure

In his essay *The Architecture of Complexity*, Herbert Simon explains that we are able to understand and describe complex systems that exhibit structure. By exploiting the structure of complex systems we can describe them economically so that the description does not itself have to be complex. The structure of a system can be exposed and exploited by identifying the "redundancy" in the system (Simon 1962, p.478).

Garner describes this as "the informational concept of redundancy" (Garner 1974, p.5). *Redundancy* occurs when two or more properties of a stimulus are "correlated" thereby allowing one of them, any one, to be retained as representative of the stimulus and the others to be subsumed.

Consider the stimulus shown in Figure 1 (taken from (Garner 1974, p.83)). The components of the stimulus are formed by combining the properties *shape* (i.e. circle or triangle); *internal line style* (i.e. one vertical line or two); *opening* (i.e. on the right or left); and *dot position* (i.e. above or below). What Garner calls the "total set" is derived from the Cartesian product of the four dichotomous properties (i.e. 2 shapes \times 2 internal line styles \times 2 openings \times 2 dot positions) yielding a total set with sixteen members. Clearly, the *total set* is an enumeration of the all of the combinatorial possibilities and in the example depicted in Figure 1 a single instance of each member is included.

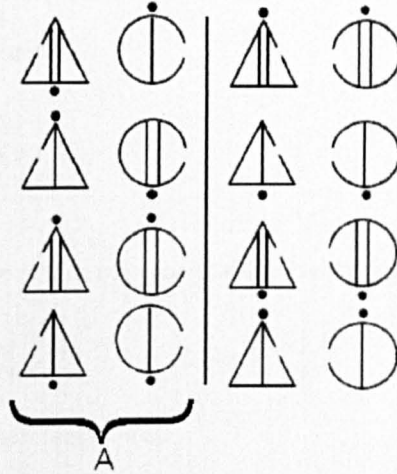


Figure 1

Taking the eight components on the left hand side, labelled A, the properties *shape* and *opening* are correlated. If we were asked to draw one of the members of subset A (i.e. any one of our choice) that has the opening on the left, our drawing would have to be in the shape of a triangle. If we were asked to draw one of the circles the opening would have to be placed on the right. In each of these examples one of the properties, *shape* or *opening*, is redundant. We could dispense with either one, but not both.

Now consider the matrix shown in Figure 2 (Simon 1962, p.478). Although not as immediately obvious as is the case in the foregoing graphically-based example, this stimulus also has high redundancy.

A	B	M	N	R	S	H	I
C	D	O	P	T	U	J	K
M	N	A	B	H	I	R	S
O	P	C	D	J	K	T	U
R	S	H	I	A	B	M	N
T	U	J	K	C	D	O	P
H	I	R	S	M	N	A	B
J	K	T	U	O	P	C	D

Figure 2

For example, if we assume the following assignments

$$W = \begin{vmatrix} AB \\ CD \end{vmatrix} \quad X = \begin{vmatrix} MN \\ OP \end{vmatrix} \quad Y = \begin{vmatrix} RS \\ TU \end{vmatrix} \quad Z = \begin{vmatrix} HI \\ JK \end{vmatrix}$$

the matrix can be represented as

$$\begin{pmatrix} WX & YZ \\ XW & ZY \\ YZ & WX \\ ZY & XW \end{pmatrix}$$

Further, this representation affords the opportunity to identify additional redundancy as follows

$$F = \begin{pmatrix} WX \\ XW \end{pmatrix} \quad G = \begin{pmatrix} YZ \\ ZY \end{pmatrix}$$

yielding a (possible) final representation as

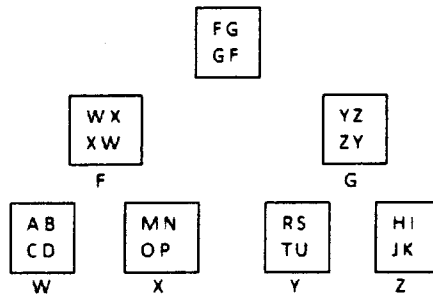
$$\begin{pmatrix} FG \\ GF \end{pmatrix}$$

Thus, by identifying and exploiting the redundancy in the original matrix we can capture the detail using just seven components

$$W = \begin{pmatrix} AB \\ CD \end{pmatrix} \quad X = \begin{pmatrix} MN \\ OP \end{pmatrix} \quad Y = \begin{pmatrix} RS \\ TU \end{pmatrix} \quad Z = \begin{pmatrix} HI \\ JK \end{pmatrix} \quad F = \begin{pmatrix} WX \\ XW \end{pmatrix}$$

$$G = \begin{pmatrix} YZ \\ ZY \end{pmatrix} \quad \begin{pmatrix} FG \\ GF \end{pmatrix}$$

As an aid we may chose to arrange the components in a representation that highlights the structural relationships between them



One final example is presented with two purposes in mind. First to counter the idea that the first two examples have been conveniently manufactured to portray redundancy and structure as a self-fulfilling prophesy. Second, and more importantly and significantly, to emphasise that

despite the complexity and infinite variability of the physical and social environment we inhabit it does in fact have structure.

Eleanor Rosch and her colleagues have shown that real-world attributes do not occur independently of each other. Some combinations are quite probable; some rare; and some have been shown empirically not to occur.

"Creatures with feathers are more likely also to have wings than creatures with fur, and objects with the visual appearance of chairs are more likely to have functional sit-on-ability than objects with the appearance of cats." (Rosch, Mervis et al. 1976, p.383)

In one exercise they enumerated a total set, a *l  Garner*, of the attributes normally used when classifying animals. The attributes used were *coat* (fur or feathers), *oral opening* (mouth or beak) and *primary mode of locomotion* (flying or on foot). This resulted in eight hypothetical animals with the following characteristics

- a. those with fur and mouths, which move about primarily on foot
- b. those with fur and mouths, which move about primarily by flying
- c. those with fur and beaks, which move about primarily on foot
- d. those with fur and beaks, which move about primarily by flying
- e. those with feathers and mouths, which move about primarily on foot
- f. those with feathers and mouths, which move about primarily by flying
- g. those with feathers and beaks, which move about primarily on foot
- h. those with feathers and beaks, which move about primarily by flying

Of the eight combinations just two of them "comprise the great majority of existent species in the world." Entry (a) is the subset of mammals and entry (h) the subset of birds (Mervis and Rosch 1981, p.91).

The world is structured. It has inherent, built-in, or what Garner termed "intrinsic" structure. This structure derives from the abundance of redundant properties in the world.

Our limited cognitive resources can deal with the world by using our ability to exploit its structure.

2.2.1 Structure: Discovered and Imposed

To exploit structure we must be able to discover and recognise it. In addition, we sometimes find it convenient to impose or invent structure to augment existing structure or to introduce structure where it does not exist (i.e. where there is an absence of structure).

Observation, investigation and research can allow intrinsic structure to be discovered, though discovery is sometimes not easily achieved and the structure is not always directly observable. Simon has described how the creation of an appropriate representation can reveal structure that was previously unnoticed or undetected (Simon 1962, p.478). For example, Mendeleev's arrangement of the elements into the *periodic table* revealed the intrinsic though previously unrecognised structure of the elements and the associated relationships between them. On the basis of this arrangement other elements were identified because gaps in the table invited investigation. Similarly, in diagrammatic representations of the structure of the brain the placement by phrenologists of various mental faculties in particular parts may be empirically discoverable using technology but not directly observable. Thus, they may know a particular part of the brain is responsible for vision but they cannot see what the brain is seeing.

Medin notes that "people may impose rather than discover structure in the world" (Medin 1989, p.1469). Structure can be imposed where none exists, or in addition to that which already exists. For instance, humans have imagined a grid of longitude and latitude lines on the globe to facilitate navigation and other activities. Similarly, the specification of time zones and the adjustment of clocks for summer and winter times are constructed conveniences. As structuring devices which are not dependent on the inherent internal order, they can be altered. China has uniquely chosen to impose a unified time structure across the entire country despite the fact that the country spans several international time zones.

Even where structure exists, additional structure can be added. For example, the absence of empirical evidence for the existence of unicorns does not inhibit our ability to impose them on any intrinsic structure of animals we may have already discovered. Murphy and Medin describe this type of structure as "empty" because it has no actual instantiations but they also note that many cultures are full of mythical and fictional entities like this (Murphy and Medin 1985, p.294).

A much more significant example is provided by the discipline of mathematics where Hadamard describes Cardan's development of imaginary numbers and how they fundamentally transformed mathematical science. Hadamard described it as follows

Let us recall what an imaginary quantity is. The rules of algebra show that the square of any number, whether positive or negative, is a positive number: therefore to speak of the square root of a negative number is mere absurdity. Now, Cardan deliberately commits that absurdity and begins to calculate on such "imaginary" quantities.

One would describe this as pure madness; and yet the whole development of algebra and analysis would have been impossible without that fundament – which, of course, was, in the nineteenth century, established on solid and rigorous bases. It has been written that the shortest and best way between two truths of the real domain often passes through the imaginary one. ((Hadamard 1954, p.122)

Hadamard also explains why there is little difference cognitively between intrinsic structure that is discoverable and imposed, or invented structure that may be empty and incapable of empirical instantiation.

We speak of invention: it would be more correct to speak of discovery. The distinction between these two words is well known: discovery concerns a phenomenon, a law, a being which already existed, but had not been perceived. Columbus discovered America: it existed before him; on the contrary, Franklin invented the lightning rod: before him there had never been any lightning rod.

Such a distinction has proved less evident than appears at first glance. Toricelli has discovered that when one inverts a closed tube on the mercury trough, the mercury ascends to a certain determinate height: this is a discovery; but in doing this, he has invented the barometer; and there are plenty of examples of scientific results which are just as much discoveries as inventions. Franklin's invention of the lightning rod is hardly different from his discovery of the electric nature of thunder.

This is a reason why the aforementioned distinction does not truly concern us; and, as a matter of fact, psychological conditions are quite the same for both cases. (Hadamard 1954, p.xi)

Terms like imposed, invented, imaginary and empty can prejudice our acceptance and valuation of structure that is not intrinsic. Indeed, it may be alluring to harbour a preference for intrinsic structure because it actually exists, may be possible to empirically isolate and is an inherent or naturally occurring phenomenon. However, it should not, indeed cannot, be preferred over imposed or invented structure. The unions, intersections and differences of structures, whether intrinsic or imposed, are the materials used in the architecture of complexity described by Simon. As Hadamard confirms we don't distinguish cognitively between the sources of the structures we exploit. We simply exploit them.

2.3 Categorization

The process of discovering or imposing structure is variously described as “partitioning” (Medin 1989), “subdivision” or “decomposition” (Tversky 1989), “classification” (Tversky and Hemenway 1984) but more commonly as “categorization” ((Rosch 1978); (Zentall, Galizio et al. 2002); (Solomon, Medin et al. 1999); (Lakoff 1987); (Medin 1989)).

Zentall *et al.* regard categorization as the “process of determining what things ‘belong together’ and a *category* is a group or class of stimuli or events that so cohere” (Zentall, Galizio et al. 2002, p.237). Douglas Medin describes it as a strategy for “treating two or more distinct entities as in some way equivalent” (Medin 1989, p.1469). Kwasnik views it as “the meaningful clustering of experience” (Kwasnik 1999, p.24). Smith and Medin believe categorization captures the notion “that many objects or events are alike in some important respects, and hence can be thought about and responded to in ways already mastered” (Smith and Medin 1981, p.1).

Mervis and Rosch argue that without “categorization an organism could not interact profitably with the infinitely distinguishable objects and events it experiences” ((Mervis and Rosch 1981, p.94) and Van Loocke suggests that “in order to cope with its environment every living system must categorize ‘things’ and ‘events’ into classes that provoke similar reactions” ((Van Loocke 1999, p.1). Smith and Medin provide a useful synopsis

If we perceived each entity as unique, we would be overwhelmed by the sheer diversity of what we experience and unable to remember more than a minute fraction of what we encounter. And if each individual entity needed a distinct name, our language would be staggeringly complex and communication virtually impossible. Fortunately, though, we do not perceive, remember, and talk about each object and event as unique, but rather as an instance of a class or concept that we already know something about. (Smith and Medin 1981, p.1)

Rosch *et al* describe categorisation as “one of the most basic functions of all organisms” (Rosch, Mervis et al. 1976, p382). This view is elaborated by Lakoff when he writes “Categorization is not a matter to be taken lightly. There is nothing more basic than categorization to our thought, perception, action, and speech. Every time we see something as a *kind* of thing, for example, a tree, we are categorizing” (Lakoff 1987, p.5).

Despite their variety, a deck chair, a kitchen chair and a wheelchair might all be categorised as *chairs* on the basis of what might be described as their sit-on-ability. Treating these distinct

entities as equivalent (i.e. as chairs) means that we do not have to cognitively instantiate, paraphrasing Jackendoff, all chairs there ever have been and will be, or all possible chairs. Thus, an obvious benefit of categorization is its support for the parsimonious use of our limited cognitive resources.

That infrastructural benefit is important but it is superseded by the vastly more significant informational benefit which allows us to exploit our categorizations in novel situations and build "knowledge" ((Solomon, Medin et al. 1999); (Medin 1989)). As Solomon *et al* note "Categorization is not an end in itself, but rather it serves to connect old to new: categorizing novel entities allows the cognitive system to bring relevant previous knowledge to bear in the service of understanding the novel entity. Recognizing some unusual shape as a 'toothbrush' allows one to understand its parts and their functions" (Solomon, Medin et al. 1999, p.99). Similarly, Medin highlights the predictive and inferential benefits of categorisation when he states "Categorization involves treating two or more distinct entities as in some way equivalent in the service of accessing knowledge and making predictions" (Medin 1989, p.1469).

In his classic text on mathematical problem solving (Polya 1990) details the steps required to solve a problem. Step one is the requirement to understand the problem. Step two asks "Have you seen it before? Or have you seen the same problem in a slightly different form?" Thus, having established what it is we are required to do we immediately consider the possibility that we did something similar before and attempt to exploit the benefits of that experience.

Similarly, Medin provides a number of examples of how medical and para-medical practitioners try to "find points of contact between previous situations and the current context" noting that "diagnostic categories" allow them to "predict the efficacy of alternative treatments and to share their experiences." Categorization also allows them "to learn about aetiology. People who show a common manifestation of some problem may share common precipitating conditions or causes" (Medin 1989, p.1469).

(Solomon, Medin et al. 1999) review the type of cognitive activities categorization supports and among them include recognition, inference, explanation, reasoning, learning and communication. For example, on hearing someone repeatedly shouting the word "United" in a particular prosodic style we might infer that they are an instance of our categorization *football fan*. On encountering them our categorization would explain why they are wearing a red jersey with a distinctive logo. This would allow us to recognise them as a fan of a particular team. We

might reason that they are shouting because their team was victorious. We could communicate with our friends to advise them that we think "United" won their game. We would have learned that football fans are happy when their team wins and this information could be incorporated into to our *football fan* categorization.

2.4 Purposeful Categorization

Categorization is about structuring and the latitude associated with the ability to invent structure affords the opportunity to engage in laissez faire categorization. However, this is cognitively difficult to instantiate and manage so the strategy is tempered by pragmatic considerations, as Rosch and her colleagues have observed,

"it would appear to the organism's advantage to have as many properties as possible predictable from knowing any one property...a principle that would lead to the formation of large numbers of categories with the finest possible discriminations between categories. On the other hand, one purpose of categorization is to reduce the infinite differences among stimuli to behaviourally and cognitively usable proportions. It is to the organism's advantage not to differentiate one stimulus from others when that differentiation is irrelevant to the purposes at hand" (Rosch, Mervis et al. 1976, p.384).

This is described by Lloyd Komatsu as the "trade-off between informativeness and economy" which, he explains, is inherent in categorization. Thus,

"If categories are very general, there will be relatively few categories (increasing economy), but there will be few characteristics that one can assume different members of a category share (decreasing informativeness) and few occasions on which members of the category can be treated as identical. If categories are very specific, there will be relatively many categories (decreasing economy), but there will be many characteristics that one can assume different members of a category share (increasing informativeness) and many occasions on which members can be treated as identical" (Komatsu 1992, p.501).

Eleanor Rosch succinctly encapsulates this trade-off in her *Principle of Cognitive Economy*, which she states as "The task of category systems is to provide maximum information with the least cognitive effort. To categorize a stimulus means to consider it, for purposes" (Rosch 1978, p.29).

All categorizations are inextricably linked to some purpose. *How* one or several individuals categorize a single or several stimuli depends on the utility they derive from that categorization.

Thus Garner's arrangement of the stimuli in Figure 1 identified four categories or subsets (i.e. A, B, J, K) which suited the purposes of his exposition. Likewise, Cardan's introduction of the "imaginaries" category suited his computational purpose. However, the general principle of purposeful categorization is rich in variety.

Gregory Murphy identifies some of the categorizations a putative dog named Wilbur might participate in as follows "Wilbur might simultaneously be a bulldog, a dog, a mammal, and an animal. At various times he might be considered a pet, a friend, a guard dog, or even a weapon...different categories are most relevant or useful at any given time" (Murphy 2002, p.199).

Similarly, we categorize numbers as numbers but also as odd, even, integers, or real; shapes as shapes or squares, rectangles, circular, elliptical; chairs as chairs or dining, kitchen, lounge. The purpose of these categorizations is taxonomic and the items in the categories are reasonably predictable.

(Burbules 2000, p.171) notes that "Any two things can be linked, including a raven and a writing desk; but *what the link means* can be a deep puzzle." For example, it is nearly impossible to determine the link between children, dog, stereo, and blanket but when they are identified as "things to take from one's home during a fire" the link appears obvious and the existence of a category that includes children, dog, stereo and blanket makes sense (Barsalou 1983, p.214). One can introspect on potential members of the categories "things that could fall on your head" and "things to eat on a diet," which Barsalou describes as "ad-hoc" and "goal" based categories respectively, but whatever items one includes in the membership of those categories they will appear self-evident by virtue of their purposes. The purpose of a category can explicate the identities of the members and make the category "sensible or coherent" (Murphy and Medin 1985, p.289).

Equally, the absence of a purpose can render a category incoherent and worthless. For example, a category described by Murphy and Medin includes plums and lawnmowers on the basis that they both weigh less than 10,000 kg, did not exist 10,000,000 years ago, cannot hear, can be dropped, take up space, and so on. It is possible to create an infinite number of categories like this one but the absence of a purpose for them, except as examples of a worthless category, prevents their instantiation in any pervasive or persistent way (Murphy and Medin 1985, p.292).

A category may also be incomprehensible if it is formed for a purpose, but a user is ignorant of the purpose. Consider the example provided by Murphy and Medin which they introduce as follows

A somewhat unusual, but nonetheless useful, example arises from an old puzzle of biblical scholarship, the dietary rules associated with the abominations of Leviticus, which produce the categories *clean animals* and *unclean animals*. Why should camels, ostriches, crocodiles, mice, sharks, and eels be declared unclean, whereas gazelles, frogs, most fish, grasshoppers, and some locusts be clean? What could chameleons, moles, and crocodiles have in common that they should be listed together? That is, what is there about clean and unclean animals that makes these categories sensible or coherent? (Murphy and Medin 1985, p.289)

Later in the text the authors offer a possible explanation

Over the years there have been many speculations concerning what properties of animals gave rise to their being listed as clean or unclean, as the overall similarity of the animals in each group is so low. To our minds, the most cogent speculation concerning this classification rule, developed in Mary Douglas's (1966) intriguing book, *Purity and Danger*, is that there should be a correlation between type of habitat, biological structure, and form of locomotion. Creatures of the water should have fins and scales, and swim; creatures of the land should have four legs and jump or walk; and creatures of the air should fly with feathered wings. Any class of creature not equipped for the right kind of locomotion in its element is unclean. For example, ostriches would be unclean because they do not fly. Crocodiles are unclean because their front appendages look like hands, and yet they walk on all fours. If this analysis is correct, then there was a theory of appropriate physiological structure associated with each type of environment, and any animal that did not meet its standards was unclean (Murphy and Medin 1985, p.312-313)

This analysis would suggest that the purpose of the categorization was to ensure that adherents to the dietary rules ate only food derived from animals with an "appropriate physiological structure," food that was 'flawless' or 'pure.' As the authors note, it is a matter of speculation as to whether the conclusion is valid but it is clear that access to the possible purpose of the categorization provides the categorization with a greater degree of sense and coherence.

However, they do note that "the conceptual scheme associated with the division of clean and unclean animals is more elaborated and more intertwined with the culture that gave rise to these concepts than this example implies" (Murphy and Medin 1985, p.313).

Jackendoff offers a similar view when he argues that a category's purpose "cannot be explicated without at the same time sketching the background against which it is set...cannot be evaluated

without at the same time evaluating the world-view in which it plays a role...the evaluation of a world-view is at least in part dependent on one's purposes" (Jackendoff 1989, p.68).

Thomas Kuhn provides a striking example to support this view when he shares the following anecdote

An investigator who hoped to learn something about what scientists took the atomic theory to be asked a distinguished physicist and an eminent chemist whether a single atom of helium was or was not a molecule. Both answered without hesitation, but their answers were not the same. For the chemist the atom of helium was a molecule because it behaved like one with respect to the kinetic theory of gases. For the physicist, on the other hand, the helium atom was not a molecule because it displayed no molecular structure. Presumably both men were talking of the same particle, but they were viewing it through their own research training and practice. Their experience in problem-solving told them what a molecule must be, undoubtedly their experiences had much in common, but they did not, in this case, tell the two specialists the same thing. (Kuhn 1996, p.50-51)

Likewise, Medin et al. investigated the categorizations of tree experts and found differences between experts who were taxonomists, landscapers, and parks maintenance personnel. All provided taxonomic botanical categories but the landscapers also had a large number of goal-related categories including *street trees*, *shade trees*, and *ornamental trees*. Both landscapers and parks maintenance personnel had a *weed trees* category which they described as trees that are fast growing and require a lot of maintenance. The experts' world-view and purposes heavily influenced their categorizations (Lynch, Coley et al. 2000)

Medin provides a usefully summary of the importance of world-view and purpose when he writes

"The world could be partitioned in a limitless variety of ways, yet people find only a miniscule subset of possible classifications to be meaningful. Part of the answer to the categorization question likely does depend on the nature of the world, but part also surely depends on the nature of the organism and its goals. Dolphins have no use for psychodiagnostic categories" (Medin 1989, p.1469).

Categorisation is not an end in itself. How we categorize things is determined by how we view the world and what we are trying to achieve.

An educationalist may categorise a list of topics in a particular discipline as perfunctory, important, fundamental or core. A layperson may view them as a list of topics of equal importance all of which must be acquired to become knowledgeable in the discipline.

We may categorize the same thing in several ways because it suits our purpose or because in different settings or scenarios it is necessary to do so. For example, I may categorize the Java programming language as a programming language; a high-level programming language; an object-oriented programming language; a widely used programming language; my best (i.e. the one I am most proficient with) programming language. In contrast, on the basis of a different world-view someone may object to the description of Java as a language because it cannot be spoken and may prefer to describe it as a programming notation. This world-view may include symbols and formulae used by mathematicians as mathematical notation; chemistry equations as chemical notation; stenography as transcription notation.

As Rosch observed, organisms categorize for purposes. Knowing the purpose facilitates comprehension of the category.

2.5 Concepts

As has been frequently observed categories are concepts ((Murphy 2002); (Van Loocke 1999); (Perkins 2006); (Lakoff 1987); (Medin and Smith 1984)).

Some authors like to keep this equivalence simple. For example, Van Loocke notes “A simple living system may categorize its environment into things to approach versus things to avoid, or into things to eat and things to mate with. These are concepts” (Van Loocke 1999, p.1). Likewise, Perkins notes that “concepts function as categorisers. They carve up the world we already see and often posit the unseen or even the unseeable” (Perkins 2006, p.41).

Other authors, notably in the disciplines of psychology and cognitive science, like to introduce a distinction between what they describe as the “mental representation” of concepts in contrast to the partitioning characteristic of categories. For example, Komatsu states “a concept is assumed to be the mental representation of a category or class” (Komatsu 1992, p.500). Similarly, Murphy notes that “if we have formed a concept (a mental representation) corresponding to that category (the class of objects in the world), then the concept will help us understand and respond appropriately to a new entity in that category” (Murphy 2002, p.1).

The equivalence provided by Sloman *et al* is particularly insightful

Concepts and categories are, to a large extent, flip sides of the same coin. Roughly speaking, a concept is an idea that characterizes a set, or category, of objects. We

construe the terms "concept" and "category" to refer to two different perspectives that a subject can take on a class of objects, what Tversky and Kahneman (1983) call the "inside" and the "outside" views. The inside view regards the internal structure of a concept, its features and what binds them together. The outside view regards some or all of the instances that are believed to be included in the category referred to by the concept...Because the inside and outside views of a class of objects are usually compatible, measures of conceptual and category structure usually coincide. (Sloman, Love et al. 1998, p.192)

Taking a more functional perspective Lakoff notes "Conceptual systems are organized in terms of categories, and most if not all of our thought involves those categories" (Lakoff 1987, p.xvii).

Concepts and categories are inextricably linked - you can't have one without the other. It is a matter of choice as to whether one calls them concepts or categories and many authors use both interchangeably. I choose to refer to them as concepts and will do so for the remainder of the text.

2.6 Theories of Concepts

How different entities are grouped into categories to form concepts has occupied philosophers, anthropologists, psychologists, linguists, cognitive scientists, computer scientists and many other disciplines throughout history. To date the study of categorization and the evolution of theories of conceptual acquisition has been delineated by three important periods, or eras, which coincide with the adoption and subsequent reassessment of three views, or paradigms, of conceptual structure.

The following sections trace the evolution of the three views. In the conceptual literature they are variously referred to as (1) the traditional or definitional or classical view, (2) the exemplar or prototype or family resemblance view, and (3) the explanation-based or knowledge-based or theory theory view. I identify them as the classical, family resemblance, and theory theory views.

Because the material is drawn from a number of sources and disciplines the terminology used is not uniform. For example, the terms concept and category are used interchangeably in some disciplines and uniquely in others. Likewise, members of a category or concept may be described as having particular features, characteristics, or attributes. I have not sought to impose a terminology and have retained the nomenclature of the original.

2.6.1 Classical View

The *classical* (Medin and Smith 1984, p.115) view of concepts, also known as the *traditional* view (Laurance and Margolis 1999, p.8) and the *definitional* view (Komatsu 1992, p.503), derives its name from its peripatetic origins and the longevity of its existence having maintained its position as the dominant paradigm of concept structure for more than two thousand years.

The philosophical basis for the classical view dates back to antiquity and an essay by Aristotle with the title *On Categories* which the philosopher Bertrand Russell described as having had "considerable importance in the history of philosophy" (Russell 2009, p.255). In experimental psychology the classical view "can be traced to Hull's 1920 monograph on concept attainment" (Smith and Medin 1981, p.22).

Murphy quotes Hull's concise description of the classical approach which read "All of the individual experiences which require a given reaction, must contain certain characteristics which are at the same time common to all members of the group requiring this reaction and which are NOT found in any members of the groups requiring different reactions" (Murphy 2002, p.13).

He identifies its two key elements "The first we call *necessity*. The parts of the definition must be in the entity, or else it is not a member of the category...The second aspect we call *sufficiency*. If something has all the parts mentioned in the definition, then it must be a member of the category" (Murphy 2002, p.12).

The utility of these terms has encouraged their universal adoption in the concepts literature. As a consequence, a typical description of the classical view of concepts holds "that all instances of a concept share common properties that are necessary and sufficient conditions for defining the concept" (Medin and Smith 1984, p.115).

This succinct description is essentially a definition with a simple, identifiable focus and a determinable binary outcome of inclusion or exclusion. Application of the classical view involves enumeration of the necessary properties and check-list type confirmation of their presence. In that regard it is an attractive tool because it offers very clear lines of demarcation. In addition, the presumption of logic and certitude make it appear incontestable.

2.6.1.2 Dominant Paradigm

These features established the classical view as the dominant paradigm for two millennia. For example, writing about the concepts Sun and Gold the seventeenth century philosopher John Locke (cited in (Laurance and Margolis 1999, p.9)) states

The idea of the *Sun*, what is it, but an aggregate of those several simple ideas – bright, hot, roundish, having a constant regular motion, at a certain distance from us, and, perhaps, some other....

The greatest part of the ideas that make our complex idea of *Gold*, are yellowness, great weight, ductility, fusibility, and solubility...

Even in popular culture the ubiquity of the classical view of concepts was evident. For example, the writer Charles Dickens cleverly exploited it as one of his themes in his classic novel *Hard Times*, published in 1854, and which includes the following passage

Mr. Gradgrind (to Sissy Jupe) "Give me your definition of a horse."

(Sissy Jupe thrown into the greatest alarm by this demand.)

"Girl number twenty unable to define a horse!" said Mr. Gradgrind, for the general behoof of all the little pitchers. "Girl number twenty possessed of no facts, in reference to one of the commonest of animals! Some boy's definition of a horse. Bitzer, yours."

...

"Quadruped. Graminivorous. Forty teeth, namely twenty-four grinders, four eye-teeth, and twelve incisive. Sheds coat in the spring; in marshy countries, sheds hoofs, too. Hoofs hard, but requiring to be shod with iron. Age known by marks in mouth." Thus (and much more) Bitzer.

"Now girl number twenty," said Mr. Gradgrind. "You know what a horse is."

The classical view also formed the basis for scientific practice in the early literature (i.e. pre 1970) of concepts research. Most of the experimental work assumed a definitional theory of concepts. The approach wasn't explicitly stated but it was apparent from the stimuli used in experiments that the researchers took it for granted. In most of the studies the stimuli that were expected to be categorised as members all had elements in common. The stimuli expected to be classified as non-members did not have these elements (Murphy 2002, p.14).

Smith and Medin identify the work of Hull as the catalyst for this. They describe it thus

"When Hull started his experimental study of the classical view in 1920, he used novel visual forms that were composed of multiple features. This allowed him to control precisely which features occurred in all instances of a concept, that is, which

features were necessary. Had he used more abstract features, like functional ones, Hull would have had either to give his subjects real manipulable objects and let them discover the function (a messy task at best), or to give them pictures of objects that instantiated the function to varying degrees (which is again a relatively uncontrolled paradigm, though very likely a more ecologically valid one). Hull's emphasis on easily manipulable perceptual features proved so attractive that more than two generations of experimental psychologists have bought it" (Smith and Medin 1981, p.27)

The longevity and ubiquitous adoption of the classical view of concepts has resulted in all developments and alternative theories of concept formation being reactions to or modifications of the classical view (Laurance and Margolis 1999, p.8).

2.6.1.3 The Illusion of Simplicity

The simplicity of the classical view may explain why it retained its status as the dominant paradigm from the time of Aristotle up to the middle of the twentieth century. From every perspective the classical view has a powerful intuitive appeal because its realisation appears so straightforward. Its simple account of how we form categories, and thus concepts, is compelling. We encounter many stimuli, identify the singly necessary and jointly sufficient properties or features that instances of a particular category have in common and separate the members of the category from the non-members.

In this scenario concepts act like sets with members and non-members included or excluded on the basis of the application of the membership rules. The rules support the principle of cognitive economy because they provide a single representation that encapsulates the membership requirements. Application of the rules yields taxonomic structures with clear boundaries and highly coherent members providing specific information about the concept instances. In the cognitive trade-off between economy and informativeness the classical view is biased in favour of economy.

George Lakoff articulates the application of the scenario as follows

Most categorization is automatic and unconscious, and if we become aware of it at all, it is only in problematic cases. In moving about the world, we automatically categorize people, animals, and physical objects, both natural and man-made. This sometimes leads to the impression that we sometimes categorize things as they are, that things come in natural kinds, and that our categories of mind naturally fit the kinds of things there are in the world. But a large proportion of our categories are not categories of things; they are categories of abstract entities. We categorize events, actions, emotions, spatial relationships, social relationships, and abstract

entities of an enormous range: governments, illnesses, and entities in both scientific and folk theories, like electrons and colds. (Lakoff 1987, p.6)

Thus the classical approach fostered the impression that concepts were well-understood phenomena and that the necessary and sufficient features were always directly observable and, if not, they were at least discoverable. Regrettably such an intuitive sense now appears misplaced.

For example, the necessary and sufficient properties of *happiness* are impossible to articulate in any definitive way yet the pursuit of happiness is believed to be the steadfast goal of every human being. Indeed, the pursuit of happiness is included as one of the "unalienable Rights" identified in the United States Declaration of Independence. Many people claim to be happy and many people claim to be able to recognise, and identify evidence of, the happiness of others, despite their inability to enumerate the necessary and sufficient properties of happiness.

It is not just happiness that eschews simple conceptual classification. We "float in a world of familiar and comfortable but quite impossible-to-define abstract patterns" (Hofstadter 2007, p.177-8). A huge number of concepts are "vague, blurry, unbelievably elusive abstractions." What are the necessary and sufficient properties for something to be accepted as an unequivocal instance of *tackiness*, *wackiness*, *sleazeball* or *dread*? Hofstadter asks if it is possible to teach them to your children. Despite the fact that, upon casual enquiry, most people would be prepared to offer an explanation of what *tackiness* or a *dread* might be, the explanations are unlikely to be precise, equivalent or infallible. Yet it is possible to identify instances of *tackiness* and *dreads*.

Wittgenstein identified a similar definitional limbo for the concept *game* which he says is "uncircumscribed." In relation to games he questions "What is common to them all? - Don't say: There *must* be something common, or they would not be called 'games'...if you look at them you will not see something that is common to *all*" (Wittgenstein 1958, Part 1:66-71). He also noted that despite the absence of a definition for "game" someone can learn about games and can discuss, evaluate and compare games. In addition, people can learn how to play a specific game "without ever learning or formulating rules" and their knowledge about the concept of game may be accumulated by "watching" games and progressing to more and more complicated ones (Wittgenstein 1958, Part 1:31).

Michael Polanyi notes the human strategy of *ostensive definition* when we have to resort to pointing at an object so that the "person addressed will discover that which we have not been able to communicate" with words (Polanyi 1983, p.5).

Even resorting to the perceived benefit of a formal definition of a concept may not be enough to facilitate the identification of concept instances. Consider the definitions set out in Euclidian Geometry for a *point* and a *line* which are stated as (1) A point is that which has no part, and (2) A line is of breadthless length. As Cassirer makes clear

"There are no real things which precisely agree with the definitions of geometry; there are no points without magnitude, no perfectly straight lines, no circles whose radii are all equal. Moreover, from the standpoint of our experience, not only the actual reality, but the very possibility of such contents must be denied; it is at least excluded by the physical properties of our planet, if not by those of the universe" (Cassirer 1923, p.13).

Describing approaches to the teaching of geometry (Dienes 1963) notes how the topic is typically introduced using the concept of a two-dimensional plane. This strategy is used despite the fact that no real object can be described in this way. As Dienes explains

The only kinds of real-objects are solid, three-dimensional objects. It would seem common sense that we should start the study of geometry with the study of movements of three-dimensional, i.e. real objects. There is no such thing as a plane, and so it is impossible to provide experiences which correspond exactly to the structures which form part of two-dimensional or plane geometry. Yet, common sense or otherwise, the first geometry lessons usually consist in the treatment of 'lines', 'points', and their measurement in some way or another...The only valid kinds of statement about points and lines are about their interrelatedness" (Dienes 1963, p152-3)

Despite this, the discipline of mathematics exploits these concepts extensively (as with Cardan's concept of imaginary numbers), even though the concepts are "empty" and not a single example that matches the definition can be instantiated.

2.6.1.4 Dominance Wanes

The dependency of the classical view on the identification and description of both the necessary and sufficient features of a concept meant it was unable to deal with concepts which could not, and would not, fit its constraining requirements. Relying on a definitional approach inherited all of the linguistic difficulties highlighted by Dijkstra when he wrote

"So-called natural language is wonderful for the purposes it was created for, such as to be rude in, to tell jokes in, to cheat or to make love in (and Theorists of Literary Criticism can even be context-free in it), but it is hopelessly inadequate when we have to deal unambiguously with situations of great intricacy, situations which unavoidably arise in such activities as legislation, arbitration, mathematics or programming" (Dijkstra 1996).

The presumption that the whole world could be viewed as discrete and neatly ordered was exposed as a fallacy by the overwhelming number of concepts that didn't fit the classical view. The belief that all concepts could be uniquely identified by an enumeration of specific mandatory and equally dependent properties undermined the legitimacy of the classical view and its position as the only theory of concepts became increasingly untenable. This led to the development of alternative theories of concepts.

2.6.2 Family Resemblance View

Wittgenstein's portrayal of the difficulties associated with the concept *game* is generally credited with opening the first crack in the classical view of concepts (Lakoff 1987, p.16). Wittgenstein had refuted the idea that all concepts had clearly specified boundaries and properties and that membership was a straightforward boolean-like yes or no decision. Instead of being viewed as well-defined and predictable, concepts could now admit to a degree of vagueness and uncertainty.

2.6.2.1 Vagueness

With vague concepts the indeterminacy of the boundaries introduces a sorites paradox, in which, logical arguments start from what appears to be a true premise, proceed little-by-little on the basis of undisputed reasoning, but terminate with an apparently false conclusion. A "vague concept" is one that "picks out a category with no clearly defined boundaries" (Hampton 2007, p.355). There are (reportedly) forty shades of green - which one *is* green? Similarly, "There is arguably no precise height at which a man or woman becomes tall, and so the class of TALL MEN is not a well-defined set" (ibid.). Thus, if someone who is X metres tall is considered tall then so is someone who is $(X - 0.0001)$ m tall. Repeatedly applying this deduction includes everyone, even people who are one meter tall, in the set of tall people. (The process can also be inverted by starting with someone who is considered small and the resulting set of small people includes everyone).

Many everyday concepts are susceptible to this type of paradox - heap, bald, rich, hot, cold, windy, diversity. In these cases we often have a strong sense of what the necessary and sufficient requirements might be even though we are unable to articulate them precisely. For example, people speak of someone who is balding but not bald; well-to-do or comfortable but not rich; big but not tall.

Consider the concept *windy*. What do we mean when we say it is windy? In 1805 Francis Beaufort developed the Beaufort Wind Force Scale⁴. The scale allowed naval officers to describe the wind as calm, a breeze or a gale. Using more finely grained graduations a breeze could be described as light, gentle, moderate, fresh or strong and the gales as moderate, fresh, strong or whole. The original scale was based on a naval officer's qualitative assessment of the conditions of the sails, then later the sea (because of the demise of sail ships) and more recently, accurate technology-based wind speed measurements. Despite the availability of a numeric wind force scale, calibrated using the values 1 to 12, many meteorologists still use Beaufort's descriptors for weather forecasts and severe weather warnings, particularly in shipping contexts.

Zadeh proposed a logic that incorporated vagueness and supported graded set membership, which he called *fuzzy set theory* but which is more often referred to as *fuzzy logic* (Zadeh 1965). Fuzzy logic is an extension or superset of conventional logic. To support modes of reasoning which are approximate rather than exact fuzzy logic set membership is graduated and not simply bivalent. Thus, degrees of tallness or baldness or richness can be indicated by an appropriate value between 0 and 1. Fuzzy logic is an example of an attempt to model our cognitive ability to deal with questions of the type "Is it windy?" and to determine whether or not, and to what degree, the answer can be yes.

2.6.2.2 Typicality

In her seminal work on concept membership Eleanor Rosch showed that in graded membership concepts, some members are considered more representative of the concept than others (Rosch 1978). For example, robins are considered more representative of the concept *birds* than are chickens, parrots, penguins or ostriches. A penguin has a beak and it's offspring

⁴ http://www.metoffice.gov.uk/media/pdf/4/4/Fact_Sheet_No._6_-_Beaufort_Scale.pdf last accessed 19 January 2012

emerge from eggs but it doesn't nest in trees and instead of flying it swims. The ostrich has wings but is too heavy to fly and some birds, because they spent all their time on the ground, have 'forgotten' how to fly, the kiwi being just one example. Robins are considered a "prototype or best example" (Murphy 2002, p.41) or "cognitive reference point" (Lakoff 1987, p.41) because they share "more features in common with a prototype representation of the category birds than do penguins" (Hampton 2007, p.358). Wittgenstein used the term "family resemblance" to characterise this coincidence of "build, features, colour of eyes, gait, temperament, etc. etc." which "overlap and criss-cross in the same way" but which are not mandatory or uniform, just like the features of members of a family (Wittgenstein 1958, Part 1:67). Thus, instances of a concept may not have all of the features that are perceived to be necessary and sufficient, and even if they do they may not have them to the same degree or in precisely the same style.

In these cases the "*typical* category members are the good examples - what you normally think of when you think of the category. The *atypical* objects are ones that are known to be members but that are unusual in some way" (Murphy 2002, p.22). *Family resemblance* is flexible enough that we can accept unusualness, so we still know penguins to be birds, and three-legged dogs to be canine, even if we do not produce them as typical examples of their respective categories.

More formally, an instance is considered typical when the "combined relevance weights of the attributes...exceed a certain level (what might be called the membership *threshold* or *criterion*)" (Komatsu 1992, p.503). For example, satisfying the criterion causes someone to be categorised as bald and not balding; or tall and not big; or rich and not comfortable or well-to-do. Conversely, atypical instances have a subset of features which in sum are not enough to cross the threshold. For example, the concept of a bachelor is used to identify an unmarried adult male but neither the Pope nor a man in a long-term unmarried relationship would be viewed as representative of bachelors.

In addition to being the ones "you normally think of" typical instances of a concept are processed more rapidly, learned more quickly, and remembered better than other concept instances (Hampton 2007, p.356). This has important implications for our uses of typical instances. For example, a programmer asked for an example of a data type would normally think of a primitive type such as integer, real or character but not boolean. A Java programmer asked for an example of a collection would normally think of an array or an ArrayList but not a

HashSet. A programming instructor asked for an example of a difficult concept for novice programmers to learn would normally think of recursion or polymorphism but not assignment or the use of variables.

2.6.2.3 Basic Level

The family resemblance view "stresses naturalness" and so "...certain partitionings of the world are privileged, more immediate or direct" because they reflect "the natural partitioning of objects in the real world by our perceptual systems" (Komatsu 1992, p.505). Rosch et al. used the term "basic level" to describe these privileged concepts (Rosch, Mervis et al. 1976).

For example, when we see a bird we do not identify it specifically by its ornithological classification (say a spotted flycatcher) or more generally as an animal. One provides too much information and the other too little. We simply identify it as a bird, using the basic level.

Basic level concepts are our most cognitively efficient because they have two distinguishing properties - they are informative and distinctive - and it is these properties that sustain their privileged status (Murphy 2002). We tend to identify things as chairs because identifying them as furniture reduces the information provided and would be considered too abstract whereas identifying them as dining-room or classroom chairs may provide too much information and would be considered too specific. Basic concepts tend to be associated with the "functional structure" of our world (Brown 1958).

(Murphy 2002) provides a convenient tabulation of the empirical findings in relation to basic level concepts which shows that they are easier to learn, are the first acquired by children, are more informative, pictures of them are identified faster and they are more frequently used in text.

2.6.2.4 Difficulties Arise

Whilst the family resemblance approach ameliorates some of the difficulties associated with the definitional requirements of the classical view, it shares some of its difficulties as well. These difficulties arise because both views are similarity-based, and explain membership in terms of the attributes shared by the members. For example, in ad-hoc and goal directed categories (e.g. surprising birthday presents) the similarity of the members plays little or no role in explaining membership. In these cases membership is determined by similarity not to each

other, but to the goal or ideal associated with the category. Neither the classical nor the family resemblance views take account of this type of relationship (Barsalou 1983).

These and other difficulties prompted the development of an approach that considered the influence of a broader range of factors on category membership decisions.

2.6.3 The Theory Theory

Murphy and Medin argue that the focus of the classical and family resemblance approaches on the similarity of the members of a category provides "a language for talking about" why those items were included and others were not. They posit that the perceived similarity of the members may in fact be a by-product of a theory which explains why they are the members of the category in the first place. Inverting the argument in this way involves "having a theory that relates the objects" and showing that this is what makes them appear similar. For example, they note that in sport, winning teams are similar in the sense that they score more than their opponents "but one must turn to more basic principles to explain why they score more" (Murphy and Medin 1985, p.291).

Formulating a 'theory' obliges us to bring additional factors into our decisions of category membership and to draw on our existing concepts and prior knowledge. These factors include functional and causal relationships, within and between concepts, as well as our existing knowledge. For example, a similarity-based classical or family-resemblance conception of *water* might preclude it from consideration of *things you can walk on*, but a broader theory-based conception might include it under the constraint that it is ice. In this case our existing knowledge of the different states water can assume, allows us to exploit a previously unconsidered functional relationship. Likewise, a similarity-based conception of *good behaviour* may not include jumping in a pool with your clothes on, but a theory-based conception might admit it in the context of an heroic act to save a life.

"Theory" is used quite loosely in this context and it should be interpreted as "a host of mental explanations rather than a complete, organised, scientific account" (Murphy and Medin 1985, p.290).

Gudmundsdottir uses the term "narrative" to describe an analogous type of structure that creates "a reasonable order out of experience." The narrative has cognitive dimensions that

equate to the benefits that accrue to categorisation and concepts, such as, economy, selectivity and familiarity.

Economy means that the narrative order can be applied to all aspects of our lives - past, present and future. *Selectivity* is vital since we cannot pay equal attention to all our experiences. The narrative schema classifies and assigns significance to information and places it in the narrative. *Familiarity* is achieved by repetition and the creation of similar accounts of typical events. (Gudmundsdottir 1995, p.32 emphasis in original)

Gudmundsdottir's economy is equivalent to Rosch's principle of cognitive economy; her selectivity is a manifestation of feature selection in the classical view of categorisation; and her familiarity is the act of knowing through the application of the concept to subsequent encounters with instances of the concept.

Observations, events and interactions which have occurred in the temporal sequence of life are initially considered distinct happenings with no systematic cognitive connections. The narrative is formed through reflection which yields a thoughtful explanation of the past happenings. By assimilating our experience the narrative allows us to discover new meanings and transforms an incomplete story into a more complete and compelling one. For practitioners of everyday life (i.e. humans) narratives make sense of everyday experiences and organise them into a body of practical knowledge (Gudmundsdottir 1995). The narratives are analogous to the types of theories that Murphy and Medin are describing.

Murphy uses the more generic term "knowledge effects" to describe "influences of prior knowledge of real objects and events that people bring to the category-learning situation." He provides a survey of the research on knowledge effects and concludes that knowledge influences our acquisition and deployment of concepts at every level. This knowledge can influence what we consider important, how we learn to identify what is important and how it affects our decisions (Murphy 2002, Chapter 6). Thus a person witnessing a magician performing a faux hemitorporectomy can categorise it as an illusion simply by the absence of blood or more comprehensively on the basis of the physiology of the human body and the surgical difficulties associated with severed limbs. Similarly, a person witnessing a séance may choose to be sceptical of the notion of communicating with the dead despite the fact that the gathering appears to offer evidence to the contrary.

Cognitive scientists have adopted the term *mental model* to describe the type of knowledge that might be captured in a 'theory' (Gentner and Stevens 1983). Donald Norman describes it thus

"[when] interacting with the environment, with others, and with artifacts of technology, people form internal, mental models of themselves and of the things with which they are interacting. These models provide predictive and explanatory power for understanding the interaction." (Norman 1989)

Norman notes that these models are incomplete; unstable, because we forget details; unscientific; and naturally evolving (Norman 1989). The construction of these models is influenced by our background knowledge and the inferences we make from it. Our background knowledge is the collection of theories we have about how the world works, some or many of which may not be coherent but may be sufficient to explain the parts of the world we have experienced so far (Komatsu 1992, p.519). For example, a child living in France has a mental model of *things eaten by humans* that includes frogs and snails, whereas a child living in another part of the world may not.

2.6.3.1 Exposition of the Theory Theory

A more explicit, coherent exposition of how the 'theory' for a concept might operate is provided by (Michalski 1989). He hypothesises that a concept can be viewed as a two-tiered knowledge structure that is formed from (1) memorised knowledge accumulated in our interactions with the world and (2) knowledge generated from the memorised knowledge using our inferential abilities. He describes the memorised knowledge as the *base concept representation* (BCR) and the inferentially derived knowledge as the *inference concept interpretation* (ICI).

The BCR includes representative examples of the concept as well as exceptions and counterexamples. It contains the "general characteristics" of the concept that include "typical, easily definable, and possibly context-independent assertions" about it. These may have been deduced from examples and analogies or defined by a third party (e.g. parent, teacher). The general characteristics "tend to capture the principle, the ideal or intention behind a given concept" (Michalski 1989, p.123).

The role of the ICI is to apply inferential methods to the knowledge contained in the BCR. The ICI uses deduction, approximate deduction, analogical and metaphorical inference to shape the

relevant prior knowledge into an interpretation that is appropriate for the context. This process involves recognising, extending or modifying the concept consistent with the context.

Consider the following abridged version of an example taken from (Johnson-Laird 1983).

Alcock and Brown were the first to fly *an airplane* from the USA to Ireland.
Alcock and Brown were the first to fly *the Atlantic* from the USA to Ireland.

Here, the concept "fly" admits to multiple meanings which the reader has to resolve in a fashion appropriate to the context. This requires the ICI to temper the BCR knowledge of what it means to "fly" with how one flies and what can be flown. The presence of feathers and wings on an ostrich does not commit us to concluding that they fly because our (possibly crude) general knowledge of aerodynamics will invite us to question how the wing span and feather density will be able to support the bird's very large body. Likewise, the shape of penguins' "wings" and the nature of their habitat allow us to account for their preference to swim instead of flying, notwithstanding the fact that they have beaks, lay eggs and so on.

Michalski argues that the two-tiered model is consistent with the goals of cognitive economy and informativeness because "inference allows us to remember less and know more" (Michalski 1989, p.123). The model also captures the benefits of the other views (i.e. classical and family resemblance) because the application of metaphorical and analogical inference facilitates the identification of similarity-based features by literal similarity or mere appearance.

2.6.3.2 Status of the Theory Theory

Although viewed as a better model than its more simplistic predecessors the theory theory of concepts is not seen as a panacea and variants of it, including Michalski's, are viewed as incomplete. Komatsu has noted that it can be difficult to distinguish the types of knowledge (background, topic specific or general) used in inferences. In addition, it is not clear what constraints can or should be set on the explanations used (Komatsu 1992).

Murphy notes that the theory theory is more complete because it captures the fact that

"people do not rely on simple observation or feature learning in order to learn new concepts. They pay attention to the features that their prior knowledge says are the important ones. They make inferences and add information that is not actually observed in the item itself." (Murphy 2002, p.63).

Our knowledge of a concept is not static, pre-recorded detail that we retrieve and apply for each encounter. We use our knowledge actively to shape what we are learning and how we use it after we have learned it. The evolution of concepts has reached the theory theory stage but there is still much to discover about them (Murphy 2002, p.197).

2.7 Concepts, Threshold Concepts and Programming

The theory of concepts is inextricably linked with the two central aspects of this research - threshold concepts and programming.

In the first case the connection is obvious. Threshold concepts are concepts and it is axiomatic that the theory of concepts applies to them. As we shall see in Chapter 3, application of the theory is situated in the theory theory view of concepts and is primarily concerned with the beneficial pedagogical perspective provided by identifying and categorising some concepts as threshold concepts.

The connection between concepts and programming, in particular object-oriented programming, has been explored explicitly by only a handful of authors ((Madsen, Möller-Pedersen et al. 1993); (Rayside and Campbell 2000a); (Rayside and Campbell 2000b); (Rayside and Kontogiannis 2001)). It is primarily concerned with the representation of concepts in the processes that control automata and how those representations are used to build software artefacts.

In programming scenarios the act of categorisation, or classification, leading to the identification of a concept is usually described as abstraction; the resulting concepts are usually termed classes; and instances of the classes are termed objects. It is situated in the classical view of concepts and focuses on concepts as models of their properties and behaviours and how this facilitates the evolution of a software solution (Madsen, Möller-Pedersen et al. 1993).

Rayside and Campbell describe programming as a kind of argument and note that "when we deal with argument...we must deal with only one meaning at a time" (Rayside and Campbell 2000a, p.240). Programs reflect the syllogistic form of argument in that they begin by setting the initial state or premise, proceed through a series of operations or predicates which follow from the initial state and finally conclude with an outcome that realises the desired state. The goal of the process is to facilitate reasoning "with order, with ease, and without error" (Rayside

and Campbell 2000b, p.337). However, the process cannot guarantee the outcome because it is under the control of a fallible human.

Programming abstractions require the use of concepts that can be defined to have the necessary and sufficient properties of concepts adhering to the classical view of concepts, admitting to only one meaning for the purposes of determining category membership and providing "sharp concept borders" (Madsen, Möller-Pedersen et al. 1993, p.295). This is frequently described in the programming literature as an Aristotelian view of concepts. For example, Rayside and Campbell attempt to contribute to the theory of objects and a better understanding of object-oriented programming through an "Aristotelian understanding of object oriented programming" (Rayside and Campbell 2000b, p.337). Madsen et al note that BETA, their object-oriented programming language, "is mainly useful for representing Aristotelian concepts" (Madsen, Möller-Pedersen et al. 1993, p.295).

Programming abstractions that model this type of "systematic and 'scientific'" concept (Madsen, Möller-Pedersen et al. 1993, p.293) are appealing because their identification and utilisation adheres to the orderly process of definition, predication and inference characteristic of Aristotelian logic (Rayside and Campbell 2000a, p.237). This affords a high degree of equivalence and coherence between the real-world system and the software model of it. However, it is not always possible to move conveniently and effortlessly through the process.

Wittgenstein's exposition of the inherent vagueness (see section 2.6.2) in a large number of everyday concepts that renders them impossible to describe as Aristotelian concepts has implications for their representation as objects in programming systems. In the absence of a precise definition a degree of pragmatism is required when arriving at a suitable representation. Often it is necessary to "find a collection of properties that may characterize these concepts" (Madsen, Möller-Pedersen et al. 1993, p.293). This leads Madsen and his colleagues to acknowledge that the modelling process can involve "a great deal of invention" (ibid. 285) and "practical experience" (ibid. 283).

The purposeful nature of categorisation (see section 2.4) influences the choice of properties included in the collection used to characterise a concept modelled in a system. The designers world view, or what Madsen and his colleagues describe as one's "perspective for looking at phenomena" (ibid. p285), influences the choice of properties and the type and degree of inventiveness required to arrive at a workable representation. For example, they note that "a

post office may be regarded as an information process, as an economic process or as a social process" depending on whether one is a computer scientist, an economist, or a sociologist. How the post office is modelled will be influenced by that perspective (ibid. p287).

Invention may also justify the introduction of an "imaginary" concept which facilitates some goal of the model. For example, object oriented programming systems support the introduction of abstract classes which cannot be instantiated as objects but which make it possible to deal neatly and efficiently with other model requirements and goals. An object oriented program developed in the BETA programming language is described as "a physical model, simulating the behaviour of either a real or imaginary part of the world" (ibid. p16).

Bjarne Stroustrup has sought to introduce the term "concepts" directly into the C++ programming language standard. He uses the term to identify proposed language extensions that are intended to improve the C++ template system and make generic programming more accessible (Gregor, Järvi et al. 2006).

The centrality of concepts and classification to object oriented programming is well rehearsed in the everyday practice of professional software engineers.

2.8 Summary

Concepts provide us with structure, whether it is the naturally occurring structure of the world or an imposed structure which suits a purpose. Knowledge of the purpose is essential to explain, justify and successfully deploy the concept.

The view that concepts are determined by a set of necessary and sufficient features (or properties or attributes or characteristics) is an historical artefact which has been superseded. Except in a very small number of simple cases (e.g. prime number, triangle, a vertebrate) concepts are not limited to feature lists or similarity tests. For most concepts the instances are rarely homogeneous and the context of their deployment can expand their interpretation. Concepts operate more like theories which help to provide a rationale for the assigned features and suggest modes of application in given contexts.

Chapter 3

Threshold Concepts as Concepts

Threshold concepts are concepts but nowhere in the threshold concepts literature are they treated or examined using the theory of concepts as a framework. In this chapter the features attributed to a threshold concept are described in the context of their conceptual implications in contrast to their more frequent treatment from the standpoint of their affective implications. We also consider two documented criticisms of threshold concepts and show that the objections raised are without foundation.

3.1 The Threshold Concept Metaphor

Threshold Concept is a metaphoric term, which draws on ideas in anthropology and couches them in terms of a visual-spatial metaphor. In particular it draws on the types of situations and activities that a participant in a rite-of-passage style event experiences, as they progress from one status to another. This is viewed as crossing a threshold from the existing status to the new, transformed status. For example, a person crosses the threshold of being a child to becoming an adolescent and subsequently from adolescent to adult (Meyer and Land 2006).

The spatial aspects of the metaphor are derived from the work of the anthropologist Turner, who used the term "liminality (from the Latin *limen*, boundary or threshold)" to characterise the transitional state or space that is occupied by the participant in the rite-of-passage event. Turner described it as the "betwixt and between space" and the time spent in the space as the

liminal period. The threshold is also sometimes described as a "gateway" or "portal" and passing through it provides access to whatever resources are available on the other side (Meyer and Land 2006, p.22).

The imagery of a journey, specifically a "transformational journey," is frequently used to depict progress from one status to another (Land, Meyer et al. 2010, p.xv). In a learning context the journey is a metaphor for conceptual development and the learner is considered to be on a journey trying to reach a destination - a personal, intended learning outcome (Meyer 2010). The threshold, or transitional space, is a point of difficulty, a form of obstacle or delay, a "stuck place" (Meyer 2010). These stuck places, or "learning thresholds," are part of the conceptual journey and cannot be avoided. They are difficult because the transition to conceptual understanding proves troublesome (Meyer and Land 2006, p.3). The liminal period (i.e. the time spent experiencing the difficulty) may be protracted over a considerable period of time (Land, Meyer et al. 2010, p.x).

The visual aspects of the metaphor are derived from the characterisation of the effect of crossing the threshold. Once the threshold is crossed new things unfold as part of the journey because new scenery, landmarks and views of the world have become visible. In addition, new destinations previously unreachable, and possibly unknown, become reachable from this point. Prior to crossing the threshold this landscape was unheard of or inaccessible. From a conceptual development perspective crossing the threshold has a transformative effect on student's "internal view of subject matter, subject landscape, or even world view" (Meyer and Land 2006, p.3).

The visual-spatial metaphor corresponds with the types of cognitive and ontological transformations that are said to result from the acquisition of a threshold concept. The spatial metaphor relates to the experience of the threshold difficulties and the new conceptual understanding (i.e. the new location that has been reached in the conceptual development or journey). The visual metaphor relates to the ontological view one has (i.e. what one sees, or how one interprets what one sees, from this new location) as a result of the outcomes or effects of crossing the threshold.

3.1.1 Threshold Concept Feature Types

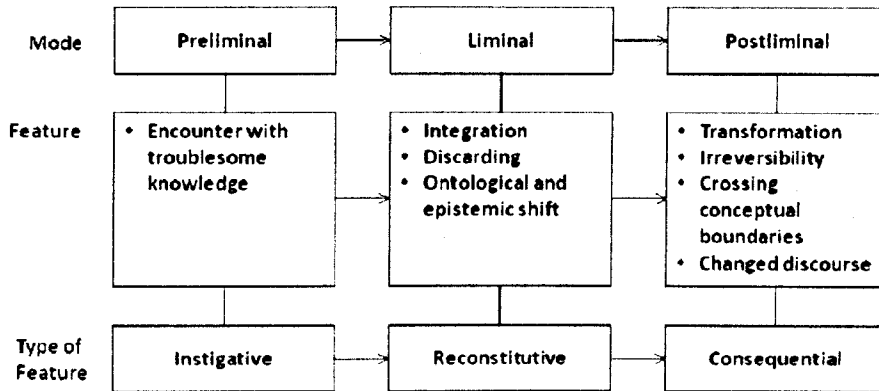
The language of journeys has prompted the adoption of descriptors such as "breach" and "hitting a brick wall" (Savin-Baden 2006) and induced notions of sequence. The journey itself has been diagrammatically represented and, on the basis of their relationship to the threshold, the point of liminality, or the stuck place, four of the five threshold concept features (i.e. transformative, irreversible, integrative, and troublesome) have been partitioned into three categories or types. The three types have been labeled *instigative*, *reconstitutive* and *consequential* (Land, Meyer et al. 2010, p.xi). The boundedness feature is not included because it is not associated with the learner experience.

The diagram is reproduced below. In the diagram the journey is partitioned into three parts, with the threshold or liminal space at the centre, bracketed by the pre- and post-liminal states.

The preliminal state is concerned with a single threshold feature - the encounter with the troublesome knowledge that impedes conceptual development and provokes a sense of difficulty or stuckness. Troublesome knowledge is categorised as instigative because it triggers or instigates efforts to resolve the conceptual difficulty.

In the liminal state acquisition of the threshold concept resolves the conceptual dilemma by integrating concepts previously viewed as disparate and establishing a new conceptual structure. In the process some existing conceptual structure may be reshaped or discarded to facilitate the formation of the new structure. For this reason integration is categorised as reconstitutive because it has the effect of reconstituting the conceptual structure. Again, this part of the diagram is associated with a single threshold feature - integration.

Acquisition of the threshold concept may take time. Integration may not proceed in an orderly, structured fashion and may be patchy and unclear. Realisation of the integrative nature of the threshold concept may only occur after several unsuccessful encounters with the material associated with the concept. This is the betwixt and between nature of the liminal space and the uncertainty associated with the duration of the liminal period. Land et al. have described this as a kind of "conceptual peristalsis" where the learner makes conceptual progress as a result of alternating wormlike waves of conceptual contraction and relaxation that "oscillate between earlier, less sophisticated understandings, and the fuller appreciation of a concept" (Land, Cousin et al. 2006, p.196).



Source : (Land, Meyer et al. 2010, p.xii)

The post-liminal stage concerns the transformative outcomes that result from the new conceptual configuration. The new, more robust conceptual structure provides a different, transformed view of the subject matter. The learner's perspective of the material is more coherent. The previous conceptual structure that has been superseded cannot be preferred to the new structure and this ensures the transformation is irreversible. In addition, there may be a shift in the learner's identity which changes their view of themselves (e.g. they may begin to believe they are bright) and their relationship with the subject area (e.g. confidence in their knowledge level increases) or even the world (e.g. they begin to describe themselves as programmers or historians or journalists). The association of this part of the diagram with outcomes explains the categorisation of transformation and irreversibility as consequential features.

3.1.2 Relationship Between the Features

In the threshold concepts literature no precedence among the features has ever been explicitly stated. However, the sequence in which they were originally introduced (i.e. transformative, irreversible, integrative and troublesome) may indicate a subconscious ranking on the part of Meyer and Land, the originators of the idea. For example, as editors of all three volumes of the biennial conference material they have commenced each Editor's Preface with an analogical instance of transformation. Indeed, the third volume has the title *Threshold Concepts and Transformational Learning*.

Offering advice to curriculum designers who might wish to use the threshold concepts idea as a basis for course design Land et al. list in order of priority (1) the need to "define potentially powerful transformative points in the student's learning experience", (2) the importance of forms of engagement that provoke the "transformations in understanding" that we would wish for, (3) an appreciation of the "tough conceptual and emotional journeys" students have to make, and (4) the need to recognise that "Grasping a threshold concept is never just a cognitive shift; it might also involve a repositioning of self in relation to the subject" (Land, Cousin et al. 2006, p.198-200)).

The prominence of transformation is also evident in the following caution offered by Meyer and Land in which the goal is stated in terms of the liberation of the "interned transformations"

"the troublesome nature of student learning...may stem from an active refusal of learning...This obviously renders problematic any simplistic schematic attempt to overcome troublesome knowledge by technicist redesign of curricula alone, and challenges easy assumptions that if the learning environment is suitably ordered and constructively aligned then the interned transformations will ensue" (Meyer and Land 2006, p.30).

The distinguished status ascribed to transformation has had a significant influence on the development of the threshold concept framework because transformation has come to dominate its literature. Much of the published work and many of the analogies and metaphors used to describe threshold concept acquisition have ontological themes and are inherently affective in nature and there is a marked absence of the cognitive aspects of threshold concepts.

Contributions exploring affective topics are well represented in the threshold concepts literature. They may be exemplified by "the scholarship of rupture in knowing" (Schwartzman 2010); "the relationship between learner identities and threshold concepts" (Savin-Baden 2006); "online feelings, judgements or estimates" (Efklides 2006). Likewise, the inherent cognitive nature of troublesome knowledge has been usurped by a focus on its affective consequences. Rather than consider the cognitive aspects of troublesomeness there is a tendency to dwell on the feelings of frustration, self-doubt and anxiety it can evoke and its influence on students' decisions to abandon their studies because they feel "frustrated, lose confidence and give up (Land, Cousin et al. 2006, p.196).

A great number of the contributions to the threshold concept literature have been concerned with the "ontological obstacles" and the associated ontological, or affective, aspects and few have been in connection with the "epistemological obstacles" and their associated cognitive aspects.

In that context, the diagram is a useful illustration of the relationships between the threshold concept features. In addition, it serves to highlight their individual roles in and contributions to conceptual development.

Viewed through an ontological lens the diagram may be helpful in elucidating the evolution of the transformative effect that takes place in the course of the journey and the significance the terminal, post-liminal state has for "the learner's sense of identity" and the "social re-positioning" that may accompany the acquisition of a threshold concept. If one's principal focus is on the ontological aspects then the diagram is a useful and informative schematic of the process.

However, such a focus is largely concerned with the result, or that part of the diagram that represents only one-third of the process. In fact, with such a focus the operational part of the process, the part where the actual work takes place, the part where the epistemic shift is substantiated, the part that causes the transformation to happen, is largely ignored.

Viewed through a cognitive lens the diagram reveals that integration is at the heart of the process, establishing its status as the engine of conceptual change. If we were to undertake the task of setting down a precedence list for the features of threshold concepts then integration would have to be given primacy. If there is no integration then the whole project is doomed.

As the diagram highlights, integration is motivated by an encounter with troublesome knowledge. Although it has an important role troublesome knowledge is not on a par with integration. For that reason it takes second place on our precedence list. This is followed by transformation and finally irreversibility. These two features play important supporting roles but they are associated with outcomes and their dependency on the first two ranks them in the second half of the precedence list.

While the affective aspects of the features of threshold concepts have been given extensive coverage in the threshold concept literature very little consideration has been given to the cognitive aspects. Here that approach is reversed and the cognitive aspects are considered.

The sequence in which the features are presented is not arbitrary. We begin with the primary feature of integration and then consider troublesome knowledge. That combination provides us with a description of the conceptual process associated with the acquisition of a threshold concept. We then consider the outcomes of that process by examining the features identified as transformative and irreversible.

3.2 Integration

Meyer and Land identified integration as a feature of threshold concepts because they claim that threshold concepts expose the "previously hidden interrelatedness of something" (Meyer and Land 2006, p.7). They subsequently categorised integration as a "reconstitutive" feature because identifying the interrelatedness that was previously unnoticed alters the learners conceptual structure of the body of knowledge associated with a particular discipline (Land, Meyer et al. 2010, p.xi).

The inclusion of integration as a feature should not come as a surprise. In Chapter 2 our discussion of concepts highlighted how all concepts are based on purposeful abstraction which seeks to discover the inherent structure of the world or impose a useful structure on it. Identifying the inherent relatedness of distinct entities, or imposing relatedness on them, allows the cognitive system to build "knowledge" capable of supporting cognitive activities such as recognition, inference, explanation, reasoning, learning and communication (Solomon, Medin et al. 1999). In the specific contexts of educational environments and learning Jerome Bruner has noted "Grasping the structure of a subject is understanding it in a way that permits many other things to be related to it meaningfully. To learn structure, in short, is to learn how things are related" (Bruner 1969, p.7).

Integration is a feature of all concepts so it is, in fact, the absence of integration as a feature of threshold concepts that would have been surprising. What distinguishes threshold concepts from other concepts is that they are associated with the integration of concepts that were previously viewed as unrelated because the connections were "hidden." The French mathematician Poincaré described as "facts worthy of being studied" those things which "reveal to us unsuspected kinships between other facts well known but wrongly believed to be strangers to one another" (Poincaré, cited in (Mednick 1962, p.221)).

Polya and Szegő use the term "concentration" to describe the outcome of "The unification of concepts which in the usual view appear to lie far removed from each other." Concentration is the result of "condensing a large amount of nutritive material into an essence" (Pólya and Szegő 1972, p.viii).

Threshold concepts encapsulate the type of facts that Poincaré was alluding to and the resulting concentration that Polya and Szegő describe because the acquisition of a threshold concept exposes the previously hidden relatedness and allows new connections to be uncovered, thereby establishing the essence of the material.

3.2.1 Grand Linkages

Revealing unsuspected kinships is what sustains the exalted position of threshold concepts over other types of concepts (e.g. key, core or fundamental) and is the source of the "qualitatively different view of the subject matter" attributed to their acquisition (Meyer and Land 2006, p.6). As the following examples show, in the history of scientific discovery significant conceptual connections have often been unearthed, not because of the possession of superior knowledge, but because of an integration of ideas that heretofore had not been considered related. The examples are not offered as instances of threshold concepts, although someone with expertise in the disciplines concerned may find it possible to make an argument to support such a view. The purpose for their inclusion is to use known and documented instances to highlight the significance of conceptual integration.

The Russian alchemist Mendeleev is credited with identifying the connections between the "elements" of the periodic table. His role as an educator was pivotal to his discovery that the elements had associations and clustered in a structured and logical fashion. Whilst in the process of writing a text book Mendeleev decided to stay indoors one cold winter's day to contemplate how best to present the elements in the text. In what is arguably the first example of knowledge elicitation through card-sorting the story goes that he had, on a series of cards, written the names, atomic weights and some other properties of the sixty-five elements known at that time. He began to arrange the cards in various ways when he saw that there was an underlying pattern and also that there were identifiable gaps in the pattern. He made predications about the properties of the elements that could fill the gaps and when they were discovered years later they were just as he had suggested (Tallack 2001, p.196). Mendeleev's

identification of the previously hidden connections between the elements irrevocably altered how chemists and scientists viewed chemistry and science.

Likewise, Bodanis records the fact that for centuries energy and mass stood alone and were considered separate things. He describes how when in 1905 Einstein published $e=mc^2$ "no one else was putting them together this way - hardly anyone even had a hint that one could try" (Bodanis 2001, p.73) He documents how Lavoisier and Faraday, despite their great work, had seen what he describes as "only part of the truth." Bodanis provides a history of the equation and notes that when it was first published it was almost entirely ignored because it did not fit in with what most other scientists were doing. He cites the renowned chemist Chaim Weizmann who after a long Atlantic crossing with Einstein in 1921 stated "Einstein explained his theory to me every day and soon I was fully convinced that he understood it." (Bodanis 2001, p.ix-x). Einstein identified the previously hidden connection between energy and mass and, literally, changed the world.

Stigler has described as the "most important event in seventeenth-century medicine" the discovery by William Harvey that blood flow was controlled by the heart. Harvey analysed material published in sources going back as far as Aristotle and, coupled with some of his own work, showed that the prevailing theory of how blood behaved in the body was totally unsustainable. Harvey used purely mathematical techniques to establish the relationship between the quantity of blood and the pumping capabilities of the heart. Stigler notes that Harvey didn't just claim that blood flow was controlled by the heart, he "established this fact unequivocally." Harvey's argument was so compelling that it inspired a "school of medical theory" and the application of experimental and scientific reasoning in medicine was "radically changed forever" (Stigler 2000, p.207).

Hartmanis recounts the story of Rutherford's discovery of the "planetary model of the atom" after an experiment showed scattered alpha particles coming backward as they passed through thin sheets of metal. Hartmanis notes that the experiment showed that the old concept of matter was wrong and forced Rutherford to consider the question "What could stop a particle rushing through space at a speed of some 10,000 miles a second?" Rutherford claimed "It was quite the most incredible event that ever happened to me in my life." The question led Rutherford to a fundamentally new model of the atom which he described using the motion of the planets (Traub 1981, p.357).

We could summarise these examples with the observation that Mendeleev did no more than notice a pattern; that Einstein merely identified the relationship between mass and energy; that Harvey simply applied the scientific method; or that Rutherford just reflected on what he had observed. That might be interpreted as an attempt to trivialise these great insights. That is not the purpose. Instead, it should be viewed as a reminder that the importance of integration, of identifying a previously obscure but pivotal conceptual association, cannot be measured by its apparent or potential simplicity.

3.2.2 The Scholarship of Integration

In his review of academic practice and what he called "the priorities of the professoriate" Ernest Boyer identified four key categories of scholarship that he felt were fundamental to the work of an academic. He listed them as (1) discovery, (2) integration, (3) application and (4) teaching (Boyer 1990, p.18-20).

He characterised the scholarship of discovery and the scholarship of integration on the basis of the types of questions that are posed. Those with a focus on discovery ask "What is to be known, what is yet to be found?" and those with a focus on integration ask "What do the findings *mean*? Is it possible to interpret what's been discovered in ways that provide a larger, more comprehensive understanding?"

Those characterisations may be evident from the examples of the previous section. Mendeleev was not trying to find new elements, he just wanted to structure his presentation of the existing ones and so he sought a "larger, more comprehensive understanding." Significantly, that understanding allowed him to postulate the existence of as yet unknown elements and to provide an initial hypothesis and narrower research focus for would-be investigators. Harvey's analysis of what was known of blood flow could not be assigned a coherent meaning and he was prompted to identify one that could. Resolving the mathematical relationships between the documented physiological values provided an understanding that could give those values the required coherence. Rutherford already had a model of the atom but the unexpected experimental results forced him to consider what they meant for that model. They meant the model was incorrect but establishing that truth provided an understanding that led to the discovery of a better model.

Under the heading "the scholarship of integration" Boyer identified the work undertaken by scholars to "give meaning to isolated facts, putting them in perspective." Boyer claims that in the absence of integration scholars risk an attachment to pedantry. In contrast, the pursuit of integration can change what a scholar has discovered from being "information" to "knowledge" and ultimately "even wisdom" (Boyer 1990, p.18-20).

The mathematician Keith Devlin describes things that connect and bind together as "more important, more valuable, and more beautiful" than things that do not. Devlin highlights the mathematical elegance that can be captured when "surprisingly intimate connections" can be established between "seemingly different concepts." In support of his claim he cites Euler's Identity equation, $e^{i\pi} + 1 = 0$, which he describes as the mathematical equivalent of the Mona Lisa painting or Michaelangelo's statue of David.

"Five different numbers, with different origins, built on very different mental conceptions, invented to address very different issues. And yet all come together in one glorious, intricate equation, each playing with perfect pitch to blend and bind together to form a single whole that is far greater than any of the parts. A perfect mathematical composition" (Devlin 2004)

Aristotle once observed that "Teaching is the highest form of understanding" and such a view inextricably links teachers to the scholarship of integration. Indeed, Van Doren has argued that the type of work involved in the pursuit of integration and the identification of connections between concepts is primarily the responsibility of educators because, he explains,

"The connectedness of things is what the educator contemplates to the limit of his capacity. No human capacity is great enough to permit a vision of the world as simple, but if the educator does not aim at the vision no one else will, and the consequences are dire when no one does. (Mark Van Doren cited in (Boyer 1990, p.19)).

Educators have shown themselves quite willing to take on this task. As noted in Chapter 1 the community of threshold concept researchers is widespread and active. An educator's pedagogic content knowledge is built on a foundation of knowing more than just the individual concepts of a discipline. In addition, educators are required to achieve and sustain the status of "scientific Intellectual" (Gal-Ezer and Harel 1998). The centrality of integration to the investigation of threshold concepts firmly establishes them as a new strand in the scholarship of integration.

3.2.3 Integration and Liminality

Integration does not adhere to a nice, orderly sequence of conceptual connections that in sum reveal the hidden relatedness associated with a particular threshold concept. Indeed, Bruner has observed that the idea of learning in sequence "is a fiction" (Bruner 1962, p.2).

(Zuckerman and Lederberg 1986) have described how this can be true for whole disciplines and even the entire scientific community. They identify what they describe as episodes of premature and postmature discovery that introduce "temporal discontinuity" into what is the "broadly incremental" development of scientific knowledge.

A *premature discovery* is one that was not attended to when it occurred either because it was "passively neglected " or "actively resisted" by members working in the discipline or scientific community generally. In these cases the community did not or could not appreciate the utility of the discovery and proceeded in the absence of any sense of significance attaching to the new information. In essence, the import of the discovery was hidden from the community. With hindsight, premature discoveries are often categorised as being "ahead of their time."

In contrast a *postmature discovery* "must be judged to have been understandable, capable of being expressed in terms comprehensible to working scientists at the time, and its implications must have been capable of having been appreciated" (Zuckerman and Lederberg 1986, p.629) p629). Postmature discoveries often answer questions that were missed by scientists because prevailing "assumptions, beliefs and images" may have impeded the community's perception of the profitable lines of inquiry. Thus the community should have been able to identify the importance of the information but was unable to do so and only later realised or comprehended the implications (Zuckerman and Lederberg 1986).

Individual learners attempting to acquire a threshold concept can experience their own episodes of premature discovery which prolong their stay in the liminal space. In their efforts to uncover the hidden connections required to cross the threshold they may ignore important concepts already known or they may be impeded by their perception of the benefit of these concepts because they cannot make the connections. After they have acquired the threshold concept the exposure to the hidden connections allows them to appreciate the significance of concepts they knew but didn't exploit. This type of postmature discovery is part of the transformation experienced by the learner after threshold concept acquisition.

Land et al. provide a useful summary of these learner experiences in the liminal space or state which are not "predicated on linear or staged notions of intellectual development." They describe how the learner will experience the need to attempt "different takes on the conceptual material" involving "digression and revisiting (recursion) and possible further points of departure and revised direction (Land, Cousin et al. 2006, p.202).

The pivotal role integration plays in the consolidation of knowledge is eloquently described using an analogy provided by Polya and Szegő.

"There is an analogy between the task of constructing a well-integrated body of knowledge from acquaintance with isolated truths and the building of a wall out of unhewn stones. One must turn each new insight and each new stone over and over, view it from all sides, attempt to join it on to the edifice at all possible points, until the new finds its suitable place in the already established, in such a way that the areas of contact will be as large as possible and the gaps as small as possible, until the whole forms one firm structure" (Pólya and Szegő 1972, p.vii)

As the diagram above shows, liminality and the crossing of the threshold are dominated by and dependent on integration because they involve the learner in turning concepts over and over, considering them from all angles and attempting, removing and retrying to join them to the existing conceptual structure to make it firmer. This involves the learner in conceptual activities that appear at times to move one step forward and two steps back. This type of activity terminates when threshold concept is acquired and the areas of contact are as large as possible and the gaps are as small as possible.

3.2.4 Integration - A Summary

The liminal experience of being betwixt and between ends when integration is achieved. Liminality and understanding are the two sides of the metaphoric threshold and integration *is* the threshold.

Integration can be difficult but it does not of necessity involve difficult concepts. The difficulties associated with integration are manifest in our failed attempts at making connections between what initially appear to be unrelated concepts. Threshold concepts enable us to make the connections and give new meaning to our existing knowledge. The threshold concept itself may be simple but the effects of making the connection can be profound.

The difficulties are associated with troublesome knowledge. We now consider the types of trouble that can cause these difficulties.

3.3 Troublesome Knowledge

Troublesome knowledge has been cast as an "instigative" feature because it triggers or instigates the whole threshold concept "journey" and marks the beginning of the learner's efforts to deal with the troublesomeness. Despite its centrality to the experience and identification of threshold concepts it is the feature that attracts the least amount of exploration in the literature and, conversely, appears to invite the most restrictive interpretation of all the features. This may be because Perkins, and subsequently Meyer and Land, provided reasonably comprehensive descriptions of instances of troublesome knowledge as part of the initial reports on the aetiology of threshold concepts ((Perkins 2006); (Meyer and Land 2006)).

In those reports several forms of troublesome knowledge were identified. The list included knowledge characterised as *inert*; *ritual*; *tacit*; *counter-intuitive*; *alien*; *conceptually difficult*; as well as *troublesome language*. Despite the multitude of entries in this list only one, *conceptually difficult knowledge*, has tended to appear in the literature and troublesome has become a euphemism for conceptually difficult. For example, Schwartzman talks of "deeply challenging knowledge"; Stokes et al. refer to threshold concepts as being "entwined with entrenched or difficult-to-change misconceptions"; ((Schwartzman 2010); Stokes et al., 2007).

Other forms of defined troublesomeness (e.g. ritual, inert, tacit) share the characteristic that they are associated with knowledge that has already been acquired - the learner possesses the knowledge but fails to retrieve it appropriately. For example, learners often use existing knowledge in a "misplaced" fashion (i.e. using knowledge where it does not belong) or in a "conglomerated" fashion (i.e. several disparate pieces of knowledge are just jammed together erroneously). In these contexts the trouble is associated with deployment and not acquisition (Perkins and Martin 1986, p.215).

This type of trouble is far more prevalent than the trouble associated with an inability to comprehend or acquire knowledge in the first place. Despite this consideration and examination of these forms of troublesome knowledge is noticeably absent from the threshold concept literature.

In what follows the different forms of troublesome knowledge are described and examples drawn from computing scenarios are used to situate the exposition in the discipline of computing.

3.3.1 Ritual Knowledge

Perkins describes ritual knowledge as "routine and rather meaningless in character. It feels like part of a social or individual ritual: how we answer when asked such-and-such, the routine that we execute to get a particular result...such as the notorious *invert and multiply* to divide fractions" (Perkins 2006, p.37).

3.3.1.1 Control Variables in Counting Loops

Even a cursory glance at a handful of programming texts or a small sample of publicly accessible source programs will confirm the ritualised nature of the naming schemes used by programmers for the control variables in loops or the indexing variables associated with collections. The use of variable names drawn from the subset of the alphabetic characters I to N is so pervasive that an encounter with a program that did not adhere to the convention would attract attention and invite some investigation. The letters I and N are the first two in the word integer. The ritual was inherited from mathematics in combination with the historical coincidence that FORTRAN, the first high-level programming language, was named and aimed at formula translation. In early versions of FORTRAN any variable name that started with an alphabetic character in the range I to N was, by default, considered an integer data type unless explicitly stated otherwise. Having been established in the conventions of the first programming languages generations of programming tutors have passed-on the ritual as a side-effect of their teaching and examples.

For some novice programmers the ritual can lead to problems associated with the concept of variable "scope." Consider the following examples of loop nesting which for convenience we will describe as *direct* and *indirect*.

In the Direct Nesting scenario the novice failed to comprehend the implications of using the same control variable for two separate activities. On foot of the encounter the novice learns to use different variables for the different activities. However, when the novice subsequently encounters the Indirect Nesting scenario they intuitively apply the direct nesting strategy to resolve the "perceived" potentially ambiguous reference to the variable *i*. Their success in

solving a non-existent but perceived problem leads to a sense of understanding. However, as the number of methods deployed increases, and every one containing a loop uses the variable *i*, the novice now faces what appears to be an insurmountable problem. How will the code they write correctly interact with code written by others? How will they know which variables to avoid and which to use? Their deployment of the ritual knowledge leads them into trouble despite the fact that they know and understand what role the variable has in the loop and how the loop works.

Direct Nesting	Indirect Nesting
<pre> for(i=0...) { //Outer loop code for(i=0...) { //Inner loop code } } </pre>	<pre> public void methodA() { for(i=0...) { // 'Perceived' Outer loop code methodB(); // invoke methodB } } public void methodB() { for(i=0...) { // 'Perceived' Inner loop code } } </pre>

3.3.1.2 Java Method Header for *main*

Another example of ritual knowledge involves the Java method header for a program's *main* method. The header has the style

```
public static void main(String[] args)
```

The single line encapsulates the concepts of method use, access modifiers, return types, method names, parameters, and collections. It is so laden with conceptual nuance that it is understandable that tutors would adopt an "I'll-tell-you-later" approach. Indeed novice programmers are frequently invited to avoid requesting an explanation and to just "do it like that" (Hong, 1998).

Most tutors, and subsequently most novices, use this construct ritualistically, even in circumstances where they will not be handling any parameters supplied to the program at runtime through the string array called *args*.

Problems arise from the distinguished nature of the *main* method in that it is allowed to breach rules that other methods must adhere to. In every method header except *main*, the inclusion of

one or more parameter declarations establishes an obligation on the user of the method to pass the specified information at the point of use. The compiler will enforce the requirement and flag as an error any use of the method which fails to supply the required data. This is inconsistent with the novices experience of the ritual associated with *main* and may lead to them to replicate the strategy of always having some piece of data passed to a method even if it is not required.

3.3.2 Inert Knowledge

Perkins describes inert knowledge as "knowledge that sits in the mind's attic, dusted off only when specifically called for by a quiz or a direct prompt...Unfortunately, considerable knowledge that needs active use proves to be inert" (Perkins 2006, p.37).

3.3.2.1 Pointers and References

Mobile phone users know that they can interact with other phone owners as long as they have the phone number of the person they want to interact with. The person can be anywhere in the world and it is still possible to share information with them by text message, or to talk to them. Simply dialling the number causes the resources of the network provider the caller is affiliated with to locate the phone and make a connection. Typically the caller has absolutely no knowledge of how the mechanisms used to establish the connection work. Similarly, having sent a text to a particular phone number the sender may receive a reply within seconds. Of course, neither the sender nor the receiver of the text needs to have any idea where the other person is, and they rely on precisely the same strategy of using just a phone number to participate in the interaction.

The notion of a single piece of information that can provide access to something of interest without full knowledge of exactly where it is located is a very powerful tool. Humans use it all the time. Once I tune my radio to the appropriate frequency for a particular station I can receive the output of the station without any knowledge of where it is located (e.g. it could be on a boat in the North Sea) and how the broadcasting is handled. The same is true of television. Of course, once I have tuned in I can only receive the output of that particular station or channel.

All of the students in an introductory programming class will have used a web-browser and will have entered a URL to 'connect' to a particular web-page and peruse the material it provides.

Typically they have absolutely no idea of where the page is physically stored. Access to the page allows them to engage in the activities that the page supports. The page may facilitate the purchase of an airline ticket or a book but maybe not a coat. Another page may facilitate the purchase of a coat but not a musical track or song. Yet another page may handle song purchase. World wide web users are aware of the fact that a URL 'brings' them to a particular page, wherever it is, and the page then provides a collection of facilities that can be manipulated in the style supported by the operations provided "on" the page, but in no other way.

The beauty of this arrangement is that it is only necessary to remember, or make a note of, a single piece of information - the URL or the phone number or the radio station frequency or the name of the TV channel. That single piece of information provides access to the resources available. The resources provided may be minimalist or very sophisticated and the manipulations possible may be simple or elaborate. None of those details are encoded in the piece of information that provides access. But, once access is established, all of the resources are available.

Absolutely none of these scenarios need to be explained to a typical class of novice programmers. Indeed, referring to them in any specific or detailed way can be perceived as boring because they believe it to be so obvious.

Programming systems use pointers, or references, in the very same way as novice programmers use mobile phone numbers, URLs, radio station frequencies and so on. Dynamic memory allocation and the provision of a single value access mechanism is an exact replica of the process used to "create" a mobile phone (i.e. somewhere in the world a phone is registered with a particular number and can be accessed using only that number). We don't care too much about how it happens and we just retain the result of the process (i.e. the number). Using a pointer or reference to access an object and utilising the methods it provides is precisely the same activity as using a URL to access a page created somewhere in the world and utilising the facilities provided by that page.

Despite this the concept of "pointers" is persistently reported as one of the most difficult for novices to learn ((Milne and Rowe 2002); (Goldman, Gross et al. 2008); (Lahtinen and Ahoniemi 2005)). Milne and Rowe found that concepts that rely on "a clear understanding of pointers" were reported as the "most difficult to cope with" by students and tutors (Milne and Rowe

2002). Pointers are troublesome because the novices' knowledge of equivalent mechanisms remains inert and they fail to deploy that knowledge in the new context.

3.3.3 Counter-intuitive Knowledge

Meyer and Land describe counter-intuitive knowledge as knowledge that is "intellectually absurd at face value" (Meyer and Land 2003).

3.3.3.1 Sequential Search

Sequential search is one of the classic algorithms that novice programmers encounter relatively early in their programming course. Like all search algorithms the solution has to traverse a list (i.e. the space we want to search or what we might simply refer to as the "search space") and test for two things (1) has the list been exhausted (i.e. we didn't find the key in the list) and (2) is the current list item the value we are looking for (i.e. we found the key in the list). In a language like Java this is typically coded as something like

```
for(i = 0; i < list.length && list[i] != key; i++) ;
if(i < list.length) {
    // found
} else {
    // not found
}
```

However, experienced programmers know that the performance of this apparently simple algorithm can be improved dramatically using a very simple strategy. By introducing one additional element into the list (i.e. by making the list size one element bigger than is required or necessary) we can eliminate one of the tests in the for loop. Since it is the for loop that does all of the work the change has a significant effect on the performance because half of the tests are removed.

With the additional list element the search strategy is modified slightly. We identify the first or the last element as the "additional" element, sometimes referred to as the sentinel, and all of the other elements are just as they were before (i.e. they represent the search space). We start by putting the key value we are looking for into the sentinel position and then we search from the other end of the list towards the sentinel. The interesting thing is that we have to find the key because we have put it in the list at the sentinel position. As a consequence we can write the code using just one test in the loop as follows

```
//put key in sentinel (last) position
```

```
list[list.length - 1] = key ;
// find key - it has to be there!
for(i = 0; list[i] != key; i++) ;
// check where it was found
if(i < list.length - 1) {
    // found - we DIDN'T find it in the sentinel position
} else {
    // not found - we DID find it in the sentinel
}
```

This simple algorithm is a major source of difficulty for most novice programmers. It seems absurd to put something into a list and then to presume to search for it with the possibility that it may not be found! How could it not be found? How could it not be in the list? The idea is profoundly counter-intuitive and troublesome.

The dilemma is eventually resolved when the novice can appreciate that the list is in fact made up of two things - the list, or search space, and the sentinel position, or additional element. When it is possible to think of these things as separate entities co-existing in a single structure (i.e. the list) one can appreciate that reaching the sentinel position means that the search space was completely traversed so the key value could not have been present (i.e. the key is not in the list). Similarly, if the traversal stops before reaching the sentinel then the key value must have been in the search space (i.e. the key is in the list).

The counter-intuitive nature of the search strategy can make it very difficult for a novice programmer to arrive at this understanding.

3.3.4 Foreign or Alien Knowledge

Perkins describes foreign or alien knowledge as knowledge that "comes from a perspective that conflicts with our own" and that many situations "allow multiple serious, sincere, and well-elaborated perspectives that deserve understanding" (Perkins 2006, p.39).

3.3.4.1 Number Systems and Representation

Our everyday experiences with number systems bring us in constant contact with a wide variety of counting strategies and increments. On their own the counting and grouping of time and calendar elements are a rich source of variance. For very short time periods of less than a second we count in powers of ten. Thus we have thousandth, hundredth and tenth of a second.

Once in seconds we count in sixties for minutes and continue with that unit into the hours. Twenty four hours represent one day but we optionally partition it into either two twelve's or a single twenty-four hour block. Moving on to days we can group them in sevens for weeks or any one of twenty eight, twenty nine, thirty or thirty one for months. Months are counted in twelve's to form years and we then switch back to powers of ten for decades, centuries and millennia. As an exception, a month can only have twenty nine days if the year is a century year and divisible by 400 or not a century year and divisible by 4.

Broadening our consideration to units for measurement, weight, and volume would introduce classifications based on inches, feet, yards, furlongs, chains, miles, ounces, pounds, stones, tonnes, pints, quarts, gallons and many more.

Even though they are perfectly logical and systematic, number systems have proven to be an anathema for many students of computing science. By comparison with the much more elaborate undertaking associated with the number systems described above the decimal number system simply involves the uniform shifting of powers of ten from units to tens to hundreds to thousands and so on. Other number systems have the same consistent shifting of powers but use a different base. Whilst many students grasp the powers of ten idea changing to powers of two or eight or sixteen is often marked by anxiety and disinterest among students. For many the representation of 126_{10} as 01111110_2 or 176_8 or $7E_{16}$ may already be viewed as foreign or alien, but tolerable. However, the introduction of two's complement arithmetic simply compounds the difficulties.

For the confident students the exposition of two's complement is a useful introduction to the importance of representation and the benefits that can accrue from the use of an appropriate data representation. For others it is no help at all to argue that the representation simplifies the design of the electronic circuits, increases the precision of the numbers that can be handled and ensures that zero has a single representation. The suggestion that negating a number and adding one to it is the most common method of representing signed integers in an electronic device truly is foreign or alien and many students balk at it.

3.3.5 Conceptually Difficult Knowledge

Perkins notes that an essential characteristic of serious study in third level education is its ability to create encounters with conceptually difficult knowledge that is troublesome (Perkins

2006, p.38). Meyer and Land have noted "When knowledge ceases to be troublesome, when students sail through the years of a degree programme without experiencing conceptual difficulty, then it is likely that something valuable has been lost" (Meyer and Land 2006, p.xiv)p.xiv). In this context troublesomeness can evoke feelings of frustration, self-doubt and anxiety that can encourage students to abandon their studies.

Barnett argues that students are

"perforce required to venture into new places, strange places, anxiety-provoking places. This is part of the point of higher education. If there was no anxiety, it is difficult to believe that we could be in the presence of a higher education" (Barnett 2007, p.147)

Whilst acknowledging that a "transformative effect" would be at least associated with something troubling, and "probably *should* be troublesome", Meyer and Land present threshold concepts as a means of developing ways of helping tutors and students "not to avoid the troublesomeness, but to feel more confident in coping with it, resolving it and moving on in confidence" (Meyer and Land 2006 notes at start of book).

A catalogue of computing concepts that are considered conceptually difficult would likely include paradigm transition (e.g. procedural to object-oriented); recursion and induction; pointers and references; proofs of correctness and invariant specification; the distinction between classes, objects, and instances; and polymorphism. But regardless of what concepts are included in such a list they are, by their very membership, inherently troublesome for students.

3.3.6 Tacit Knowledge

The characterisation of tacit knowledge as troublesome is due to Meyer and Land. They describe tacit knowledge as "that which remains mainly personal or implicit at the level of 'practical consciousness' through its emergent but unexamined understandings are often shared within a community of practice" (Meyer and Land 2006, p.12).

3.3.6.1 Face Recognition

Polyani noted that there are many things that we know that we cannot put into words. For example, he describes how we are able to recognise a person's face even when it is "among a thousand, indeed among a million" but we "cannot tell how we recognize a face we know"

(Polanyi 1983, p.4). He notes that not only can we recognise a face but we can often tell from the way the features are aligned or contorted or smoothed what mood the person is in. And yet we are unable to tell "by what signs we know it."

3.3.6.2 Specifying Algorithms

Knuth has noted "It is often said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e., express it as an algorithm" (Knuth 1974). Even expressing simple algorithms can prove difficult because we are unable to express the tacit assumptions we depend on when we are solving the problem without the aid of a computer.

For example, Shackelford and Badre replicated (in slightly modified form) an experiment originally used in 1982 by Soloway and others ((Shackelford and Badre 1993); (Soloway, Bonar et al. 1983)). In the original study novice programmers were asked to solve three simple problems. All three were averaging problems, of the type "generally taught in 5th grade...to 11 year old children," but the characteristics of the data varied as follows

Problem 1. Write a program which reads 10 integers and then prints out the average.

Problem 2. Write a program which repeatedly reads in integers until their sum is greater than 100. The program should then print the average of the numbers.

Problem 3. Write a program which repeatedly reads in integers until it reads the integer 99999. The program should then print the average (not including the 99999).

In both the original and replicated experiments more than half of the participants were unable to solve the problems but the performance in the replicated version did show improvements which were attributed to the intervention developed by Shackelford and Badre. What is significant about the results from both versions is that the participants' difficulties had nothing to do with their ability to understand the problem, structure a solution or write syntactically correct code. Their Achilles Heel was their inability to control the repetition so that the correct number of values was acquired from the input. Thus, they failed to "teach a computer" a solution to a problem generally taught to 11 year old children because they were unable to describe the component of the algorithm that leveraged their tacit knowledge about how and when to stop acquiring values for the computation.

This type of outcome is not uncommon. Many people are unable to explain how they know whether a number is odd or even. If asked about a specific number, such as 9, they will provide the correct answer. However, when asked to state how they would determine if some value N is odd or even they are unable to do so. In more sophisticated cases they can't even begin to explain how they go about solving a problem. For example, if I write a short piece of text on a board and ask a class of novice programmers to identify the longest and shortest words they can do it without any difficulty. But very often they are incapable of describing how they do it.

3.3.7 Troublesome Language

Meyer and Land identify language as a source of trouble because "Specific discourses have developed within disciplines to represent (and simultaneously privilege) particular understandings and ways of seeing and thinking." In addition "The discursive practices of a given community may render previously 'familiar' concepts strange and subsequently conceptually difficult" (Meyer and Land 2003).

3.3.7.1 Parameter Passing

The term "parameter passing" is used by programmers to identify the act of data sharing between components. The problem is that the terminology is misleading because it is grounded in the language of the system software (i.e. compiler) and not the programmer.

Parameter passing, the act of causing the parameter data to be passed from one program component to another, is a task facing a compiler and as such is of very little interest to a programmer.

A programmer is only interested in how the receiving component can affect the data being shared with it - what it can do, or needs to be able to do, with the data. This is a really important task for the programmer because how the component manipulates the data will have consequences for subsequent operations.

Typical pedagogy of parameter passing ignores this distinction and focuses on the compiler perspective.

Labelling the process parameter passing completely distorts its focus. As a consequence parameter passing causes many difficulties for novice programmers.

Note: This topic is developed in significantly greater detail in *Chapter 5 : State as a Threshold Concept*.

3.3.8 Troublesome Knowledge - A Summary

The trouble in "troublesome knowledge" is more often associated with how we deploy our existing knowledge than with our inability to acquire new knowledge. Narrowly interpreting "troublesome" to mean "conceptually difficult" limits our consideration of a broader range of more mundane, and more frequently occurring, troubles. Knowledge that is inert, ritual, tacit, counter-intuitive or alien, and how we use language, are far more likely to be the troubles that inhibit our integration of concepts than our difficulty with the acquisition of a concept that stretches our cognitive abilities. In many respects it is the simplicity and subtlety of these troubles that makes them an elusive, almost invisible, cabal of conspirators. In contrast, it is immediately obvious to us when we face a concept that we find difficult to comprehend and such failures are more finely etched in our minds. This may explain why troublesome knowledge has been usurped to mean conceptually difficult in the threshold concept literature.

3.4 Transformation

Transformation is the threshold concept feature that has attracted the most attention amongst the threshold concept community. It has been labelled a "consequential" feature of threshold concepts because it is concerned with the consequences or outcomes of acquisition and describes how the "effect on student learning and behaviour is to occasion a significant shift in the perception of a subject, or part thereof" (Meyer and Land 2006).

Transformation is often viewed and portrayed as a type of "Ah-Ha" event. However, as described earlier, integration, the source of the transformation, is not a linear or sequential process and the evolution of a particular understanding of a concept does not always culminate in a single, identifiable point of "knowing." Indeed, Pioncaré expressed the view that "what appears to be a sudden illumination is in fact the result of a previous long subconscious work" (Ruelle 1998).

Gould makes the same point when he writes about our "intellectual history" and describes how it assembles and explains "disparate and unrelated information." He notes that it is not because "Galileo trained his telescope on the moons of Jupiter or because Darwin took a ride on

a Galápagos tortoise" that we know the planets are configured in a heliocentric fashion or that similarities among organisms are the consequences of evolutionary effects. "Instead, little facts are assimilated into large theories. They may reside there uncomfortably, bothering the honourable proponents. Large numbers of little facts may eventually combine with other social and intellectual forces to topple a grand theory" (Gould 1993, p.440-1).

This idea is not unlike Per Bak's analogy for complex behaviour. Bak invokes the image of a child dropping individual grains of sand to form a pile. Initially the grains of sand form a flat arrangement having stayed close to where they landed. Slowly a pile forms which becomes steeper as the process continues. Even the addition of a single grain of sand can disturb the existing sand and cause little sand slides to occur in small sections of the pile (local transformation). Over time the extent of the sand slides increases and from time to time they can span almost the whole pile (global transformation) (Bak 1996, p.2).

If we equate each grain of sand with the acquisition of a concept the sand pile can represent a body of knowledge. Initially it is flat as concepts are acquired in the absence of a view that might connect them and they just stay where they landed. Slowly the connections are formed and the body of knowledge begins to rise in the form of a structure. Each concept added may just settle where it landed or it may disturb the existing conceptual structure and cause a transformation. The degree of transformation can vary. Some concepts may just cause local transformations. Others may effect global transformations.

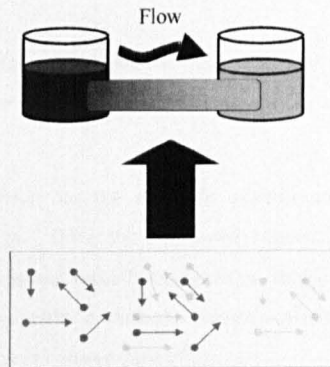
One drawback of the sand pile metaphor of transformation relates to the visual cues it affords. For example, inspection of the sand-pile would allow critical points to be anticipated, not necessarily precisely but at least in advance. Thus, as the steepness of the pile increased one would entertain the prospect of an impending sand slide, possibly with some knowledge of its degree. In addition, there is a sense of discreteness about the process as the grains of sand are added in sequence and appear distinct.

In the nineteenth century the French writer Stendhal used the word *crystallisation* to describe how "the mind discovers new perfections" and the notion of crystallising ideas is part of everyday language (Gellner 2005, p.1). Around the same time the British philosopher John Stuart Mill used the term "mental chemistry" to describe how our cognitive resources worked. Mill used the metaphor of a "chemical combination" and described it as follows

"The laws of the phenomena of the mind are sometimes analogous to mechanical, but sometimes also to chemical laws. When many impressions or ideas are operating in the mind together, there sometimes takes place a process of a similar kind to chemical combination. When impressions have been so often experienced in conjunction, that each of them calls up readily and instantaneously the ideas of the whole group, those ideas sometimes melt and coalesce into one another, and appear not several ideas but one; in the same manner as when the seven prismatic colors are presented to the eye in rapid succession, the sensation produced is that of white. But in this last case it is correct to say that the seven colors when they rapidly follow one another generate white, but not that they actually are white; so it appears to me that the Complex Idea, formed by the blending together of several simpler ones, should, when it really appears simple, (that is when the separate elements are not consciously distinguishable in it) be said to result from, or be generated by, the simple ideas, not to consist of them. . . . These are cases of mental chemistry: in which it is possible to say that the simple ideas generate, rather than that they compose, the complex ones." (cited in (Murphy and Medin 1985, p.296-7)).

More recently, in an educational context, David Perkins has described the collective consequences of many small "knowledge" interactions as "emergent effects" (Perkins 2006, p.36). Thus, because of their affiliations to the chemical sciences, a more appropriate metaphor for transformation might be the type of phenomena described as "emergent processes" (Chi 2005, p.166).

Consider the diagram below, which shows two beakers containing liquid. One contains clear water and the second contains water with dark blue dye mixed into it. Initially the two beakers are separated. Later, the beakers are connected and the liquids in both are allowed to come in contact with each other. The molecules of the liquids interact in unpredictable ways as they "jitter" and "bounce" around creating a "collective (or net) effect." The pattern of interaction is an emergent one and the summative outcome is that "both beakers reach the same color of medium blue" (Chi 2005, p.166).



Source : (Chi 2005, p.167)

In this scenario the visual clues only allow us to talk about the process but not the individual, separable interactions that are taking place. The notion of sequence is vacuous because the interaction of the liquids is, at best, explained on the basis of Brownian motion. We can confirm the transformative outcome of the interactions but it is impossible for us to isolate or pinpoint with certainty when or precisely how the transformation occurred.

Transformation is the result of the emergent process of integration. Whilst it is possible to describe the process and its outcomes at a macro level it is impossible to specify how the process will unfold for a specific learner. In addition, it is impossible to review the process to identify milestones or watersheds with any certitude.

3.4.1 Transformation - A Summary

Having successfully overcome the troubles induced by whichever instance of troublesome knowledge caused them, a learner integrates the disparate knowledge components and acquires a threshold concept. In Stendhalian fashion the learner's knowledge has become crystal clear, transformed by the interaction of the knowledge elements into a different, unified conception. The process is an "emergent" one, characterised by Brownian-like interactions, with the result that the learner's knowledge provides a different perspective, possibly inexplicably so because the process is immune to systematic review.

3.5 Irreversibility

Irreversibility is the change of perspective occasioned by the acquisition of a threshold concept that is unlikely to be forgotten or unlearned without considerable effort (Meyer and Land 2006, p.7).

As described above, integration has the effect of unifying conceptual material into a new conceptual structure or form. Once the improved conception of the material has been established there is no conceptual benefit in returning to the previous, more impoverished view. In fact, it may not be possible to undo the transformation that results from an emergent process, at least not without very considerable effort.

Continuing the chemistry metaphor we note that chemical combinations can form mixtures or compounds. With a mixture the original elements can appear to be totally combined or integrated. However, the constituents of a mixture can usually be separated relatively easily regardless of the apparent degree of integration. For example, water and oil can be mixed and later separated by simply letting them settle.

In contrast, integration that results in a compound creates a completely new substance, independent of the individual substances used to form it. The new substance has unique properties that distinguish it all other substances. Where the constituents of a compound can be separated it is usually not a simple process.

3.5.1 Irreversibility - A Summary

Threshold concepts are distinguished from other concepts because they integrate concepts to form compounds which tend to be irreversible.

3.6 Threshold Concept Features - A Summary

Using a theory of concepts framework the features of a threshold concept as set out by (Meyer and Land 2003) have been examined from a cognitive perspective. That examination has shown integration to be the central feature. The types of troublesome knowledge have been elaborated and situated in a computing context. The subtle nature of many forms of troublesome knowledge has been highlighted to improve their visibility as significant sources of trouble and to counterbalance the understandable prevalence of conceptual difficulty in the

literature. The type of cognitive transformation that results from the acquisition of a threshold concept has been described using the analogy of chemical compounds to emphasise the emergent and irreversible nature of it.

3.7 Critiques of Threshold Concepts

It is tempting to believe that having listed the features of threshold concepts we can simply identify concepts that exhibit those features and classify them as threshold concepts. However, as Meyer cautions, it is "not helpful to ask what the definition of a threshold concept is. Threshold concepts cannot be described as an essentialist, definitive list of characteristics. The classificatory pursuit of threshold concepts in any scientific sense is a pointless one" (Meyer 2010, p.205).

Meyer is right but his comments are not uniquely applicable to threshold concepts and are, in fact, true of every concept. As we have seen in Chapter 2, the definitional or classical view of concepts and concept formation based on feature lists is an historical artefact which has been superseded by the family resemblance and theory theory approaches.

Notwithstanding this, the threshold concept literature includes two critiques that rely exclusively on an argument based on the classical view of concepts. The first, by Darren Rowbottom, was published in 2007 and the second, by Rod O'Donnell was published in 2010 ((Rowbottom 2007); (O'Donnell 2010)). Whilst there is a large amount of overlap between the critiques, O'Donnell has broadened the discussion by posing a series of specific questions which, he believes, undermine the coherence of the features specified for concepts qualifying as threshold concepts. The questions are listed as follows

1. How many characteristics are required?
2. Can probable characteristics be defining characteristics?
3. How important is troublesomeness?
4. Can threshold concepts have all five characteristics simultaneously?
5. How many learners must undergo the specified experiences?
6. How many threshold concepts are there?

3.7.1 Dependence on the Classical View of Concepts

Both Rowbottom and O'Donnell argue that the vagueness of the definition thwarts efforts to provide empirical evidence to support threshold concept identification. Rowbottom believes it

is because "they have been defined in a way that makes it impossible, even in principle, to empirically isolate them". Questioning "even if there are such things" he cites the following extracts from the feature definitions (1) a threshold concept...is likely to be (2) probably irreversible (3) possibly often (though not necessarily always) bounded, and (4) potentially (and possibly inherently) troublesome. O'Donnell describes these descriptions as "extraordinary elasticities." Both Rowbottom and O'Donnell believe they pose serious difficulties to empirical exploration of threshold concepts.

The threshold concept definition and the concerns that accompany it are not unique. Many authors, committees and other groups have sought to classify concepts and provide lists and catalogues of *important, fundamental, core, central, difficult, even great* concepts of computing ((Nievergelt 1980); (Denning 2003); (Zendler and Spannagel 2008); (Goldman, Gross et al. 2008)). It is often assumed that the properties that determine a concept's membership of a particular category are intuitively comprehensible and universally acceptable.

For example, it is sometimes presumed that everyone knows what it means to be a core concept or a fundamental concept, even in the absence of a formal definition (Zendler and Spannagel 2008). Probed for their personal list of, say, fundamental concepts, respondents with a knowledge of a discipline are usually happy to offer one. In a programming context these concept enumerations typically include staples such as programming construct, function, parameter passing, scope, recursion, abstraction and so on (Boustedt, Eckerdal et al. 2007). Thus, even in the absence of a formal list of features for fundamental concepts people are still prepared to enumerate what they consider to be a list of fundamental concepts.

Rowbottom argues that "it is necessary to understand what the essential properties of Xs are, in order to be able to determine if there are any Xs." This obliges us to provide a precise list of properties because "Providing an account only in terms of accidental properties, even likely ones, is not sufficient." In the absence of such a list he believes we are engaging in the pointless exercise of positing inert (or opaque) things (Rowbottom 2007).

O'Donnell states that the "purpose of a theoretical definition is to be definitive rather than conditional" and that, with respect to threshold concepts, the "lack of clear foundations...allows practitioners to do as they please in accumulating instances" (O'Donnell 2010, p.6).

These and similar arguments are unsustainable. Our inability to precisely articulate a concept in language does not inhibit our ability to model and manipulate it cognitively or identify instances of it. It is futile to assume that simply enumerating the essential properties of a concept is a pre-requisite or a guarantee that facilitates the uncontested classification of that concept, whether as a threshold concept or some other classification.

In Chapter 2 we noted that an aversion to vagueness is understandable but it is also unfounded. Many concepts do not have clearly defined boundaries and language is not as precise as we would like it to be. It is quite easy to produce a list of Xs which enjoy widespread acceptance and usage but for which the essential properties are so nebulous and subjective as to render them elusive (e.g. game, bald, tall, green, dread), and also of Xs for which the so-called essential properties are meticulously defined but it is known that no such Xs exist (e.g. imaginary number, point, line).

We tend to ignore the fact that when listing the properties of things we are predisposed to stating them affirmatively. For example, a typical description of a bird would be - has feathers, has wings, can fly, has a beak, lays eggs, lives in a tree, sings. However, a more correct description would be - is likely to have feathers, has wings, probably can fly, has a beak, lays eggs, possibly often (though not necessarily always) lives in a tree, is likely to sing, possibly often (though not necessarily always) can swim. This would be more consistent with our actual experience of birds.

3.7.2 Typicality

Inevitably the perceived imprecision in the threshold concept features invites questions of the type O'Donnell poses. For example, "How many characteristics are required," "Can probable characteristics be defining characteristics," and "Can threshold concepts have all five characteristics simultaneously" (O'Donnell 2010). These questions are misguided, as the eminent metallurgist Robert Pond has ably demonstrated

There's a big group of people who don't know what a metal is. Do you know what we call them? Metallurgists!"

Here's why metallurgists don't know what a metal is. We know that a metal is an element with metallic properties. So we start to enumerate all these properties: electrical conductivity, thermal conductivity, ductility, malleability, strength, high density. Then, you say how many of those properties does an element have to have to classify as a metal? And do you know what? We can't get the metallurgists to

agree. Some say three properties; some say five properties, six properties. We really don't know. So we just proceed along presuming that we are all talking about the same thing. (Pond cited in (Murphy 2002, p.18).

Again, as detailed in Chapter 2, concepts can have graded membership and some may be more representative than others. Robins, chickens, parrots, penguins and ostriches are all birds but robins, are considered more representative of the concept *birds* than the others. In addition, Eleanor Rosch and her colleagues have shown that real-world attributes do not occur independently of each other and some combinations are quite probable whilst others are rare. Creatures with feathers are more likely to have wings than creatures with fur.

Similarly, a typical threshold concept may not possess all of the features prescribed but concepts that are integrative are more likely to transform than concepts that are not. A transformative outcome will probably be irreversible. A typical threshold concept will be associated with troublesome knowledge that leads to conceptual integration.

3.7.3 Threshold Concepts and the Theory Theory of Concepts

As described in Chapter 2, the *theory theory of concepts* captures the fact that concepts are not constructed by the mere observation or identification of specific features. The acquisition of a concept involves the use of prior knowledge to distinguish between the features that are important and those that are not. Applying that knowledge requires the use of inference and the introduction of information not directly available from observation alone. Used in combination these components provide a conceptual structure to sustain the concept.

In their exposition of threshold concepts Meyer and Land describe how they looked at their existing knowledge of learning and education and inferred from it that every discipline had concepts that inhibit learning and impede progression to expertise in the discipline. They also report how their interactions with practitioners in a range of disciplines led them to conclude that troublesome knowledge could be a useful marker for concepts that inhibit progress in a discipline. (Meyer and Land 2006, p.4).

It is clear that the concept of a threshold concept was developed on the basis of the theory theory of concepts, leveraging the experience and expertise of the originators. Arguments seeking to refute the basis for threshold concepts which rely on the classical view of concepts are inherently flawed, not least because the classical view of concepts is no longer viewed as viable.

3.8 Summary

For the first time, here, the concept of a threshold concept has been situated in the more general theory of concepts. The features attributed to a threshold concept have been set out and elaborated using examples drawn from the discipline of computing. The key role played by troublesome knowledge and the centrality of integration have been identified. The status of transformation and irreversibility as consequential features has been established.

There are threshold concepts. The concept of a threshold concept is independent of the existence of any given instance of it. As Wittgenstein pointed out "Something red can be destroyed, but red cannot be destroyed, and that is why the meaning of the word 'red' is independent of the existence of a red thing" (Wittgenstein 1958, Part 1:57). In the same way that we can talk about the concepts of Santa Claus, abstract classes, the tooth-fairy, virtual methods, happiness, long integers, dreads, recursion, birds, complex numbers, beautiful people and the Higgs-Boson particle, we can talk about threshold concepts, explore them, find them useful and even believe that we have experienced them.

In the remaining chapters we will examine how we can identify threshold concepts and provide a description of one that we have identified in the discipline of computing.

Chapter 4

Identifying Threshold Concepts

Contrary to the assertion that the definition of a threshold concept causes problems for the accumulation of empirical evidence to identify threshold concepts we believe the problem is instead associated with the approaches used thus far to gather the empirical data. This chapter documents a review of some of the approaches that have been used to categorise concepts under various headings including fundamental, central, difficult and in particular, threshold concepts. The review identifies some of the difficulties that application of the approaches has unearthed and helps to explain why they have not been as successful as might have been expected. Examination of these difficulties leads to an investigation of the reasons why researchers were unable to accumulate sufficient empirical evidence to identify concepts in various categories, especially threshold concepts. The findings of this review prompted the development of a methodology that sought to maximise the pedagogic expertise of the researcher at the point of learner liminality and troublesomeness, where learners experience the greatest challenges to their acquisition of conceptual structure. Focusing on threshold concepts I propose a methodology that leverages the pedagogic content knowledge of tutors to document the empirical justification of a particular concept as a threshold concept.

4.1 Approaches to the Accumulation of Evidence

A number of attempts at concept classification have been documented in the literature. These attempts have had as their goal the classification of concepts based on empirical evidence as opposed to subjective decisions. In many cases subjectiveness can be irrelevant but in contexts where a particular concept may be threatened with exclusion from a list or there is a dispute about a particular concept's inclusion on a list, empirical support is deemed essential. Typically, the source of the evidence has been discipline experts and would-be discipline experts in the guise of learners.

The necessity for the authoritative view of experts to validate the inclusion of concepts in a classification is based on the belief that experts "know" their discipline and this knowledge provides them with the tools to dissect and analyse the body of knowledge to identify the concepts that are key or fundamental or important or whatever euphemism of significance is deemed appropriate or preferred (Zendler and Spannagel 2008). As a consequence of this view, systematic and empirical investigation based on the ratings of expert groups, including the use of processes such as Delphi (Clayton 1997), is a common strategy for the accumulation of empirical evidence.

A variant of this approach seeks empirical data gathered from partial or would-be experts in the guise of students, often described as apprentices or novices, at various stages of completion of their programmes of study at university or other places of learning. In this scenario the subjects' lack of competence or knowledge gaps are viewed as sources of evidence to support or confirm the assignation of a particular status to a given concept.

4.1.1 Fundamental, Central and Difficult Concepts

(Zendler and Spannagel 2008) undertook a study to identify the central concepts of computing. They chose forty-nine concepts from the ACM Computing Classification System (1998 Version). Only concepts which were mentioned more than 10 times in the classification system were used. The list and a questionnaire was sent to ninety-eight computer science experts at fourteen high ranking research universities in Germany. Using the four dimensions specified in the definition of a fundamental concept provided by (Schwill 1994), the subjects had to rate the forty-nine concepts using a Likert scale rating. Thirty-seven valid questionnaires

were returned (i.e. about 38%). The data were analyzed using clustering techniques and produced the following overall ranking

algorithm • computer • data • problem • information • system • language • program
• test • communication • software • process • model • computation • network •
error • hardware • structure • architecture • application • representation •
method/methodology • interface • reliability • performance • distribution • theory •
parallelism • processing • memory • design • processor • database • logic •
technique • automation • analysis • control • code • graphics • verification •
standard • simulation • device • management • generation • approximation •
equation and statistics

As a second example, consider the usage of the Delphi process to identify concepts that are important and difficult for students to learn (Goldman, Gross et al. 2008). The process was applied to three introductory computing courses, one of which was programming. Honouring the requirements of the Delphi approach a panel of experts was established for each course. To ensure that no emphasis was placed on a specific language, paradigm, or environment the programming panel had twenty participants with a variety of backgrounds and experience. The panellists were categorised as experts because they had taught programming frequently and had published textbooks or pedagogical articles on the topic.

The initial phase of the process invited panel members to nominate ten to fifteen concepts they considered both important and difficult. In the second phase a master list of all the concepts nominated by more than one expert was produced and circulated to the panel members. Each expert then rated the concepts across three dimensions (i.e. importance, difficulty, expected mastery) using a '1 to 10' scale. In the third phase rating averages and some other statistical measures were provided to the panellists and, with the benefit of this information, they were asked to rate the concepts again. In cases where their rating was at odds with the outcomes from the second phase they were required to provide a "convincing justification" for their new rating. A new set of statistical measures was generated and together with an anonymised set of the justifications received in the third phase, was circulated for the final rating phase. From these final ratings a Top-10 list was produced with the following entries

procedure design • conceptualize problems and design solutions •
abstraction/pattern recognition and use • designing tests • debugging and exception
handling • issues of scope (local vs. global) • functional decomposition and
modularization • inheritance • parameter scope and use in design • memory model
and references/pointers

Unlike Zender and Spannagel, Goldman et al. explicitly embraced the requirements of the learner and sought concepts that were both important and difficult for students to learn. The feedback features of the process provided experts with a review of the complete set of ratings at each phase. Where the ratings of an individual expert were perceived as outliers the expert was obliged to convince the others of their reliability.

Significantly, three of the concepts share an affiliation with the concept of *variable* - namely, issues of scope (local v. global), parameter scope and use in design, and memory model and references/pointers - but no recognition of the connection is evident from the reported analysis. The process does not appear to have encouraged any reflection on the resulting concept set and as a consequence the presence of a linking thread among one-third of the concepts appears to have passed unnoticed.

One possible explanation for this is that the concept of *variable* may have been, tacitly, considered too simple by the experts. It is significant that in their conclusions Goldman et al advise that "the difficulty ratings should be taken with a grain of salt" because "teachers have an incomplete (at best) understanding of student learning."

4.1.2 The Threshold Concepts Group

In 2005 a group of computer science education researchers launched in a multi-institutional, multi-national effort to identify threshold concepts in computer science. The group members were drawn from several institutions across Europe and the United States. The group used a variety of tools and strategies in its attempts to provide empirical evidence supporting the identification of one or more threshold concepts. The work has been reported in a series of publications ((Eckerdal, McCartney et al. 2006); (Boustedt, Eckerdal et al. 2007); (Eckerdal, McCartney et al. 2007); (McCartney, Eckerdal et al. 2007); (Moström , Boustedt et al. 2008); (Zander, Boustedt et al. 2008); (Thomas, Boustedt et al. 2010)).

The group's efforts represent the most persistent and varied attempt at threshold concept identification undertaken in the discipline of computing. The project evolved over five phases (Thomas, Boustedt et al. 2010) which the group labelled (1) preliminary investigation (2) threshold/troublesome concepts (3) liminality (4) integration and object orientation (5) transformation and abstraction. Because of the project's distinguished status the processes and outcomes are briefly reviewed here.

The first phase (Boustedt, Eckerdal et al. 2007) involved an informal survey of attendees of two international computer science education conferences (i.e. ITiCSE and Koli 2005). At ITiCSE thirty-six instructors from nine countries were interviewed in an informal, unstructured and conversational style. Subjects were made aware of the five properties of a threshold concept and asked to suggest candidate concepts. Thirty-three concepts were nominated including

functions • parameters • scope • iteration • recursion • abstraction • abstract data type • lists (arrays) • trees • balanced trees • graphs • objects • class • polymorphism • encapsulation • aggregation • cohesion • threads • templates • modelling • algorithm • complexity • computer architecture • programming language grammars • state • addressing memory (pointers, references) • control structures (e.g. if, loops)

A similar exercise was undertaken with conference attendees of Koli. This time the data was gathered more systematically, using interviews and a questionnaire, but the results were similar to those collected at ITiCSE. The group noted that subjects found the concept of threshold concept "compelling: nearly everyone we spoke with was immediately interested" and they tended "to focus on "difficult to learn" more than any other aspects of the concepts they discuss" (Boustedt, Eckerdal et al. 2007, p.505). The group followed up the surveys with a literature search for "likely candidates" which resulted in their identification of *object-orientation* and *abstraction* as potential threshold concepts.

The second phase investigated threshold concepts from the students perspective using semi-structured interviews (Boustedt, Eckerdal et al. 2007). Sixteen subjects from seven institutions across three countries participated in interviews ranging in duration from twenty minutes to over an hour. The subjects were graduating students and it was hoped that their mastery of the discipline would enable them to provide a perspective on the acquisition of the concepts involved. The interview script used in the interviews is shown above. It began by inviting subjects to nominate a concept where they were "stuck at first but then became clearer." This language was intended as a euphemism for troublesome. The interviews were recorded, transcribed and analysed.

A review of the interviews prompted the selection of two concepts that seemed promising, namely, object-oriented programming (OOP) and pointers. The group closely re-examined the interview transcripts for content relating to those concepts and concluded that for both they "found evidence that they satisfy the criteria for Threshold Concepts."

The group interpreted the results from this phase as (1) providing evidence that threshold concepts exist in computer science, and (2) that object-oriented programming and pointers were threshold concepts. In particular, object oriented programming had an integrative affect on other concepts.

1. Could you tell me about something where you were stuck at first but then became clearer? (Subject answers <X>.)

The rest of this session will now focus on <X>.
2. Can I start by asking you to tell me your understanding of <X>?
3. Assume that you were explaining <X> to someone just learning this material, how would you do it?
4. Tell me your thoughts, your reactions, before, during and after the process of dealing with <X>.
5. Can you tell me what helped you understand <X>?
6. Can you describe how you perceived/experienced <X> while you were stuck and how you perceived/experienced it afterwards?
7. Based on your experience, what advice would you give to help other students who might be struggling with <X>?
8. Please tell me what other things you need to understand in order to gain a good understanding of <X>.
9. Can you tell me how your understanding of <X> has affected your understanding of other things?
10. Was your understanding of <X> something that you had to keep reviewing or having learned it once were you OK with it?
11. Describe how and in what context you have used <X> since you learned it?
12. Is there something more you want to tell me about <X>?
13. To finish the interview, can you tell me whether there are any other things where you were stuck at first but then became clearer? I promise I won't ask you about them in detail!

Source: (Bousteadt, Eickerdal et al. 2007)

Phase three kept the focus on students. It was devoted to the investigation of liminality and the group's effort to capture the effects the transition (i.e. from not understanding to understanding a threshold concept) had on students. The group re-examined interview transcriptions from phase two (Eckerdal, McCartney et al. 2007) "looking for quotes related to liminal space." The selected quotes were discussed by the group members in the context of the framework proposed by (Meyer and Land 2005). They describe this type of analysis as a

triangular conversation, that is, an ongoing conversation and negotiation between the researchers, the data, and the liminal space as it is described by Meyer and Land. The questions we asked are inspired by the characteristics of the liminal space, but also by the data, the observed characteristics from the quotes. The answers we found are shaped by the research questions, the data, and our lengthy experiences as teachers in the subject domain.

Reporting the results the group noted that students experience the transition differently and take different routes through it.

In phase four (Sanders, Boustedt et al. 2008) the group returned to the task of accumulating empirical evidence to support their nomination of object oriented programming as a threshold concept. In this phase they used a different research tool and a different type of subject. Seventy-one novice programmers, drawn from six institutions, were given a sample concept-map that included the concepts "kitchen" and "dinner" and a number of meal-related concepts. They were then asked to complete the following task

Put the concept map here that starts with the two concepts "class" and "instance" with labelled arrows and other concepts that creates a partial map of object-oriented programming, as you have learned it so far.

For some of the subjects the exercise was part of an examination and for others a voluntary exercise after a lecture on Friday afternoon. Twelve of the students had been given a concept map practice session but the authors do not state explicitly whether any of the other fifty-nine participants had seen, used or created a concept map before. The results confirmed widely reported misconceptions students have about concepts such as "class", "object" and "instance" but no additional evidence regarding the status of object oriented programming as a threshold concept was reported.

The fifth and final phase of the project focused on the transformative properties of threshold concepts (Moström , Boustedt et al. 2008). It was motivated by the view that transformation "is arguably the most important criterion" of a threshold concept and the group felt it had been

given little attention by students in the earlier phases. Eighty-six students were asked to "interpret their experiences in retrospect, adding their attitudes and opinions" by creating a "transformation biography" in which they identified and described a computing concept that transformed the way they see and experience computing. To help them structure their biography two sample "transformation biographies - one about modularity, another about abstraction" were provided with the instructions.

The students were drawn from five institutions in three countries but the majority (i.e. 87%) of the biographies were collected at two institutions in different countries. The biographies were written as part of an assignment in courses taken by computing majors in the second half of their programme.

The results show that the sample biographies, described by the group as "lure stories," heavily influenced the student responses. Fifty-nine responses (i.e. 69%) dealt with concepts very closely related to them. Twelve responses referred to modularisation in one guise or another. Forty-seven responses discussed a concept or experience closely related to abstraction, although only ten explicitly used the word *abstract* or *abstraction* in their biographies. Earlier in the project the group had proposed abstraction as a potential threshold concept and guided by that they decided to examine the forty-seven biographies related to abstraction in more detail.

An initial analysis of the abstraction biographies revealed a degree of similarity and on that basis they were partitioned into seven more general categories - modularity, data abstraction, object-oriented concepts, code reuse, design patterns, complexity and other concepts. Each category was assigned to a member of the research group and independently analysed. This detailed analysis led the group to conclude (Moström , Boustedt et al. 2008, p.133)

Abstraction is certainly a *key* concept in computer science. It appears, however, from our students' descriptions of transformative experiences, that abstraction *per se* is not a threshold, but that the particular abstractions discussed in this paper may be.

Summarising the results of the whole project (Thomas, Boustedt et al. 2010) the group note "Using a combination of approaches has allowed us to vary our focus and triangulate across data sources, across groups of students, and across modes of analysis." Yet despite these herculean efforts the group has not managed to identify a single threshold concept that is empirically supported using the data provided by their students. Referring to the data provided by instructors the group note



Faculty interviews in Phase 1 suggested abstraction (in general) as a threshold concept, but students describe particular examples of abstraction. It may be that instructors naturally try to view the bigger picture, abstracting away from the real thresholds.

Concluding the project, the group list the "potential threshold concepts that have been identified" as

pointers • object-orientation • data abstraction • complexity • modularity • code reuse • design patterns

The inclusion of the last four items in the list is a little surprising given that they only surfaced in the last phase of the project, on foot of the "lure stories" used as exemplars in the student biographies study. In their report of the study the group accepted that the lure stories had biased the biographies. In those circumstances the four nominations appear entirely speculative. The group still tentatively consider their original threshold concept proposals (i.e. object-orientation, abstraction, now modified to data abstraction, and pointers) as possibilities, but with muted confidence.

In 2010 the group suspended their search for threshold concepts in computer science (Sanders 2010, private communication).

4.2 Observations Regarding the Approaches Used

The approaches described above are characterised by the fact that they have all yielded *basic level* concepts (Rosch, Mervis et al. 1976). Recall from Chapter 1 that basic level concepts are informative and distinctive because they are not too abstract and not too specific. They tend to be associated with the "functional Structure" of our world and reside in the taxonomic space between maximally general and maximally specific where there are relatively few categories. This maximises the information value of the concept and provides generic information. Concepts at this level are recognised more rapidly; are acquired before concepts at other levels; are unmarked linguistically (i.e. are used in normal everyday conversation) and are used spontaneously to name objects. They are "the most natural, preferred level at which to carve up the world conceptually" (Murphy 2002, p.210).

The table below lists the top five concepts resulting from the deployment of the approaches described in the studies above. With the possible exception of *pointers* all of the concepts fall into Rosch's basic level category. One could argue that in Zendler and Spannagel's study the use

of the concepts in the ACM classification list predetermined the results because the classifications used in that list are general anyway. Notwithstanding this, the pervasive presence of basic level concepts is significant.

Zender and Spannagel	Goldman et al	TC Group
1. Algorithm	1. Procedure Design	1. Pointers
2. Computer	2. Problem Conceptualisation/Solution Design	2. Object-Orientation
3. Data	3. Abstraction/Pattern use	3. Data Abstraction
4. Problem	4. Test Design	4. Complexity
5. Information	5. Debugging/Exception Handling	5. Modularity

The approaches have not facilitated any sort of fine-grained consideration of the conceptual space and the features that distinguish concepts and allow them to be classified in a purposeful way. Despite taking different routes the authors' destinations have been the same, in fact, almost identical. In all three cases the outcome is an enumerated list with no accompanying insight into the anatomy of the conceptual space or the physiology of the conceptual elements identified. In the context of threshold concept identification the anatomical and physiological issues are pivotal for realising the integrative and transformational properties.

4.3 Critical Incident Interviews

The acquisition of a threshold concept that causes knowledge integration and transforms the learners view would appear to qualify it as a "stand out" experience of profound learning and one that a learner would be able to remember and identify as a critical learning incident. Flanagan developed a technique for gathering and analysing first hand reports of "important facts concerning behaviour" in what he called *critical incidents* (Flanagan 1954). A critical incident is described as "one that makes a significant contribution, either positively or negatively, to an activity or phenomenon" (Gremler 2004). The technique is designed to elicit the cause, description and outcome of a self-identified incident. It also takes cognisance of feelings and perceptions, any actions taken during the incident and any changes that the incident precipitated. Once the "story" of the incident has been recorded content analysis can proceed.

The technique is inductive in nature and can be used as an exploratory method to help increase our knowledge about an unknown or little-known phenomenon. The technique does not dispose the respondent to any preconceived, idiomatic or idiosyncratic perspective. The respondent is simply asked to recall the incident and it is left to them to determine what is important or worthy of inclusion in their recall of the incident. Thus the "story" is developed entirely from the respondent's perspective and is documented using their own language and terminology.

The respondent determines what qualifies as a critical incident and what qualifies as the important aspects of it. This can make the captured data diverse and offers the potential of identifying incidents and aspects of incidents that are unexpected or unknown. A high incidence of reporting the same incident, or aspects of it, may confirm a hypothesis or refocus the line of enquiry.

The approach is used widely because it offers a cost-effective way of capturing data. The respondents are located in their normal (working) environment and performing the task being studied. This authenticates the data. Because the occurrence of the incident is part of the respondent's normal activities they may be able to indicate how to recover from it or how to reduce its impact.

Self reporting eliminates the need for observation. This is particularly significant in situations where incident frequency is high. It also means that all incidents viewed as important by the person doing the task get reported, regardless of their severity. An observer may inadvertently filter important incidents or report unnecessary ones.

4.3.1 Participant Profile and Interview Protocol

To investigate if threshold concepts could be equated and analysed using Flanagan's critical incident technique I undertook a series of semi-structured interviews based around a critical incident protocol developed by Judith Lambrecht. She had used the protocol in studies of concept acquisition by learners in the development of computer literacy skills ((Lambrecht 1999); (Lambrecht 2000)). The protocol was appealing because, like Lambrecht, I was interested in concepts as they were experienced by students. I based the interviews on an adaptation of Lambrecht's protocol with the specific focus on programming.

The adapted protocol is reproduced below. It makes no mention of threshold concepts and does not refer to the features of threshold concepts. Conscious that reflective learning can occur anywhere I didn't constrain the location of the incident to any particular setting. I prepared a series of questions that I could use to assist the participant if I felt it would be useful during the interview.

I interviewed 30 students from three universities: one research-intensive, which provided thirteen participants; one teaching-intensive, providing nine participants; and one mixed, which provided nine participants. Twenty five of the students were male and five were female.

Nearly half (i.e. 14) of the students were in their final (or penultimate) year of study; nine of them were near the end of their first year and seven were pursuing postgraduate studies. This provided a rich mix of programming knowledge and skills.

All interviews were carried out between March 1 2007 and May 25 2007 and were professionally transcribed. Each interview lasted approximately 45 minutes and was facilitated in an interview room in the university.

Interview Protocol

I want you to think of an occasion when you finally caught onto a concept that you were having a hard time understanding. This might be an occasion when you were in a computing class, private study or interacting with other students in your course. Describe the key elements of the activities that caused this noticeable impact on your learning and understanding.

If possible, can you provide enough detail so that the effect can be clearly understood by others?

I may ask you some questions to assist you with telling your story.

Support Questions

Why do you think it was a problem for you?

What do you think you were lacking or might have been missing that caused it to be a problem?

What do you think helped you to move from being troubled by it to feeling you understood it?

When you felt you understood – how did you know?

When you understood, did it change the way you viewed things afterwards?

If you were trying to describe it to someone now, what would you do so that they would avoid having the same difficulty?

4.3.2 Interview Transcript Analysis

The results of this empirical study were not what was expected. Firstly, students were (largely) unable to recall a "critical" moment in their learning.

"I mean it's quite difficult to look back now and see why you'd have trouble with that because obviously it's quite easy. I don't know." (DB)

"I don't know, thinking of one occasion. I'm really having trouble with [this] actually because it was just a kind of "Oh, I didn't know it", and then I learned it." (EU)

Even when students were aware of the general case, that they had learned something they previously did not understand, they were frequently unable to recall details of the situation - time or place - that it happened:

"Well, there are a lot. It's just I can't - I'm trying to think of actual examples, because like working on my [final year project], definitely there were moments where I said, "Oh, I don't know how to do that. I really don't know how to do that," and then click. It just worked and then I went back to my computer. "Oh, well, that's how it works". But I can't actually remember what it is I was trying to do" (IM)

" ... unfortunately I can't actually remember the time I actually twigged it. It's a very simple thing of course, but I just ... I can't remember now whether it was a lecture or tutorial, or just was it playing around with it or what was it that done it" (BL)

In response to questions, there were some students who recalled revelatory moments, but they were rarely unique, in the way that I was expecting. Rather they reported them as an ongoing experience.

"So it always seems to be the way that sorta we had - our assessed exercises handed out that would seem really intimidating until a couple of weeks after they'd been handed out and then suddenly the concepts would start to click ..." (JL)

"There's been a good few of those experiences now in the last year or so, where I'd be in a module or something that I heard about in a previous module the year before. Something would click because - it'd be something along the lines, but it helped explain it that little bit more than it was explained previously. And you're like, oh, yes, now I understand this, and you get to use it." (DF)

" ... what annoys you two weeks ago, you understand two weeks later. "S*?t, what's a variable?" Two weeks later you're going, "S*?t, what's a string." And it's the same in March when you don't understand methods, it's two weeks later you understand a fair bit of methods. So now I understand methods. So now I'm hitting on classes ... and I know that in three and four months' time, that won't be such an issue to me, you know."(CS)

This was not the picture of learning that was anticipated would be painted as the result of the critical incident elicitation, nor did it support the picture of threshold concepts and their acquisition that had been painted in the literature.

This raised the prospect that like other researchers before me I had come to a research dead end and led me to question the choice of methods and to query what it was those methods were trying to illuminate. Regardless of whether the method used was interviews, structured tasks or questionnaires almost every study had asked students to look back and recall a time when they had a conceptual difficulty that had been resolved and to recount how it had been resolved.

In the case of my own interviews it might have been possible to discount the nine first year students on the basis that their knowledge structures could be fragile (Perkins and Martin 1986) and their ability to command the language required to articulate their experiences might be suspect. However, the knowledge structures and meta-learning skills of the seven postgraduates students should have been sufficiently well advanced to support review and examination. Likewise, the fourteen undergraduates in the final or penultimate year of their programmes.

In every case the outcomes had been unproductive. These outcomes were consistent with the work reported by Kevin Dunbar in his in-vivo studies of conceptual change in research laboratories (Dunbar 1997).

Dunbar spent a year immersed in the work of a number of research laboratories with the goal of identifying the points in time at which innovative scientific thinking occurred. He reports his discovery that one of the central places in which new ideas and concepts were generated was the regular meetings of all the laboratory researchers. At these meetings the researchers externalised much of their thinking by talking through the questions, proposals, hypotheses and interpretations that were generated by the group discussion. This made it possible for Dunbar to gain access to what he describes as their "online" thinking and reasoning.

Dunbar reported his findings in relation to the use of analogy and the reasoning mechanisms used by the research teams but he also reports that "the scientists had little memory" of how the various thinking and reasoning elements contributed to the conceptual change experienced by the research group. He documents the exchanges at one of the meetings where a significant

discovery was made and records the various ideas and reasoning lines that evolved during the discussion. He then describes the crucial point at which it became clear to the group that they had made a breakthrough.

"It was at this point the everyone in the lab realized that a conceptual change had occurred and all shouted in excitement. This was followed by some further analogies in which other post-docs suggested other experiments. Finally a post-doc made an analogy to the methods that other researchers have used and the methods that the post-doc had used, explaining why their rivals lab had not made the discovery that they had just made." (Dunbar 1997, p486-487).

One week later, one month later, three months later, and nine months later Dunbar asked the post-doc who conducted the research how the "discovery" was made. he reports that "On none of these occasions did he recall the spontaneous analogies used, nor that distributed reasoning was involved. Thus, much of the online cognitive processes that went into the conceptual change would have disappeared without a record if I had not taped the original meeting" (Dunbar 1997, p.488).

The evidence indicated that retrospective accounts of learning were not going to reveal the source of integrative and transformative learning experiences and this led me to question the retrospective nature of the commonly-used methods.

4.4 Retrospection and Hindsight

4.4.1 Hindsight Bias and The Curse of Knowledge

Our view of events in hindsight is not the same as our view in foresight and *hindsight bias* affects our judgements, including the judgements of experts (Fischhoff 1975). Fischhoff described *hindsight bias* as the difficulties we experience when, knowing a particular outcome, we try to assess or 'judge' the likelihood of that outcome in a foresightful manner (i.e. as if we didn't know the outcome). In a series of experiments he conducted subjects overestimated their ability to predict the outcome of an event when they knew the outcome. The subjects also overestimated what uninformed others would have known. This general finding has been replicated in a variety of settings ((Hoch and Loewenstein 1989); (Guilbault, Bryant et al. 2004)).

Fischhoff coined the term *creeping determinism* to describe our tendency to view the outcomes as inevitable, and cites (Florovsky, 1969, p.369) as follows

In retrospect, we seem to perceive the logic of the events which unfold themselves in a regular or linear fashion according to a recognizable pattern with an alleged inner necessity. So that we get the impression that it really could not have happened otherwise. (Fischhoff 1975, p.288)

Providing a rationale for the effect (Fischhoff 1975, p.295) suggested

subjects fail to properly reconstruct foresightful (before) judgments because they are "anchored" in the hindsightful state of mind created by receipt of outcome knowledge.

The anchoring effect occurs because once they have received the outcome knowledge they

immediately assimilate it with what they already know about the event in question. In other words, the retrospective judge attempts to make sense, or a coherent whole, out of all that he knows about the event.... Assimilation of this type would tend to induce creeping determinism. (ibid., p.297)

Fischhoff also showed that even when instructed to do so subjects were unable to ignore their knowledge of the outcome and denied their judgements were influenced by that knowledge.

(Hawkins and Hastie 1990) note that thinking backwards with knowledge of the outcome "sharply inhibits thinking" and results in "narrow-minded thinking" because only one outcome requires explanation whereas several possible outcomes must be considered when foresight is involved. Thus, the projection of acquired knowledge into the past can affect the selection of evidence, the evaluation of that evidence and the way in which the evidence is integrated into decisions about the knowledge. For example, when considering possibilities in hindsight "outcome-congruent information becomes more accessible" and once a piece of information is chosen it may need to be further refined or calibrated (e.g. estimates of speed, time, distance, strength) but always with the implications for the final outcome available. (Henriksen and Kaplan 2003, p.46) argue that possession of outcome knowledge changes the way we consider "a bewildering array of non-convergent events" because they have now become "a coherent causal framework for making sense out of what happened."

One interesting feature of hindsight judgements is that whilst the outcome is known with certainty it is rarely possible to identify a well-defined starting point for the combination of actions and events which caused the outcome. (Rasmussen 1987) discusses the difficulties associated with deciding when to stop an "explanatory search after the fact." Ideally the search

should continue until the root cause is identified but "generally the search will stop when one or more changes are found which are familiar and therefore applicable as explanations." However, such a decision may be based on pragmatic or arbitrary constraints (e.g. time, money or accessibility and availability of the participants) and not informational or causal requirements. In many cases the outcome of the search "depends entirely on the stop rule applied" and this can have serious implications for the conclusions which are or can be drawn.

In the context of economic analyses (Camerer, Loewenstein et al. 1989) investigate what they describe as "violations of normative theories of judgement" which are aberrations of the idea that better-informed people make better decisions. Hogarth called it the *curse of knowledge* because the better-informed agents are unable to ignore their informational advantage even when it is in their best interests to do so. For example, the buyer of a house may pay more simply because they know they can afford it. The seller of a house may reduce the price because of their knowledge of some of its flaws, even if the flaws are unobservable to the buyer. The better-informed person is cursed because their judgements of the less-informed person are biased by their own knowledge and they cannot ignore it. As a consequence the better-informed agent may suffer losses. (Camerer, Loewenstein et al. 1989) argue

This exaggeration interferes with the evaluation of decision quality. Outcomes are an imperfect indicator of decision quality; good decisions can lead to bad outcomes and vice versa. But principals must often judge decisions of agents on the basis of outcomes because actions or decision criteria are unobservable.

They concluded

The curse of knowledge suggests that informed subjects will be unable to ignore the information that they have that the uninformed subjects lack, causing bias in their predictions.

Thus, an expert's evaluation of the outcome of an exercise or the material gathered as part of a study may be erroneously based on the knowledge they possess but which the subject lacked.

4.4.2 The Illusion of Memory

The term *illusion of memory* (Chabris and Simons 2010, p.45) attempts to capture "the disconnect between how we think memory works and how it actually works." Bartlett's classic work on remembering (Bartlett 1932) is usually credited with distinguishing between the *reproductive* and *reconstructive* properties of memory. Most people believe memory is reproductive and that when we remember something we retrieve an accurate, video-style

version of the material from memory. For example, (Chabris and Simons 2010, p.45) report a survey in the US in 2009 in which 63% of respondents supported this view and nearly half of the respondents (47%) believed that once we have formed a memory of an event it doesn't change.

Bartlett's work showed that memory recall was a process of construction and not reproduction. He had subjects read an Indian folktale and recall it repeatedly. He noted

The first notion to get rid of is that memory is primarily or literally reduplicative, or reproductive...In the many thousands of cases of remembering which I collected, a considerable number of which I have recorded here, literal recall was very rare. (Bartlett 1932, p.204)

Bartlett developed a theory of remembering to provide a coherent explanation of how "the past operates as an organised mass rather than as a group of elements each of which retains its specific character." His theory emphasises the constructive style of remembering which is an active process, frequently error-prone because remembered components are distorted during construction and missing components are filled in. Bartlett describes it thus

when a subject is being asked to remember, very often the first thing that emerges is something of the nature of attitude. The recall is then a construction, made largely on the basis of this attitude, and its general effect is that of a justification of the attitude. (Bartlett 1932, p.207)

(Deese 1959) provides an interesting example of how a particular disposition or attitude can evoke recollections that are consistent or justified by the disposition but that are in fact erroneous. Deese chose a theme or *critical word* and created a word list containing twelve words closely associated with the critical word but that did not include the critical word. For example, the list for the critical word *needle* contained *thread, pin, eye, sewing, sharp, point, pricked, thimble, haystack, pain, hurt, and injection*. Subjects were read the word list and then immediately asked to recall it. Deese reported a high incidence of "extralist intrusions" (i.e. recall of the critical word) induced by the words appearing in the list. (Deese 1959, p.21) concluded that "in the process of recollection, words and concepts associated with the remembered items will be added." (Roediger and McDermott 1995) replicated and extended Deese's work and report that their "subjects confidently recalled and recognised words that were not present and also reported that they remembered the occurrence of these events." What Roediger and McDermott describe as "false memories" are a consequence of the constructive characteristics of remembering.

Without specifying the task involved (Brewer and Treyns 1981) recruited subjects for an experiment. The subjects were brought to a graduate student's office and asked to wait whilst the experimenter checked if the previous group had finished the activity. This was a ruse used by the experimenters. In fact, the time spent by the subjects "waiting" in the office was the experiment. After approximately thirty seconds the subjects were taken to another room and asked to write a list all of the things they had seen in the office. In addition to listing objects that were in the office about 30% of the subjects listed books and around 10% included a filing cabinet. Unusually, there were no books or filing cabinets in the office. The subjects' lists included things they had actually seen as well as things they would normally associate with an office of that type. Their false memories of the presence of some objects emerged from the context they associated with the location.

Loftus and Palmer note

two kinds of information go into one's memory for some complex occurrence. The first is information gleaned during the perception of the original event; the second is external information supplied after the fact. Over time, information from these two sources may be integrated in such a way that we are unable to tell from which source some specific detail is recalled. (Loftus and Palmer 1974, p.588)

An article in the New York Times⁵ provides a powerful real-life example of this. The January 6, 2009 edition of the paper included a story about the American spiritual writer Neale Donald Walsch who was accused of plagiarising a story he claimed was about a pageant at his son's kindergarten. As part of the pageant the children participating held up letters spelling the title of a song called *Christmas Love*. The child with the letter *m* inadvertently held the letter upside down which had the effect of changing the text to "Christ was love." Recalling the story Walsch found it so endearing he used it for a personal message posted on a spiritual website around Christmas 2008. Unfortunately, the story wasn't his and didn't involve his son. It had been written by another writer and published in a magazine ten years earlier. The original author had copyrighted the story with the US Copyright Office in 2003. Coincidentally, both authors happened to have a son with the same name. As part of their reporting of the story the New York Times quoted Walsch as saying

All I can say now — because I am truly mystified and taken aback by this — is that someone must have sent it to me over the Internet ten years or so ago....Finding it

⁵ (http://www.nytimes.com/2009/01/09/books/07book.html?_r=1).

utterly charming and its message indelible, I must have clipped and pasted it into my file of 'stories to tell that have a message I want to share.' I have told the story verbally so many times over the years that I had it memorized ... and then, somewhere along the way, internalized it as my own experience.

As we acquire information it does not just accumulate and lie passively in our cognitive system waiting to be recalled from memory. As time passes new information adds to or alters it and when we recall it we recall the morphed version and not the original, individual components of the recollection ((Loftus and Palmer 1974); (Bartlett 1932)).

4.4.3 Expert Blind Spot

(Davies 2003) argues that approaches based on fundamental or core concepts, which have been used in many disciplines including his own discipline of Economics, are unlikely to lead to the identification of threshold concepts because of their focus on the conceptual structure of the subject and their indifference to the learner's acquisition of understanding. He suggests that fundamental or core concepts, and their equivalents, conflate understanding and structure, and divorce understanding from the learner's previous experiences.

Davies' use of language is revealing. Conflation is not transformation and divorce is an antonym for integration. A taxonomic approach to mapping the conceptual structure of a discipline does not reflect the integrative nature of some concepts in the taxonomy. Chapter 2 highlighted the pivotal role played by *purpose* in any categorisation process. The purpose of a core or fundamental concept is significantly different from that of a threshold concept. A core or fundamental concept is agreed within a discipline as an identifiable, important component of the discipline's body of knowledge. Labelling a concept as core or fundamental is the outcome of decisions that are immune to the active learning process. In contrast, a threshold concept is viewed as an identifiable, important concept that *acts* on the (partially) acquired body of knowledge as a restructuring tool during the learning process.

Consistent with Davies' observations, Nathan and Petrosino invoke the blind spot metaphor of visual perception to highlight some difficulties experienced by educators with advanced subject-matter knowledge. Students' conceptual development and instruction should be guided by the tutors knowledge of the learning needs and developmental profiles of the students. However, tutors tend to be guided by "powerful organizing principles, formalisms, and methods of analysis" that serve as the foundations of the discipline and are embodied in the concepts

identified as core or fundamental. The tutor then falls victim of expert blind spot because they “tend toward views of student development that align more closely with the organization of the discipline than with the learning processes of students” and are often “entirely unaware of having such a blind spot” (Nathan and Petrosino 2003, p.906).

This view resonates strongly with the conclusions drawn by Goldman et al. about the incompleteness of tutors' understanding of student learning. The results of their Delphi process were greeted with caution because, as noted earlier, they felt “teachers have an incomplete (at best) understanding of student learning.” That conclusion persuaded them to interview their students “to validate the difficulty ratings asserted by our experts” in the expectation that “some topics that our experts ranked as easy will, in fact, be rife with student misconceptions” (Goldman, Gross et al. 2008).

The view is also consistent with the distinction drawn by Gal-Ezer and Harel between the knowledge requirements of a computing science educator and those of a practitioner or researcher. Their description of a “scientific intellectual” whose knowledge structure needs to be far more nuanced than the other two emphasises the need for educators to be actively alert to the dynamics of student learning and the nature of knowledge acquisition (Gal-Ezer and Harel 1998). This requirement is completely absent from the workbench toolset of a practitioner or researcher.

4.4.4 The Influence of Language

In a series of experiments (Loftus and Palmer 1974) demonstrated how the language used in a question can evoke false memories and cause a change in our recollection of an event.

In one experiment, subjects who had viewed film-footage of a car accident were divided into groups. The groups were asked the same questions about the accident but different words were used in the questions. For example, the groups were asked to estimate the speed the cars were travelling when they *contacted*, *hit*, *bumped*, *collided* or *smashed* into each other. Questions using *smashed* elicited higher speed estimates than the other verbs.

A week later the same subjects were recalled and again questioned about the accident, this time without being shown the film footage. They were specifically asked about the presence of broken glass. There was no broken glass visible in the film they had been shown the previous

week. The subjects who were asked questions using the word *smashed* were more likely to report seeing broken glass than the others.

In the first scenario when the subjects had actually seen the film-footage an actual feature of the event (i.e. the speed of the car) was distorted by the use of language. In the second scenario even though only a week had elapsed a fictitious feature was introduced by the use of language.

4.4.5 Emotion

A number of medical studies investigating lifestyle choices have found that the sources of evidence documenting some of those choices can be resistant to recall. For example, recall associated with choosing to start or give up smoking is markedly different from the recall of choices such as whether a baby was breastfed or vaccinated. In many of these cases the accuracy of the recalled material is paramount.

An important issue that emerged from the studies was the nature of the material recalled and its relationship to the significance of the original incident. Comparing remembered incidents to documentary sources, or, in some cases, to earlier accounts from the same participant (Berney and Blane 2003), medical researchers discovered that the *type* of information recalled made a difference (Seldon 1983). Somewhat counter-intuitively they found that "hum-drum events which carry little emotional charge and the barely noticed background routines of life appear to be recalled most accurately" and that "Emotionally laden events are least likely to be recalled accurately" (Blane 1996).

The affective consequences of acquisition of a threshold would qualify it as an "emotionally laden" event and the medical study findings would be consistent with our experience in the critical incident technique interviews where the participants recollections were closer to the "hum drum" and "background" events of everyday life than they were to the significant event of the acquisition of a new conceptual understanding.

4.4.6 Retrospection and Hindsight: A Summary

It is almost impossible for individuals to look back with hindsight and accurately recall critical cognitive events which, because they occurred spontaneously, could not be consciously monitored for the circumstances enabling their occurrence (i.e. the pre-event state) or the

effect and consequences of their occurrence (i.e. specific identification of the post-event state). Memory is constructive and not reproductive and our memory of something is not recalled with a clinical ability to identify the individual components which provide the specific character of the recollection. Those individual components have been transformed into a compound by the mental chemistry that operates on our cognitive processes and subsumes them.

Interpretations of questions that are phrased using particular language can encourage the recall of events in a particular way or associate properties with an event that may not be factually correct or justified. In the threshold concept literature the word "troublesome" is invariably associated with difficulty, typically conceptual difficulty, and the literature is dominated by a perspective of troublesomeness that is almost uniquely characterised by that association. In the previous chapter we showed that, in fact, troublesome knowledge is more frequently associated with various knowledge components already acquired but not deployed appropriately when it would be beneficial to do so.

It is somewhat ironic that attempts to elicit the remembered responses of subjects based on their past experience is actually biased against the identification of information we are seeking. Investigations based on asking experts or would-be experts to articulate their own experience of acquiring a threshold concept are completely open to the combined effect of the exigencies associated with expert blind spot, hindsight bias, the curse of knowledge, the illusion of memory, the nuances of language and the influence of emotion. The "betwixt and between" state of liminality and its rhizomatic properties, that are induced by the emergent nature of the learning process, inhibit the detailed cataloguing of the transformation.

4.5 A Better Source of Identification

Having established that threshold concepts are not to be located in retrospective accounts of learning, we considered the possibility that evidence of their existence could be found elsewhere.

In Chapter 1 we noted the frequent observation that threshold concepts have a special attraction for pedagogues in all disciplines. The idea is seen by them as having immediate relevance to their own practice and allows them to leverage their expertise of their own discipline. This is an attractive alternative to having to feel like informed amateurs in their

attempts to unpack and demystify the "big gap between reading about, being taught about, reflecting on, and discussing" the generic and largely technical view of teaching and learning provided in the education literature (Meyer 2010, p.196).

This disposition among pedagogues encouraged us to consider teachers' expert knowledge as a fruitful place to locate and identify threshold concepts because that, after all, is where the reality of student learning is lodged, in the day-to-day classroom experience; in the teachers' practice; in the teachers' presentation of concepts; rather than in the learners acquisition of them. Cousin framed the site of enquiry rather eloquently when she described how threshold concept research:

"does not require the academic to learn another discipline; on the contrary, it requires that she goes more deeply into her own for the purposes of formulating the best ways of teaching and learning it. By staging the exploration at the site of the subject and of its difficulties, threshold concept research promises to harness an academic's research curiosity for his subject with a new curiosity about how best to teach it" (Cousin 2010, p.7)

4.5.1 Teachers' Expert Knowledge

However, there are some difficulties associated with investigating teachers' expertise.

Firstly, teacher expertise is largely professional, embodied and contingent: a teacher may know 5 or 6 or 10 ways to teach recursion, to exemplify it, but will not use them all every time they teach the topic. They will draw on their expertise and select what is appropriate to the circumstances of the particular students on a particular day. This is typical of Schön's notion of "reflection in action" that characterizes professional performance (Schön 1983) and the classroom expertise that van Manen calls "noncognitive, nondiscursive confidence" (van Manen 1995).

Secondly, it is hard to classify exactly what "teacherly knowledge" consists of. Anderson and Page identify four areas they have described as follows

- *Technical knowledge* which is a form of equivalence in the sense that "academic knowledge is technical knowledge"
- *Local knowledge* which "includes the narratives that are idiosyncratic to a local school or community setting ... included within this domain is knowledge of local politics, and local cultures and sub-cultures"

- *Craft knowledge* which "consists of the repertoire of examples, images, understandings and actions that practitioners build up over time," and
- *Personal knowledge* which consists of things that I know by virtue of my life experience, such as being a parent, having a mortgage, and so on.

Rather than encompass four disparate areas, Max van Manen defines just two, *cognitive knowing* (knowledge and facts) and *noncognitive knowing*. Focusing on his less-intuitive noncognitive knowing he subdivides it into four subcategories which he enumerates as

- Knowledge that resides in action as lived (in our confident doing, style, and practical tact; in habituated acting and routine practices)
- Knowledge that resides in the body (in an immediate corporeal sense of things; in our gestures and our demeanour)
- Knowledge that resides in the world (in being with the things of our world; in situations of at-homeness, dwelling), and
- Knowledge that resides in relations (in the encounter with others; in relations of trust, recognition, intimacy).

Of the various taxonomic classifications one of the most influential has been formulated by Lee Shulman. Shulman's classification has seven components (Shulman 1987) - although he did not arrive at this classification on his first attempt. He has listed them under the following headings:

- *Content knowledge*
- *General pedagogical knowledge*, with special reference to those broad principles and strategies of classroom management and organisation that appear to transcend subject matter.
- *Curriculum knowledge*, with particular grasp of the materials and programmes that serve as 'tools of the trade' for teachers.
- *Pedagogical content knowledge*, that special amalgam of content and pedagogy that is uniquely the province of teachers, their own special form of professional understanding.

- *Knowledge of learners and their characteristics.*
- *Knowledge of educational contexts*, ranging from the workings of the group or classroom, the governance and financing of school districts, to the character of communities and cultures.
- *Knowledge of educational ends*, purposes and values, and their philosophical and historical grounds.

One final issue that presents itself in relation to teaching expertise is that, even if teacher's expert knowledge can be identified, it is hard to represent. Documentation of teaching practice is rare and there are few representational forms. Where there exist collections of representations they are largely non-standard and therefore non-comparable.

4.6 Pedagogic Content Knowledge

Of all these attempts to examine the constituents of teachers' expertise by far the most significant has been that of Shulman's *pedagogical content knowledge* which has a take-up among researchers far beyond any of the other classifications. This is principally because it encapsulates what is considered a unique and distinguishing indicator of teaching expertise. For example, teachers may have high content knowledge, but with low pedagogic knowledge they are often characterised as brilliant subject experts who cannot teach. Similarly, Ben-David Kolikant has reported on how teachers used their high pedagogic knowledge to mask their lack of content knowledge when faced with the task of teaching a programming language which took them out of their comfort zone and required a paradigm shift from procedural programming to object-oriented programming (Lieberman, Kolikant et al. 2009)). In contrast, teachers characterized as having high pedagogic content knowledge are able to draw on a repertoire of approaches, to respond to student needs, to present and explain material in a wide variety of ways, to draw disparate areas of the curriculum together, and to show how the material being learned is linked to more advanced concepts (or to concepts already well-known).

Shulman described "the study of subject-matter and its interactions with pedagogy" as the missing paradigm in research on teaching (Shulman 1999, p.ix). If threshold concepts are present within a discipline, and they can be detected within teachers' knowledge, pedagogic

content knowledge would seem to be the site to investigate. Threshold concepts are clearly more than content knowledge (which is pedagogically unordered and impervious to how hard students find it to learn), and pedagogic knowledge alone would be insufficient to characterize and present them (they are not generic, but specific to the subject being taught). It is in pedagogic content knowledge that they are expressed by the teacher and made apparent to the students.

4.6.1 The Pedagogic Content Knowledge Model

Pedagogic content knowledge is a conceptual model of teaching which attempts to capture the idea that teaching is more than a transmission activity that delivers subject content to students and learning is more than the passive absorption of subject content for accurate regurgitation later. Teachers develop their pedagogic content knowledge by making a conscious decision to teach "for understanding" instead of simply delivering content. It evolves as part of their professional learning and their constant pursuit of deeper levels of understanding of the pedagogy of their discipline. This requires a commitment to devising innovative pedagogic strategies and then implementing and evaluating them, whilst at the same time honouring the prescribed curriculum requirements (Loughran, Berry et al. 2006).

Pedagogic content knowledge is learned on the job, through practice. Interpreting and reflecting on their practice has a transformative effect on a teacher's knowledge and perspective. The resulting compound of pedagogic and content knowledge distinguishes expert teachers by allowing them to make deeper interpretations of events and contextual clues and respond appropriately to students' requirements.

The theory of teacher cognition associated with pedagogic content knowledge is explained as the integrative and transformative development of a tutor's subject content knowledge and their general pedagogic knowledge. This requires knowledge organisation that integrates diverse practices and ideas, suggests alternative explanations, and supports the evolution of new insights (Gess-Newsome and Lederman 1999). Shulman described pedagogic content knowledge as the

"blending of content and pedagogy into an understanding of how particular topics, problems, or issues are organised, represented and adapted to diverse interests and abilities of learners , and presented for instruction" (Shulman 1987, p.8)

Pedagogic content knowledge and threshold concepts share a high degree of similarity and coherence. As noted in Chapter 1, threshold concepts and pedagogic content knowledge represent a synergy of ideas because they both have deep understanding of content knowledge at their centre. Both are student focused but not student centred. Both rely on the expertise of reflective pedagogues. They are distinguishable from each other by the sharpness of the focus in their respective lenses. Pedagogic content knowledge has a wide view whereas threshold concepts zoom in on specific content knowledge that derails student learning. Awareness (albeit tacit) of threshold concepts and the difficulties they may cause is subsumed into pedagogic content knowledge. The two ideas are linked inextricably and mutually supportive.

4.6.2 Documenting Pedagogic Content Knowledge using a CoRe

It is difficult to capture, portray and share knowledge of practice in ways that are amenable to articulation, representation and meaningful use by others. One approach that has proven successful at capturing pedagogic content knowledge was developed by Loughran and his colleagues. The approach grew out of their attempts at sharing teaching expertise and skill through brief anecdotes and stories of in-class experiences. Through these conversations and shared observations they developed a representation that captures important aspects of the knowledge that underlies pedagogic content knowledge. The representation provides access and support for the two foundational components of pedagogic content knowledge (i.e. content knowledge and general pedagogic knowledge) with the intention of bringing them to the forefront of the teachers thinking and thereby allowing them to be better understood (Loughran, Berry et al. 2006).

The representation is called a *CORE*, a mnemonic for *Content Representation* using the first two letters of the words, and written as CoRe. (Loughran, Berry et al. 2006) have provided several examples of CoRe's developed for topics in science.

A CoRe is a two-dimensional grid. Along the vertical axis of the grid is listed a series of questions or *prompts*, that are intended to provoke consideration of a particular concept and motivate elicitation of a teacher's pedagogic knowledge in relation to the concept. Along the horizontal axis the teacher captures their content knowledge of the concept by listing what they consider to be the "big ideas" associated with this concept. The number of ideas is not restricted and individual teachers can include as many as they wish. At the intersection of each

CHAPTER 4 : IDENTIFYING THRESHOLD CONCEPTS

row and column the teacher makes an entry documenting their response to the row prompt for the big-idea in the column. A blank template is shown below.

	<i>Big Idea A</i>	<i>Big Idea B</i>	<i>Big Idea C</i>	<i>Big Idea D</i>	<i>Big Idea E</i>
What you intend the <u>students</u> to learn about this idea.					
Why is it important for students to know this.					
What else <u>you</u> know about this idea (that you do not intend students to know yet).					
Difficulties/limitations connected with teaching this idea.					
Knowledge about the students' thinking which influences your teaching of this idea.					
Other factors that influence your teaching of this idea.					
Teaching procedures (and particular reasons for using these to engage with this idea).					
Specific ways of ascertaining students' understanding or confusion around this idea (include likely range of responses).					

The CoRe is a facilitation mechanism that allows a teacher to document how *they* conceptualise the target concept. The big-idea columns are aimed a provoking consideration of and making explicit the supporting ideas and concepts that scaffold or characterise the target concept. The big-ideas represent what the teacher views as important about the concept and how the ideas can shape the student's learning of the concept. In some cases the entries will be the ritual-like entries that are well-known within the discipline as being necessary or problematic. However,

expert teachers may have a novel way of handling them and will also draw on their reflective experience of student difficulties to include previously unknown entries they have unearthed and dealt with in their own classes.

The teacher's level of expertise may be evident in the number of big-ideas they include. For example, a small number of ideas might suggest that too much is being encompassed in each entry. Conversely, a large number of entries might suggest the teacher's experience of this idea is extensive and a potentially rich source of data. As a source of guidance Loughran and his colleagues provide a rule of thumb metric that indicates experienced practitioners usually settle on between 5 and 8 big-ideas. However, of far more importance is how the big-ideas interact and integrate to form the target concept.

The prompts in the first column oblige the teacher to document their view of how and why each big-idea can and should be taught. The teacher's responses to the prompts elicit the basis on which they make a variety of decisions about how they present the idea to students. In addition, it provides a useful insight into how the teacher came to appreciate why a particular concept proves difficult for students to grasp. What the teacher records is what has evolved as their (current) best strategy for dealing with this difficulty. Thus, the entries made by the teacher represent what they consider exemplary or best practice for teaching this idea.

Loughran and his collaborators describe their formulation as a "powerful, accessible and useful representation" of pedagogic content knowledge (Loughran, Berry et al. 2006, p.26). They identify several benefits that accrue from the representation. First, the grid format is simple to use, particularly in soft-copy format. The intersecting rows and columns can act as a bridge between theoretical and practical aspects of teaching but the focus is always on bringing the teacher's expert knowledge to the fore. Populating the grid with entries obliges the teacher to document, explore and analyse their own practice and makes what is normally implicit, private and individual for the them explicit, clear and meaningful for both themselves and others. This reduces the influence and dependence on generic descriptors of good pedagogy and provides a powerful, more sophisticated approach to unpacking the expertise associated with teaching a particular concept.

4.7 Identifying Threshold Concepts

The earlier analysis of our ability to recall with hindsight the cognitive transitions that constitute learning episodes showed it to be an unreliable source of evidence because it actually mitigated against the discovery of the type of information we sought. In contrast, the pedagogic content knowledge of expert teachers is current, active and intrinsically rich with the type of evidence we seek because it represents the accumulated experience of dealing with exactly the type of episodes we are investigating.

Using their pedagogic content knowledge expert teachers can identify candidate threshold concepts, whether they are obvious instances associated with concepts that are inherently difficult conceptually, or the less obvious instances that are associated with more subtle forms of difficulty such as tacit, ritual, inert knowledge. Their high levels of pedagogic content knowledge allow them to unpack the constituent concepts integrated by a threshold concept, using their content knowledge, and articulate the substance of the difficulties associated with them, using their pedagogic knowledge .

Using a CoRe representation to analyse and document a concept identified as a potential threshold concept can provide an explicit and accessible method for examining the evidence to determine whether the concept qualifies as a threshold concept or not.

In Chapter 5 we use this methodology to explicate the case for state as a threshold concept in computing.

Chapter 5

State as a Threshold Concept

In this chapter we propose the concept of *state* as a threshold concept of computing. The definition of a threshold concept requires a qualifying concept to integrate a body of knowledge and irreversibly transform our way of viewing it. These requirements tend to encourage us to look for significant concepts, particularly because they have to transform, and the notion of transformation seems to carry with it an intuitive sense of momentous effect. Significant is an acknowledged metaphor for big and there is a danger that our search for threshold concepts may be influenced by such a view.

5.1 The Lure of the Big Concepts

The lure of the big concepts is a common pitfall. The surveys of experts reported in Chapter 4 showed that when asked to identify important concepts, regardless of whether *important* was interpreted as *fundamental* or *core* or *threshold*, the experts tended to pick concepts that maximised information value and provided generic information. They were "big" or very inclusive concepts like object-orientation, data abstraction, complexity, modularity, procedure design, computer, data, problem and so on.

In his book, *Life in the Undergrowth*, the legendary naturalist Sir David Attenborough reminds us that we are "greatly prejudiced by our size" and we find it very difficult to believe that creatures that are many thousand times smaller than ourselves, like "bees and blowflies,

beetles and butterflies," should be considered as anything other than "mere automata, mindless robots reacting automatically to the simplest stimuli" (Attenborough 2005, p.7). In the book and the accompanying BBC television series, he describes how these tiny creatures play a central role in the continued stability of the entire eco-system. In his closing commentary from the BBC series he reminds us of the folly of disregarding these tiny creatures

"If we and the rest of the backboned animals were to disappear overnight, the rest of the world would get on pretty well. But if [the invertebrates] were to disappear, the land's ecosystems would collapse. The soil would lose its fertility. Many of the plants would no longer be pollinated. Lots of animals, amphibians, reptiles, birds, mammals would have nothing to eat. And our fields and pastures would be covered with dung and carrion. These small creatures are within a few inches of our feet, wherever we go on land — but often, they're disregarded. We would do very well to remember them." (Sir David Attenborough, in concluding *Life in the Undergrowth*)

The *role* played by the constituents of a mechanism does not always equate with their size. Indeed, as Britcher has observed "Large things are scarce on this planet: large animals, large structures of any kind. There is a reason for this. Large things, without nature's kindest nurturing, cannot stand under their own weight" (Britcher 1995, p.20).

Peter Denning has described how, in what he calls the "mature disciplines" like physics and biology, rich structures are built from sets of small but important components (Denning 2003). Small things, interwoven in useful ways, are not only capable of forming larger structures they *allow* them to be formed because working with small(er) components is much easier than working with components that are themselves large. Simply because something is large and perceived as important or significant does not make it so.

Things that bind are usually concealed, or at least not prominent, and facilitating concealment requires binding agents to be small and innocuous (e.g. stitches, mortar, joints). In Chapter 3 integration was identified as the central feature of threshold concepts and the smaller concepts appear to provide more utility for that task than the more grandiose ones.

5.2 Proposing State as a Threshold Concept

The proposal of state as a threshold concept in computer science nominates a seemingly humble concept. The choice appears consistent with the proposals that have emanated from other disciplines. For example, the economists have identified *opportunity cost*; *precedent* has

been offered by the schools of law; and *pain* by the physiotherapists. All of these concepts are within the conceptual grasp of a typical young child.

For instance, what child is not vividly aware of the nuances of opportunity cost when faced with the choices arranged around them in a sweet shop? What parent has not been reminded that the pleasure they are currently proposing to deny one child was granted to a sibling on a previous occasion? In any environment where children are present 'knocks' will result and consolation and tear wiping go hand-in-glove. Of course, the child's conception is rough-edged and lacks the sophisticated reasoning that will accompany it when it has been refined and smoothed, but the seedling presence of these conceptions in children of very tender age is astounding nonetheless.

It seems paradoxical that these small or simple concepts could have the ability to irreversibly integrate and transform a body of knowledge that attempts to explain the behaviour of the global economy, the operation of international law or the treatment of back injuries.

5.3 A Content Representation (CoRe) for State

The proposal of state as a threshold concept is developed from a Content Representation, or CoRe, of the type described in Chapter 4, which has been constructed for the concept of state and derived from my pedagogic content knowledge. The CoRe identifies a set of so-called *big ideas*, unfortunate but understandable terminology for the reasons set out above, that have been chosen by me. The choices have been informed by my pedagogic content knowledge of the troublesome knowledge that envelops the concept of state.

The CoRe structure has considerable explanatory power especially in its use of focussing questions to elicit underlying structures of tacit disciplinary pedagogic expertise. Additionally, it has considerable representational power in presenting this knowledge in a compact and apprehensible form.

5.3.1 Content

A discipline expert would not view the ideas included in my CoRe as troublesome and so the exposition of the ideas may induce a sense of incredulity in a reader with such a status. Discipline experts have little understanding of student learning (Goldman, Gross et al. 2008) and

are subject to *expert blind spot* which can make them blind to the instructional needs of novices (Nathan and Petrosino 2003, p.906). Often they are completely unaware of having these deficits. When considering problems and problem solutions expert knowledge can be a disadvantage because it can act as a mental set and promote fixation which inhibits consideration of a wider range of possibilities (Wiley 1998). It is very difficult for a discipline expert to appreciate that the assignment statement can be problematic for novice programmers, yet pedagogic expertise and familiarity with the computer science education literature reveals that it is. Similarly, it is very difficult for a discipline expert to appreciate that the concept of a method could be problematic, and yet it is. The perceived simplicity of these concepts can disguise their troublesomeness.

5.3.1.1 Whose Content?

In its originating context, and in subsequent adoptions (Saeli, Perrenet et al. 2012), the CoRe structure has been used exclusively as a group-based tool to collect and collate ideas from a range of teachers in regard to teaching a particular subject (or topic area) to a particular age group. In this, it has frequently been used in a workshop setting.

Following (Shulman 1987) and (Gudmundsdottir 1995) I have taken pedagogic content knowledge to be the construction and property of an individual. A teacher draws on their own knowledge of the discipline, and combines it with their own understanding of how learning is accomplished (whether that knowledge is formally or informally acquired). As Olson and Bruner say “Teaching ... is inevitably based on teachers’ notions about the nature of the learner’s mind. Beliefs and assumptions about teaching, whether in school or any other context are a direct reflection of the beliefs and assumptions the teacher holds about the learner” (Olson and Bruner 1999, p.11).

From these twin sources, combining to form pedagogic content knowledge, teachers devise teaching interventions which are structured around their own “curriculum stories” to lay out learning opportunities for their students. A “curriculum story” encompasses the choices teachers make about what parts of a syllabus are important, which have similarities, in what order they are to be taught and what connections may be emphasised between parts. Gudmundsdottir shows how these stories grow and develop with accumulating teachers’ experience and deepening pedagogic content knowledge (Gudmundsdottir 1990).

Of course, teachers may agree on “the right way” to teach, and so be able to share approaches and materials. A teacher who believes that “objects first” is the best way for students to learn Java will be drawn to a different range of textbooks and exercises than one who believes that “procedural first” is more effective. However, it is hard to imagine how an entire body of pedagogic content knowledge may be arrived at jointly, or how it may be precisely replicated from one teacher to the next. As it builds on specific, personal experiences of teaching and being taught it cannot be objectively “learnt”, and even if it may be painstakingly explained, if there is no fit with another’s pedagogic content knowledge, it will be disputed and/or discarded. Even the contributors to a CoRe developed in a workshop setting may view it as just an agglomeration of parts from disparate teachers and backgrounds combined into a Frankenstein-like CoRE.

5.3.1.2 An Autoethnographic Approach

I believe there is greater value in exploring an individual’s pedagogic content knowledge to capture and articulate how a concept proves troublesome for students and how, when it is acquired, it integrates the existing knowledge with a transformational effect.

Choosing *which* individual to study, however, is a methodologically challenging question. There are several associated problems.

Firstly, there is the problem of eliciting tacit knowledge. A teacher may well find it hard to say why they teach something in a specific way because “that’s the way they’ve always done it” or simply “that’s the way it works”. Secondly, the experiences that form our pedagogic understanding are unbounded by time or type: they may range from early childhood to the day before yesterday and encompass informal learning experiences as well as structured understanding of learning theory and classroom exposure. Knowing where to look and what to ask become significant obstacles for enquiries of this nature. Thirdly, there is the problem of how pedagogic content knowledge is expressed. It would require considerable resources to investigate lecture notes, slides, student assessments, classroom and lab activities, marked student work, etc. as they develop over many years, to observe the expression of pedagogic content knowledge in instructional design and classroom artefacts.

Consequently, to demonstrate the articulation of threshold concepts within pedagogic content knowledge, I have chosen to harness the power of the CoRe form through an individual’s

expertise, using an autoethnographic approach. Chang describes autoethnography as a self-narrative model with distinctive characteristics that supports a systematic approach to data collection, analysis and interpretation and facilitates validation by others. It combines three aspects. The content orientation which is focussed on the self ("auto"); the interpretative orientation with its emphasis on the cultural ("ethno"); and the methodological orientation that examines autobiographical data through a critical, analytical lens as part of the research process ("graphy"). Thus autoethnography "combines cultural analysis and interpretation with narrative details" but the stories are "reflected upon, analyzed, and interpreted within their broader sociocultural context" (Chang 2008, p.46).

This approach situates the researcher in a critical relationship with others in the same cultural situation (in my case, the teaching of computing in tertiary education). It is this that distinguishes autoethnography from other narrative-oriented writings such as autobiography, memoir, or journal and has established it as a "rigorous ethnographic, broadly qualitative research method" (ibid. p.57). The strength of autoethnography means that it is a frequently used method in educational enquiry ((Hayler 2007); (Granger 2012)).

Chang has enumerated five points of potential difficulty that demand consideration and resolution to ensure the integrity of the process, the data and the analysis. He describes them as follows

1. excessive focus on self in isolation from others
2. overemphasis on narration rather than analysis and cultural interpretation
3. exclusive reliance on personal memory and recalling as a data source
4. negligence of ethical standards regarding others in self-narratives
5. inappropriate application of the label "autoethnography"

Coupling an autoethnographic approach together with the CoRe form is a strategy that provides two significant benefits. The first accrues from the explicit documentation of expert reflective practice and its capture using a simple structure (5.3.1.3, below). The second accrues from the inherent properties of the CoRe which neutralise some of the potential difficulties attributed to autoethnography.

The strategy used has not only remained faithful to the term "autoethnography," addressing Chang's point 5, it has also reinforced it by the adoption of Anderson's "analytic

autoethnography" approach (Anderson 2006). Anderson describes how evocative autoethnography, the traditional approach to autoethnographic research, is based on a free-form style that requires considerable narrative and expressive skills and seeks to create an empathetic or emotional resonance with the reader. In contrast Anderson advocates a more "scientific" approach that emphasises more formalised rhetoric and method. Analytic ethnographers are not content with accomplishing the representational task of capturing "what is going on" in an individual life or social environment but are directed towards theoretical development, refinement and extension (ibid. p.387). The CoRe structure captures what is going on and the accompanying narrative adds value and quality by providing a broader generalisation. This embeds the analysis in a coherent disciplinary framework.

Autoethnography can be criticised for its singular focus and the conclusions drawn from that one perspective, for whilst it is de facto a valid interpretation, it may not be reliable or replicated. The CoRe analysis is not limited to a single person (i.e. self) because people do not accumulate their experiences in a social vacuum. It is informed by the outcomes of personal reflective practice but also by the individual's awareness and knowledge of the reflective practice of others, acquired through working with colleagues (sometimes across various institutions), as well as knowledge of the curricular and pedagogic literature, and from personal interaction and exchange. The CoRe is reflective of, and draws on, the individual's immersion in the culture of the discipline and its pedagogy, in marked contrast to an isolationist perspective (addressing Chang's points 1 and 2).

The CoRe documents a conceptual framework through the lens of a (proposed) threshold concept and exposes the integrative effects it has on what were previously viewed as disparate concepts. It is populated through an analytical process that obliges the person filling it in to reflect on, question and justify their choice of "big ideas" and the pedagogic strategies that can be applied to realise the transformative outcome, in a process similar to Schön's "reflection on action". Thus completing a CoRe is an act of reasoned analysis, not memory recall (addressing Chang's point 3).

The CoRe form is "ethics neutral" because, although it draws on others, and the work of others – students, colleagues, text-book authors – its analytic form means they are not involved (or implicated) as participants, as is more common in other autoethnographic forms, such as oral histories or emic accounts of native ethnography (addressing Chang's point 4).

Anderson acknowledges that analytic autoethnography is not meant to produce "un-debatable conclusions" but it does attempt to ground the connections between the insider's perspective provided by the researcher's presentation of the research and the theoretical understandings of broader phenomena. The user of autoethnography is engaged in a process of reflective practice that includes awareness of the discursive milieu and ongoing critical consciousness and self-examination. None of these things begin or end when one adopts or discards the title of researcher. The autoethnographic approach used in this work enables the researcher to pay close attention to the separation of their representation and interpretation roles, to manage the research work effectively and to ensure the integrity of the process, the data and the analysis.

5.3.1.3 Content Exposition

In the rest of this chapter, I have set out from first principles the pedagogic content knowledge framework that I have for the concept *state*.

The exposition begins with a tabular CoRe depicting my pedagogic content knowledge with regard to state. The aspects of state that are embodied in my chosen "big ideas" range from the very basic concepts of programming through to what are considered the most powerful and most intellectually demanding. Each idea made explicit within the CoRE structure is situated in the context of others' practices, drawn from literature, to demonstrate their wider currency and to show that whilst they are individual, they are not idiosyncratic. The literature documents the history of difficulties novice programmers experience with the concept of state and references to that literature have been supplemented with a description of how the concept is troublesome.

The presence of state throughout the conceptual space associated with learning to program establishes its credentials as an integrative concept of computing. The pivotal role state has played in the evolutionary development of programming paradigms, from indisciplined *big balls of mud*⁶ to architecturally sensible object-oriented systems, marks it as a transformative

⁶ Foote and Yoder (1997) Big Ball of Mud. *Fourth Conference on Patterns Languages of Programs* (PLoP '97/EuroPLoP '97)

concept. This transformed view of computing systems as state space partitioned into objects is irreversible.

The set of big ideas developed here is not exhaustive and the exposition is followed by a series of summary descriptions of other big ideas that could have acted as alternative or supplementary ideas to allow the integrative power of state to be successfully articulated. Thus, the purpose of the exposition is to justify the identification of state as a threshold concept and not to provide a curriculum for teaching state.

The central feature of the pedagogic content knowledge displayed here is, in fact, a threshold concept for computing. The strength of this idea is in its simplicity.

5.3.2 Representation

As demonstrated in Chapter 4 a CoRe is normally represented as a two-dimensional tabular structure with the static, discussion prompt headings listed on the vertical axis; the dynamic, big-idea headings provided by the discipline teaching expert(s) listed across the horizontal axis; and the dynamic, pedagogic content knowledge being articulated at the intersections of the two. This structure is suited to capturing the contributions of a number of participants in group or workshop scenarios using the dynamic properties of whiteboards, expandable wall charts and onscreen views of spreadsheets where, when necessary, presentation constraints can be supplemented with linked documents or even supporting verbal contributions.

The single voice and absence of such facilities in a document like this demands an alternative format and in what follows the CoRe for the concept of state has been transformed into what is essentially a one-dimensional representation composed of a sequence of individual textual descriptions set out over several pages. A summary style tabular CoRe structure that encapsulates the exposition which follows is presented here.

CHAPTER 5 : STATE AS A THRESHOLD CONCEPT

Year Level for which this CoRe is designed Year 1 Introductory Programming	State Space	Notional Machine	Variables	Methods and Parameter Passing	Design/Decomposition
<p>What you intend the <u>students</u> to learn about this idea.</p>	<p>What state space is and that it is vast, even for small problems.</p>	<p>ONLY two operations (inspect and alter) a computer can perform.</p>	<p>Variables are used to record state and are central to all programming environments.</p>	<p>The state space is too large to attempt to comprehend as a single unit and needs to be broken down into more manageable units. This obliges us to develop information sharing facilities so that the units can interact.</p>	<p>Breaking problems down is a fundamental feature of solving problems. Programming is not about writing a 10,000 line program but about writing 1,000 10 line programs.</p>
<p>Why is it important for students to know this.</p>	<p>The vastness is the source of difficulty.</p>	<p>ALL programs are collections of inspect and alter operations clothe in 'sophisticated' programming constructs which provide a more abstract implementation.</p>	<p>Without variables we cannot flexibly utilise and control the inspect and alter operations.</p>	<p>Modern design techniques attempt to partition the state space into logically coherent units or objects. Methods are a stepping stone to this.</p>	<p>Every "real" and example program the students encounter will be structured as a series of relatively small, cooperating methods which are the result of design decisions that attempt to structure the solution for reliability and comprehension.</p>
<p>What else <u>you</u> know about this idea (that you do not intend students to know yet).</p>	<p>Programming expertise = intelligent state space management.</p>	<p>This notional machine is called a Turing Machine and is the basis for the theory of computation.</p>	<p>Variables can record very trivial states or very sophisticated states and this may give rise to different notations being used to manipulate them even though the operations are the same (i.e. inspect and alter).</p>	<p>Methods and sharing information between them will dominate the development of the students programming expertise.</p>	<p>Dijkstra on importance of structure and knowing what it is (glass box equivalent). Regardless of the paradigm used system design is based on the partitioning of state into a "useful structure" (EWD).</p>

CHAPTER 5 : STATE AS A THRESHOLD CONCEPT

<p>Difficulties/limitations connected with teaching this idea.</p>	<p>This is the student's first formal introduction to abstraction.</p>	<p>This idea is deceptively simple and therefore NOT perceived as important by students. This view may invoke "defended" learning in the form of unbelievability.</p>	<p>Variables are discrete NOT continuous values and must be explicitly manipulated. Initialisation is a crucial operation</p>	<p>The discipline of computing has entrenched but confusing and inappropriate language for describing this concept.</p>	<p>Partitioning problems is difficult and it requires a lot of practice. The strategies used for partitioning them are all based on state space although not always obviously so.</p>
<p>Knowledge about the students' thinking which influences your teaching of this idea.</p>	<p>It is the absence of students thinking about this item that influences my teaching of it.</p>	<p>Popular view that "cool" computing stuff is derived from complexity and sophistication when in fact it is from simplicity.</p>	<p>Students often think the variables "act" independently and "behave" as the students expect them to, especially if they have meaningful names.</p>	<p>Students frequently experience the issues associated with this concept (e.g. on their Facebook pages) but they remain part of their inert knowledge.</p>	<p>Students will tend to focus on the processing required to solve a problem and this can lead to decompositions that are unworkable. Early and often examples of state-based decomposition is a "start as you mean to continue" approach.</p>
<p>Other factors that influence your teaching of this idea.</p>	<p>Introduce the idea of problem boundaries and conscious decisions to bound problems.</p>	<p>This idea is pervasive and completely independent of the programming paradigm used to teach novices.</p>	<p>Variables tend to have "roles" and fall into idiomatic categories which can assist acquisition of the idea of a variable.</p>	<p>This concept plays a pivotal role in the development of design knowledge. Most follow-on courses will depend on this concept.</p>	<p>The use of objects is notoriously problematic for novices. We need to be cognisant of these difficulties.</p>

CHAPTER 5 : STATE AS A THRESHOLD CONCEPT

<p>Teaching procedures (and particular reasons for using these to engage with this idea).</p>	<p>Invite students to determine the state space for imperial weights (i.e. ounces, pounds, stones, etc. and distances (inches, feet, yards, furlongs, miles, etc.); for time and dates; possible phone numbers used by telecommunication providers in a particular country and globally;</p>	<p>Using examples from domains that the students are familiar with to explore the inspect/alter idea (e.g. mobile phone contacts list, spreadsheet creation).</p>	<p>Introduce the "roles" applicable to variables and develop the students understanding of their importance. See RoV papers</p>		
<p>Specific ways of ascertaining students' understanding or confusion around this idea (include likely range of responses).</p>	<p>Ask questions like "What are the consequences of two mobile phone top-up machines issuing the same number?"; "How could it be avoided?"; or invite students to explain "How the GPS works" or "Why and how they restrict access to their Facebook page."</p>	<p>Use everyday examples with "obvious" inspect/alter behaviour. Get students to trace short programs. Ask students to explain the purpose and benefits of slow-motion replays. Mavaddat's maze machines and Light Bot "game".</p>	<p>See RoV papers</p>		

5.4 Notional Machine

Albert Einstein once described the dilemma facing a man who is trying to understand the mechanism of a closed watch and noted "He sees the face and the moving hands, even hears its ticking, but he has no way of opening the case. If he is ingenious he may form some picture of a mechanism which could be responsible for all the things he observes, but he may never be quite sure his picture is the only one which could explain his observations" (Einstein and Infeld 1938).

If we are constrained to treating something as a "black box" (i.e. purely on the basis of its external specifications and behaviour) then it is impossible for us to begin to understand how it actually works. Our only hope is to treat it as a "glass box" and seek to comprehend the internal structure so that we can reason about its operation.

This introduces the necessity for the tutor to present a view of the mechanism that allows the novice to understand what is going on. Du Boulay notes "The learner needs a very unsophisticated explanation of what is inside, but she or he does need some explanation." He terms a description of this type a *notional machine*. Du Boulay warns that in the absence of a mechanism offered by the tutor to explain how a computer works novice programmers cannot help but form their own mechanism. The danger is that like the man and the closed watch the novice cannot be sure if the mechanism provided is a correct explanation. Generally, novices develop what Du Boulay termed "impoverished" mechanisms, or models, of how a variety of concepts actually work in computer systems (du Boulay 1986).

The provision of a notional machine is important to avoid the pitfalls of allowing novice programmers to construct their own. Du Boulay and his collaborators describe a notional machine as "an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed". They argue that using a notional machine helps novice programmers overcome their knowledge deficit in relation to what a computer "can be instructed to do and about how it manages to do it." A notional machine is especially effective if it is "conceptually simple" and "visible" so that the novice "can observe certain of its workings in action" (Boulay, O'Shea et al. 1981).

5.4.1 Notional Machines Can Help Learners

One drawback with this idea is that the notional machines, or "match box" machines as Du Boulay has dubbed them, tend to be tied to specific languages. Any language could have a notional machine defined for it and a number of machines has been documented in the literature, each imbued with an abundance of idiosyncrasies foisted on it by the preferences and disposition of the original programming language designers ((Mayer 1976); (Du Boulay et al. 1981)).

Notwithstanding this, Mayer developed a series of studies that showed the use of a notional machine, or model as he preferred to call it, can improve programming performance.

In one experiment two groups were asked to solve a small collection of programming problems, some of which were similar to sample problems provided in the instructional material. One group had been instructed using a model, for a "BASIC-like" language, and the other group had not. Both groups performed well on problems similar to those presented in the instructional material and poorly on more complex problems. However, the "model group" did better on problems that required "moderate amounts of transfer" and Mayer concluded that the model "provided an assimilative context in which novices could relate new technical information" (Mayer 1989).

In a second experiment Mayer provided the model for the same BASIC-like machine to two groups. However, in this case the model was presented to one group before the instructional material was provided and to the other group afterwards. Then both groups were given a recall test. The after group performed well on "retention-like problems" but the before group "excelled on problems requiring creative transfer to new situations." Mayer concluded "subjects who use a concrete model during learning develop learning outcomes that support broader transfer" (Mayer 1989, p.141).

In a variant of the second experiment subjects in the before and after groups were asked to recall all they could about portions of the instructional material. In this case Mayer found that "subjects given the model before learning showed evidence of more integrated and conceptual learning of technical information" (ibid., p.143).

Finally, in an attempt to establish if the results would generalise, Mayer varied the type of language used and presented a model of a "file management language based on SEQUEL."

Again two groups were involved with one being presented with the model while the second group were not. Both groups were then given a 20-item test and as in the previous experiments the model group outperformed the other group "on longer problems that require creatively integrating" the statements of the language.

Mayer's results indicate that providing an explanatory mechanism can be quite successful, especially with respect to solving novel problems. Indeed, Mayer noted that the outcomes "tended to be strongest for low-ability subjects" because they were "more likely to lack prerequisite knowledge" whereas high-ability students "already possess their own useful models for thinking about how a computer works" (ibid., p.146).

5.4.2 Atomistic Thinking

Novice programmers often view their programs as analogous to a human conversation. This gives rise to assumptions that multiple operations are executing in parallel, the system can establish the intention of the program or can apply natural language semantics to the programming instructions (Kurland and Pea 1989, p.323).

When students bring these anthropomorphic analogies and metaphors into their attempts at solving problems using a computer they mislead rather than illuminate. Early work by the programming group at Yale University lead to the development of "bug lists" detailing the stereotypical mistakes novice programmers are in the habit of making and repeating ((Sophrer, Soloway et al. 1989); (Sophrer and Soloway 1989)). For example, novices often misinterpret programming language keywords because of confusion with the semantics of their natural language equivalents. They also write code in the belief that the machine will know their intention and will execute the operations correctly even if they didn't write the operations in the correct sequence. Pea described the confluence of several of these bugs occurring in the conversation analogy as the novice programmers' "superbug" (Pea 1986).

Kurland and Pea have suggested that many of the problems stem from the absence of "atomistic thinking about how programs work (Kurland and Pea 1989, p.323).

Almost a decade earlier Farhad Mavaddat had developed a small set of notional machines for use in his introductory programming courses with the intention of developing his student's atomistic thinking abilities. One of his objectives he describes as somewhat "destructive" because it sought to undo the popular but misleading image of computers. Mavaddat wanted

to show his students that computers were nothing more than obedient servants capable of following exact but incredibly trivial instructions at incomprehensible speeds. He believed that undoing the flawed perception would provide him with a platform to explore the "constructive" aspects of computing that would allow his students to realise that computers are capable of many things but they are not achieved in the way the students currently think (Mavaddat 1981). Mavaddat was trying to encourage and develop his students' atomistic thinking. Appendix A includes some examples of Mavaddat maze machine ideas (A-2.1).

5.4.3 The Turing Machine as a Notional Machine

Notional machines are variants of Turing machines because regardless of what language a notional machine is developed for it has to support the basic operations described by a Turing machine. Alan Turing first described his machine as an aid to investigating the extent and limitations of a theory of computation. He recognised that using a simple abstract device allowed the description to be independent of any specific implementation details. A generic, language independent notional machine offers tremendous utility because it is universal, uniform and independent of the influences of programming language and paradigm. In addition, Turing's machine showed that with just a few simple operations a computing device could be used to implement the logic of any computer algorithm.

A Turing machine is a type of state machine that can be in any one of a finite number of states at any time. The machine is controlled by setting the initial state and specifying instructions that cause it to remain in the current state or change to one of the other states. Thus, a Turing Machine is a notional machine that has only two operations

1. It can inspect the current state of the machine
2. It can alter the current state of the machine

Of course, this is the basis for all of today's computing devices. In some languages the operations are referred to as get (i.e. inspect) and set (i.e. alter).

All programming languages provide statements that implement these two operations regardless of paradigm. For example, even a superficial categorisation of the statements provided in a modern programming language like JAVA would reveal that the assignment statement stands uniquely as the only operation capable of altering the state and all other statements are inspection operations. In languages like JAVA the selection statements provide decision support

on the basis of state inspection(s) and iteration statements are examples of repeated state inspections supported by an assignment mechanism that will eventually alter the state to facilitate termination. If we switch paradigm and consider a logic programming language like PROLOG these facilities are provided by predicates in which only free variables can be altered and bound variables can only be inspected.

It is important to note that the foregoing presentation has not provided a detailed description of the theoretical background or formal specification of Turing machines principally because it is unnecessary to introduce that material to novices. What has been presented is the simplicity of a device that is state-based with just two apparently trivial operations that are completely focused on state.

A Turing machine is a notional machine. It is an "idealized, conceptual computer" with the advantage that its properties are implied by the constructs provided by *any* programming language used to program it. In addition it adheres to the Du Boulay principles of simplicity and visibility. The machine has only two operations (i.e. inspect and alter) and the ability to record the current state of the machine. It couldn't be simpler. The operations and their effect on the recorded state are highly visible because you cannot avoid them. In addition, programming a Turing machine requires atomistic thinking. Appendix A includes some examples of a virtual machine called the LightBot machine which uses simple operations to guide a "bot" through the solution of a grid-style problem that includes some obstacles (A-2.2).

5.4.4 Difficulties with Inspecting and Altering State

The precision required to build something using just two apparently trivial operations is, as (Dijkstra 1989, p.1398) described, a "radical novelty" and without precedent in the experience of most novice programmers. Atomistic thinking is hard work and proves troublesome for novice programmers.

Several studies have tested novice programmers' ability to comprehend the assignment operations required to swap the values in two variables ((Corney, Lister et al. 2011); (Simon 2011); (Murphy, McCauley et al. 2012)). In all cases large numbers of the participants have performed poorly and Simon reports the problem persisting through to the student's third programming course. Swap is a deceptively simple but important operation in computing.

In everyday situations the participants would have absolutely no difficulty swapping things but when set out explicitly as a series of three apparently trivial assignments operations it is a source of trouble. The radical novelty of atomistic thinking can be deceptively hard to conquer.

Donald Norman provides a useful explanation of why our expertise with everyday operations can be a source of troublesome knowledge.

Everyday activities must usually be done relatively quickly, often simultaneously with other activities. Neither time nor mental resources may be available. As a result, everyday activities structure themselves so as to minimize conscious mental activity...These characteristics restrict everyday tasks to those that are shallow (having no need for looking ahead or backing up) and those that are narrow (having few choices at any point, and therefore requiring little planning)...the mental effort required for doing the task is minimized. (Norman 1989, p.125)

We do not reason about everyday tasks. We perfect them to the point of automation. Consequently, we do not recognise or explore in any meaningful way the implications of the concepts packaged in the task.

Marvin Minsky describes how a person asked about stacking blocks is unlikely to even consider and thus mention their ability to distinguish between the blocks already on the stack and those available for use; how a block is chosen for use; or the implications of the stack starting to sway. He refers to this type of knowledge as "layered knowledge", associated with our maturing skills and abilities which make it all seem so obvious and natural.

As time goes on, the layers become increasingly remote until, when we try to speak of them in later life, we find ourselves with little more to say than 'I don't know'. (Minsky 1986, p.22)

In a similar vein William Thurston talks of "compressed knowledge" which, in the context of mathematics, he describes as follows

...you may struggle a long time, step by step, to work through some process or idea from several approaches. But once you really understand it and have the mental perspective to see it as a whole, there is often a tremendous mental compression. You can file it away, recall it quickly and completely when you need it, and use it as just one step in some other mental process. (Thurston 1990, p.848)

The layered or compressed nature of everyday knowledge enables us to "use it as just one step" and the illusion of singularity of action means we rarely ever think about the layers or what has been compressed. We encountered this in Chapter 2 when we noted that humans categorise

and classify a vast variety of stimuli into concepts often in the absence of any consciously available explanation of how they do it.

5.4.5 State as Troublesome Knowledge

State is pervasive in everyday situations. In spoken language we often hear people (parents especially) advising others (children especially) to look at the state of themselves, their room, their clothes, their hair and so on. We talk about the state of the economy, our personal finances, the road system, school buildings, our rivers and the broader ecosystem.

Everyday-processes have state characteristics. We monitor the state of food as it roasts, boils, fries or sets. The food has to be brought to the correct state for each phase of the preparation. These state transitions have to be managed and are dependent on the current state. For example, we apply more or less heat depending on the current state to ensure the next required state is reached correctly. In addition, the sequence in which the state changes are arranged is important. You have to open a bank account before you can lodge or withdraw money and you can only progress through infancy, childhood, adolescence and adulthood in that sequence.

In sporting events the *state of play* is recorded using what we call the score. Indeed, when screening highlights of a sporting event, sports editors restrict the content to occasions when the state changed (i.e. someone scored) or had the potential to change (i.e. near misses). Thus, when we need to review a long process in a short period of time we identify the critical state-changing points and examine the actions at that time to determine how or why the state changed.

We have symbols and rituals for identifying state. For example, females often wear a ring on their ring-finger to acknowledge their betrothed state. Married males and females frequently observe a similar ritual acknowledging their married state. A person's position in a hierarchy is often evident by the uniform and accoutrements they wear or the title by which they are addressed. We talk of their status and our reactions and responses are often dependent on what we perceive that status to be.

Because of the everydayness of state, few of us think of it in explicit or conscious terms. Yet all of the things described above are mimicked in a Turing machine and will be a companion to

every novice programmer throughout their entire computing career, regardless of its longevity. Mastering them has serious implications for the development of their expertise.

5.5 State Space

Edsger Dijkstra ranks as one of the few, and possibly the only, author of programming texts to explicitly and formally discuss the implications and consequences which follow from the fact that computing devices are state based. In his classic text *A Discipline of Programming*, and also in his contribution to *Structured Programming*, Dijkstra sets out the profound intellectual challenge the concept of state presents ((Dijkstra 1976); (Dahl, Dijkstra et al. 1978)).

5.5.1 Vastness of State Space

Following the tradition of mathematics, Dijkstra uses the term "state space" to describe the collection of values associated with a particular scenario or problem. "The state space describes the amount of freedom of the system; it just has nowhere else to go" (Dijkstra 1976, p.11). For example, a problem that involves a single binary digit has a state space collection that contains only two values; a single decimal digit has a state space collection with 10 values; and a Java program that involves a single long integer has a state space with 2^{63} values in it.

When a problem involves more than a single entity the state space is derived from the Cartesian product of the state space of each individual entity. Thus a Java program to solve a problem that involves just two long integers has a state space collection with 2^{126} values in it; three integers has 2^{189} values; and so on. Clearly, the magnitude of the state space can grow very rapidly. In fact, even for very trivial problems the state space can be vast.

During its life-time a solution will perform only a negligible fraction of the potential computations defined by it. However, we have no idea which ones they will be. Despite this a programming solution has to provide the guarantee that it will correctly solve whichever instances of the problem are presented.

It is futile to think that we will be able to test a solution to a problem for every value in the state space. For example, to test a Java program involving just a single long integer for every value in the state space would require 2^{63} tests (i.e. 10^{19} tests). If we assume a test can be performed with a single instruction then a machine that can do a billion instructions a second (i.e. 10^9

instructions) would be able to do 10^{17} tests in a year (i.e. in 10^8 seconds). For a problem involving a single long integer testing every case would take a little over a year to complete. Of course, it is far more likely that a test will require several instructions, possibly even hundreds, so the time would be much greater. We can conclude that except for trivial problems the state space is effectively infinite and we have absolutely no chance of considering or dealing with every single state in it.

State space is also a feature of everyday situations. In everyday conversation we use specific language for articulating, recording or comparing state. For example, the state of the stock market may be described as bullish or bearish; security risks can be coded green, blue, yellow, orange or red; food can be raw, rare, medium-rare, well-done or overcooked.

Thus, associated with each state of interest to us we have a domain of values used for measuring and identifying the possible states. As the examples highlight, the domain of values may be a trivial two-valued set or a more elaborate multi-valued set. The value domain associated with a state may be numeric (e.g. the inflation rate), alphabetic (e.g. someone's name), alphanumeric (e.g. someone's address), boolean (e.g. bullish or bearish), an enumerated list (e.g. security risk identifiers), a set of values (e.g. the lottery numbers), a host of items amalgamated into a single entity (e.g. a Facebook page), or a dynamic collection of any of the foregoing (e.g. the collection of Facebook pages in your Friends list). The domain of values associated with a state could be vast.

When several values contribute to an amalgamated or aggregated state then, as before, the state space is derived from the Cartesian product of the constituent domains. For example, the state of the economy might be measured by reference to the budget deficit, the rate of inflation, the unemployment level and gross domestic product. Enumerating the Cartesian product of just these four items would result in a state space of immense, effectively infinite proportions. Consequently, like software developers, economic model builders are unable to test their models with every possible state to determine which combination yields a target equilibrium.

5.5.2 Useful Structure and Abstraction

The vastness of the state space, whether in everyday or software scenarios, requires us to develop techniques for "mastering multitude and avoiding its bastard chaos as effectively as

possible" (Dahl, Dijkstra et al. 1978, p.6). To achieve this the state space has to be structured, or what Dijkstra more accurately described as "usefully structured," so that we can manage it more efficiently or conveniently or both.

Humans don't consciously realise or appreciate that to be able to cope/manage/handle/deal with the vast array of things that we encounter in our everyday lives we utilise what was described in Chapter 2 as *categorisation* or *classification* (i.e. structuring or grouping related items into concepts like cat, dog, bird and so on). These categorisation skills are highly developed and sophisticated but as was described in section 5.4.4 they are layered and compressed.

Categorisation (i.e. recognising or imposing structure) is a significant example of the inert knowledge novice programmers possess which they find difficult to deploy when it would be useful in programming situations. Because we categorise so effortlessly in our everyday lives we find it difficult to isolate the activity and reflect on it. Novice programmers are confounded by the act of thinking about what and how they categorise things even though their categorisation skills are very sophisticated.

One advantage we have when categorising in everyday situations is that many of the things we categorise are visually and physically tangible so, for clarity, we can (literally) point to the features that encouraged us to put something in a particular category. Thus, putting a particular object in the *ball* category is relatively easy. Similarly, putting the same object in the *beach ball* category or the *base ball* category or the *bowling ball* category will be straightforward.

One drawback in computing contexts is that many of the categorisations will lack this physical and visible tangibility. For example, consider JAVA's categorisation of integers on the basis of how they are represented internally and realised in the programming language notation using keywords such as `byte`, `short`, `int`, `long`, `Integer` and `BigInteger`. This may be the first example novices encounter of the necessity and subsequent utility of introducing structure in a computing scenario. These are primitive categorisations used to structure the potentially vast memory space. The categorisations are based on the role the category plays within the machine or at the machine level. A byte is eight bits; a short sixteen; and so on.

We can use the category *int* in the same way we use the category *dog*. We don't need to enumerate all of the properties. We don't need to say *int*, with 32 bits, stored in two's complement and so on just as we don't say *dog* with four legs, barks and chases cars. Using the category *dog* liberates use from having to specify the details.

Dealing with *ints* liberates us from the machine specifics. The fact that the integer 10 can be in any one or all of the categories is a decision we make (e.g. it may not be appropriate to consider 10 a *BigInteger*), just like the decision we made about the ball (i.e. it's a ball and a beach ball but not a bowling ball).

5.5.3 Abstraction as Troublesome Knowledge

We are so accustomed to making these types of decisions that in everyday scenarios students can make correct, instantaneous decisions without even knowing that they have made them. For example, if students are informed that a taxi can take four passengers and asked to determine how many taxis are required to transport ten people to a party, they immediately respond with the (integer) value 3. If presented with a list of integers (e.g. 1, 2, 4) and asked to calculate the sum, they respond with (integer) 7, but for the average they respond with (real) 2.3 - note, rounded because the 3 is recurring. This is part of inert and tacit knowledge that is compressed, layered and minimised for everyday use.

Novices are also confounded by the introduction of a new terminology. In computing "categorisation" is more usually termed "abstraction" and for many people abstraction is associated with vagueness. However, as Dijkstra noted, the "purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise" (Dijkstra 1972, p.864).

Students' inert and tacit knowledge about categorisation combined with their unfamiliarity with the term abstraction, can make it difficult for them to appreciate the significance of this concept. Just as categorisation is ubiquitous in everyday life abstraction permeates the whole discipline of computing. The primitive data typing of components of the state space may be the novice programmer's first conscious exposure to the fact that "recognizing the sameness among things allows us to expose commonality within key abstractions and mechanisms, and eventually leads us to smaller and simpler architectures" (Booch 1994, p.145). The power of

abstraction makes it our main mental technique because it allows us to partition the vast state space into a useful structure. This will be a persistent theme in the following sections.

5.6 Variables and their Roles

The combination of the notional machine using just two operations (i.e. inspect and alter) and the state space (i.e. the collection of states that may be associated with a particular task) represents the specification of the environment available to programmers for solving programming problems. The specification is implemented using a chosen programming language and, when available, any associated programming support tools, such as libraries, development environments, etc.

5.6.1 Variables

To be able to selectively manipulate the states in the state space we have to be able to uniquely identify them so that during the development of our programs (i.e. our atomistic processes) when the time arrives for us to use whichever one of the inspect or alter operations is most appropriate we can specifically target the state that requires manipulation. This obliges us to establish a mechanism for separating and identifying the states. Every programming language uses the concept of a *variable* for this task.

Consistent with his adoption of the state space analogy Dijkstra described a variable as the "Cartesian coordinates of a state space." This is essentially another name for memory address, which is precisely what a variable is at the machine level - the location in the state space where the state is recorded. A variable, or address, is required for each state that needs to be represented in the state space.

As a convenience high-level programming systems allow programmer-defined names, or variable names, to act as aliases for the state space addresses. Each reference to a defined variable name is a reference to the associated position in the state space where a particular state value is being stored. This makes it much easier to identify and manipulate individual states in a program.

Of course, the manipulations can only be alter or inspection operations. To alter a state you have to replace it with one of the values from the domain of values associated with it. Even

though the domain of values may be vast the recorded state (i.e. the value stored at the location in memory used to record the state) is a single value drawn from the domain of values associated with the state. This is the current state (i.e. the current value of the variable).

Any reference to a variable that is not an alteration operation can only be an inspection operation. If we consider the following variable references we can easily identify the type of operation involved

```
bigger = number > max ; // number inspected, max inspected THEN bigger altered
max = number ; // number inspected THEN max altered
n = n + 1 ; // n inspected, some addition THEN n altered
```

5.6.2 Novice Programmer Difficulties with Variables

The concept of variable is hugely problematic for novice programmers.

For example, encouraging novices to judiciously choose variable names that can assist comprehension of the state manipulations can lead to the "superbug" phenomenon as they attach intent to the names and assume that the machine knows what they mean as well (du Boulay 1986).

Madison documents an account provided by a tutor of her "variable instruction" which includes an "elaborate definition" and several example analogies and metaphors. The concept of variable is introduced early in the course and the teacher assumed that students had "a principled understanding of variables." However, later in the course Madison interviewed eight of the students and discovered that "the expectation was only partially borne out." Most of the students mentioned "change" but only two of them "referred to a variable as a memory location" (Madison 1995).

Students with a mathematical background can have problems with the meaning associated with the term variable. Mathematicians use the term variable to mean something that represents no specific value at all whereas programmers use the term for something that **always** has some value, which may include the value "undefined" or "unusable" (e.g. a null reference). In addition, the value of a programming variable is temporal and once it has been assigned a value (i.e. set to a specific state) it stays like that unless we explicitly alter it. This is an important antidote to the "superbug" phenomenon because it crystallises the obligation on the

programmer to assume all responsibility for ensuring that the variables always have the required states to ensure the atomistic process completes satisfactorily.

(Samurcay 1989) used a number of categorisations to examine student difficulties with the concept of variable. In one case he distinguished between "external" variables and "internal" variables. External variables "are values controlled by the program users" such as variables used for storing user inputs or system outputs. Internal variables are "values controlled by the programmers: i.e. variables which are necessary only for the programmed solution of the problem." Students found internal variables "conceptually more difficult" because they require "representation of the computer operation in terms of systems states" whereas the meaning of external variables was "more easily constructed" (Samurcay 1989, p.165).

Input and output operations are not without their problems. Students sometimes confuse single character variable names and the character itself. For example, Bayman and Mayer found that students misinterpreted statements of the type INPUT A to mean that the letter A would be stored in memory. Similarly, they could not understand the difference between the statement PRINT C and PRINT "C" (Bayman and Mayer 1983, p.678).

5.6.2.1 Problems with Assignment

The basic mechanism for altering or changing state is the *assignment statement* and (Samurcay 1989) distinguished four types of assignment which he labelled (1) constant (2) calculated (3) duplication (4) accumulation and which can be exemplified as follows

CONSTANT	CALCULATED	DUPLICATION	ACCUMULATION
a = 3 ; found = false ; word = "Java" ;	a = 3 + 5 ; b = 3 * k ; r = Math.rand() ;	k = m ;	x = x + 5 ; n = n + 1 ; sum = sum + number ; expox = expox * x ;

Obviously all of the assignment categories involve alteration operations since that is their purpose but, with the exception of the constant category, they also include inspection operations in the expressions on the right hand side of the assignment operator.

All four types are problematic for novice programmers, as reported by various authors.

Novice programmers' interpretation of assignment is often reversed and $k = m$ is interpreted as the value of m is changed to the value of k which leads to erroneous conclusions about what is being altered, or duplicated ((Putnam, Sleeman et al. 1989, p.304); (du Boulay 1989, p.290)). In addition, when writing assignments involving a constant many novices reverse the positions of the variable name and constant value resulting in $3 = a$ ((du Boulay 1989); (Putnam, Sleeman et al. 1989)).

A notational style that specifies the computation followed by the result is part of the ritual knowledge students have developed when learning to solve computational problems.

When a solution is required for a particular problem the details of the problem have to be specified first. This allows the relevant information contained in the problem specification to be identified. Using their problem solving capabilities the person attempting to solve the problem attempts to determine the result. This sequence is embedded in any query related to a computation because one has to vocalise or specify the required computation first to enable the result to be computed. In the case of arithmetic problems the task facing the problem solver is to apply the computational rules correctly.

When learning their tables by rote during their early schooling children recite "2 multiplied by 7 equals 14" to vocalise what had been printed in their Tables book as $2 \times 7 = 14$. Here the computation is on the left hand side of the "equals sign" and the result is on the right.

The syntax of the programming language assignment statement is not adhering to the conventions for problem specification. It and not the student appears to be reversing the sequence. The assignment statement is inconsistent with the students ritual knowledge and is troublesome.

One of the difficulties with assignment is that often the symbol used in programming languages to specify it is the same symbol students typically use in everyday scenarios to specify equality. Thus, $k = m$ is read as k is equal to m . As a consequence, calculated assignments are sometimes interpreted as operations to store the expression or equation instead of the result. Thus, the statement $b = 3 * k$ assigns the formula or arithmetic expression $3 * k$ to b and not the result of computing $3 * k$ (Bayman and Mayer 1983). Alternatively, an assignment may be viewed as incomprehensible because the equality rule cannot be affirmed by any logical means. For example, Putnam et al report that a study participant found the operation $c = c + 1$ "impossible"

to interpret because in his program he had previously assigned the value zero to c and the computation $0 = 0 + 1$ did not make sense (Putnam, Sleeman et al. 1989, p.304).

This troublesome notational nuance has encouraged many programming language designers to try and circumvent it by using a modified version of the $=$ symbol (e.g. $:=$) or by using a different symbol altogether (e.g. \leftarrow) or by using an alternative symbol for equality (e.g. $==$).

In addition, the term "assignment" is linguistically misleading because the concept of assignment in programming languages is widely different from what natural language users of the term mean by it. In a program the assignment operation is used to alter, or overwrite or replace, the current state of a variable. Consider the following sample collection of definitions for the word assignment: (1) to set apart for a particular purpose; designate (2) to select for a duty or office (3) to give out as a task; allot (4) to ascribe; attribute (5) to transfer (property, rights, or interests) from one to another (6) to place (a person or a military unit) under a specific command. None of these have any sense of altering or overwriting or replacing, with the possible exception of a transfer of ownership, which is an interpretation that would be completely at odds with assignment in a programming context.

Notwithstanding the notational difficulties associated with the assignment operator (i.e. $=$), reading $c = c + 1$ as "c is altered to $c + 1$ " is far more accurate than the widespread practice of reading it as "c is assigned $c + 1$ ". The first reading has the effect of emphasising the fact that the variable on the left hand side is the one being altered. It also invites consideration of what it is going to be altered to and, presumably, we will need to know the answer to that *before* we can do the alteration. This implies that the expression on the right hand side has to be calculated *first*. Calculating the value of the expression on the right hand side involves an inspection operation on the variable c . Once the value of c has been inspected and used to calculate the result only then can the alteration operation be executed. Now the asymmetrical and sequential nature of assignment is being exposed and the dilemma faced by someone who finds $c = c + 1$ impossible to understand can be resolved.

Assignment is troublesome for novice programmers.

5.6.2.2 Initialising and Using Variables

The last example, $c = c + 1$, provides a very simple but important lesson. The machine can only inspect or alter the states recorded by the variables but by combining those operations we

can get it to do a variety of things. The combination $c = c + 1$ causes the machine to count but other combinations can achieve other outcomes.

To examine novice programmer conceptions of variables (Samurcay 1989) devised an experiment that obliged novice programmers to complete four unfinished programs by adding appropriate statements at the required locations in the code. In the first two programs the subjects were required to insert appropriate initialisation operations (i.e. alteration); the third problem required a loop termination test to be inserted (i.e. inspection); and the fourth problem required some initialisation, counting and multiplicative accumulation, as well as a loop termination test (i.e. a number of alteration and inspection operations).

Samurcay found that initialisation was the operation that proved the most difficult for the participants. They tended to initialise internal variables (i.e. variables used for computation, not input and output) using input operations instead of assignment. Accumulating a counter was easier than a more general accumulation because it was more obvious that the increments should be 1. The other type of accumulation was more difficult because the method for increasing the total involved multiplication and not just addition.

Samurcay makes some interesting observations about initialisation which are centred on the participants ritual and inert knowledge of problem solving. He associates the participants tendency to avoid initialization with the fact that it is an operation that is done automatically in hand-solutions and "awareness by the subject is not needed." In respect of counting variables he notes that the participants have a "canonical schema constructed about the counter variable" which ensures that it starts at zero and is incremented by one. The absence of a similar schema for the more general accumulator explains their inability to reliably initialise and increment it. Samurcay describes initialisation as "a complex cognitive operation" because of the difficulties associated with "making an hypothesis about the initial state of a system" (Samurcay 1989).

In a state-based machine (e.g. a Turing machine or a notional machine based on it) the initial state of the machine is absolutely crucial. If the machine is not in the correct initial state then it does not matter how ingenious the operations which manipulate it are, the whole project is doomed to failure. Many programming languages automatically initialise variables to "expected" or "reasonable" values drawn from their value domains to ensure that the initial state of the machine is at least discernible, although not necessarily what is required. It is

possible to reason from a discernible but erroneous initial state, if only to show that the discernible initial state is erroneous. It is impossible to reason from an indiscernible initial state. Appendix A includes some examples of the importance of initial state for traversing a maze and the difficulties that can arise in even simple cases when the initial state is ambiguous (A-5.1).

The distinguished and privileged status of initialisation is recognised in object-oriented programming languages by the pivotal role played by constructor methods. In the absence of a constructor method the system will usually resort to "reasonable" initialisation of the object instance variables.

5.6.2.3 Using Variables in Plans

Combinations involving a small number of alteration and inspection operations can form a component of a program and, as Samurcay has noted, "the concepts of variables and assignment take their full programming meaning" in these cases (Samurcay 1989, p.164). Early work by Soloway's programming group at Yale showed that expert programmers possessed a collection of such components or plans that they used and merged to form more sophisticated components and plans (Sophrer, Soloway et al. 1989). Dijkstra used the analogy of the collection of proofs in plane geometry that mathematicians draw on when required, without reformulating them *ab initio*. Plans are a set of tools whose usefulness has been generally accepted and a competent programmer possesses as part of their tacit programming knowledge (Dahl, Dijkstra et al. 1978, p.10).

The collection of plans includes combinations of alteration and inspection operations to swap or exchange the values in two variables; to aggregate (e.g. sum, average) the values in an input stream or list; to find the largest or smallest value in an input stream or list; and so on. When one is immersed in the daily activity of using these combinations the pattern of use and the resulting outcome appear self-evident. However, when one is encountering them for the first time they can be daunting.

(Meerbaum-Salant, Armoni et al. 2010) investigated whether the visual programming language Scratch could be used as a means of facilitating the learning of CS concepts. Scratch is a high-profile development system aimed at encouraging young people develop an interest in computing and programming. The authors developed a set of Scratch based learning materials specifically intended to support the investigation. The materials were structured so that each

section was aimed at teaching a specific CS concept and designed according to the constructionist philosophy of Scratch which promotes self-learning and learning-by-making exploration. The materials sought to "de-emphasize aspects of appearance" which often dominate visual programming environments like Scratch. The results showed that participants had problems with initialization, variables and concurrency, three important CS concepts.

The authors note that "Unlike many other CS concepts which appear in an unusual guise in Scratch, the support for variables in Scratch is quite similar to that in regular programming languages: One has to declare a variable (as either global or local to a sprite) and then assign values using blocks such as set or change...It seems as if this concept is less likely to be acquired by self-learning in a trial-and-error manner" (Meerbaum-Salant, Armoni et al. 2010, p.74). They concluded that the use of variables was problematic because initialization requires more general cognitive skill beyond programming and variables involve relationships between instructions whereas many other concepts involve just a single instruction.

Several studies have highlighted the difficulties novice programmers have with *relational reasoning* and providing explanations of what code does in terms of its purpose as opposed to a line-by-line description. Relational reasoning obliges the novice to consider the contribution of each variable to, and the affect of each operation on, the state space. Relational reasoning synthesises the variables and operations to establish the effect or function of the code. Poor performance with relational reasoning tasks early in an introductory programming course has been shown to be a marker for poor programming ability later ((Murphy, McCauley et al. 2012); (Lopez, Whalley et al. 2008); (Corney, Lister et al. 2011); (Simon 2011)).

Analyses of publicly available Scratch projects consistently emphasise the low utilisation of variables in projects ((Maloney, Peppler et al. 2008); (Dahotre, Zhang et al. 2010); (Meerbaum-Salant, Armoni et al. 2010); (Meerbaum-Salant, Armoni et al. 2011)). In this regard Scratch would appear to be mimicking its equally high-profile predecessor LOGO. Pea and Kurland noted that developmental studies of LOGO programming skills showed that even participants extremely interested in learning to program wrote programs that reached a moderate level of sophistication and their grasp of the concept of variables was highly context-specific (Pea and Kurland 1984).

5.6.2.3 The Role Of Variables

Despite this, as (Samurcay 1989)observed, some of the stereotypical plans associated with variables resonate with novices. Thus, the participants in Samurcay's experiment exploited their "canonical schema" about counters to correctly initialise the counters to zero and increment them by one. In this case the role of something classed as a counter made it possible to predict the requirements for handling it correctly and to comprehend the pattern of usage in an existing program.

Sajaniemi's development of the *roles of variables* idea exploits this connection. Sajaniemi analysed all of the programs in three introductory programming textbooks and noted that there were "stereotypic usages of variables that occur in programs over and over again" (Sajaniemi and Kuittinen 2005, p.60).

The focus of the analysis was on the dynamic nature of the variables and how their values (or state) changed over time. From the analysis Sajaniemi was able to determine the role played by the variables in a program. He identified ten roles that cover 99% of variables used in "novice level programming" (ibid.). The roles identified include *stepper*, *gatherer*, *follower*, *most-recent holder*, *most-wanted holder*, *temporary* and *fixed value*. The three most frequently occurring roles (i.e. *fixed value*, *stepper* and *most-recent holder*) accounted for 84% of the variables used.

Roles are cognitive rather than physical entities. The role of a variable is an abstraction of its dynamic behaviour in a program. The roles succinctly articulate the ways in which the behaviour of the notional machine achieves the type of dynamic activity associated with the role. Roles provide a model of the state changes that occur in the state space that is associated with a particular variable. Roles tell us what variables do rather than how they do it. Naming them appropriately can reinforce that information.

Explicitly teaching roles of variables to novice programmers has yielded very positive results. In one study Sajaniemi and Kuittinen found that learners had a "better vocabulary for talking about programs" and a "conceptual framework that they could use to better process program knowledge." They explain the benefits thus

"in the traditional approach, a variable has no special meaning by itself but is only the object of some—to a novice apparently more or less incidental—assignments. Role knowledge turns this situation upside down: a variable is an active subject taking care of some specific task and all assignments can be seen to support this

task. Thus the roles make the deep program knowledge more accessible to students" (Sajaniemi and Kuittinen 2005, p.79-80)

5.6.3 Variables and Troublesome Knowledge

Dijkstra has observed that "once a person has understood the way in which variables are used in programming, he has understood the quintessence of programming" (Dahl, Dijkstra et al. 1978, p.11). Novice programmers struggle with the concept of variable. Difficulties arise because the notation used can be counter-intuitive; the language of description is nebulous; crucial operations like initialisation are tacitly known; and recognition of the net effect of dynamic state change may be inert.

Stereotypical usage of variables in plans is a consequence of the operations provided by the notional machine (i.e. alteration and inspection) and achieving the required state changes to a variable can only be effected in predictable ways. For example, counting and its more general equivalent of accumulation or summation cannot be achieved without a state alteration of the type $c = c + n$. The capabilities of the notional machine influence the strategies that can be adopted by programmers. Expert programmers are aware of this influence and know the strategies that are required to harness it and make use of them as part of their tacit programming knowledge.

Variables provide an interface to the state space and facilitate targeted state alterations and inspections to a single element of it. By abstracting the dynamic behaviour of a variable its role in a program can be identified. This facilitates program construction and comprehension. Roles of variables can document and articulate tacit programming knowledge.

5.7 Methods and Parameter Passing

Identifying the individual roles of variables helps to partition the state space into manageable segments whose abstract behaviour we are able to comprehend. However, as the number of variables increases the multiplicity of variable roles becomes a surrogate for the multiplicity of states. Even a modest number of variables will tax our mental ability to cope with what we might refer to as the burgeoning role space (i.e. the Cartesian product of the roles).

As noted in section 5.6.2.3 above, plans involving combinations of assignment statements on a single variable are useful building blocks. They can be composed in sequence, or merged, into a single structure and understood as a separate single component that implements a functionally useful task (e.g. display something on the screen, check membership of a list, generate a random number in a specified range). In these cases the interaction of a small number of variables can be understood completely independently of the other variables in the state space - it is not even necessary to know that the other variables exist.

With the exception of trivial problems this type of separation, or factorisation, allows programming solutions to be built, understood and reasoned about as networks of interacting components, or *methods*. Whilst it is possible to write a 10,000 line monolithic program, the use of factorisation is more likely to produce 1000 ten-line program methods that are interwoven to create a solution structure that is comprehensible and manageable. Factorisation localises the manipulation of the state space associated with the variables involved in a functionally useful task and reduces the cognitive load on the programmer trying to comprehend the task (Dromey 1982). Appendix A includes some examples of student's intuitive uses of methods, including parameters, in the solution of maze problems.

5.7.1 The Linguistics of Parameter Passing

Managing the interactions of the methods is one of the most challenging aspects of programming because the methods manipulate the state space to produce the desired result. Interactions that erroneously or inadvertently alter the state can be difficult to locate and correct, especially in large or sophisticated component networks. Interactions are managed and controlled by the careful use of programming language features for data sharing between methods. Methods and data sharing are notoriously problematic concepts for novice programmers ((Madison 1995); (Fleury 1991)).

Language is one source of trouble. The term "parameter passing" is used by programmers to identify the act of data sharing between components. The problem is that the terminology is misleading because it is grounded in the language of the system software (i.e. compiler) and not the programmer.

When a compiler encounters a method invocation involving information sharing (i.e. one that includes parameters) it has to decide how to deal with it - it has to decide how to make the

state information stored in the variables of one component (i.e. identified by the parameters) available to the other component. Unfortunately the details required to achieve this are not specified at the point of the invocation. This is an extremely important point - how the information is to be shared is not specified at the point of invocation; at the point where the decision to share has been made; at the point where the component offering to share information acts on its offer.

How the information is shared is actually specified in the header of the method being invoked. Thus, the method being offered access to the state information stored in another component is the one that decides how that access will be effected. Access to the state space of a method is being offered to another method and control of how it will happen has been relinquished to, or bestowed upon, the recipient.

The compiler has to determine the sharing mechanism specified in the recipient method header and then has to generate the code necessary to "pass" the data from the point of the invocation to the invoked method, and also create a mechanism to enable program execution to resume at a point immediately following the invocation.

The "passing" metaphor has had the affect of creating its own vocabulary. Parameters are referred to as *actual* parameters before they are passed and *formal* parameters when they are passed. In addition, parameters can be passed by *value* (i.e. copied) or by *reference*, depending on how they will be used by the receiving method. Students develop mental models based on information "being passed into the procedure...and then it is passed back out" (Madison 1995, p.95) and explain problems on the basis that "you don't have any way to send them back" (Fleury 1991, p.284).

Many introductory textbooks go to great lengths to explain the process and they detail the stack of activation records that is created as well as providing pictorial representations of the mechanism in operation. This is an explanation of parameter passing, which is not a trivial task, but it is a task facing a compiler and as such is of very little interest to a programmer because it is about the mechanics of compilation and implementation as opposed to the purpose of useful programming features. It has a place in a programming languages course but not an introductory programming course.

A programmer's interest in parameter passing is principally concerned with how the receiving method can affect the information being shared with it - what it can do, or needs to be able to do, with the information it is receiving. This is a really important task for the programmer because how the method manipulates the program state will have consequences for subsequent operations. This is the programmer using abstraction to stay focused on the problem solution instead of the mechanics of machine operation.

5.7.2 The Linguistics of Parameter Receiving

Of course, there are only two things the method can do with the shared information because there are only two operations available for manipulating state. The method can either inspect the information or it can alter it. It could do both but the net effect would be equivalent to an alter operation so the three possibilities are covered by the first two.

When a system is being designed the functional goal of a method will determine whether information that is shared with it needs to be inspected or altered or both. The method header will indicate the outcomes of those decisions. In the header the programmer will specify the potential effects that use of the method can have on the information shared with it. The method header is actually a specification of the information *receiving* requirements and the potential consequences for the information of using the method. In essence, the method header indicates what can happen to the program state when the method is used.

Invoking a method leverages the contribution the method makes to the problem solution. The detail of how that contribution is realised is hidden in the method. For the method to be able to make its contribution it has to be able to access the state space and inspect or alter it appropriately. The only component that can specify or articulate the type of access necessary to allow the contribution to be made is the method itself. That is the role of the method header - to specify the access necessary to realise its contribution.

The role of the method invocation is to identify the variables in the state space that the receiving method should manipulate. This explains why a method invocation does not specify how the information will be shared.

Labelling this process parameter passing completely distorts its focus. Parameter *passing* is a compiler task. Parameter *receiving* is a programmer task and is all about specifying the type of access requirements required to achieve a particular goal. Parameter receiving advises

potential users of a method what they are committing themselves to in terms of program state effects. Sight of the receiving method header is sufficient to inform a potential user of the method whether the program state will be protected or exposed by use of the method.

5.7.3 Trouble with Parameter Passing

Parameter passing causes many difficulties for novice programmers. (Fleury 1991) interviewed twenty-three novice programmers taking a university level introductory programming course. The participants had been tutored by four different instructors. The participants were shown a working program and six other "working and almost working" variants of it. The almost working versions had problems related to "parameter passing or variable access." Fleury found that "rote-learning" of rules associated with parameter passing "contributed to student confusion." In addition, the same student misconceptions were evident throughout the group despite the variety of instruction sources. These misconceptions were formed from "overly strict (and/or slightly misguided)" interpretations of the parameter passing mechanisms of the programming language. Even successful students admitted to having guessed some of their correct answers. In addition, Fleury noted that novices can produce working programs by including "extraneous declarations or inappropriate uses of parameters". For example, substituting a reference parameter for a more appropriate value parameter may not produce an erroneous result but it may introduce a problem elsewhere in the program.

In a separate study Madison interviewed eight students enrolled on a university introductory programming course. She also observed the class instruction, interviewed the instructors about their teaching strategies and examples as well as reviewing some of the students programming work. Madison found that students' understanding of parameter passing "was fragile at the end of their introductory course." Their conceptual understanding of parameter passing was surpassed by their ability to write code that incorporated parameter passing techniques. Successful coding was often possible by mimicking working examples or, like Fleury's observation, making "seemingly innocuous adjustments to procedure heading lines" that produced the correct results. The most commonly observed mistake was the students inability to distinguish between the use of a value or reference parameter. Invariably students used a reference parameter because of the benign affect it appeared to have on compilation errors (Madison 1995).

Although not specifically studying parameter passing a number of authors have observed student misconceptions regarding parameter passing appearing in their data. (Ragonis and Ben-Ari 2005) report that students felt uncomfortable with parameters and had difficulties understanding where parameter values came from. (Hristova, Misra et al. 2003) found that students were confused by the requirements to identify the type of a parameter in a method header but not in a method invocation. (George 2000) noted that students had misconceptions about the values assigned to parameter variables both during and after method invocations.

5.7.4 State and the Pedagogy of *Parameter Receiving*

One of the advantages using the Java programming language is that it avoids the dilemmas associated with making copy or pass by reference decisions. Java has a very simple parameter passing mechanism which only allows data copying and eliminates the ability to pass by reference. This simple plan is rationalised on the basis of its ability to guarantee the reliability of the manipulation of the state space. All state space changes are local to the method or, in cases where an object is passed, controlled by the availability of set and get methods providing disciplined manipulation of the object state. The significance of this insight cannot be overstated because, like roles of variables, it is an inert piece of expert knowledge.

In other cases complete control may be assigned to someone who is trusted. For example, people often assign a power of attorney to a trusted family member, friend or an acquaintance working in a professional capacity on their behalf. Similarly, when interacting on the Internet users can choose to allow *cookies* be stored, retrieved and updated on their machines to facilitate secure and intelligent interaction with another site. The use of cookies is, in fact, an imposition of state on what would otherwise be a stateless mechanism. The user in receipt of a cookie request makes a decision to trust the site they have connected to and give it permission to inspect and alter a package of data stored on their machine.

This is a dynamic interaction so the request for permission and the granting or denial of approval is handled appropriately as an ephemeral act and the interaction continues in a style configured by the cookie decision. However, in the context of a static program compilation where only one party can make decisions (i.e. the compiler) an attempt to use a method inappropriately can only be denied by the compiler and the whole compilation process has to stop. To allow programmers make appropriate use of a method the interaction requirements

are published publicly so that a decision to use the method or not can be taken in advance of compilation. Publishing the interaction details simply involves publishing the method header because it contains all the interface requirements and the potential consequences for the information involved.

Information sharing decisions are asymmetrical and favour the recipient of the information. Unfortunately, historical baggage associated with solving the technical difficulties of making flexible information sharing possible in programming systems shifted the focus to how the information could be sent or passed. This has had the effect of situating the concept of information sharing in the wrong framework, in the framework of parameter passing. The correct framework for reasoning about information exchange focuses on how the information needs to be used, how the recipient plans to handle it and how handling it will impact on the state space of the program. This is a more coherent mental model of information sharing than that provided by parameter passing.

This is consistent with the presentation of the resources in a library as a list of method headers. The only things the library developer needs to share with the library users are the mechanisms for using the library and the consequences of those mechanisms. This type of description is presented in the JavaDoc resources provided to Java programmers. The user simply knows what to do but not how it is achieved. This is a very valuable lesson for novice programmers.

Of course, well designed libraries and methods don't have a widespread impact on the state space. In the main the impact is localised and side effects are frowned upon because they can interfere with the programmers efforts to control the alterations to the state space. So programmers may make a decision not to use a particular feature because of its implications. In contrast, a well designed library will limit the impact on the state space and this will encourage programmers to use the facilities because they know they are well behaved. This is a discourse about program design not about the mechanics of activation records and stacks. This is the discourse we want our students to be involved in. This is a discourse that will sustain the development of their programming expertise. This is a discourse about state.

Program design is about managing program state, about partitioning it into a useful structure that is comprehensible and robust. Program reliability is dependent on controlled, secure access to program state. The pedagogy of parameter receiving is a more appropriate

framework for presenting program design to novice programmers than the pedagogy of parameter passing.

5.8 Debugging

Brian Kernighan once observed that debugging a program is twice as hard as writing it in the first place. That might explain why novice programmers often replace or rewrite erroneous code rather than fixing it ((Nanja and Cook 1987); (Ahmadzadeh, Elliman et al. 2005)). To make a repair you have to be able to understand the programmer's intention and what it is that has prohibited the intention from being realised. This is difficult because a programmer has to apprehend two things at once. The first is the static text of the program which documents the underlying structure of the program; and the second is the dynamic state of the evolving execution of that text. The shorter the conceptual gap between the two the better. Obviously, the gap can be eliminated altogether if you can identify the point in the text where the programmer's intention is *not* being realised and can replace it with an alternative that realises the intention. Otherwise you have to make the effort to comprehend what is happening dynamically in time from the information laid out sequentially in the program text. You have to be able to follow and comprehend the program's traversal of the state space.

5.8.1 Close Tracking

(Perkins, Hancock et al. 1989) describe this process as "close tracking" that maps the effects of the program text onto "a status representation." The creation of a status representation is straightforward for novice programmers learning in graphic-based environments like LOGO or SCRATCH. The low utilisation of variables in these types of projects (see section 5.6.2.3) to record internal program states results in the graphic display becoming a readymade visual status of the progression of the computation. Errors are visible in identifiable flaws on the screen display and learners can actually see the program state unfolding by observing the output on the screen (Lewis 2012). The program state is visually exposed and debugging involves ensuring the state changes to the display are consistent with the programmer's intention.

When there is no graphic status representation available an alternative has to be created. In these circumstances Perkins et al note that the "status is best conceptualised as a matter of the

values of the variables and the appearance of accumulated output at a given point in time; the student must interpret accurately how each piece of code adds to the output and alters the value of a variable" (Perkins, Hancock et al. 1989, p.269). Thus, the programmer has to create some physical or mental representation of the current state of the state space.

Close tracking is not easy. Lister et al found that novices' abilities at systematically carrying out "tracing" tasks was "fragile" and they recommended that students are first taught "systematic tracing as a base skill" which would allow them to build "higher-level comprehension skills upon that base" (Lister, Adams et al. 2004). (Kaczmarczyk, Petrick et al. 2010) also report that an inability to "trace code linearly" was a major source of difficulty for novice programmers.

(Perkins, Hancock et al. 1989) identify three types of reaction to debugging problems which manifest themselves in novice programmers.

- Some learners just stop. They make no effort whatsoever to investigate or comprehend why the program fails to execute as required. These are classified as "stoppers".
- "Movers" try different things. They makes changes to the program and try them out. If one option fails they try something else. In most cases this is good. Tinkering with the code can reveal a silly assumption or incorrect initialisation.
- "Extreme movers" tinker based on thoughtless alterations and can get themselves mired in a vicious circle of ineffectual changes because they try too many things without reflecting on what they are changing and where it is leading them.

Movers sometimes project their expectations on to the text instead of reading it clinically. Perkins and his colleagues used the term "extrapolating impressionistically" to describe this type of behaviour. In contrast, good debugging requires the text to be read as it is actually written and with an eye to form.

Effective tinkering involves close tracking of the state of the variables in the program. As Perkins et al note "Those rare students who track very closely are not tinkering at all; they know exactly what is going on" (Perkins, Hancock et al. 1989, p.272).

They track the state changes to the program state space meticulously. They are patient and not perturbed by the simplicity of individual alter and inspect operations. They recognise idiomatic

plans and stereotypical uses of variables but they still read the operations as they are presented in the program and verify that they are being correctly deployed. They record the changes to the affected variables in the state space and they assimilate this information with what the program is supposed to do. They confirm that the current state is the required state to realise the programmers intention. They continue in this vein until they locate the problem. And then they fix it because they comprehend the error and can identify the correct state to realise the programmer's intention. In many environments this type of activity is supported by debugging tools that assist with creating and maintaining the "status representation" (i.e. the values of the variables in the state space).

When debugging, the concept of state is unavoidable.

5.8.2 A Good Understanding of State Makes Good Programmers

Ahmadzadeh et al did a study of 155 novice Java programmers in which they were given a program that had some "compiler errors and logical mistakes" introduced into it. The participants were given two hours to correct the program. Some 59 of the participants had scored more than 70% in their programming exams and were categorised as good programmers, but only 23 of them (39%) corrected the errors and were classified as good debuggers. The authors concluded that "many students with a good understanding of programming do not acquire the skills to debug programs effectively, and this is a major impediment to their producing working code of any complexity. Skill at debugging seems to increase a programmer's confidence and we suggest that more emphasis be placed on debugging skills in the teaching of programming" (Ahmadzadeh, Elliman et al. 2005, p.87).

(Fitzgerald, Lewandowski et al. 2008) used a modified replication of an experiment originally developed by (Katz and Anderson 1987). In the study 21 novice programmers in the second part of their Java programming course completed a programming exercise. They were then presented with a "buggy version" of a solution to the same problem and given 20 minutes to remove the errors. Participants found 70% of the errors and fixed all bar two of them. One of the errors not fixed was an incorrect variable initialization. The results showed that "novice programmers now are similar to those in the past." The authors were also pleasantly surprised to have their "anecdotal expectations as computing instructors" contradicted by the finding that once participants found an error they were usually able to fix it. Finding them was the greatest

challenge and the authors recommend that "guidance and practice with bug location should be emphasized." Like Ahmadazdeh et al the study revealed that students who were good debuggers were also good programmers but not all good programmers were good debuggers (Fitzgerald, Lewandowski et al. 2008).

The Ahmadazdeh et al and Fitzgerald et al findings show that the development of programming expertise can be hampered by a poor understanding of state and state space. In the studies even some good programmers experienced liminality and stuckness in relation to state. In contrast, the solid understanding of state exhibited by the good debuggers was shown to be a marker for good programming ability.

5.8.3 A Poor Understanding of State Makes Bad Programmers

In programming scenarios where the concept of state is neglected the consequences can be significant. For example, as part of their efforts to use the Scratch programming language to teach computing science concepts (Meerbaum-Salant, Armoni et al. 2011) and her colleagues analysed projects developed by their course participants and discovered that the students were developing two programming habits "very much at odds with accepted practice...which could be detrimental to learning programming." The habits led to poor design and difficulties with testing and reliability.

The first they described as *programming by bricolage*. When presented with a programming task the students simply collected all the operations they thought would be useful for solving the task and placed them on their programming workspace. Then they started combining them into a script or method. As each combination was assembled it could be tested and viewed on the screen to determine if it was achieving the required outcome. Program development was guided by the visual effects on the screen.

The second they termed *extremely fine-grained programming*. In this case the students developed very large numbers, sometimes hundreds, of scripts which "usually lacked logical coherency." They provide some examples, one of which is included here. In a game a missile moves until it touches a target.

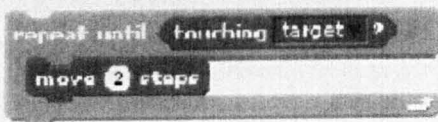
A simple implementation, provided by Meerbaum-Salant and her colleagues, is shown below on the left hand side. A loop structure is used to move the missile until it reaches the target. This

implementation exploits the fact that the position of the missile (i.e. the state of the missile stored in variables associated with the missile) is altered and updated by each move operation.

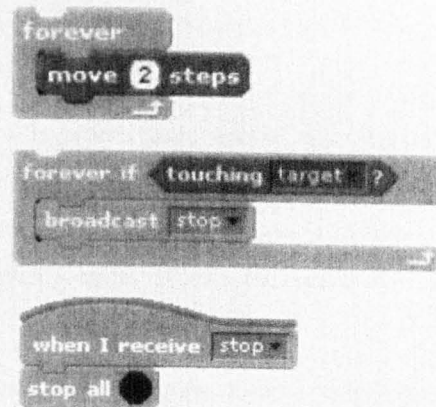
A sample student solution is provided on the right hand side. The strategy used by the simple loop used in instructor's solution has been partitioned into three separate scripts where control of the execution is external, or global, and heavily dependent on the concurrent features of Scratch. Concurrent execution is not simple to understand and actions can be interleaved in ways that are unwanted or unexpected.

Spreading the simple loop structure over three distinct scripts increases the cognitive load on the programmer and makes it very difficult to read and understand. Indeed, the authors note that their students were "frequently helpless when faced with unanticipated problems caused by concurrency issues" and that the massive use of concurrency made the programs difficult to debug. This led to a paradox where "the motivation that results from the ability to program interesting games can dissolve when the debugging process becomes difficult and frustrating" (Meerbaum-Salant, Armoni et al. 2011).

Simple Solution



Sample Student Solution



These findings replicate the findings presented in evaluations of LOGO, Scratch's predecessor. (Perkins, Hancock et al. 1989) point out that visual programming environments support the production of relatively complex outputs in the absence of structured and disciplined use of high level programming concepts. This is a double-edged sword because "it makes programming accessible even to rather young children, but does so through a simple linear pattern of thinking that does not generalize well to more sophisticated problems...such

programming is not cognitively demanding, but requires mainly stamina and determination" (Perkins, Hancock et al. 1989, p.274).

The two habits documented by (Meerbaum-Salant, Armoni et al. 2011) are a consequence of an absence or ignorance of state. As noted earlier (see section 5.6.2.3) Scratch users tend to avoid the use of variables and rarely create localised state space associated with the objects they create. Instead they rely heavily, in truth almost totally, on the external state provided by the screen display so that their work is more akin to a videogame-like manipulation of the two-dimensional screen space than a structured, algorithmic solution to a problem . This leads to an impoverished concept of state and the requirement to record state in a script or method.

Programming by bricolage predisposes novices to focusing on the external, visible state. There is little if any cognisance of the internal state and this leads to extremely fine grained programming as the novice tries to cobble together some (any) collection of operations that will make the display (i.e. the external state) correct.

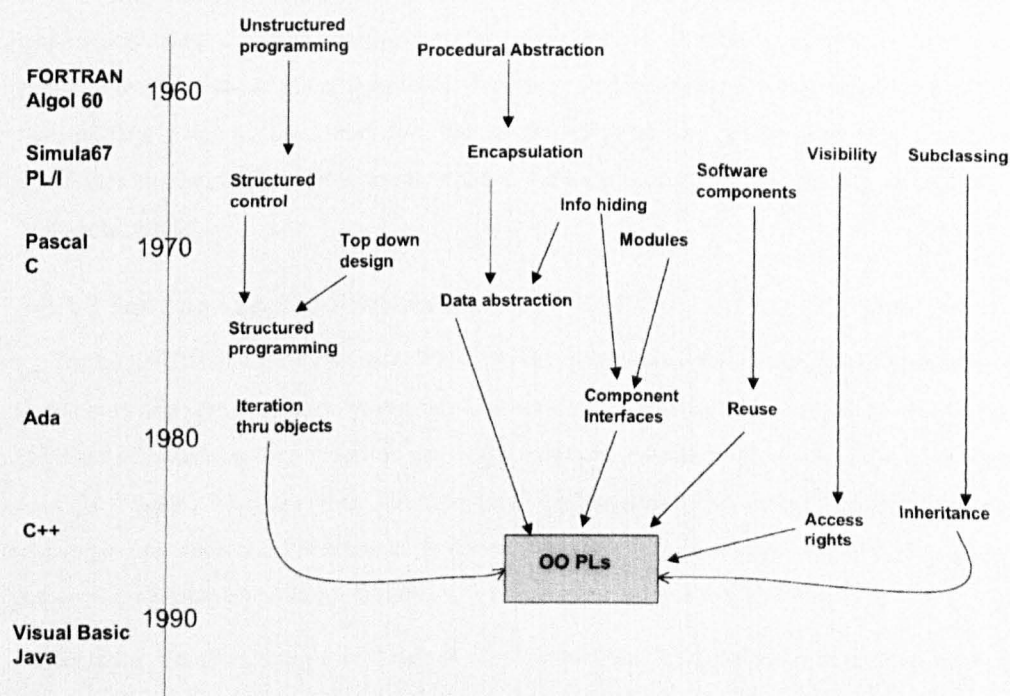
Scratch programmers depend on a superficial representation of the state space, one that shows them the result or outcome of its manipulation but provides little or no insights into the internal trail of changes that has led to this outcome. As a consequence, they are often incapable of resolving a programming problem by debugging it because they have had little or no exposure to, or experience of, meaningful debugging activities.

These difficulties resonate with Samurcay's observations cited earlier regarding external variables (i.e. variables used for input and output) and internal variables (i.e. variables necessary only for the programmed solution). Students found external variables easier to comprehend whereas the internal variables were conceptually more difficult because they required representation in terms of systems states.

Novice programmers have difficulty traversing a problem state space systematically. Even when the operations are relatively simple sequencing and assignment like those required for swapping two values they can experience difficulties. Reliance on visual feedback about the state can help but when the states are internal their ability to reason about them is severely diminished.

5.9 Program Decomposition and Design

"We all know how we cope with something big and complex: divide and rule, i.e., we view the whole as a composition of parts and deal with the parts separately" (Dijkstra 1989, p.1399). Breaking problems down is a crucial skill for programmers and good software is structured by design not accident. Indeed, good, enduring solutions to any type of problem are consciously designed and not serendipitous. The way a solution is designed can be influenced by the intellectual skills of the designer as well as the programming language used.



Source : Ryder et al. 2005

The facilities provided by programming environments and languages to support design have evolved on the basis of research and practice in software engineering. Ryder et al. have traced the evolution of programming language features and tools and their analysis reveals that all of the major developments have centred around the efficient and effective management of state. A summary of their analysis is provided by their timeline diagram which is reproduced above. The diagram has two parts, or sides, which group developments on the basis of their contribution to one or other of two types of abstraction identified as (1) control abstraction and

(2) procedural and data abstraction. The most widely used programming languages provide facilities and constructs to support these forms of abstraction (Ryder, Soffa et al. 2005).

5.9.1 Control Abstraction

Ryder et al. describe control abstraction as the separation of the abstract concepts of iteration and selection from their implementation. Implementing control abstraction requires the use of operations which control the flow of execution by testing the state and branching. With a lot of effort the connections between the states and the branching destinations can be understood. Nested combinations of iteration and selection significantly complicate the effort required to comprehend the relationships. Program comprehension is heavily influenced by the control flow patterns used and this led to the development of programming language constructs for iteration and selection that had a single entry point and a single exit point (Ryder, Soffa et al. 2005).

5.9.1.1 State and Control Abstraction

The constructs and their characteristics are extensively described in programming language documents and text books. However, what is rarely if ever highlighted is that single entry/exit constructs for iteration and selection can make programs simpler to read and write, as well as easier to modify, because they structure and explicate the connection between a state inspection operation and the scope of its effect. The reasoning associated with making choices between programming language constructs is about the manipulation of states.

For instance, iteration using a test before is more common because the programmer cannot presume the system is in the correct state when the iteration construct is reached and is obliged to verify whether it is or not before initiating a potentially repetitious action.

Pedagogy that eschews consideration of states can compound the difficulties a novice is experiencing. For example, very often the categorisation of iterations as definite and indefinite occurs in the absence of zero being considered one of the possibilities, or is structured so that it cannot be (Roberts 1995). This can lead to the exclusion of this possibility as part of the novice programmer's already fragile knowledge (Soloway, Bonar et al. 1983).

More generally, unless an iteration construct is trivial and uses only constants, there can be nothing whatsoever definite about the number of iterations it will require. Iteration has

nothing to do with whether the number of iterations is definite or indefinite. The number of iterations is a side-effect of the initial or repetitiously evolving system state. The iteration will be infinite instead of indefinite if no mechanism is provided to (at some stage) alter the state to the terminating state.

In almost all of the widely used programming languages the prominence of the *for* iteration construct is a consequence of the fact that it is a test before iteration (i.e. the most commonly used type) and its convenient syntax collects in one place the three most important components of an iteration - the setting of the initial state; the testing of the continuation state; and the state alteration mechanism that ensures the iteration will terminate at some point. All three components are state related inspections or alterations. With other iteration constructs these elements are delocalised and spread throughout the program text. Localising them reduces the cognitive load required for comprehension. With the *for* construct one can have a good grasp of its behaviour without, or in advance of, looking at the loop body it controls. This is because one has a good grasp of the state manipulations it is effecting, all of which are arranged conveniently on the first line of the construct.

5.9.2 Procedural and Data Abstraction

Procedural and data abstraction are two views of the same thing, distinguished from each other by source of the reasoning which guides them. In 1972 a seminal paper by David Parnas introduced the idea of decomposing programs into modules using *information hiding* as the criterion. In this regime each module is associated with a difficult design decision or a design decision which is likely to change. A module hides its knowledge about the decision and reveals as little as possible about its inner workings to the other modules. Parnas noted that because design decisions transcend time it is inevitable that modules will not correspond to identifiable steps in the processing (Parnas 1972).

Typically modules hide information about data structures, data formats or specific operations associated with the design. Thus, modularity is intelligent partitioning of the state space on the basis of design decisions and how they might change over time. Jean Ichbiah has suggested that the introduction of modularity solved the problem of scalability because "we learned how to scale with linear cost" (Ryder, Soffa et al. 2005, p.466).

Around the same time Barbara Liskov argued that while the basic idea of modularity seemed very good it sometimes led to the introduction of additional complexity because a system may be partitioned into modules which had too many functions, a common function may be distributed over several modules, or complex connections between modules may interact in unexpected ways (Liskov 1972). Liskov addressed the question of how good modularity could be achieved and developed the idea of using data abstraction as a way of partitioning systems. This would lead to "an improvement in program quality: programs will be more modular, and easier to understand, modify, maintain, and prove correct" (Liskov and Zilles 1974).

These developments had a profound influence on the design of programming languages which ultimately led to a transformative paradigm shift within the entire computing community with the widespread adoption of the "objects" paradigm.

5.9.2.1 State and Procedural and Data Abstraction

In some OO languages the conventions used by programmers writing in the language encourage them to identify methods that inspect the state by prefixing the method name with the word *get* and methods that alter the state with the prefix *set*. There is no obligation on the programmer to adhere to this convention but ignoring the collective experience and advice of the community of programmers expert in that language is unthinkable because the community rely on the convention to assist them in comprehension, debugging and maintenance tasks.

Indeed, in C++ the language provides a mechanism to allow the programmer document and signal this type of information to readers of the program with the added advantage that the compiler enforces the decision. By suffixing a method header with the keyword *const* the C++ programmer highlights it as a method that does not alter an object's state. The decision is enforced by the compiler which checks that the method does not alter member variables. Readers of the program know that the omission of the keyword *const* identifies a method that can alter the state of an object instance.

Both these examples show that even for professional, experienced programmers the pervasive nature of state, embodied in the humble inspect and alter operations, is still considered so pivotal that it warrants explicit consideration and identification.

5.9.3 Trouble with Decomposition

Novice programmers often err when decomposing a programming problem. In some cases it is because they do not or cannot recognise stereotypical plans involving a small collection of variables and fail to identify appropriate or workable problem chunks that could form a method (Sophrer, Soloway et al. 1989). They very often factor the problem on the basis of the identifiable steps in the processing (e.g. the layout of the output) which can in fact complicate matters instead of easing them (Perkins, Hancock et al. 1989). Object-oriented design is about identifying the objects that usefully structure the state space (i.e. the program data structures) of the system. This is one of the reasons why.

(Sorva 2012) provides a comprehensive list of the well-documented difficulties objects present for novice programmers. For example, novices tend to conflate the idea of class and object and find it difficult to grasp the static and dynamic distinction between the two. The role of constructors and what tasks they perform are misinterpreted. Students misconstrue the relationship between a class and an instance believing the instance to be a part of the class.

Novice programmers, and more generally novice designers, often falter when trying to design or structure a solution because they are not be familiar enough with the concept of state to identify the benefits provided by objects and the sorts of structures they can support. This inhibits their ability to discriminate between good designs and bad ones.

Object-oriented programming systems realise the outcomes of fifty years of programming research. They are built on a very simple concept. Partition the problem by partitioning the state space associated with the data and provide operations to protect, maintain and allow disciplined access to each partition. Structure the state space partitions so that they interact correctly to solve the problem. No one would claim that this is a simple task but the centrality of state to its successful completion is beyond dispute.

5.10 Additional "Big-Ideas"

The following list offers a summary of some additional big-ideas that could be used to supplement the case for state as a threshold concept in computing. The conceptual span ranges from the simplest concepts associated with software development to the most advanced.

- Requirements Engineering marks the commencement of a software project. Even at this stage state plays an important role. Requirement Engineers “produced the first extensive categorization of requirements engineering problem domains” by categorizing each problem domain using eight dimensions all of which are implicitly or explicitly state based (Maiden and Hare 1998).
- Almost all of the system design techniques developed (e.g. Jackson System Design, Unified Modelling Language and their equivalents) are state based
- Proofs of correctness, pre and post conditions, invariants, formal specifications and related topics, generally viewed as advanced computer science topics, are all inextricably bound to the concept of state. Typically their notations specify the initial and goal states and the transition required to reach the goal state. The method used to make the transition is usually not included. Thus, the focus is completely dominated by the states.
- Concurrent and/or parallel programming, generally perceived as the most complex of programming tasks, is essentially about protecting shared variables (representing states) from interference. Monitors, semaphores and other tools used to control synchronisation in concurrent systems all arise because of the need to manage process state and in particular interleaved process states.

5.11 Summary

Computers are very simple devices that can record, alter and inspect state. Novice programmers learning to harness that simplicity can find it difficult because it requires them to think atomistically using trivial operations to build sophisticated software artefacts.

State management is the essence of programming. The vastness of the state space requires the use of abstraction (i.e. categorisation or classification) to manage individual states as variables, combinations of related states as methods and interacting states using information exchange. Encapsulating state behaviour in objects is an advanced form of state management.

Learners with a good grasp of the concept of state make good programmers. Learners with an impoverished grasp of state can find some programming concepts difficult to acquire.

The centrality of the concept of state has been hidden in plain sight of the computing community for fifty years. It integrates the body of knowledge associated with software engineering. It is troublesome for learners at every stage of their learning. State exhibits all of the properties of a threshold concept.

Chapter 6

Conclusions and Future Work

This thesis is concerned with the difficulties experienced with the pedagogy of programming. The difficulties have been well documented over half a century but, nevertheless, continue to persist. I have argued that some of the difficulties have been a consequence of the discipline's success at delivering the tools that have propagated and sustained the ubiquitous proliferation of computing in everyday life. This outstanding 'production' success has been a distracting pressure that has dissipated the resources that could have been devoted to the development of our 'understanding' of the discipline and its pedagogy.

The distracting pressure has come from the rapid evolution of system platforms, programming languages and paradigms, interaction mechanisms, hardware improvements and smarter techniques for managing software engineering projects. The discipline has been running simply to stand still. This has been reflected in the discipline's professional and academic literature, including the literature of computing pedagogy which has been characterised as reflective of *being a CS educator* in contrast to the goal of *doing science* (Fincher and Petre, 2005).

Peter Denning has described how the "mature disciplines" such as physics, biology, and astronomy have, over their long histories, developed a small set of "great principles" which they use to portray the essence of the discipline. He notes

"Newcomers find a principles-based approach to be much more rewarding because it promotes understanding from the beginning and shows how the science transcends particular technologies...Indeed, you cannot understand a principle without knowing where it came from, why it is important, why it is recurrent, why it is universal, and why it is unavoidable." (Denning 2003,CACM Nov 2003 p15)

The discipline of computing is beginning to develop an 'understanding' of the nature of the discipline and to portray the essence of it. In relation to pedagogy it is worth noting three initiatives aimed at operationalising the goal of forming a shape and culture for the embryonic area of computer science education research and moving it into a defined territory with the status of doing science.

The first is the description by Tenenberg of the Bootstrapping Research in Computer Science Education project which aimed to develop the skills for designing, conducting and managing computer science education research and to establish and maintain "a research community able to sustain a constructive discourse as well as ongoing collaboration" (Teneberg, 2004, p1). The second is the publication of Fincher and Petre's seminal work on Computer Science Education Research which promotes a framework for the pursuit of research quality and excellence (Fincher and Petre, 2005). Finally, the establishment of the International Computing Education Research (ICER) Workshop provides an annual conference that focuses on high-quality contributions which are theoretically based and empirically supported contributions to the development of computer science education research.

This thesis is a contribution to the evolving status of computer science education research and the movement to formalise and develop its research credentials. As part of that contribution I have introduced the concept of a threshold concept to this discipline and sought to use it as a means of examining, comprehending and improving the pedagogy of programming, in particular the learning difficulties of novice programmers. I have also sought to develop threshold concept research itself and to consolidate the theory of threshold concepts.

In the following sections I document what the thesis has accomplished with respect to that contribution and what may be accomplished in the future.

6.1 Contribution to Computer Science Education Research

Although still in its embryonic stages computer science education research has established itself and attracted a very active community of scholars that continues to grow. This thesis

makes a contribution to computer science education research by introducing an innovative approach that has yielded important insights into novice programming pedagogy.

6.1.1 First Threshold Concept Identified in Computer Science

I have proposed the concept of *state* as a candidate threshold concept in computer science and provided a rationale and justification to support that classification. This establishes a theoretical framework capable of intellectually sustaining the concept of a threshold concept, identifies a fruitful source of evidential material and frames the examination of a candidate concept using those resources. The time invested in the foundational steps that facilitated understanding and yielded a coherent methodological process.

Marking the way opens up new routes for those. By successfully identifying *state* as a threshold concept this thesis has established that threshold concept research is a viable research topic for computer science education research.

6.1.2 Providing Insight into a Fifty Year Old Problem

The identification of *state* as a threshold concept has exposed the tacit and previously unidentified connection between the disparate difficulties that have been persistently reported in the computer science education literature as stumbling blocks for novice programmers. Those difficulties range from, for example, an inability to appreciate the mechanics of a swap operation to the benefits of classes. *State* as a threshold concept provides an insight into the discipline's fifty year old problem of why novices find programming hard and why our teaching has not been as effective as we would have liked.

At the least, this insight offers a useful starting point for the further development of our understanding of the discipline and its pedagogy. It represents a point of maturity for the pedagogy of the discipline and a stimulus for the development and evaluation of discipline specific pedagogic knowledge.

6.1.3 Moving to Teaching for Understanding

By exposing the discipline to the centrality of *state* researchers can begin to consider the implications of focusing on the teaching of *state*. This may raise awareness within the computer science community to the fact that *state* has been central to all of the programming language developments to-date and that for novices the concept can be a block to progression.

Classifying state as a threshold concept may stimulate the development of a pedagogic approach that recognises the pervasive influence of state, in programming specifically and computing more generally.,

6.2 Contribution to Threshold Concept Research

This thesis makes several significant contributions to the development of threshold concepts and threshold concept research.

6.2.1 Embodying Threshold Concepts in the Theory of Concepts

Threshold concept research has been criticised for the absence of a theoretical framework that can act as an organising principle to explicate existing research and provide a coherent focus for further research. This thesis has addressed that problem and shown that the *theory of concepts* subsumes the classification of certain concepts as threshold concepts. The proposal that threshold concepts can and should be understood as concepts, albeit of a specific type, appears self-evident but this thesis represents the first time the proposal has been articulated, justified and used to undertake investigative work. This is a valuable contribution to the development of threshold concept research.

6.2.2 Addressing the Criticisms of Threshold Concept Features

Two papers criticising the threshold concept idea have been published in the literature. However, the criticisms that are offered in these two papers are based on an Aristotelian view of concepts. As demonstrated in Chapter 2, the association of threshold concepts with the theory of concepts has shown these critiques to be unsustainable. This represents a significant contribution to threshold concept research.

6.2.3 Clarifying the Nature of Troublesome Knowledge

The importance of troublesome knowledge in the threshold concept process is never doubted or undermined by researchers but what they describe as constituting troublesome knowledge is almost always one-dimensional. Troublesome knowledge has been appropriated and interpreted unilaterally to mean *conceptually difficult*. This view can impede the consideration of a range of concepts as threshold concepts and distort the type of investigative protocols used to examine and evaluate candidate concepts.

For completeness, emphasis, and clarity I have examined various forms of troublesome knowledge within the discipline and situated the examples in a computing context. This exposition has highlighted the fact that *conceptually difficult* is one of the least frequently occurring causes of troublesomeness. This finding provides an important source of reflection for researchers attempting to identify threshold concepts.

6.2.4 Establishing the Primacy of Integration

The threshold concept literature is dominated by a focus on the transformative effects of threshold concept acquisition. This has resulted in transformation being considered the pivotal feature of threshold concepts. Rather, this thesis proposes that integration is the central feature because it is the conduit for cognitive development and only after concepts have been integrated is it possible for the consequential effects of transformation to become evident. In our ongoing search for threshold concepts researchers need to be cognisant of this emphasis and attribute more weight to integration.

6.2.5 Clarifying the Nature of Transformation

Using the analogy of chemical mixtures and compounds I have explained why the transformation which results from integration is irreversible. The emergent nature of the integration process renders it immune to articulation after the event. In addition, because the outcome is a compound and not a mixture it is impossible to undo or redo in precisely the same way. This has implications for the choice of research method and data gathering instruments used in threshold concept research. Identifying these issues provides research guidance for appropriate investigative approaches and future investigations.

6.2.6 Innovative Research Approach

The research approach used here may be used by other investigators interested in examining threshold concepts, both in computer science and other disciplines. This thesis contends that investigative approaches based on asking experts or learners to recall instances of critical cognitive development of the type that is attributed to threshold concepts are ineffective.

The use of the CoRe (Content Representation) for data collection is a novel and innovative approach for sourcing and recording data pertinent to the identification of a threshold concept.

The CoRe has considerable explanatory and representational power. To date it has been used exclusively as a group-based instrument to collect and collate ideas from a range of teachers and develop aggregated views. In this thesis the CoRe has been used as part of an autoethnographic approach to provide a principled exploration and exemplification of my pedagogic content knowledge which is then corroborated by and augmented with documentary evidence from the literature, especially the computer science education literature.

This thesis proposes the approach for the examination, documentation and evaluation of candidate threshold concepts in any discipline. It provides a mechanism for exploring and identifying threshold concepts which to-date has been unavailable to the threshold concepts community.

6.2.7 'Small' Concepts Are The Most Likely Candidates

The identification of state as a threshold concept identifies what might be considered a 'small' concept. The 'small' concepts may be the most difficult for learners to surmount because they are not in the foreground and are lurking in the tacit, inert and ritual knowledge categories which are all known to be troublesome. This may also be the reason they are overlooked by researchers searching for "big" concepts. Small concepts are perceived as innocuous and not credited with providing anything more than simplistic contributions. However, as the case for state reveals, small concepts can be pervasive so it should not surprise us that they can be at the heart of any difficulty a learner may have.

6.3 Future Work

Useful ideas tend to stimulate the thinking and scholarship of everyone. Threshold concepts oblige us to view the organisation of pedagogic knowledge in new ways, invite us to seek out explanations for why things are viewed the way they currently are, and to suggest how we might usefully view them differently. The identification of a threshold concept in the discipline of computer science should stimulate research to reveal new relationships and better understanding. Future work may pursue such understanding in several ways.

6.3.1 Are There More Threshold Concepts in Computer Science?

The identification of one threshold concept in computer science begs consideration that there might be more. Clearly there are many opportunities for additional research in this area. In addition to state I propose *delimiter/event* and *offset* as candidate threshold concepts of computing. Providing a rationale and supporting evidence for these proposals based on appropriately developed CoRe's is an immediate personal research goal.

6.3.2 How Can Threshold Concepts Influence Our Teaching?

We need to understand the circumstances which will best facilitate the introduction of threshold concepts into our teaching. If they continue to be "small" concepts they are susceptible to being overrun by their grand or great peers. When is the best time to introduce a threshold concept? How can the introduction be managed and what kind of conceptual infrastructure is required to scaffold the introduction?

If the concept gets lost, as it seems to do at present, how can we avoid or deal with this? If someone is finding the acquisition of the threshold concept difficult what can be done to ameliorate the problem? When do we know for sure that the threshold concept has been acquired? What performance or behavioural marks can be detected to show us that the concept has been acquired?

We need a clear sense of the possible pathways to the acquisition of the threshold concept and what developmental sequences are most helpful or successful.

6.3.3 Identifying Pedagogic Knowledge Gaps

Learning is inherently complex and messy and any instrument that assists with untangling learner difficulties warrants application and evaluation. Threshold concepts attempt to make sense of the part of the learning process which addresses the problems learners encounter and how and why those problems can inhibit the learner's progress. They have revitalised the study of teacher knowledge and provided a new analytical framework for organising and collecting data on the difficulties experienced by learners. This analysis can help clarify and consolidate existing pedagogic knowledge but it may also be capable of identifying pedagogic knowledge gaps. How can threshold concepts help identify gaps in the pedagogic knowledge base?

Appendix A

State - Sample Teaching Materials

A-1 Introduction

In this appendix I am providing some sample materials used in my classroom to introduce the concept of *state*. The structure of the Appendix is aligned with the "big-ideas" listed in the CoRe provided in *Chapter 5 State as a Threshold Concept*. I have sketched out the type of material introduced and in some cases provided exemplars from student class work to support the text.

Students in the class are all novice programmers. They may have had some exposure to programming before but not in a formal setting, unless on an individual basis they participated in commercially operated training or had an interested and innovative teacher at their school. In the main the students have had no previous exposure to programming.

The first two weeks of the course are devoted to setting the scene for the rest of the semester and, indeed, their programming careers. Two notional machines are used to set out the types of activity the class will be involved in during the course and in programming scenarios in general. The goal is to draw attention to a number of important concepts that will be elaborated and developed more fully in the class work and the individual practice work the students do themselves.

The key concepts are

- the machine provides only two operations - inspect (or get) and alter (or set) - which allow the programmer to manipulate the state of the machine
- the simplicity of this mechanism can be a source of difficulty. Programming languages provide facilities to assist with harnessing this simplicity. We will learn about the essential facilities provided by programming languages and build on those facilities as we progress.
- how we organise and manage the machine state is crucially important. Initially we will deal with reasonably straightforward problems to allow us develop some expertise with these activities. Our treatment will evolve to encompass more sophisticated features.

A-2 A Notional Machine

A-2.1 The Maze Machine

In 1980 Farhad Mavaddat developed a small set of notional machines for use in his introductory programming courses (Mavaddat 1980). One of his objectives he describes as somewhat "destructive" because it sought to undo the popular but misleading image of computers. Mavaddat wanted to show his students that computers were nothing more than obedient servants capable of following exact but incredibly trivial instructions at incomprehensible speeds. He believed that undoing the flawed perception would provide him with a platform to explore the "constructive" aspects of computing that would allow his students to realise that computers are capable of many things but they are not achieved in the way the students currently think.

He described one of his machines as the Maze Machine. In the simplest maze machine only two operations are available or provided by the machine, STEP and RIGHT. To use this machine users are obliged to write the sequence of operations necessary to traverse a maze from the starting position to the exit position.

The operations are collectively referred to as a program. The users of the machine are called programmers because they must program it to traverse the maze.

The RIGHT operation allows the machine to alter its orientation or direction. The operation changes the direction by rotating 90° to the right. Thus if the machine is currently oriented or facing forward (i.e. →) then obeying or executing a RIGHT operation will alter the orientation and the machine will be facing downwards (i.e. ↓). In this orientation a RIGHT operation will cause the machine to be facing to the backwards (i.e. ←). A third RIGHT operation will cause the machine to face upwards (i.e. ↑). In summary, the RIGHT operation simply alters the machine orientation and does **nothing** else.

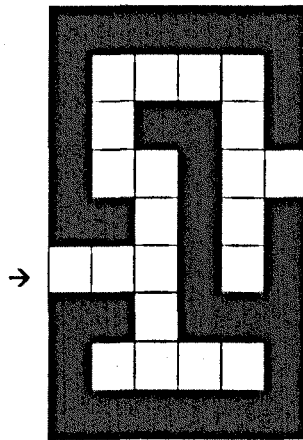
We can of course establish from this behaviour that the RIGHT operation is capable of setting the machine to "face" in any direction and from that setting to "turn" or alter the direction. Using one, two or three RIGHT operations allows us to face the machine in any direction we choose.

The STEP operation allows the machine to change its position by advancing one unit or location in the direction the machine is currently set to. If the path is blocked by a maze wall the position is not changed. Such an operation could be considered an "error" or an inappropriate action even if it does not "appear" to affect the possibility of a successful traversal of the maze.

I renamed the operations FORWARD and FACERIGHT for two reasons. The first was to make them longer and therefore a little more irritating to write repeatedly. I want the students to appreciate that we can whinge and moan all we like about how a particular objective is achieved in a particular programming language but it won't change it. A programmer who uses several languages may compare and contrast the features provided by those languages and may even express a preference of have a favourite but when using those languages they must accept what the language offers and work with it. We can whinge about the modulus arithmetic operator being a % symbol in one language and the abbreviation *mod* in another but it isn't going to change the languages. What is important is that we appreciate their equivalence and especially their purpose and utility.

The second reason was to emphasise that the RIGHT operation only alters the orientation and doesn't alter the position. I felt FACERIGHT would more clearly indicate that there is no change of position whereas RIGHT is somewhat ambiguous. These are minor alterations but they explain why the student samples included below use different operations to those specified by Mavaddat.

Consider the following maze taken from Mavaddat's paper which I have used as a first example to get students programming a maze machine. Students are presented with this maze in the first session and after the two basic operations have been described. They are asked to write a solution that will traverse the maze with the minimum number of operations.

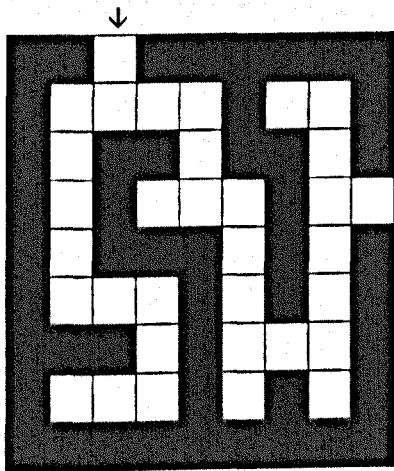
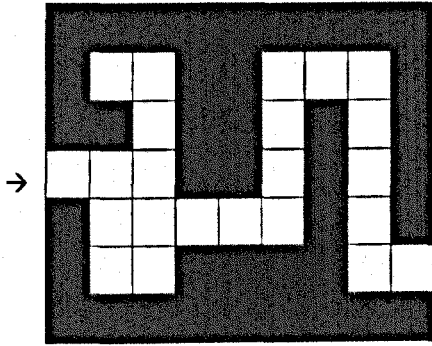


To traverse this maze the student must present a solution like the following

FORWARD, FORWARD, FORWARD, FACERIGHT,
FACERIGHT, FACERIGHT, FORWARD, FORWARD,
FACERIGHT, FACERIGHT, FACERIGHT, FORWARD,
FACERIGHT, FORWARD, FORWARD, FACERIGHT,
FORWARD, FORWARD, FORWARD, FACERIGHT, FORWARD,
FORWARD, FACERIGHT, FACERIGHT, FACERIGHT,
FORWARD, FORWARD

(Subject : EL1)

The provision of a variety of mazes allows the students to write programs for several maze problems all of which use the same machine operations permuted as required by the problem. For example,



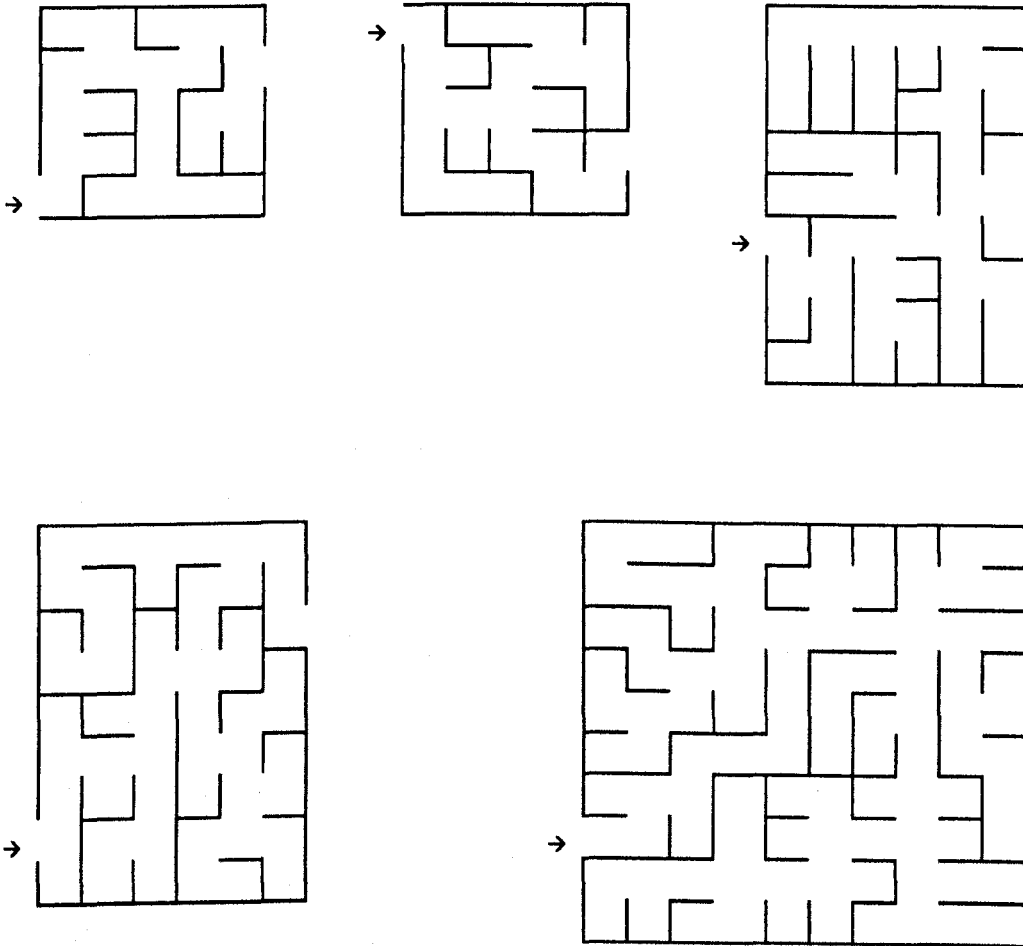
It is, of course, simple but tedious to write a solution to these problems. The reason the solutions are simple is because the current state of the machine is visible all of the time. Traversing the maze simply requires us to note our current location and to figure out a path to the exit. As we write the individual operations that follow the path we can see what effect each has on the state of the machine and easily identify our new location. From this location we can establish what has to happen next to get to the destination.

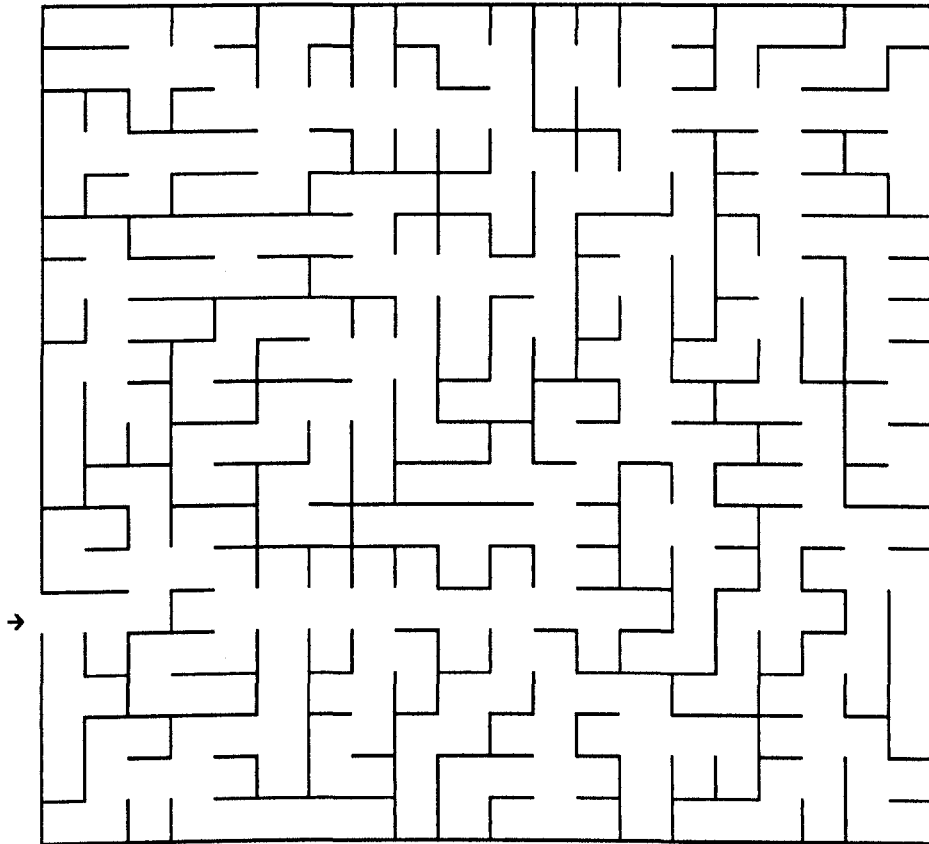
The tedium arises because what each operation is capable of achieving is trivial. The absence of a FACELEFT operation is immediately obvious. A FACELEFT operation can easily be realised using three FACERIGHT operations but when it occurs several times it becomes distracting and unhelpful. Similarly, the inability of the FORWARD operation to vary the distance moved makes the programs longer and annoying to write. It seems ridiculous to have to write several FORWARD operations in a row to traverse a straight line section of the maze.

APPENDIX A : STATE - SAMPLE TEACHING MATERIALS

I developed a maze generator to make it possible to quickly generate a large quantity of maze problems that have to be "programmed" by the students in my class. Providing students with a personal copy of the generator affords them the opportunity to write many maze programs. However, the main advantage of a maze generator is that it allows the scale of the problem to be varied very easily.

For example, consider the small selection of generated maze problems shown below. Few students would relish the prospect of having to write solutions to these problems mainly because they would take a lot of time to write and check for correctness. In addition, few of us would look forward to the prospect of having to check someone else's solution.

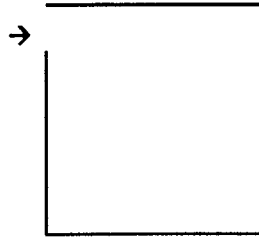




However, a much more significant difficulty associated with solving maze problems becomes evident when we do not have the benefit of an aerial view like those provided in the pictorial representations of the mazes shown above. Although the solutions to those mazes may be tedious to write we at least know how to go about finding a solution because at all times we can see the state of the maze machine and we can verify the evolving solution. How would we solve the maze if we didn't have an aerial view?

Consider the following maze which is described to the students as a maze with a cover placed over it so that we can still see the entry and exit points but we cannot see the actual layout of the maze. I generated the maze using the maze generator but had the colour of the internal walls set to white with the effect that they become invisible.

We have to write a program for the maze machine to traverse this maze. How can we do it?



As things stand it is impossible for us to solve this problem because the maze machine only provides operations capable of altering the state. We can only alter the orientation or the position. We have no way of finding out what the current orientation or position is and in the absence of the maze layout we have no way of recording our current state (i.e. where we are in the maze). We have no way of inspecting the maze machine's current state. As a consequence we have no way of working out what has to be done, or needs to be done, to solve an unknown maze layout. Without the ability to record, inspect and alter the state of the maze machine we will not be able to solve maze problems for which we cannot see the maze layout. It is impossible for us to build any intelligence into the solution to enable us to "work out" how to traverse the maze. The absence of inspect operations is a significant weakness in the design of the machine.

A-2.2 Light Bot Machine

The Maze Machine is not dynamic enough for many students. You cannot run your maze programs and after a little while mazes lose their appeal. The Light Bot⁷ machine provides a more contemporary setting for exploring the nuances of state and its accompanying set and get operations as well as providing some interesting "programming style" challenges for students.

The Light Bot machine is a web-based game application that requires the player to write code to solve a series of problems that increase in difficulty. It has many of the characteristics of the Maze Machine. For example, the game is played on a tiled grid which can be configured in a three dimensional style. The number of operations is small and not unlike the maze operations. In total Light Bot provides just five operations. They are Forward, FaceRight, FaceLeft, Hop (hops up the height of one tile only), LightOn (turns on the tile light in tiles that have a light). Like the maze machine the operations either change the orientation (i.e. FaceLeft and

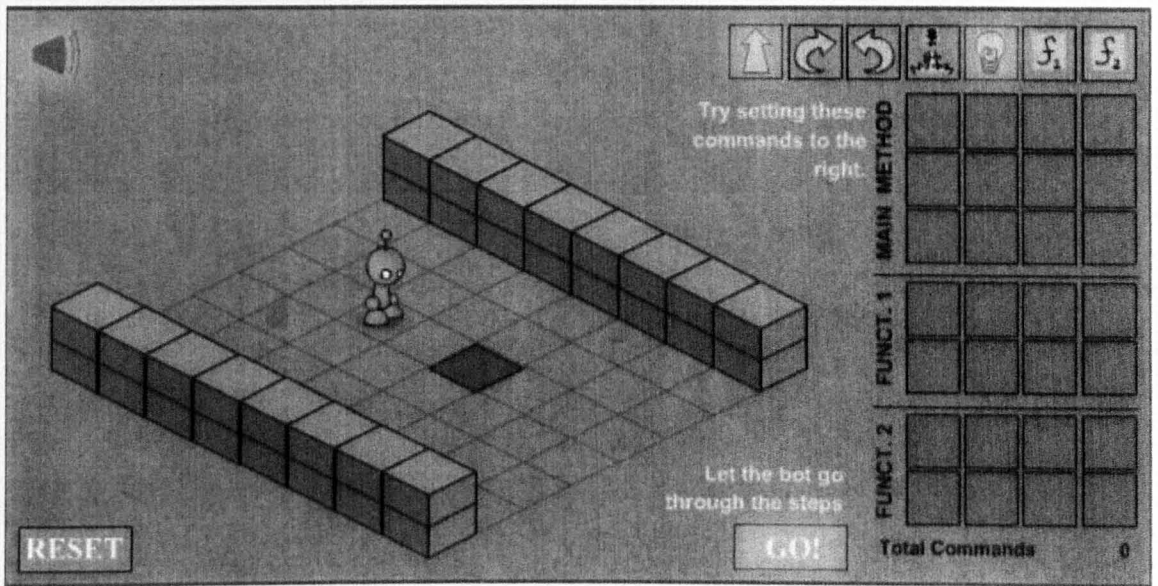
⁷ Available at http://www.kongregate.com/games/Coolio_Niato/light-bot. Last accessed 10/07/2013

FaceRight) or change the position (i.e. Forward and Hop). In addition, the Bot has an operation for turning on the light (i.e. LightOn). Like the Maze Machine the Light Bot machine can only alter the state but it cannot inspect it so it suffers from the same design weaknesses as the Maze Machine. We can only solve Light Bot problems for which we can see the entire layout of the game.

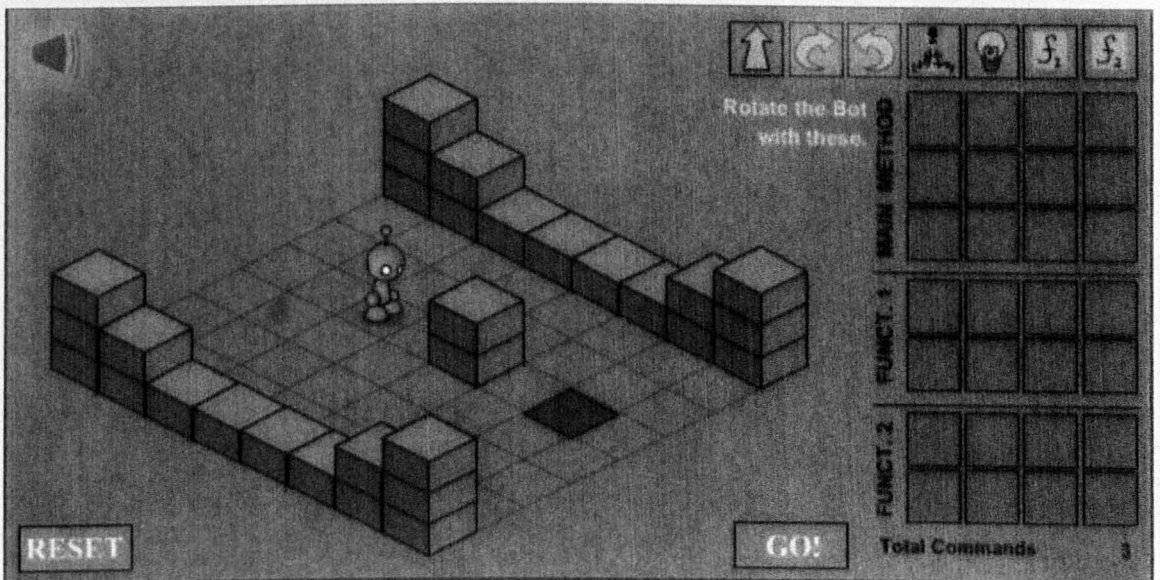
The Light Bot machine also provides three named methods (i.e. *main*, *f1* and *f2*) that allow the user to decompose the solution if they find it useful or necessary. Methods *f1* and *f2* can be used at the user's discretion but the operations to solve the problem must start in the *main* method. In addition, the current state of the machine is permanently visible on the screen and is updated as the program instructions are executed. If the program is not correct the user can see from the animated execution where the problem arose. The incorrect solution can be altered and rerun as many times as necessary to remove any errors in the solution.

The additional method and debugging features if Light Bot provide a nice basis for familiarising students with similar features in real programming languages like C, C++ and Java. However, the main benefit of the Light Bot environment is the simplicity of the operations and the opportunities it provides to solve reasonably difficult problems using a small collection of trivial operations. This is another opportunity to highlight the idea that even small collections of operations can be used to build solutions to difficult problems.

The screen shot below shows the challenge presented at Level 1. The simple problem shown is solved by placing a Forward operation in the first two positions of the main method and a LightOn operation in the third position. Clicking on the GO! button executes the program. If the user has not provided a correct solution for the problem they cannot advance to the next level.

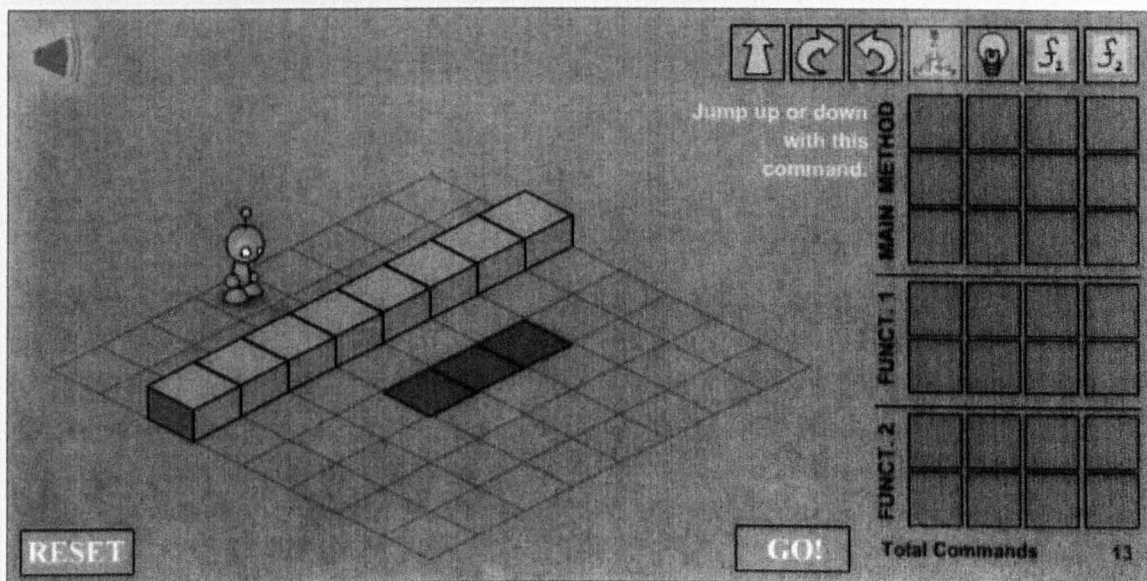


The increasing complexity of the challenges presented at each level can be seen in the screen shots for levels two to six which follow. The starting levels have hints provided to assist users unfamiliar with programming. In Level 2 the user is provided with a hint that suggests using the FaceRight and FaceLeft to get around the pillar blocking the Bot from the light, instead of trying to hop over it. The Hop operation only hops to the height of one tile so it cannot be used to solve this problem.

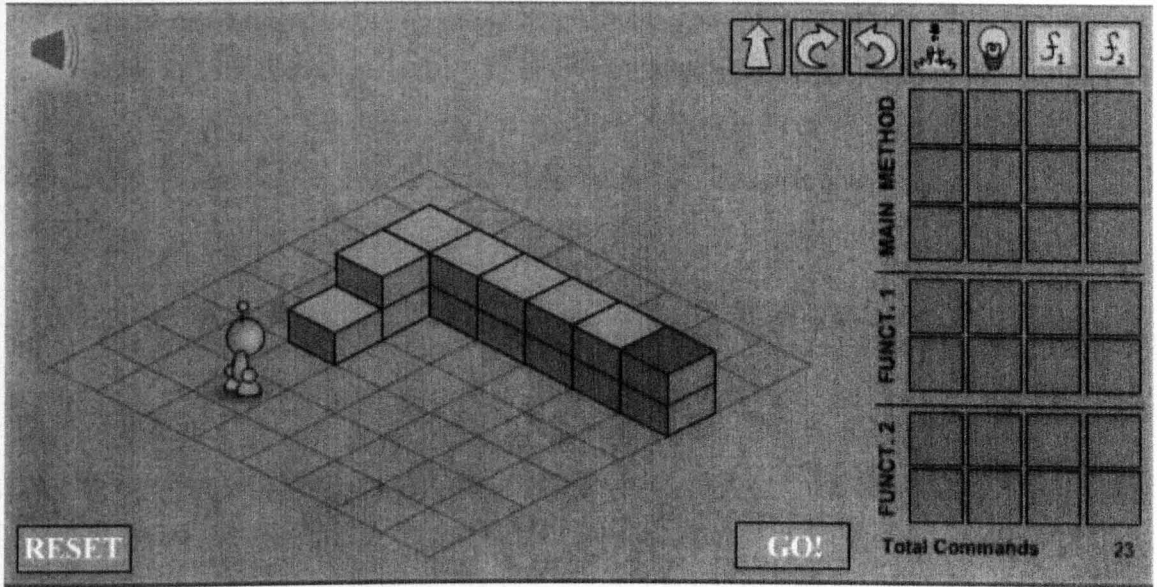


APPENDIX A : STATE - SAMPLE TEACHING MATERIALS

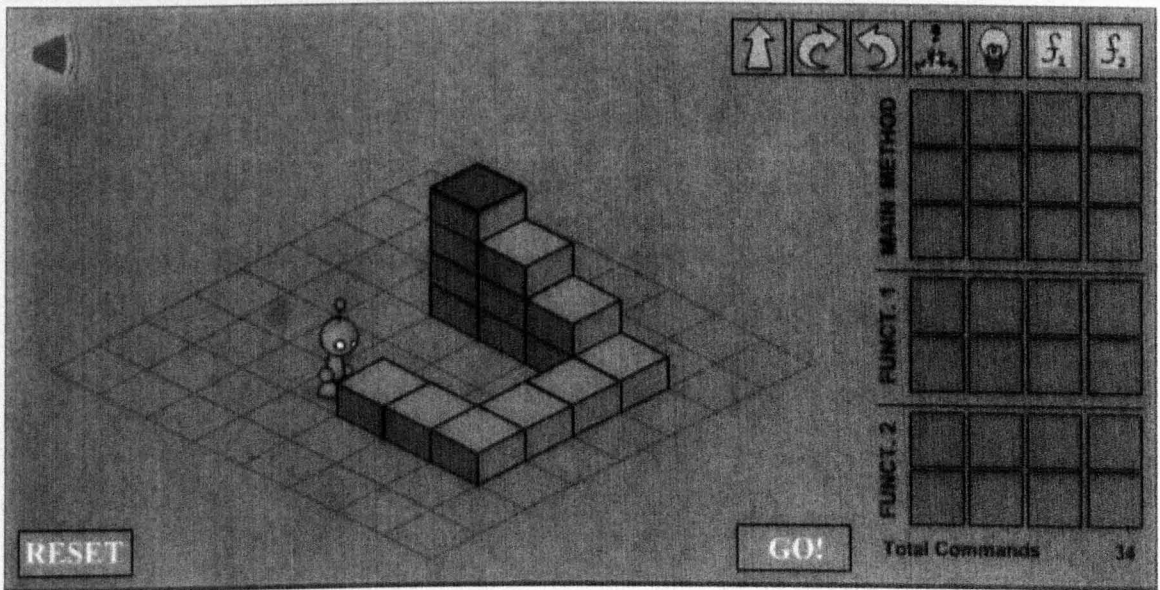
In Level 3 the user has to use the hop operation twice to jump on and then off the wall before proceeding to turn on three lights.



In Level 4 the user has to combine four of the five operations to solve the problem. As with the preceding problems it is relatively easy for the user to create a solution to the problem because the entire layout of the grid is permanently visible on the screen. The user simply has to choose the operations and sequence them appropriately. Incorrect programs can be edited and rerun until the solution is discovered.

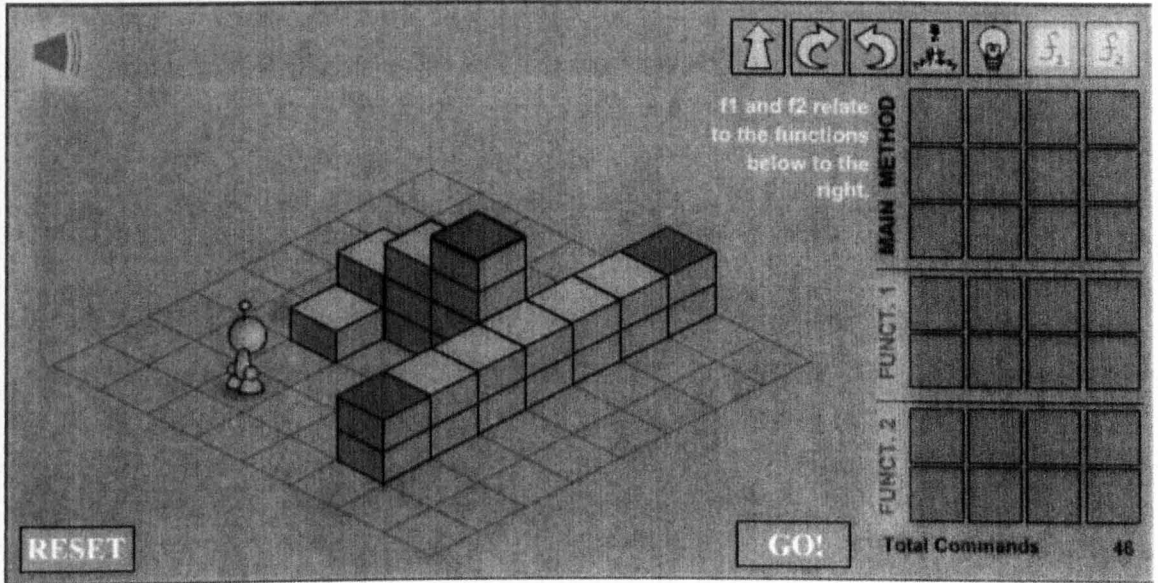


At Level 5 the main method has only one spare operation position left after the solution has been entered. The user has to use all five of the machine operations to solve this problem.



In Level 6, the last of the examples shown here, the user is provided with a hint to the effect that the problem will require the use of one or both of the f1 and f2 methods. The main

method will not be able to accommodate the entire solution and the user will need to partition or decompose the solution to be able to solve it using the resources provided by the game.



The sequence of problems from Level 1 to Level 6 leads the user from problems requiring a simple sequence of operations to a scenario where some design decisions have to be taken. The solution to Level 6 has to be decomposed into a controlling method (i.e. main) and one or two supporting methods (i.e. f1 and f2). The user is left to decide how they want to use those resources to solve the problem, but they have to think about it and they have to use them. The remaining levels in the game increase the dependence on design decisions and the utilisation of the f1 and f2 functions but the problems are always solvable using the same five, basic operations.

A-2.3 Turing Machine

In both the Maze and Light Bot machines the simple operations provided can only alter the state of the machine, either the orientation or the position. No operations are provided that allow the user to inspect the state of the machine. For example, in the Maze Machine there are no operations that allow the programmer to check if it is possible to step forward (i.e. to check if the way forward is obstructed by a maze wall); no operation to determine if the program has stepped out of the maze (i.e. has completed the maze). Similarly, in the Light Bot machine there is no way of determining if the Bot is standing on a tile with a light; no way of checking if

the way forward is obstructed by a tile or a pillar or wall of tiles; no way of determining the height of an obstruction.

Both the Maze and Light Bot machines provide examples of how problem solutions can be built from trivial operations that are combined to solve problems of varying degrees of difficulty. However, they both suffer from the drawback of not providing any operations that allow the state of the machine to be inspected. In both cases all of the operations alter the current state but none of them allow the user to inspect the state of the machine and to make decisions on the basis of those inspections.

As mentioned above this is a significant weakness in the design of both machines. It is only possible to write programs that solve problems for which the problem layout is known and we are presented with an aerial view allowing us to see the various paths and simply follow the one which solves the problem.

In 1936 Alan Turing proved that a machine that extended the facilities provided by the Maze and Light Bot machines a little and provided, in addition to alter operations, operations to inspect the machine's (current) state could be used to solve **any** computation that could be specified as an algorithm.

A Turing Machine is a type of state machine that can be in any one of a finite number of states at any time. The machine is controlled by setting the initial state and specifying instructions that cause it to remain in the current state or change to one of the other states. Thus, a Turing Machine is a machine that has only two operations

3. It can inspect the current state of the machine
4. It can alter the current state of the machine

These operations are supported by the machine's ability to record the current state of the machine. The operations are sometimes referred to as get (i.e. inspect) and set (i.e. alter). A program controls the execution of the set and get operations which are executed at incredible speed. It is the speed that disguises the trivial nature of the operations. Because the sets and gets execute at tens of millions per second most computer users are completely unaware of just how trivial the operations are.

Virtually every computer system built since Turing described his notional machine has been designed on the basis of his principles. As a consequence state is absolutely central to

everything associated with the development of computer systems. The key challenge facing programmers is the development of a representation of the state of the problem being solved in circumstances where no physical representation is available. Meeting that challenge involves the use of inspect and alter operations that manipulate the recorded state of the process. This is what the student will spend their time learning throughout the remainder of their programming course(s).

A-3 Introducing Inspections

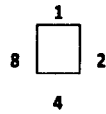
To allow inspections a machine (i.e. any machine) will have to provide some way of doing the inspection. That task has two parts. The first requires an operation that is capable of supplying the information we are required to inspect. For example, if we want to be able to inspect an individual maze cell to determine whether it has a wall on its right hand side, or on top, or on the bottom, or to the left, then the machine will have to provide an operation that supplies that information. The second requirement is that we must have some mechanism to inspect the result of the operation to establish which of the possibilities is currently applicable.

To illustrate these requirements the basic Maze Machine is modified with five additional operations; three operations providing information and two operations allowing the information to be tested. The three information operations are called OUT, FACING and WALLS. The two testing operations are called IF and WHILE.

The OUT operation produces the answer yes or true if the machine has reached the maze exit point (i.e. the position outside the exit cell). Otherwise OUT produces a no or false answer.

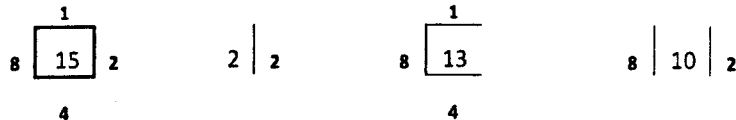
The FACING operation has the value 'R' if the current state of the machine is that it is facing to the right; the value is 'L' if the machine is currently facing left; 'U' if the machine is facing up; and 'D' if the machine is currently facing down.

The WALLS operation is a little more sophisticated. There can be up to four walls in a cell but they can be in any combination. To handle all of the combinations each of the walls has been assigned a weighting value as shown in the following layout



For a cell that has only a single wall along the top the WALLS operation returns 1. It returns 2 if the cell has only a wall on the right hand side; 4 for a bottom wall only; and 8 for a left hand side wall only.

If the cell has more than one wall the WALLS operation returns a value that is the sum of the wall weightings. For example, the following configurations show the sum in the cell with the wall weightings around the edges.



The WALLS value for the first example is 15 because the cell has all four walls and the sum of the weightings is 15. The second example has a WALLS value of 2 because the cell only has a wall on the right hand side. The third example has a WALLS value of 13 because the cell has top, left and bottom walls and the weightings sum to 13. Similarly, the final example sums to 10 because it has left and right walls.

If a cell has no walls then it's WALLS value is 0.

WALLS provides the value for the cell at the machine's current position.

Using the IF testing operation we can use the values produced by WALLS and FACING to make decisions about what to do next. For example, if the test FACING = 'U' is correct or true then if we want to alter the state of the machine so that it is facing right we need to do a FACERIGHT operation. Similarly, if the test FACING = 'L' and we want to alter the state of the machine so that it is facing right we need to do two FACERIGHT operations. If FACING = 'D' is true then we need to do three FACERIGHT operations to alter the state so that it faces to the right.

In our programs we can write the following to ensure that the machine is facing right. Note, if the machine is already facing right then the following has absolutely no effect.

```
IF FACING = 'U'
  FACERIGHT
```



```

ELSE IF FACING = 'L'
  FACERIGHT
  FACERIGHT
ELSE IF FACING = 'D'
  FACERIGHT
  FACERIGHT
  FACERIGHT

```

The WHILE operation is like the IF in that it does a test but they differ significantly in that the WHILE will repeat the test as long as it continues to be true. So unlike the IF which is executed just once the WHILE can be executed many times.

For example, suppose we want to write solutions to the two mazes shown below. We could of course write solutions like the following

```

FORWARD, FORWARD, FORWARD, FORWARD, FORWARD,
FORWARD, FORWARD
FORWARD, FORWARD, FORWARD, FORWARD, FORWARD,
FORWARD, FORWARD, FORWARD, FORWARD, FORWARD,
FORWARD, FORWARD, FORWARD

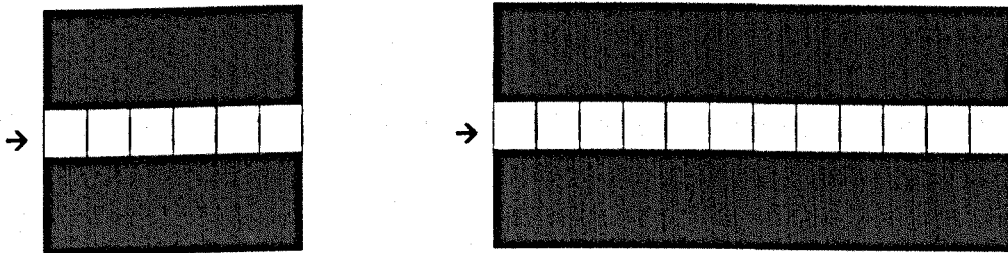
```

Alternatively, we could use the WHILE operation and the information provided by OUT to create a solution that will work for any maze of the style shown below.

```

FORWARD
WHILE OUT = no
  FORWARD

```



The addition of the information and testing operations brings the number of operations provided by the machine to six. With these meagre resources it is possible to write a solution to solve any maze without the need to write large amounts of code. This is a salutary lesson for students and emphasises the idea that even with a very small number of apparently simple operations it is possible to solve a great number of sophisticated problems. In addition, it is

important to point out that the six operations can all be classified as either inspect or alter operations which operate on the current state of the machine.

A-4 Methods

The Light Bot environment provides the user with three methods (main, f1 and f2) for coding solutions but they can only be used in the strict style available within the environment. In contrast, the Maze Machine provides no explicit decomposition mechanisms but because the machine is not subject to the constraints of a software environment it allows a lot of latitude. In the absence of an enforcement mechanism students often make pragmatic decisions about how to write Maze Machine solutions and do not strictly adhere to the rules specified for the machine.

For example, even though the Maze Machine doesn't have a FACELEFT operation some students, possibly those who have programmed already or just dabbled with programming, use one in their solutions. A typical FACELEFT operation takes the form

FL = {FACERIGHT, FACERIGHT, FACERIGHT}

(Subject : CW2)

LEFT {FACERIGHT, FACERIGHT, FACERIGHT}

(Subject : FR4)

Z{FACERIGHT, FACERIGHT, FACERIGHT}

(Subject : JMC2)

This is then used to code a solution as follows

F3, LEFT, F2, LEFT, FORWARD, FACERIGHT, F2,
FACERIGHT, F4, FACERIGHT, F2, LEFT, FORWARD

(Subject : FR4)

When students are asked why they do this they typically respond that they would sooner use the non-existent FACELEFT operation than constantly include three FACERIGHT operations. In addition, they claim that it is easier to write the solution, understand it and check it, although some of the naming strategies are not entirely consistent with the goal of improved comprehension (e.g. Z instead of FACELEFT).

APPENDIX A : STATE - SAMPLE TEACHING MATERIALS

Even after writing solutions to just a handful of Maze Machine problems most students embrace the *user-defined operations* approach. In some cases this is the result of students observing that their peers have used them and they were useful and even encouraged by the tutor. Operations with names like FORWARD2, FW3 and X become commonplace and are defined as a shorthand to eliminate the need for repeatedly writing FORWARD operations.

FW2 = Name {FORWARD, FORWARD}

FW3 = Name {FORWARD, FORWARD, FORWARD}

(Subject : BG4)

X{FORWARD, FORWARD, FORWARD}

Z{FACERIGHT, FACERIGHT, FACERIGHT}

(Subject : JMC2)

Many students quickly realise that the intelligent use of user-defined operations can help alleviate the tedium of writing long operation sequences. Harnessing the mechanism they develop solutions involving permutations and combinations of primitive and user-defined operations which can be quite sophisticated, useful and clever. For example, a solution to a maze that repeats three times could be coded as follows

L = NAME {FACERIGHT FACERIGHT FACERIGHT}

R = NAME {FACERIGHT}

F = NAME {FORWARD}

F2 = NAME {FORWARD FORWARD}

F3 = NAME {F2 FORWARD}

F4 = NAME {F2 F2}

REPEAT = NAME {F3, L, F2,L, F, R, F2, R, F3, R,
F2, L, F2}

REPEAT3 = NAME {REPEAT, REPEAT, REPEAT}

(Subject : MW4)

The students are applying programming principles and techniques that have not been mentioned or even suggested at this point in their programming course, but the sample problems are encouraging them to consider how writing their solutions can be handled in a

more intelligent and manageable fashion. The students are consciously introducing additional operations based on the primitive operations with the implication that the new, user-defined operations will ultimately be realised by substituting the appropriate primitive operations in their place.

Most importantly, they can appreciate how partitioning the operations that manipulate the machine state into logical blocks assists with the coding, improves comprehension of the solution and ameliorates some of the tedium associated with using trivial operations. The students motivation may in part be due to the avoidance of work but that is why all programmers use such strategies so in fact we should be crediting the students because they are already starting to think like programmers!

The idea that programmers would identify operations they find useful and build them from the primitive operations provided by the machine is seen by the students to be logical, practical and eminently sensible. When methods are formally introduced in the course they confirm the commonsense observations the students themselves had. Methods describe what the student already knew to be worthwhile.

A-5 Using Variables for Additional State Information

We may find it convenient to supplement the information provided by the Maze Machine (e.g. by OUT, WALLS and FACING) to assist us in developing a solution to a problem. There may be other events or values that could simplify or improve a solution to a problem.

For example, consider the increasing number of FORWARD operations created by the decision to restrict each of them to a specific number of forward steps. Students create separate methods for moving forward two steps, three steps, and so on. An operation that could move forward a variable number of steps specified at the time the operation is being used would eliminate the multitude of specific forward operations and reduce it down to one, flexible equivalent.

To be able to provide that sort of facility the machine would have to allow the programmer identify and create an additional mechanism for recording the state, inspecting it and altering it. In programming systems variables are used to support those activities. Consider the following sample solution to the problem of providing a flexible forward operation.

```
FORWARD(INT nSteps)
  INT count
  count = 0
  WHILE count < nSteps
    FORWARD
  ADD 1 TO count
```

We have introduced two additional states that store important information used for moving the correct number of steps.

The first is called nSteps. It will be given a value when we need to move forward a specific number of steps. For example, we could write FORWARD(3) when we need to indicate that we wish to move forward three steps. The variable nSteps would be altered so that it contained the value three. Using FORWARD(5) would alter nSteps to the value five; and so on.

The second is called count. We explicitly set the initial value of count to zero to ensure that it starts with the correct value. We use the WHILE operation discussed above to inspect or test the current value of count against the current value of nSteps. If it is less than nSteps we move forward one cell. Then the value or state of count is altered and it is incremented by 1. Because it is a WHILE operation we must test or inspect the values in count and nSteps again. This process continues as long as count is less than nSteps. When it is not the process stops and we will have moved forward the appropriate number of steps.

By introducing the additional states and creating them as variables we have been able to eliminate the need for several methods that do essentially the same thing and replace them with a single method that has the flexibility to allow the programmer to tailor the number of moves appropriately. Using the usual inspect (i.e. in the WHILE count < nSteps) and alter (i.e. count = 0 to initialise it and ADD 1 TO count to alter it) operations we have succeeded in creating a simpler and more flexible solution.

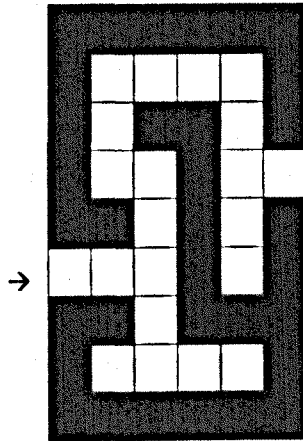
Formulating good solutions to programming problems is more concerned with choices and decisions about states than it is about writing lines of code. A well designed program provides an intelligent and efficient representation of the states required to solve the problem. Students will spend more of their time learning how to create intelligent and efficient state

representations than any other mechanism or feature of programming systems. The simple examples

A-5.1 Initialisation

The machine has no idea how additional states introduced as variables by the programmer behave so the manipulation of variables is the sole responsibility of the programmer. Any alterations to the variables must be explicitly coded by the programmer. The initial state is crucially important because if it is set incorrectly then regardless of how ingenious the solution may be it will not produce the required outcome.

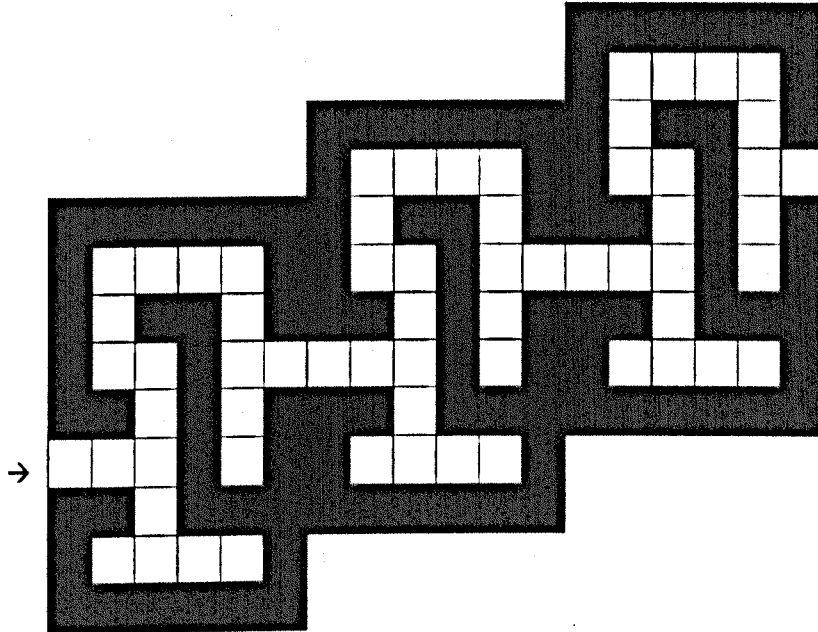
Setting the initial state is very easily overlooked and even when programmers remember to set it they can do so incorrectly.



The first maze presented to students, duplicated above for convenience, shows the initial state with the starting point marked with the → symbol (i.e. at the point immediately outside the maze). Some students assume this means they are inside the maze on the first cell and others interpret it to mean that they are at the point immediately outside the maze. The goal is to traverse the maze and exit from it, so the finishing position should be the point immediately outside the last or exit cell. Some students interpret the goal to be reaching the exit cell but not actually exiting and some solve the problem correctly and exit the maze.

The interpretation of the initial state is important because it determines how many forward steps need to be taken to reach the first turning point. The solutions produced by students who assume they are inside the maze cannot be correct even if all of the operations are correct relative to the initial position. Because the initial state is incorrect the whole process will fail.

This issue is also important, but in a much more subtle way, when students are presented with a maze formed from a repetition of the original maze like the one shown below.



Many solve the problem by simply repeating their solution to the original maze three times. However, that solution will fail because in its correct version it should terminate at the point outside the exit cell. That termination point is now the first cell of the next maze so a precise repetition of the original solution will start from an incorrect initial state (i.e. not the cell immediately outside the entrance to the maze). The solution is off by one cell for the second maze.

Initialisation is a very important operation. Its significance is recognised by the fact that most, and especially more modern, programming languages provide "reasonable" initial values for state represented as programmer variables. Even in some of the more advanced facilities provided by programming languages (e.g. classes) explicit provision is made for the correct initialisation of state represented as programmer variables (e.g. constructors). A thorough understanding and appreciation of the implications of state initialisation is mandatory for all programmers and explains many of the features provided by programming languages.

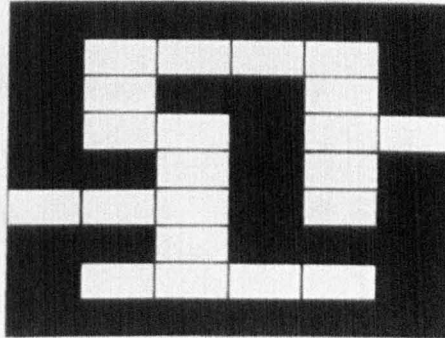
A-6 Summary

Chapter 5 set out the CoRe structure supporting the classification of the concept *state* as a threshold concept in computing. It identified a set of big-ideas developed using an autoethnographic approach from my pedagogic content knowledge and articulated them within a wider context drawn from the literature and the practice of others. This Appendix grounds these ideas in the material pedagogy of my classroom exemplifying the CoRe in my own practice.

Forward, Forward, Forward,
 Forward, Forward, Forward,
 Forward, Forward,
 Forward, Forward, Forward,
 Forward,
 Forward,
 Forward, Forward,
 Forward,
 Forward, Forward, Forward,
 Forward,
 Forward, Forward,
 Forward, Forward, Forward,
 Forward, Forward



Bart Simpson's Maze I

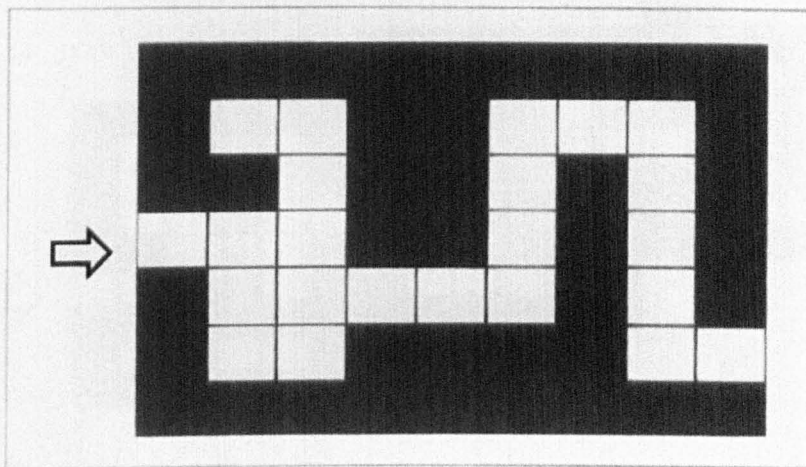


EL1

$f1 = \{ \text{face right, face right, face right} \}$
 $f = \{ \text{forward} \}$
 $f2 = \{ \text{forward, forward} \}$
 $f3 = \{ \text{forward, forward, forward} \}$
 $f r = \{ \text{face right} \}$

Bart Simpson's Maze II

$f3$
 $f r$
 $f3$
 f
 $f1$
 $f3$
 $f r$
 $f2$
 $f r$
 $f3$
 f
 $f6$
 $f2$

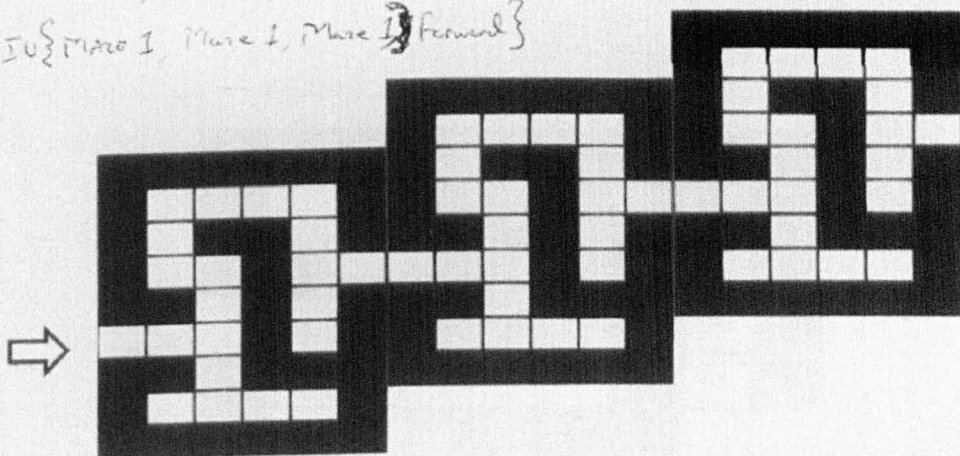


CW2

$f3 = \{ \text{forward, forward, forward} \}$
 $LEFT = \{ \text{Face Right, Face Right, Face Right} \}$
 $f4 = \{ \text{Forward, Forward, Forward, Forward} \}$
 $f2 = \{ \text{Forward, Forward} \}$

Bart Simpson's Maze IV

$MAZE I = \{ f3, LEFT, f2, LEFT, Forward, Face Right, f2, Face Right, f4, Face Right, f2, LEFT, Forward \}$
 $MAZE IV = \{ MAZE I, Maze I, Maze I, Forward \}$



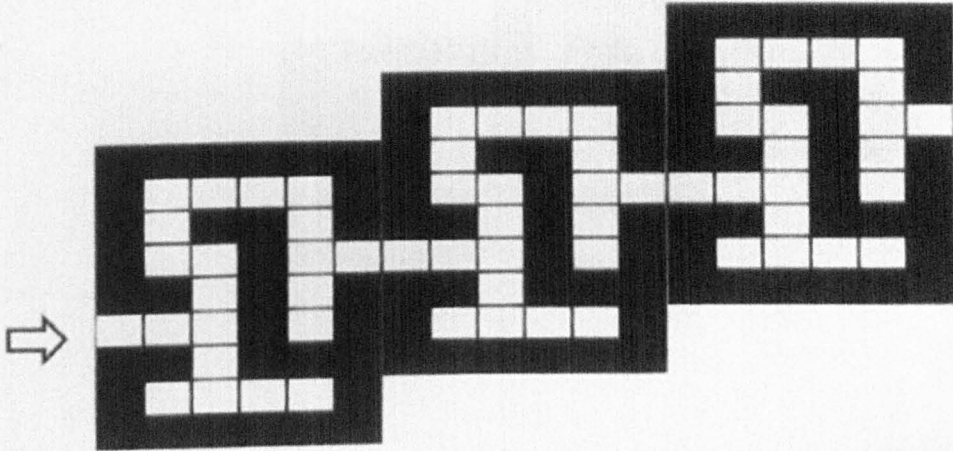
FR4

$fw2 = \text{Name } \{ \text{Forward Forward} \}$
 $fr3 = \text{Name } \{ \text{FaceRight FaceRight FaceRight} \}$
 $sm1 = \text{Name } \{ fw2 \text{ Forward } fr3 \text{ } fw2 \text{ } fr3 \text{ Forward FaceRight } fw2 \text{ FaceRight } fw2 \text{ Forward FaceRight } fw2 \text{ FaceRight Forward} \}$

Solution:

$sw1$
 $sw1$
 $sw1$
 Forward

Bart Simpson's Maze IV

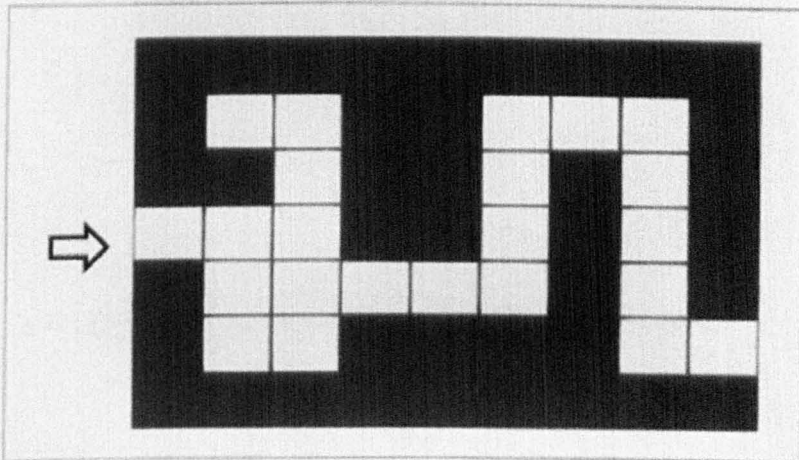


BG4

$w = \{ \text{Forward} \}$
 $x = \{ \text{Forward Forward Forward} \}$
 $y = \{ \text{FaceRight} \}$
 $z = \{ \text{FaceRight FaceRight FaceRight} \}$

Bart Simpsons Maze II

Solution



X
 Y
 Z
 X
 X
 X
 X
 Y
 Y
 Y
 Y
 X
 Z
 Z

JMC2

$L = \text{Name} \{ \text{Face Right}, \text{Face Right}, \text{Face Right} \}$
 $F2 = \text{Name} \{ \text{Forward}, \text{Forward} \}$
 $F4 = \{ \{ F2, F2 \}$

$F = \text{Name} \{ \text{Forward} \}$
 $F3 = \text{Name} \{ \text{Forward} \}$
 $R = \text{Name} \{ \text{Face Right} \}$

Bart Simpson's Maze IV

$\text{Repeat} = \text{Name} \{ F3, L, F2, L, F, R, F2, R, F3, R, F2, L, F2 \}$
 $\text{Repeat3} = \text{Name} \{ \text{Repeat}, \text{Repeat}, \text{Repeat} \}$

Solution 1

F3 F4

L L

F2 L

L F2

F L

R L

F2 F

R R

F3 R

R F2

R R

F2 F3

L R

F4 F2

L L

L L

L F2

L F

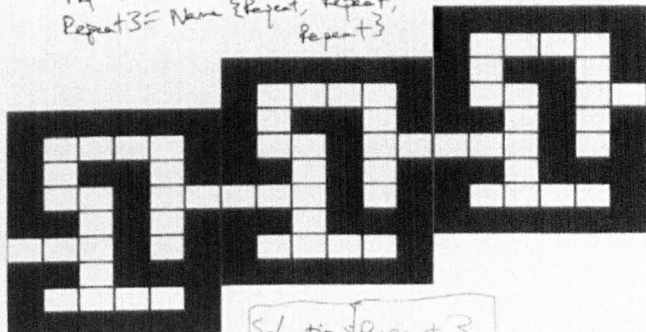
R R

F2 R

R F3

R F2

L



Solution Repeat 3

2

MW4

References

- Ahmadzadeh, M., D. Elliman, et al. (2005). An Analysis of Patterns of Debugging Among Novice Computer Science Students. ITICSE. Lisbon, Association of Computing Machinery: 84-88.
- Almstrum , V. L., D. Ginat, et al. (2002). "Import and Export to/from Computing Science Education: The Case of Mathematics Education Research." SIGCSE Bulletin **34**(3): 193-194.
- Anderson, L. (2006). "Analytic Autoethnography." Journal of Contemporary Ethnography **35**(4): 373-395.
- Attenborough, D. (2005). Life in the Undergrowth. London, BBC Books.
- Ausubel, D. P. (1968). Educational Psychology: A Cognitive View. New York, Holt, Rinehart and Winston.
- Bak, P. (1996). How Nature Works: The science of self organized criticality. New York, Copernicus.
- Barnett, R. (2007). A Will to Learn. Berkshire, McGraw Hill.
- Barsalou, L. W. (1983). "Ad Hoc Categories." Memory and Cognition **11**: 211-227.
- Bartlett, F. C. (1932). Remembering: A Study in Experimental and Social Psychology. London, Cambridge University Press.
- Bayman , P. and R. E. Mayer (1983). "Diagnosis of beginning programmers' misconceptions of BASIC programming statements." Communications of the ACM **26**(9): 677-679.
- Beaty, L. (2006). Foreward. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge.
- Berney, L. and D. B. Blane (2003). "The lifegrid method of collecting retrospective information from people at older ages." Research Policy and Planning **21**(2): 13-22.
- Blane, D. B. (1996). "Collecting retrospective data: Development of a reliable method and a pilot study of." Social Science and Medicine **42**(5): 751-757.
- Bodanis, D. (2001). E=mc2:A Biography of the World's ,ost Famous Equation. London, Pan Books.
- Booch, G. (1994). Object-oriented analysis and design. Menlo Park, CA, The Benjamin/Cummings Publishing Company.

- Boulay, B. d., T. O'Shea, et al. (1981). "The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices." International Journal of Man-Machine Studies **14**: 237-249.
- Boustedt, J., A. Eckerdal, et al. (2007). "Threshold concepts in computer science: do they exist and are they useful?" SIGCSE Bulletin **39**(1): 504-508.
- Boyer, E. L. (1990). Scholarship Reconsidered: Priorities of the Professoriate. New York, The Carnegie Foundation for the Advancement of Teaching.
- Brewer, W. F. and J. C. Treyens (1981). "Role of Schemata in Memory for Places." Cognitive Psychology **13**: 207-230.
- Britcher, B. (1995). "A Few (Proposed) Fundamental Laws of Programming." ACM SIGSOFT **20**(2): 19-20.
- Brooks, F. P. (1996). "The Computer Scientist as Toolsmith II." Communications of the ACM **39**(3): 61-68.
- Brown, J. S., A. Collins, et al. (1989). "Situated Cognition and the Culture of Learning." Educational Researcher **18**(1): 32-42.
- Brown, R. (1958). "How Shall a Thing be Called." Psychological Review **65**(1): 14-21.
- Bruner, J. S. (1962). On Knowing: Essays for the left hand. Cambridge, MA, Harvard University Press.
- Bruner, J. S. (1969). The Process of Education. London, Oxford University Press.
- Burbules, N. C. (2000). "Aporias, Webs and Passages: Doubt as an Opportunity to Learn." Curriculum Inquiry **30**(2): 171-187.
- Camerer, C., G. Loewenstein, et al. (1989). "The Curse of Knowledge in Economic Settings: An Experimental Analysis." The Journal of Political Economy **97**(5): 1232-1254.
- Carroll, J. M. and M. B. Rosson (1987). Paradox of the Active User. Interfacing Thought: Cognitive Aspects of Human- Computer Interaction. J. M. Carroll. Cambridge, MIT Press: 80-111.
- Cassirer, E. (1923). Substance and Function and Einstein's Theory of Relativity. London, The Open Court Company.
- Chabris, C. and D. Simons (2010). The Invisible Gorilla: And other ways out intuition deceives us. London, Harper Collins Publishers.
- Chang, H. (2008). Autoethnography as Method, Left Coast Press
- Chi, M. T. H. (2005). "Commonsense Conceptions of Emergent Processes: Why some misconceptions are robust." The Journal of the Learning Sciences **14**(2): 161-199.

- Clancy, M. (2004). Misconceptions and Attitudes that Interfere with Learning to Program. Computer Science Education Research. S. Fincher and M. Petre. London, Routledge Falmer.
- Clancy, M., J. Stasko, et al. (2001). "Models and Areas for CS Education Research." Computer Science Education **11**(4): 323-341.
- Clayton, M. J. (1997). "Delphi: A Technique to Harness Expert Opinion for Critical Decision-Making Task in Education." Educational Psychology **17**: 373-386.
- Corney, M., R. Lister, et al. (2011). Early Relational Reasoning and the Novice Programmer: Swapping as the "Hello World" of Relational Reasoning. 13th Australian Computer Education Conference. J. Hamer and M. de Raadt. Perth, Australia, Australian Computer Society. **114**.
- Cousin, G. (2008). Threshold Concepts: Old wine in new bottles or a new form of transactional curriculum inquiry? Threshold Concepts within the Disciplines. R. Land, J. H. Meyer and J. Smith. Rotterdam, Sense Publishers: 261-272.
- Cousin, G. (2010). "Neither Teacher-Centred nor Student-Centred: Threshold concepts and research partnerships." Journal of Learning Development in Higher Education **2**: 1-9.
- Covey, S. R. (1989). The 7 Habits of Highly Effective People. London, Simon and Schuster.
- Dahl, O. J., E. W. Dijkstra, et al. (1978). Notes on Structured Programming. London, Academic Press.
- Dahotre, A., Y. Zhang, et al. (2010). A Qualitative Study of Animation of Programming in the Wild. ESLM 10. Bolzano-Bozen Italy, Association of Computing Machinery.
- Davies, P. (2003). Threshold Concepts: how can we recognise them? EARLI Conference. Padova.
- Deese, J. (1959). "On the prediction of occurrence of particular verbal intrusions in immediate recall." Journal of Experimental Psychology **58**: 17-22.
- Denning, P. and A. D. McGettrick (2005). "Recentering Computer Science?" Communications of the ACM **48**(11): 15-19.
- Denning, P. J. (2003). "Great Principles of Computing." Communications of the ACM **46**(11): 15-20.
- Devlin, K. (2004). "When Google becomes ePlay." Retrieved 01 December, 2012, from http://www.maa.org/devlin/devlin_10_04.html.
- Dienes, Z. P. (1963). Building Up Mathematics. New York, Hutchinson Educational.
- Dijkstra, E. W. (1972). "The humble programmer." Communications of the ACM **15**(10): 859-866.
- Dijkstra, E. W. (1976). A Discipline of Programming. New Jersey, Prentice-Hall Inc.

Dijkstra, E. W. (1989). "On the cruelty of really teaching computing science." Communications of the ACM **32**(12): 1398-1414.

Dijkstra, E. W. (1996). Foreword. Teaching and Learning Formal Methods. C. N. Dean and M. G. Hinchey. London, Academic Press.

Dijkstra, E. W. (1999). "Computing Science: Achievements and Challenges. EWD1284." Retrieved 01 December, 2012, from <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1284.html>.

Dromey, R. G. (1982). How to solve it by computer, Prentice-Hall.

du Boulay, B. (1986). "Some difficulties of learning to program." Journal of Educational Computing Research **2**(1): 57-73.

du Boulay, B. (1989). Some difficulties of learning to program. Studying the Novice Programmer. E. Soloway and J. C. Sopher. Hillsdale, New Jersey, Lawrence Erlbaum Associates: 283-299.

Dunbar, K. (1997). How scientists think: Online creativity and conceptual change in science. Conceptual structures and processes: Emergence, discovery and Change. T. B. Ward, S. M. Smith and S.Vaid. Washington DC, APA Press: 461-493.

Eckerdal, A., R. McCartney, et al. (2007). From Limen to Lumen: computing students in liminal spaces. ICER 07. New York, ACM: 123-132.

Eckerdal, A., R. McCartney, et al. (2006). Putting Threshold Concepts into Context in Computer Science Education. ITICSE 06. Bologna Italy, ACM: 103-107.

Eden, A. H. (2007). "Three Paradigms of Computer Science." Mind and Machines **17**(2): 135-167.

Efklides, A. (2006). Metacognition, Affect, and Conceptual Difficulty. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge: 48-69.

Einstein, A. and L. Infeld (1938). The Evolution of Physics : From Early Concept to Relativity and Quanta. England, Cambridge University Press.

Entwistle, N. (2008). Threshold Concepts and Transformative Ways of Thinking within Research into Higher Education. Threshold Concepts within the Disciplines. R. Land, J. H. F. Meyer and J. Smith. Rotterdam, Sense Publishers: 21-35.

Fincher, S. and M. Petre (2004). Computer Science Education Research. London, Routledge Falmer.

Fischhoff, B. (1975). "Hindsight is not equal to Foresight: The Effect of Outcome Knowledge on Judgment Under Uncertainty." Journal of Experimental Psychology: Human Perception and Performance **1**(3): 288-299.

- Fitzgerald, S., G. Lewandowski, et al. (2008). "Debugging: Finding, Fixing and Failing, a Multi-Institutional Study of Novice Debuggers." Computer Science Education **18**(2): 93-116.
- Flanagan, J. C. (1954). "The Critical Incident Technique." Psychological Bulletin **51**(4): 335.
- Fleury, A. E. (1991). "Parameter passing: the rules the students construct." ACM SIGCSE Bulletin **23**(1): 283-286.
- Gal-Ezer, J. and D. Harel (1998). "What (Else) Should CS Educators Know." Communications of the ACM **41**(9): 77-84.
- Garforth, F. W. (1965). John Dewey : Selected Educational Writings. London, Heinemann.
- Garner, W. R. (1974). The Processing of Information and Structure. New York, Wiley.
- Gellner, E. (2005). Words and Things. London, Routledge Classics.
- Gentner, D. and A. L. Stevens, Eds. (1983). Mental Models. London, Lawrence Erlbaum Associates.
- George, C. E. (2000). "EROSI – Visualising Recursion and Discovering New Errors." SIGCSE Bulletin **32**(1): 305-309.
- Gess-Newsome, J. and N. G. Lederman, Eds. (1999). Examining Pedagogical Content Knowledge: The Construct and its Implications for Science Education. London, Kluwer Academic Publishers.
- Goldman, K., P. Gross, et al. (2008). Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process. SIGCSE 08. New York, ACM: 256-260.
- Gould, S. J. (1993). Eight Little Piggies: Reflections in Natural History. London, W.W. Norton and Company.
- Granger, C. (2012). Silent Moments in Education: An Autoethnography of Learning, Teaching, and Learning to Teach, University of Toronto Press.
- Gregor, D., J. Järvi, et al. (2006). Concepts: linguistic support for generic programming in C++
- OOPSLA '06 : Proceedings of the 21st annual ACM SIGPLAN OOPSLA Conference: 291-310.
- Gremler, D. D. (2004). "The CIT in Service Research." Journal of Service Research **7**(1): 65-89.
- Gries, D. (2008). Foreward. Reflections on the Teaching of Programming: Methods and Implementations. J. Bennedsen, M. E. Caspersen and M. Kölling. Berlin, Springer- Verlag.
- Gudmundsdottir, S. (1990). Curriculum Stories. Insights into Teachers' Thinking and Practice. C. Day, P. Denicolo and M. Pope, Falmer Press: 107-118.

Gudmundsdottir, S. (1995). The Narrative Nature of Pedagogical Content Knowledge Narrative in Teaching, Learning and Research. H. McEwan and K. Egan. New York, Teachers College Press.

Guilbault, R. L., F. B. Bryant, et al. (2004). "A Meta-Analysis of Research on Hindsight Bias." Basic and Applied Psychology **26**(2 and 3): 103-117.

Hadamard, J. (1954). The Psychology of Invention in the Mathematical Field. Dover, Dover Publications.

Hampton, J. A. (2007). "Typicality, Graded Membership, and Vagueness." Cognitive Science: A Multidisciplinary Journal **31**(3): 355-384.

Hawkins, S. A. and R. Hastie (1990). "Hindsight: Biased Judgments of Past Events After the Outcomes Are Known." Psychological Bulletin **107**(3): 311-327.

Hayler, M. (2007). Autoethnography, Self-Narrative and Teacher Education, Sense Publishers.

Henriksen, K. and H. Kaplan (2003). "Hindsight bias, outcome knowledge and adaptive learning." Quality and Safety in Health Care **12**(2): 46-50.

Hoch, S. J. and G. F. Loewenstein (1989). "Outcome Feedback: Hindsight and Information." Journal of Experimental Psychology: Learning, Memory and Cognition **15**(4): 605-619.

Hofstadter, D. (2007). I am a Strange Loop. New York, Basic Books.

Holloway, C. M. (1995). "Software Engineering and Epistemology." ACM SIGSOFT **20**(2): 20.

Hristova, M., A. Misra, et al. (2003). "Identifying and Correcting Java Programming Errors for Introductory Computer Science Students." SIGCSE Bulletin **35**(1): 153-156.

Jackendoff, R. (1989). "What is a Concept, that a Person May Grasp it?" Mind and Language **4**(1&2): 68-102.

Jadud, M. C. (2005). "A first look at novice compilation behavior." Computer Science Education **15**(1): 25-40.

Johnson-Laird, P. N. (1983). Mental Models. Cambridge MA, Harvard University Press.

Kaczmarczyk, L. C., E. R. Petrick, et al. (2010). "Identifying Student Misconceptions of Programming." Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10: 107-111.

Kahney, H. (1989). What do novice programmers know about recursion? Studying the Novice Programmer. E. Soloway and J. C. Spohrer. Hillside, New Jersey, Lawrence Elbaum Associates: 209-228.

Katz, I. and J. Anderson (1987). "Debugging: An analysis of bug location strategies." Human-Computer Interaction **3**(4): 351-399.

- Knuth, D. E. (1974). "Computer Science and its Relation to Mathematics." American Mathematical **81**: 323-343.
- Komatsu, L. K. (1992). "Recent Views of Conceptual Structure." Psychological Bulletin **112**(3): 500-526.
- Kuhn, T. S. (1996). The Structure of Scientific Revolutions. London, The University of Chicago Press.
- Kurland, D. M. and R. D. Pea (1989). Children's mental models of recursive Logo programs. Studying the Novice Programmer. E. Soloway and J. C. Sopher. Hillside, New Jersey, Lawrence Erlbaum Associates: 315-323.
- Kwasnik, B. H. (1999). "The Role of Classification in Knowledge Representation and Discovery." Library Trends **48**(1): 22-47.
- Lahtinen, E. and T. Ahoniemi (2005). Visualizations to Support Programming on Different Levels of Cognitive Development. Fifth Koli Calling Conference on Computer Science Education, Turku Centre for Computer Science.
- Lakoff, G. (1987). Women, Fire and Dangerous Things: What Categories Reveal about the Mind. Chicago, University of Chicago Press.
- Lambrecht, J. J. (1999). Developing employment-related office technology skills. MDS-1199. Berkeley, CA, National Center for Research in Vocational Education.
- Lambrecht, J. J. (2000). "Developing End-User Technology Skills." Information Technology, Learning, and Performance Journal **18**(1): 7-19.
- Land, R., G. Cousin, et al. (2006). Conclusion: Implications of threshold concepts for course design and evaluation. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge: 195-206.
- Land, R., J. H. Meyer, et al. (2008). Editor's Preface. Threshold Concepts within the Disciplines. R. Land, J. H. Meyer and J. Smith. Rotterdam, Sense Publishers.
- Land, R., J. H. Meyer, et al., Eds. (2008). Threshold Concepts within the Disciplines. Rotterdam, Sense Publishers.
- Land, R., J. H. F. Meyer, et al. (2010). Editor's Preface. Threshold Concepts and Transformational Learning. J. H. F. Meyer, R. Land and C. Baillie. Rotterdam, Sense Publishers.
- Laurance, S. and E. Margolis (1999). Concepts and Cognitive Science. Concepts: Core Readings. S. Laurance and E. Margolis. Cambridge, MIT Press.
- Lewis, C. M. (2012). "The Importance of Students' Attention to Program State: A Case Study of Debugging Behavior." Proceedings of the 9th International computing education research conference 127-134.

Liberman, N., Y. B.-D. Kolikant, et al. (2009). In-service teachers learning of a new paradigm: a case study. ICER '09 Proceedings of the fifth international workshop on Computing education research workshop. Berkeley, CA, ACM New York, NY: 43-50.

Liskov, B. and S. Zilles (1974). "Programming with abstract data types." ACM SIGPLAN Conference on Very High Level Languages: 50-59.

Liskov, B. H. (1972). "A design methodology for reliable software systems." Proceedings of the Fall Joint Computer Conference: 191-198.

Lister, R., E. S. Adams, et al. (2004). "A Multi-National Study of Reading and Tracing Skills in Novice Programmers. ." SIGCSE Bulletin 36(4): 119-150.

Loftus, E. F. and J. C. Palmer (1974). "Reconstruction of Automobile Destruction : An Example of the Interaction Between Language and Memory." JOURNAL OF VERBAL LEARNING AND VERBAL BEHAVIOR 13: 585-589.

Lopez, M., J. Whalley, et al. (2008). "Relationships between Reading, Tracing and Writing Skills in Introductory Programming." Proceedings of the Fourth International Workshop on Computing Education Research, ICER '08: 101-112.

Loughran, J., A. Berry, et al. (2006). Understanding and Developing Science Teachers' Pedagogical Content Knowledge. Rotterdam, Sense Publishers.

Loui, M. C. (1995). "Computer Science Is a New Engineering Discipline." ACM Computing Surveys 27(1): 31-32.

Lynch, E. B., J. D. Coley, et al. (2000). "Tall is Typical: Central Tendency, Ideal Dimensions and Graded Category Structure Among Tree Experts and Novices." Memory and Cognition 28(1): 41-50.

Madison, S. K. (1995). A study of College students' construct of parameter passing: Implications for instruction. Computer Science, University of Wisconsin-Milwaukee. PhD.

Madsen, O. L., B. Möller-Pedersen, et al. (1993). Object-oriented Programming in the Beta Programming Language, Addison Wesley.

Maiden, N. A. M. and M. Hare (1998). "Problem domain categories in requirements engineering " International Journal of Human-Computer Studies 49(3): 281-304.

Maloney, J., K. Peppler, et al. (2008). Programming by Choice: Urban Youth Learning Programming by Scratch. SIGCSE 08. Portland Oregon USA, Association of Computing Machinery: 368-371.

Marton, F. and R. Säljö (1976). "On qualitative differences in learning: Outcome and process " British Journal of Educational Psychology 46(1): 4-11.

Mavaddat, F. (1981). "Another experiment with teaching of programming languages." SIGCSE Bulletin 13(2): 49-56.

- Mayer, R. E. (1976). "Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order." Journal of Educational Psychology 68: 143-150.
- Mayer, R. E. (1981). "The Psychology of How Novices Learn Computer Programming." ACM Computing Surveys 13(1): 121-141.
- Mayer, R. E. (1989). The psychology of how novices learn computer programming. Studying the Novice Programmer. E. Soloway and J. C. Sopher. Hillsdale, New Jersey, Lawrence Erlbaum Associates: 129-159.
- McCartney, R., A. Eckerdal, et al. (2007). "Successful students' strategies for getting unstuck." SIGCSE Bulletin 39(3): 156-160.
- McCracken, M., V. Almstrum, et al. (2001). "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students." ACM SIGCSE Bulletin 33(4): 125-180.
- Medin, D. L. (1989). "Concepts and Conceptual Structure." American Psychologist 44(12): 1469-1481.
- Medin, D. L. and E. E. Smith (1984). "Concepts and Concept Formation." Annual Review Psychology 35(113).
- Mednick, S. A. (1962). "The Associative Basis of the Creative Process." Psychological Review 69(3): 220-232.
- Meerbaum-Salant, O., M. Armoni, et al. (2010). Learning Computer Science Concepts with Scratch. ICER 2010. Aarhus Denmark, Association of Computing Machinery: 69-76.
- Meerbaum-Salant, O., M. Armoni, et al. (2011). Habits of Programming in Scratch. ITICSE 11. Darmstadt Germany, Association of Computing Machinery: 168-172.
- Mervis, C. B. and E. Rosch (1981). "Categorization of Natural Objects." Annual Review of Psychology 32: 89-115.
- Meyer, J. and R. Land (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. Occasional Report 4, University of Edinburgh.
- Meyer, J. H. F. (2010). Helping our students : learning, metalearning, and threshold concepts. Taking Stock: Research on teaching and learning in higher education. Montreal, McGill-Queen's University Press: 191-213.
- Meyer, J. H. F. and R. Land (2003). Threshold concepts and troublesome knowledge (1) – linkages to ways of thinking and practising. Improving student learning – ten years on. C. Rust. Oxford, Oxford Centre for Staff and Learning Development.

Meyer, J. H. F. and R. Land (2005). "Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning." Higher Education **49**: 373-388.

Meyer, J. H. F. and R. Land (2006). Editor's Preface. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge J. H. F. Meyer and R. Land. Abingdon, Routledge.

Meyer, J. H. F. and R. Land, Eds. (2006). Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. Abingdon, Routledge.

Meyer, J. H. F. and R. Land (2006). Threshold Concepts and Troublesome Knowledge: An introduction. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge: 3-18.

Meyer, J. H. F. and R. Land (2006). Threshold Concepts and Troublesome Knowledge: Issues of Liminality. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer. Abingdon, Routledge: 19-32.

Meyer, J. H. F., R. Land, et al., Eds. (2010). Threshold Concepts and Transformational Learning. Rotterdam, Sense Publishers.

Michalski, R. S. (1989). Two-tiered concept meaning, inferential matching, and conceptual cohesiveness. Similarity and Analogical Reasoning. S. Vosniadou and A. Ortony. New York, Cambridge University Press: 122-145.

Milne, I. and G. Rowe (2002). "Difficulties in Learning and Teaching Programming – Views of Students and Tutors." Education and Information Technologies **7**(1): 55-66.

Milner, R. (2006). "Ubiquitous Computing: Shall we Understand it?" The Computer Journal **49**(4): 383-389.

Minsky, M. (1986). The Society of Mind. New York, Simon and Schuster.

Moström, J. E., J. Boustedt, et al. (2008). Concrete Examples of Abstraction as Manifested in Students' Transformative Experiences. ICER 08. Sydney Australia, ACM: 125-135.

Murphy, G. (2002). The Big Book of Concepts. Massachusetts, MIT Press.

Murphy, G. L. and D. L. Medin (1985). "The Role of Theories in Conceptual Coherence." Psychological Review **92**(3): 289-316.

Murphy, L., R. McCauley, et al. (2012). 'Explain in Plain English' Questions: Implications for Teaching. SIGCSE 12. Rayleigh USA, Association of Computing Machinery: 385-390.

Nanja, M. and C. R. Cook (1987). An analysis of the on-line debugging process. Empirical studies of programmers, Second workshop. G. Olson, S. Sheppard and E. Soloway. Norwood, NJ, Ablex: 172-184.

Nathan, M. J. and A. Petrosino (2003). "Expert Blind Spot Among Preservice Teachers." American Educational Research Journal 40(4): 905-928.

Newman, W. (1994). A Preliminary Analysis of the Products of HCI Research, Using Pro Forma Abstracts. Computer Human Interaction 94 Conference. Boston MA: 278-284.

Nievergelt, J. (1980). Computer Science Education – an emerging consensus on basic concepts. IFIP80. S. H. Lavington.

Norman, D. (1989). The Design of Everyday Things. New York, Doubleday.

O'Donnell, R. (2010). A critique of the Threshold Concept hypothesis and an application in Economics, University of Technology, Sydney.

Olson and Bruner (1999). Folk Psychology and Folk Pedagogy. The Handbook of Education and Human Development, Cambridge University Press.

Osmond, J., A. Turner, et al. (2008). Threshold Concepts and Spatial Awareness in Transport and Product Design. Threshold Concepts within the Disciplines. R. Land, J. H. F. Meyer and J. Smith. Rotterdam, Sense Publishers: 243-258.

Parnas, D. (1972). "On the criteria to be used in decomposing systems into modules." Communications of the ACM 15(2): 1053-1058.

Pea, R. D. (1986). "Language-independent conceptual "bugs" in novice programming." Journal of Educational Computing Research 2(1): 25-36.

Pea, R. D. and D. M. Kurland (1984). "On the Cognitive Effects of Learning Computer Programming." New Ideas Psychology 2(2): 137-168.

Perkins, D. (1999). "The Many Faces of Constructivism." Educational Leadership 57(3): 6-11.

Perkins, D. (2006). Constructivism and troublesome knowledge. Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge.

Perkins, D. (2006). Constructivism and Troublesome Knowledge. Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge: 33-47.

Perkins, D. (2010). Forward. Threshold Concepts and Transformational Learning. J. H. F. Meyer, R. Land and C. Baillie. Rotterdam, Sense Publishers.

Perkins, D. N., C. Hancock, et al. (1989). Conditions of Learning in Novice Programmers. Studying the Novice Programmer. E. Soloway and J. C. Spohrer. Hillside, New Jersey, Lawrence Elbaum Associates: 261-279.

Perkins, D. N. and F. Martin (1986). Fragile knowledge and neglected strategies in novice programmers. Empirical Studies of Programmers. E. Soloway and S. Iyengar. Norwood New Jersey, Ablex Publishing Co.: 213-229.

- Polanyi, M. (1983). The Tacit Dimension. USA, Doubleday and Company, Inc.
- Polya, G. (1990). How to Solve it. London, Penguin Books.
- Pólya, G. and G. Szegő (1972). Problems and Theorems in Analysis Vol. 1. New York, Springer-Verlag.
- Prosser, M. and K. Trigwell (1999). Understanding learning and teaching: The experience in higher education. Buckingham, Society for Research into Higher Education and Open University Press.
- Putnam, R. T., D. Sleeman, et al. (1989). A summary of misconceptions of high school BASIC programmers. Studying the Novice Programmer. E. Soloway and J. C. Sopher. Hillsdale, New Jersey, Lawrence Erlbaum Associates: 301-314.
- Ragonis, N. and M. Ben-Ari (2005). "A Long-Term Investigation of the Comprehension of OOP Concepts by Novices." Computer Science Education 15(3): 203-221.
- Randolph, J. (2007). Computer science education research at the crossroads: A methodological review of the computer science education research: 2000-2005. Logan, Utah, Utah State University. PHD.
- Randolph, J. J., R. Bednarik, et al. (2005). A Methodological Review of the Articles Published in the Proceedings of Koli Calling 2001-2004. 5th Annual Finnish / Baltic Sea Conference on Computer Science Education. Helsinki, Finland: 103-109.
- Rasmussen, J. (1987). The Definition of Human Error and a Taxonomy for Technical System Design. New Technology and Human Error. J. Rasmussen, K. Duncan and J. Lwplat. Chchester, Wiley: 23-30.
- Rayside, D. and G. T. Campbell (2000). An Aristotelian understanding of object-oriented programming. OOPSLA '00 : Proceedings of the 15th ACM SIGPLAN OOPSLA Conference: 337-353.
- Rayside, D. and G. T. Campbell (2000). Aristotle and object-oriented programming: why modern students need traditional logic. SIGCSE '00 : Proceedings of the thirty-first SIGCSE technical symposium on Computer science education 237-244.
- Rayside, D. and K. Kontogiannis (2001). On the syllogistic structure of object-oriented programming. ICSE '01 : Proceedings of the 23rd International Conference on Software Engineering 113-122.
- Roberts, E. (1995). "Loop exits and structured programming: Reopening the debate." ACM SIGCSE Bulletin 27(1): 268-272.
- Robins, A., J. Rountree, et al. (2003). "Learning and Teaching Programming: A Review and Discussion." Computer Science Education 13(2): 137-172.

Roediger, H. L. and K. B. McDermott (1995). "Creating False Memories: Remembering Words Not Presented in Lists." Journal of Experimental Psychology: Learning, Memory, and Cognition **21**(4): 803-814.

Rosch, E. (1978). Principles of Categorization. Cognition and Categorization. E. Rosch and B. B. Lloyd. Hillsdale, Lawrence Erlbaum Associates Publishers: 27-48.

Rosch, E., C. B. Mervis, et al. (1976). "Basic Objects in Natural Categories." Cognitive Psychology **8**: 382-439.

Rowbottom, D. P. (2007). "Demystifying Threshold Concepts." Journal of Philosophy of Education **41**(2): 264-270.

Ruelle, D. (1998). "In Retrospect: A book review of Poincaré's Science and Method." Nature **391**: 760.

Russell, B. (2009). The Basic Writings of Bertrand Russell. London, Routledge Classics.

Ryder, B. G., M. L. Soffa, et al. (2005). "The Impact of Software Engineering: Research on Modern Programming Languages." ACM transactions on Software Engineering and Methodology **14**(4): 431-477.

Saeli, M., J. Perrenet, et al. (2012). "Programming: Teachers and Pedagogical Content Knowledge in the Netherlands, ." Informatics in Education **11**(1): 81-114

Sajaniemi, J. and M. Kuittinen (2005). "An Experiment on Using Roles of Variables in Teaching Introductory Programming." Computer Science Education **15**(1): 59-82.

Samurçay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. Studying the Novice Programmer. E. Soloway and J. C. Sopher. Hillsdale, New Jersey, Lawrence Erlbaum Associates: 161-178.

Sanders, K., J. Boustedt, et al. (2008). Student Understanding of ObjectOriented Programming as Expressed in Concept Maps. SIGCSE 08. Portland Oregon, ACM: 332-336.

Savin-Baden, M. (2006). Disjunction as a form of Troublesome Knowledge in Problem-Based Learning. Overcoming Barriers to Understanding: Threshold concepts and troublesome knowledge. J. H. F. Meyer and R. Land. Abingdon, Routledge: 160-172.

Schön, D. (1983). The Reflective Practitioner : How professionals think in action, Basic Books.

Schwartzman, L. (2010). Transcending disciplinary boundaries: A proposed theoretical foundation for threshold concepts. Threshold Concepts and Transformational Learning. J. H. F. Meyer, R. Land and C. Baillie. Rotterdam, Sense Publishers: 21-44.

Schwartzman, L. (2010). Transending Disciplinary Boundaries: A proposed theoretical foundation for threshold concepts. Threshold Concepts and Transformational Learning. J. H. F. Meyer, R. Land and C. Baillie. Rotterdam, Sense Publications: 21-44.

- Schwill, A. (1994). "Fundamental ideas of computer science." European Assoc. for Theoretical Computer Science **53**: 274-295.
- Seldon, A. (1983). By world of mouth: Elite oral history. London, Methuen.
- Shackelford, R. L. and A. N. Badre (1993). "Why can't smart students solve simple programming problems?" International Journal of Man-Machine Studies **38**: 985-997.
- Shaw, M. (2003). Writing Good Software Engineering Research Papers. 25th International Conference on Software Engineering, IEEE Computer Society: pp.726-736.
- Sheil, B. A. (1981). "The Psychological Study of Programming." ACM Computing Surveys **13**(1): 101-120.
- Shulman, L. (1999). Foreward. Examining Pedagogical Content Knowledge: The Construct and its Implications for Science Education. J. Gess-Newsome and N. G. Lederman. London, Kluwer Academic Publishers.
- Shulman, L. S. (1986). "Those Who Understand: Knowledge Growth in Teaching." Educational Researcher **15**(2): 4-14.
- Shulman, L. S. (1987). "Knowledge and Teaching: Foundations of the New Reform." Harvard Educational Review **57**(1): 1-22.
- Simon (2011). Assignment and Sequence: Why Some Students Can't Recognise a Simple Swap. Koli Calling. Koli Finland, Association of Computing Machinery: 10-15.
- Simon, H. A. (1962). "The Architecture of Complexity." Proceedings of the American Philosophical Society **106**(6): 467-482.
- Sloman, S. A., B. C. Love, et al. (1998). "Feature Centrality and Conceptual Coherence." Cognitive Science: A Multidisciplinary Journal **22**(2): 189 - 228.
- Smith, E. E. and D. L. Medin (1981). Categories and Concepts. London, Harvard University Press.
- Solomon, K. O., D. L. Medin, et al. (1999). "Concepts do more than categorize." Trends in Cognitive Sciences **3**(3): 99-105.
- Soloway, E., J. Bonar, et al. (1983). "Cognitive strategies and looping constructs: An empirical study." Communications of the ACM **26**(11): 853-860.
- Soloway, E. and J. C. Spohrer, Eds. (1989). Studying the Novice Programmer. New Jersey, Lawrence Erlbaum Associates.
- Sophrer, J. C. and E. Soloway (1989). Novice mistakes: Are the folk wisdoms correct? Studying the Novice Programmer. E. Soloway and J. C. Sophrer. Hillside, New Jersey, Lawrence Erlbaum Associates: 401-416.

- Sophrer, J. C., E. Soloway, et al. (1989). A Goal/Plan analysis of buggy Pascal programs. Studying the Novice Programmer. E. Soloway and J. C. Sophrer. Hillside, New Jersey, Lawrence Erlbaum Associates: 355-399.
- Sorva, J. (2012). Visual Program Simulation in Introductory Programming Education. Department of Computer Science and Engineering. Helsinki, Aalto University.
- Stigler, M. S. (2000). Statistics on the Table: The history of Statistical Concepts and Methods. Cambridge, MA, Harvard University Press.
- Tallack, P., Ed. (2001). The Science Book. London, Cassell and Co.
- Thomas, L., J. Boustedt, et al. (2010). Threshold Concepts in Computer Science: An ongoing empirical investigation. Threshold Concepts and Transformational Learning. J. H. F. Meyer, R. Land and C. Baillie. Rotterdam, Sense Publications.
- Thurston, W. P. (1990). "Mathematical Education, ." Notices of the AMS **37**: 844-850.
- Traub, J. F. (1981). "Quo Vadimus: Computer Science in a Decade." Communications of the ACM **24**(6): 351-369.
- Tversky, B. (1989). "Parts, Partonomies and Taxonomies." Developmental Psychology **25**(6): 983-995.
- Tversky, B. and K. Hemenway (1984). "Objects, Parts and Categories." Journal of Experimental Psychology **113**(2): 169-193.
- Valentine, D. W. (2004). "CS Educational Research: A Meta- Analysis of SIGCSE Technical Symposium Proceedings." SIGCSE Bulletin **36**(1): 255-259.
- Van Loocke, P., Ed. (1999). The Nature of Concepts. London, Routledge.
- van Manen, M. (1995). "On the Epistemology of Reflective Practice." Teachers and Teaching: Theory and Practice **1**(1): 33-50.
- Wittgenstein, L. (1958). Philosophical Investigations. New Jersey, Prentice Hall.
- Zadeh, L. A. (1965). "Fuzzy Sets." Information and Control **8**: 338-353.
- Zander, C., J. Boustedt, et al. (2008). Threshold concepts in computer science: a multi-national investigation. Threshold Concepts Within the Disciplines. R. Land, J. H. F. Meyer and J. Smith. Rotterdam, Sense Publications.
- Zendler, A. and C. Spannagel (2008). "Empirical Foundation of Central Concepts for Computer Science Education." ACM Journal on Educational Resources in Computing **8**(2).
- Zentall, T. R., M. Galizio, et al. (2002). "Categorization, Concept Learning and Behavior Analysis: An Introduction." Journal of the Experimental Analysis of Behavior **78**(3): 237-248.

Zuckerman, H. and J. Lederberg (1986). "Postmature Scientific Discovery." Nature 324(6098): 629-631.