**3**

**Keeping it real**

**Helping learners navigate the concrete and abstract**

Mick Chesterman

https://orcid.org/0000-0002-8005-2390

**Abstract**

This chapter highlights the value and challenges of implementing hands-on teaching approaches in a UK schooling context. It begins by looking at interpretations of inclusion and specifically the Universal Design for Learning (UDL) framework. The terms concrete and abstract and their relevance to coding and definitions of computational thinking are explored. The second half of the chapter then turns to practical ways teachers can help learners navigate abstract concepts and benefit from hands-on experience of coding. To do this, it outlines some techniques promoted by the National Centre for Computing Education (NCCE). Throughout this part of the chapter, there is a focus on linking these educational practices with inclusive approaches and learner engagement.

**Introduction**

In the UK, computing and computational devices are all around us. Young people interact with them in many ways including general communication, games, social media and creative apps. Computing as a subject and coding as an activity can draw on the diverse ways that computing touches people’s concrete lives. A quick search of the web for creative computing or tech for kids yields a multitude of activities, devices and materials designed to engage the hobby interests of young people. Physical examples include fashion- and textiles-based computing, robotics, colourful lighting displays and programming Lego constructions. This chapter celebrates the value and fun involved in coding in these contexts. However, while the range of materials and possible creative projects is promising, aligning creative opportunities with an exam-assessed curriculum is challenging.

This chapter highlights the value and challenges of implementing hands-on teaching approaches in a UK schooling context. It begins by looking at interpretations of inclusion and specifically the Universal Design for Learning (UDL) framework. The terms concrete and abstract and their relevance to coding and definitions of computational thinking (CT) are explored. The second half of the chapter then turns to practical ways teachers can help learners navigate abstract concepts and benefit from hands-on experience of coding. To do this, it outlines some techniques promoted by the National Centre for Computing Education (NCCE). Throughout this part of the chapter, there is a focus on linking these educational practices with inclusive approaches and learner engagement.

**Inclusion and inclusive pedagogies**

The term inclusion in education no longer addresses solely children with special educational needs and disabilities (SEND) but also examines barriers of culture and other exclusionary elements of the school environment and discourse (Black-Hawkins et al., 2008). The issue of alienation from the culture of computing in schools has been identified as a concern, especially for girls and some ethnic minorities (The Royal Society, 2017). To be inclusive, schools and teachers must identify both traditional SEND issues and wider cultural barriers to participation in computing classrooms and help students overcome them. Beyond the important technical accessibility tools and assistive learning technologies – for example, text in different sizes and screen readers – a diversity of inclusive teaching strategies is also needed.

One way to address SEND issues is to use differentiation to adapt the standard lesson plan for learners needing special support. However, this view of a standard, optimal learner pathway is not supported by recent research in neurodiversity, which suggests there is no one optimal way for students to learn. Inclusive pedagogies take a different approach to differentiation, which places more power in the hands of learners to choose the path that is most appropriate for them. All students are given a greater choice of materials and activities from the start suiting the varied needs of all students. This has the benefit of removing the stigmatisation of some learners having to undertake work that is perceived as being created for *low-achieving* students. These principles – among others – are presented in a framework called UDL.

**Key concept – UDL**

UDL is a set of design principles aimed at educators to help them design learning experiences that incorporate diverse ways to engage pupils and to represent the concepts being communicated. The UDL framework provides guidelines for three key areas of representation, engagement and expression/action (CAST, n.d.). You can use the following summary of UDL principles as a checklist to help you to plan inclusive teaching activities. A more comprehensive outline is provided by the CAST (Center for Applied Special Technology) organisation.[[1]](#endnote-2)

**Multiple means of representation:** Are you presenting material and concepts in multiple formats? For example, spoken presentations, written documents, graphics, hands-on activities or audio material. Are you clarifying new language and symbols in diverse ways? Are you providing important background knowledge and highlighting important patterns and relationships in the knowledge you explore?

**Multiple means of engagement:** Have you been able to increase student choice and the relevance of your material to spark learners’ excitement? Are you using a variety of ways to allow your learners to focus on their goals, maintain self-belief and sustain their effort as individuals and in group work?

**Multiple means of expression/action:** No one particular action or format of expression will be best for all students. Are you able to offer a choice in how your students interact with materials and tools (particularly assistive technologies) and allow students choice in the media they use for communication and construction? How much are students able to set their own learning goals and monitor their own progress?

UDL places great value on the personal relevance, choice and authenticity of learning experiences. As a way to encourage engagement, UDL suggests setting choices of concrete learning goals that are relevant to the learners. This learner-led approach is very different from a traditional instruction-based, directive approach to teaching. The diverse learning pathways offered can be unfamiliar for both teachers and students. The UDL guidelines recognise this and provide information to support teachers to implement them. As educators, we may need to build our own abilities and familiarity with learner-led approaches as well as grow the autonomy of our students.

One area of UDL that teachers can implement straightforwardly is to represent concepts in the classroom in a diversity of ways. In a related study, researchers Cook et al. (2016) explored the alignment of UDL with another framework, CRA, which consists of a three-stage model to support learners to develop concepts (Fyfe et al., 2014). The researchers outline how the three stages of CRA (concrete, representational and abstract) align with key UDL principles, most specifically multiple ways to represent knowledge to aid learner perception and comprehension. In short, first teachers introduce a physical, concrete model of the concept, then progress to iconic forms, for example, graphics or pictures; finally, learners work with more abstract models of the concept. The CRA framework is an example of concept popular in mathematics research and practice called concreteness fading where concepts are introduced in concrete examples and then learners are supported to understand and represent them in more abstract ways.

When reading about different approaches to teaching computing, the terms concrete and abstract are used commonly. For example, the concrete practice of coding is a good way for learners to work with more abstract computing concepts. The following section explores the utility of these terms to explore inclusive approaches to teaching, especially in relation to an understanding of CT.

**Computational thinking: The abstract and concrete**

**Concrete and abstract learning approaches**

<TX1>In an everyday sense, concrete objects are ones you can get hold of and abstract objects exist only as concepts. A pound coin is concrete, but the idea of profit is abstract. In everyday usage, abstract knowledge may be harder to grasp than more concrete understandings. For example, we might ask for a concrete example if we don’t understand a more abstract definition. Here, it is the use of something in a context that makes something concrete. In traditional conceptions of education, abstract knowledge is often perceived to be of greater value. If you can understand a concept as it applies in different situations, then this ability to transfer it and have a more global understanding is held as a higher form of knowledge. This concept is popular in education in many forms, for example, Piaget’s influential model of developmental stages, specifically in the progression to more abstract thinking in the transition from the concrete (operational) to formal (abstract) stages (Burman, 2021).

Not all educators agree with the supremacy of more abstract ideas of knowledge and are keen to celebrate the value of concrete exploration by learners. Seymour Papert and Sherry Turkle’s work on creative computing at MIT (Massachusetts Institute of Technology) created a legacy that includes the development of the Scratch programming tool and the use of physical computing in education. Papert and Turkle (1990) thought it was vital that we value and recognise concrete approaches to computing and coding. While they do not discard the technical value of abstract approaches, they draw on feminist theory to make a convincing case that approaches like abstract planning and formal language can be off-putting to certain learners and especially girls. They argue that the process of finding solutions to coding issues for novice coders should be a matter of personal preference. As well as the terms abstract and concrete – they use the terms top-down and bottom-up approaches to learning. In a bottom-up approach, problems are tackled piece by piece, experimentally. Desmond Tutu once said that “there is only one way to eat an elephant: a bite at a time”. Bottom-up (concrete) coders take this approach.

To clarify this, Papert and Turkle give the example of a young coder Lisa, who is aware of a more formal way to approach the kinds of programming tasks she is undertaking, but maintains that way doesn’t work for her. As she continues her journey as a coder, this frequent message that she is doing things the wrong way demotivates her enthusiasm around coding. Abstract coding concepts are tools for thinking. But they are only useful to the learner if they match with her experiences. Forcing the learner to adopt an abstract approach in this example is counterproductive as it undermines her experience and progress. The authors outline that the danger of prioritising teaching and testing of abstract concepts is to devalue this bottom-up approach to coding. This way of coding has also been called a craft approach. It is a way of doing things that has been shown to have a lot of value in many professions.

Thus as inclusive educators, we have a duty to allow learners to follow a learning path that suits them as much as is practical within the constraints of the curriculum. A conceptual tool that may help teachers to help learners to navigate between the abstract and concrete is semantic profiling.

**Key concept – semantic profiles**

Semantic profiles chart the use of more concrete (high semantic gravity) language and more abstract (high semantic density) concepts and patterns as they emerge in classroom situations (Macnaught et al., 2013). Exploring semantic profiles is being promoted by NCCE as an aid to teachers wanting to plan their lessons in a way that communicates the key abstract curriculum knowledge that students will need for exams, and to also allow them to put the concepts into practice to build real coding skills and to make valuable connections to personal experience. A Quick Read on semantic profiles is available on the NCCE website.[[2]](#endnote-3)

Research carried out by Curzon et al. (2020) in a computing education context outlines the value of semantic profiles in wave shape as opposed to a flatline, which remains too much in concrete examples or more abstract concepts. This research highlights the value of unpacking, exploring and then repacking ideas during the course of a lesson. A student’s understanding of a concept may deepen a little bit each time it is applied in practice and then reconnected with the abstract.

*Semantic waves*

*Figure 3.1* A semantic profile with semantic waves.

Figure 3.1 An image that shows a diagram of semantic waves, which is a simple diagram or a drawing, which represents the planned knowledge building about a concept within a learning activity.

Examples showing semantic wave profiles usually start with the abstract or with high semantic density. See Figure 3.1 for an example. The advice of starting with more abstract terminology and concepts may seem to be in opposition to the approach of concreteness fading as explained with the CRA method outlined above. However, on examination of the research example carried out in the research above, the process of starting with concepts may only involve giving a short outline of the concept that is being explored and saying that this will be illustrated in a following concrete activity.

**Activity – recognising and exploring the concrete and abstract in your teaching**

To help you balance your lessons, it is useful to identify which parts address more abstract conceptions of computing knowledge and those that are more concrete. To do this, ask yourself the following questions about your lessons:

* What are the core skills and concepts I want to communicate in my lesson? How many of them are more abstract and how many are more concrete? Do I have a good balance?
* How do learners first meet core concepts? How are they re-enforced? Is it through an abstract definition, supported by a more concrete explained example, or via hands-on work that students are engaged in?
* What kind of semantic profile do your lessons follow? Are you able to avoid “flatlining”, where students explore only abstract or concrete language?
* Are you alternating between concrete and abstract poles in a way that helps deepen the understandings of students as they link curriculum concepts to practical experience by helping them link curriculum concepts to practical experience and back again?

**Definitions of computational thinking**

The promotion of CT has been a key factor in the development of the UK’s computing curriculum. However, the claims of early advocates that CT skills could be applied widely in subjects beyond computing are now advanced more cautiously to avoid the danger of over-promising (Tedre and Denning, 2016). We can use the distinction between concrete and abstract to examine the differences between two popular interpretations of CT. The first is an influential take from Jeanette Wing. “The most important and high-level thought process in computational thinking is the abstraction process. Abstraction is used in defining patterns, generalizing from instances, and parameterization” (Wing, 2011). Many learning resources designed to support the computing curriculum present this principle as four key pillars of CT: decomposition, pattern recognition, abstraction and algorithmic thinking (BBC Bitesize, n.d.). The essence here is to deal with concepts and principles as abstract and separate from the context of coding.

Another widely used definition of CT by Brennan and Resnick (2012) was developed in response to a thought experiment “How do we describe what Tim, Shannon, and Renita are learning as they participate as designers of interactive media with Scratch?”. The researchers took a grounded/situated approach to mapping the potential learning dimensions of students designing and coding collaborative, creative computing projects. The resulting map they created includes computational concepts, computational practices and computational perspectives.

* *Computational concepts* include sequences, loops, parallelism, events, conditionals, operators and data, thus representing the mechanics of coding structures.
* *Computational practices* include debugging, iteration, reuse and remixing, abstracting and taking a modular approach.
* *Computational perspectives* such as expressing, questioning and connecting were observed in the behaviour of learners completing their coding designs.

This interpretation of CT, based on observation of learners in action, is more accessible to teachers and learners in comparison with the more abstract interpretations of CT as they can easily recognise their own practice. To give a specific example, rather than decomposition, the applied framework outlines taking an iterative, incremental approach to problem solving and arranging code in modules.

This broader, process-driven definition of CT has been used and adapted by many organisations seeking to support the new computing curriculum. As such, it may be familiar from websites, posters and other supporting material created by groups like Barefoot Computing. Lye’s extensive review of teaching CT (Lye and Koh, 2014) used Resnick and Brennan’s definition as the basis for the review, which points to the widespread use of this more applied approach. The wider definition of CT here assumes an environment where learners are engaged in the collaborative coding of a computing project. The review above and the influential framework used by computing at school (Csizmadia et al., 2015) have included elements of this applied framework as well as other more abstract CT concepts.

The popularity of the CT frameworks describing concrete and collaborative learning help support the development of inclusive teaching approaches. However, the tension between abstract and, thus, potentially transferable knowledge and context-bound concrete knowledge exists not only in definitions of CT and but also in approaches to teaching and assessment. In the next section, I will explore this tension and look at recent responses in the field of UK computing education.

**Models for teaching computing in the classroom**

Classroom practice is strongly influenced by curriculum content and more specifically the format of exam questions. The removal of coursework from GCSE (General Certificate of Secondary Education) exams due to the widespread sharing of worked examples on-line created a real challenge for those setting exams. Exam boards had to rethink how to test the practical programming experience and ability of students in a written exam setting. At GCSE level, most exam questions test the more applied definitions of CT, particularly the process of writing, analysing and revising algorithms in the form of written code examples. Students are required to demonstrate and explain fundamental code building blocks and approaches from first principles. However, exam questions on coding are by nature fragmented and decontextualised compared to practical project coding experience. The written exam questions presenting small coding challenges are necessarily fragmented in order to test a particular part of the curriculum. A focus on written exam approaches in the classroom can reduce time to explore more authentic projects where students follow their own interests and develop skills in design, debugging and other troubleshooting practices.

Spending a lot of time in lessons addressing decontextualised exam questions has potentially negative impacts on student engagement and such disengagement is a serious barrier to inclusion (Kanevsky and Keighley, 2003). In an ideal situation, practical work can deepen and broaden more abstract knowledge. However, given the reality of schools’ focus on exam results, the teacher must balance the development of students’ hands-on coding practices with the ability to recognise and respond to more abstract written questions. Having proficiency in a variety of relevant teaching approaches can be helpful to resolve this tension. The NCCE has produced a series of resources based on research on computing practices to help teachers. They have provided a set of 12 principles for teaching computing[[3]](#endnote-4) and supporting Pedagogy Quick Read documents[[4]](#endnote-5) aimed at teachers to explain and promote key research-informed teaching techniques. The following sections summarise some of these principles.

**PRIMM:** PRIMM stands for predict, run, investigate, modify and make. This model helps learners adopt coding practices and computational concepts through providing a concrete code example that they run after predicting what it does. Learners then make changes to the existing code before finally creating code from scratch. Asking students to identify target computing concepts in code examples allows teachers to guide students towards key CT processes or algorithmic details. Thus, PRIMM is well suited to the classroom as starting with the prediction of code allows a whole class of learners to unpack and repack the same set of concepts in a restricted time scale. This process enables students to more easily tackle the formal problem solving, paper-based questions of the GCSE exams. The use of code examples and a structured set of varied activities aligns well with UDL principle of representing knowledge in a variety of means. For a more detailed summary of the PRIMM approach, see the Quick Read pedagogy article.[[5]](#endnote-6)

**Unplugged activities:** Unplugged activities are carried out away from the computer and aim to illustrate computing concepts. Unplugged activities often use very familiar non-school examples and draw on learners’ understanding of their own concrete experiences. As an example, teachers may use cooking recipes as a way to illustrate the importance of correct sequencing in a code context. Unplugged activities are also often embodied activities. Embodied ways of learning involve moving beyond a computational view of cognition to recognise the importance of physical experience and emotions in the learning process (Settoducato, 2017). Thus, the practice is very much in line with the UDL principle of allowing multiple forms of expression and action for learners and highlighting patterns of knowledge representation. To help learners integrate abstract concepts, teachers should link unplugged activities with concrete coding activities. A semantic wave approach as outlined above can help teachers explore this process.

**Pair programming:** Pair programming groups students in pairs and divides coding into two roles. One student undertakes hands-on coding while the other is free to think about the abstract design of the overall program. A benefit of pair programming is to increase coding confidence as students build their experience of the different roles involved. To help novice coders, teachers should model and break down the processes involved. Pair programming involves social learning elements and can model greater choices for students in the ways they solve problems. The process of building an identity in a community with the help of peers is key to a socio-cultural understanding of how learners pick up coding in a classroom (or other settings). The importance of a coding community is explored in this collection on design and project approaches. A summary of pair programming roles and tips on how teachers can help learners to adopt them in present in a Quick Read document from NCCE.[[6]](#endnote-7)

**Observing and assessing hands-on practices:** One way to address the tension between giving learners the freedom to pursue their own concrete coding goals as is the use of observational techniques to assess user progress. Skilled observation allows teachers to catch learners when they are engaged with their own personally meaningful project and their motivation to solve problems is high. The NCCE has created a Quick Read on observation.[[7]](#endnote-8) A summary of that document follows:

* *Structured observations:* Before lessons, teachers create a framework of the behaviour or use of concepts they want to observe during interactions with students, via recordings or created work.
* *Unstructured observations:* Teachers record or reflect on some of the more unexpected turns that happen during the lesson, often after the event from memory or recordings.
* *Verbal protocols:* Teachers assess the learning and understandings of students by asking them to talk aloud the way they are solving problems and undertaking tasks.

This kind of observation is time-consuming and can benefit from extra classroom support. However, this effort can be justified as an inclusive measure to support a diversity of learner expression and assessment recommended by the UDL guidelines.

**The use of concept maps and learning frameworks:** To facilitate structured observation, a predetermined framework of the kind of behaviours, practices and concepts that align with the project work being undertaken is extremely useful. One way that teachers can develop such frameworks is via a technique called concept maps. The NCCE has created a Quick Read guide for teachers to create concept maps, which focuses on more technical knowledge[[8]](#endnote-9). In this document, concept maps are presented as a way for teachers to model and for students to explore connections between concepts. They can also be used as a map or a matrix to help observation. Providing students with this framework can also help them navigate their learning journey. Having potential learning clearly mapped out and involving students in self-monitoring increases the efficiency of the observation process. The visual nature of the maps also aligns with UDL guidelines on presenting concepts via multiple means. I explore project approaches in Chapter 8 of this book.

**Conclusion**

The purpose of this chapter has been to celebrate the educational value of hands-on, concrete coding as an inclusive way to explore the computing curriculum. As computing educators, we are lucky to have rich and engaging resources at our disposal, which invite tinkering and learning through trial and error experimentation. This chapter has explored the value of UDL principles for teachers looking to inclusive practice and the possibility to assess content knowledge through observing students during hands-on work. We have seen researchers Papert and Turkle celebrate the value of concrete at a very early stage of computing education. More recently, Resnick and Rusk (2020) from the Scratch research community caution against recent tendencies to adopt predominately formal approaches to computing including: too much memorisation of computational terms rather than application, devaluing hands-on coding compared to abstract concepts, not enough time devoted to complete projects and, finally, learners not given enough choice in their coding projects. I share the concern of these researchers that the potential for computing to build creative and design thinking competencies is not being optimised in formal education.[[9]](#endnote-10) Teachers in the UK continue to experience tensions between encouraging experimentation and the pressure to bring students’ attention back to underlying concepts that are assessed through more abstract test material. The NCCE is playing a pivotal role in supporting teachers to recognise and navigate the abstract and concrete via teaching techniques. I explore the principles of design- and project-based approaches, which also have the potential to aid teachers to balance the requirements of the curriculum and the value of letting students spend more getting their hands dirty in the concrete of coding.

**Notes**

**References**

BBC Bitesize, n.d. Introduction to computational thinking – KS3 Computer Science Revision [WWW Document]. BBC Bitesize. URL https://www.bbc.co.uk/bitesize/guides/zp92mp3/revision/1 (accessed 3.26.21).

Black-Hawkins, K., Florian, L., Rouse, M., 2008. Achievement and inclusion in schools and classrooms: Participation and pedagogy, in: Artículo Presentado En La Conferencia de British Educational Research Association. Universidad Heriot Watt. Edinburgh.

Brennan, K., Resnick, M., 2012. New frameworks for studying and assessing the development of computational thinking. Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada.

Burman, J.T., 2021. The genetic epistemology of Jean Piaget, in: Oxford Research Encyclopedia of Psychology. Oxford University Press. https://doi.org/10.1093/acrefore/9780190236557.013.521

CAST, n.d. About universal design for learning [WWW Document]. CAST. URL https://www.cast.org/impact/universal-design-for-learning-udl (accessed 2.13.22).

Cook, S., Rao, K., Cook, B., 2016. Using universal design for learning to personalize an evidence-based practice for students with disabilities. In M. Murphy, S. Redding, & J. Twyman (Eds.), Handbook on personalized learning for states, districts, and schools (pp. 239–247). Philadelphia, PA: Temple University, Center on Innovations in Learning. Retrieved from www.centeril.org

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., Woollard, J., 2015. Computational thinking – a guide for teachers. Computing at School.

Curzon, P., Waite, J., Maton, K., Donohue, J., 2020. Using semantic waves to analyse the effectiveness of unplugged computing activities, in: Proceedings of the 15th Workshop on Primary and Secondary Computing Education, WiPSCE ‘20. Association for Computing Machinery, New York, NY, pp. 1–10. https://doi.org/10.1145/3421590.3421606

Fyfe, E.R., McNeil, N.M., Son, J.Y., Goldstone, R.L., 2014. Concreteness fading in mathematics and science instruction: A systematic review. Educational Psychology Review 26, 9–25. https://doi.org/10.1007/s10648-014-9249-3

Kanevsky, L., Keighley, T., 2003. To produce or not to produce? Understanding boredom and the honor in underachievement. Roeper Review 26, 20–28. https://doi.org/10.1080/02783190309554235

Lye, S.Y., Koh, J.H.L., 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior 41, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Macnaught, L., Maton, K., Martin, J.R., Matruglio, E., 2013. Jointly constructing semantic waves: Implications for teacher training. Linguistics and Education 24, 50–63. https://doi.org/10.1016/j.linged.2012.11.008

Papert, S., Turkle, S., 1990. Epistemological pluralism and the revaluation of the concrete. Signs, 16(1), 128–157. http://www.jstor.org/stable/3174610

Resnick, M., Rusk, N., 2020. Coding at a crossroads. Communications of the ACM 63, 120–127. https://doi.org/10.1145/3375546

Settoducato, E., 2017. Of pedagogy and potentiality: Embodied learning and collaborative storytelling through pop-up exhibits. Journal of New Librarianship 2, 117–121. https://doi.org/10.21173/newlibs/3/5

Tedre, M., Denning, P.J., 2016. The long quest for computational thinking, in: Proceedings of the 16th Koli Calling International Conference on Computing Education Research. Presented at the Koli Calling 2016: 16th Koli Calling International Conference on Computing Education Research, ACM, Koli, Finland, pp. 120–129. https://doi.org/10.1145/2999541.2999542

The Royal Society, 2017. After the reboot: Computing education in UK schools. The Royal Society.

Wing, J., 2011. Research notebook: Computational thinking—What and why. The Link Magazine 6, 20–23.

1. https://udlguidelines.cast.org/ [↑](#endnote-ref-2)
2. https://blog.teachcomputing.org/quick-read-6-semantic-waves/ [↑](#endnote-ref-3)
3. https://blog.teachcomputing.org/how-we-teach-computing/ [↑](#endnote-ref-4)
4. https://blog.teachcomputing.org/tag/quickread/ [↑](#endnote-ref-5)
5. https://blog.teachcomputing.org/using-primm-to-structure-programming-lessons/ [↑](#endnote-ref-6)
6. https://blog.teachcomputing.org/quick-read-pair-programming-supports-learners/ [↑](#endnote-ref-7)
7. https://blog.teachcomputing.org/using-observation-techniques-to-record-student-behaviour-for-research-or-evaluation/ [↑](#endnote-ref-8)
8. https://blog.teachcomputing.org/using-concept-maps-to-capture-communicate-construct-and-assess-knowledge/ [↑](#endnote-ref-9)
9. https://web.media.mit.edu/~mres/papers/CACM-Coding-At-Crossroads.pdf [↑](#endnote-ref-10)