

Rob Reed  
rob @ ode-is-simple.com  
April 9, 2010 (originally written late 2009)  
Ode and education

## **Introduction**

This paper is a survey of current educational methodologies and concepts, including:

- constructionism and discovery learning
- technological fluency and relevancy
- collective intelligence and distributed cognition

I will provide an overview of these, and consider how they might be applied to computer science. I believe there are real benefits that can be derived from the application of these methodologies in the area of introductory computer science in particular.

I propose a general class of tool that might be utilized to integrate these concepts into introductory CS education in a way that complements existing practice. Additionally, I have included an extensive bibliography which incorporates both the formative work and more recent research, serving as a resource for anyone who might choose to explore the ideas presented further.

### *Section 1*

In the first section, I will briefly introduce the educational concepts and methodologies that are the foundation of this survey (listed above). For readers who are unfamiliar with these ideas, this brief summary should be sufficient to follow along with the rest of the discussion.

## *Section 2*

In this section, which accounts for the bulk of the paper, I turn my attention to the question of how each methodology relates to computer science, and introductory computer science education more specifically. I contend that not only are these ideas important from an educational perspective, but that they are a natural complement to the transition, already well underway in industry, away from an emphasis on singular contributions, to the coordinated efforts of large teams working in concert. It should be noted that I'm using the word 'team' here in a very general sense, to mean any collection of people working together toward some common goal. These groups may be formal as is the case with a project group working in close collaboration within an organization. Alternatively, they may be informal as often happens with groups dedicated to open source software development. In these sorts of informal groups, geographically distributed individuals or smaller 'subgroups' may move in and out of an existing persistent and long standing project assuming various primary or marginal roles over the course of its lifespan. The style of collaboration may be synchronous, as is typical of relatively short term projects with fixed deadlines, or asynchronous, such as is the case over the evolution of larger systems and whenever multiple independent projects are combined, often in new and unanticipated ways (e.g. modular software, open source, systems of complementary services, interaction among services).

This progression toward more complex forms of coordinated effort is the transition from development toward software design and engineering, and there is little doubt it is the future direction of advancement in the field of computer science from research to the practical application of resulting theory.

As already stated, this transition is well underway as far as the technology itself is concerned. While the idea of reuse without significant limitation may still be an elusive goal,

the idea of repurposing software is firmly established and vital to modern, large scale software development efforts. Object oriented concepts, service oriented architecture, and increasingly sophisticated APIs are commonly seen as the hallmarks of modern software design. If there is an area where we have fallen behind it may be the mastery of these new principles and the evolution of organizations and project groups so that we might fully embrace and benefit from them. In my opinion this is first and foremost an educational issue.

I believe that the ideas discussed here provide a unique and valuable perspective on the challenge of acquiring both the requisite skills to function as a responsible and efficient practitioner and the ability to communicate and collaborate effectively.

### *Section 3*

The application of these relatively recent innovations to the context of computer science education is novel. The vast majority of existing research centers on traditional areas of education (e.g. fundamental reading and writing, basic math and science skills) and typically involves elementary school age children. It is my understanding that the basis for this bias is the belief that early childhood is a formative period particularly important for long term growth and development. There is no reason, however, to assume we must abandon these potentially effective approaches to learning later in life. Any advantage related to intellectual progress may be especially timely for students who are new to college making a potentially difficult transition that will demand more of them as productive and responsible members of an academic community than perhaps has been the case at any earlier period in their educational careers. We must not undervalue the importance of this transformative stage in the academic and intellectual development of young adults who are still establishing the base of knowledge that will sustain them through their personal and professional lives.

This means that we will need a new set of tools to apply these concepts to the context of CS education. To this end I briefly discuss a project that I had put together previously which I believe is appropriate to use for this purpose. The project to which I'm referring is an extensible personal publishing platform, which is flexible enough to be used for a wide variety of applications. Because it's extensible, it allows for unlimited creative freedom for students to explore their own interests, and can be used to bridge into more advanced topics. This project embodies all of the principles introduced here, from constructionism to collective intelligence and distributed cognition, and is representative of a general class of tools that may be used to bring these educational innovations into the field of computer science. As such, the discussion will emphasize the attributes of the platform that make it suitable for this purpose.

#### *Section 4*

In the interest of building a foundation for future research without complicating the key issues, I have chosen to focus on seminal work. To supplement this, I'm including an extensive bibliography referencing much of the subsequent published work related to the concepts and methodologies presented. As is consistent with the idea of a survey, it is intended to allow readers to quickly familiarize themselves with the existing literature, and to celebrate the breadth, flexibility, and impact of these ideas. Furthermore, it may help readers to identify related topics of interest, and orient their own present or future projects in relation to current research.

**Section 1: In which I introduce and define, in general terms, the educational methodologies being considered.**

### *Constructionism and Discovery Learning*

Constructivism is an educational theory which supposes that people gain knowledge and learn from their experiences by constructing an internal representation of the world around them. The existing representation forms a framework which can be extended to assimilate new information which in turn leads to a better understanding of the world.

Constructionism is a refinement of this idea which adds the qualification that people learn best when they create tangible constructions (e.g. physical objects, environments, narratives) that model the internal representations of their ideas. These artifacts can then be shared with others to elicit feedback and encourage discussion, debate, and collaboration, which leads, in turn, to improved understanding.

It is easy enough to formulate simple catchy versions of the idea of constructionism; for example, thinking of it as "learning-by-making."...

Constructionism--the N word as opposed to the V word--shares constructivism's connotation of learning as "building knowledge structures" irrespective of the circumstances of the learning. It then adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe (Papert 1991, 1).

With constructionism, Papert advocated creating an environment that would afford new learners the same relationship with a subject that is enjoyed by experienced professionals,

whom he referred to as “expert practitioners”. For example, allowing beginning learners to appreciate mathematics in the same way a mathematician does. When engaging in a constructionist activity participants are asked to fulfill the dual roles of learner and communicator. The advantage of the approach, at least in part, is related to the complementary nature of these two roles.

In summary, constructionism is a powerful methodology because participants are encouraged to explore new ideas by creating personally meaningful constructions and then sharing not only the artifacts but also the associated meaning and value with others.

### *Technological Literacy, Fluency and Relevancy*

We can clarify the distinction between technological literacy and technological fluency if we think in terms of the more common example of natural language. In this familiar case, it’s generally understood that a literate person is someone who has acquired a minimal comprehension of the language. He or she might be able to read and understand what has been written or spoken by others but fail to grasp the subtlety and nuance of what has been said. He or she may be capable of writing and speaking but in such a way that is heavily dependent on common phrases or idioms, without fully understanding the complexities and derivation. On the other hand, someone who is fluent is able to communicate creatively and take advantage of the expressive power of the language to describe and explore entirely new ideas, making use of the basic building blocks that are the syntax and grammar to make complex and intricate constructions.

It was Seymour Papert who first proposed the notion of technological fluency to refer to the ability to use and apply technology in a fluent way, likening it to language fluency and emphasizing the importance of utilizing it “effortlessly and smoothly, as one does with language.”

In an article titled "Positive Technological Development: Working with Computers, Children and the Internet" the author explores this idea in greater detail:

Today the term computer literacy often connotes little more than the ability to use specific software applications for well-defined simple tasks, such as word processing and e-mail, and knowing the basic principles of how a computer works (Committee on Information Technology Literacy, 1999). In contrast to this computer literacy trend of teaching instrumental skills, the technological fluency movement emphasizes the importance of teaching how to learn and think with technology (Bers 2007, 6).

A concept that I consider to be very closely related to fluency is relevancy, i.e. the close connectedness or appropriateness of what is being learned to the broader context of a learner's life and experiences, academic and otherwise. The use of this as a formal term that applies to these methodologies is unique, not appearing elsewhere in related literature, but I feel it is valuable in this context. I would argue that it's an issue that underlies much of what is being discussed here. Relevancy is a motivating influence, a contributing factor that allows us to assign meaning to new experiences and integrate new information encountered.

I contend that we can't surpass basic literacy and make progress toward fluency unless we consider what we're learning to be relevant to goals and ambitions that are personally meaningful.

*Collective Intelligence & Distributed Cognition*

The notion of collective intelligence is a shared capacity to acquire and apply knowledge that comes from the collaboration and contributions of many individuals.

The ideal of a collective intelligence is a community that knows everything and individuals who know how to tap the community to acquire knowledge on a just-in-time basis (Jenkins 44).

Collective intelligence can be considered a subcategory of the broader concept of distributed cognition, which is the idea that knowledge and learning exist not just in the minds of individuals but that both are distributed among the members of groups (often referred to as a ‘community of practice’ in the literature) and the tools that are utilized in the process of learning. The implication is that the tools we choose to use are a significant part of the context. We should strive to include tools that are comparatively more capable of helping us to retain knowledge, share what we have learned, and express complex ideas at different levels of abstraction.

Consider this assertion from Scardamalia and Bereiter, (authors of a paper titled “Computer Support for Knowledge-Building Communities”):

Schools need to be restructured as communities in which construction of knowledge is supported as a collective goal, and the role of educational technology should be to replace classroom discourse patterns with those having more immediate and natural extensions to knowledge-building communities outside school walls (265).

The value of distributed cognition as it applies to the topic of this paper is twofold.

First, in a world becoming inextricably entangled with computer technologies, we must be more than solely consumers of technology even just to function efficiently within our local

communities. Furthermore, we must be capable of negotiating the sort of rich environments that surround us in order to fully realize the advantage such technology affords, and be capable of making broader societal contributions.

Secondly, in the field of CS, the technological tools are literally inseparable from the work that we do. From simple programming tasks to complex engineering problems, the interplay between practitioners and their toolset is critically important. When we consider issues related to software engineering and design and what might be required to fully embrace engineering principles as tenets of software development, we have to ask ourselves questions about not only methodologies, design principles, and the roles of individuals and groups, but also the capabilities and sophistication of available tools and our understanding of how to best use them to our advantage.

## **Section 2: In which I discuss how these educational concepts and methodologies lend themselves to the field of computer science and CS education**

It is not these concepts and methodologies in and of themselves, important as they may be, but more to the point the ways that they might be applied to inform introductory CS education that is the true focus of this survey.

As mentioned in the introduction, the seminal work related to these ideas comes from the field of education and has typically involved young children. Adapting these methodologies to introductory CS material for high school and college aged students is a unique application of the established principles. Why am I proposing that we change the current practice?

An important concern with targeting the introduction of educational innovation exclusively to younger learners is that it means essentially skipping an entire generation of young adults who represent the next best chance to make substantial academic contributions and to impact

the world. I fail to see the inherent benefit in waiting a generation before we introduce innovative strategies which are intended to enable students to more effectively utilize technology in a variety of fields so that they might help us answer pressing and long standing problems. More importantly, I think it's ill-considered to expect the world to wait.

I propose that we should strive to introduce computer science and related topics to more young adults as they are deciding what to do with their professional lives, so that we might help them see its relevance to their goals and ambitions, in hopes of bringing more people into the field while equipping students with new approaches to problem-solving that can then be applied to other areas of interest as well. We need strategies to help the current generation of adults see the relevance of technology to their lives, interests, and all of their pursuits, academic and otherwise. What's more, if parents and other adults are not technologically fluent, they may struggle to assist the next generation in becoming proficient with these tools. By working toward establishing the applicability of CS to a broader range of problems we essentially widen the utility of the discipline. This serves to encourage individuals to pursue computer science and fosters greater adoption of CS as a mainstream course of study.

It seems obvious to say that computer science is an increasingly important field of study given that many 21st century problems stem directly or indirectly from the introduction and integration of new technologies as it inevitably plays a role in virtually every facet of human culture and community. Not surprisingly, this has led to any number of advancements, but also corresponding problems: We are able to extend human life expectancy and sustain a larger world population through better healthcare (i.e. increased effectiveness and availability of medical technologies) and improved sanitation among many other factors. But this has contributed to devastating ecological issues. In the same way, increased connectedness among the people of the world thanks to improved travel and global communications has led to a better understanding and appreciation of our current diversity and accomplishments but also, increased unprecedented competition and financial

interdependence on a global scale in a way that we seem to struggle to comprehend much less control and coordinate.

We need to equip ourselves to deal with the associated challenges. There are any number of hard problems to be solved. The best chance of dealing with any of them is to involve as many people as we can. As I have already mentioned, computer technologies are playing an increasingly important role in all of our lives, and the cost of technology is decreasing dramatically. At the same time, accessibility and pervasiveness continue to increase along with the power and sophistication of the technology itself. However, CS is still seen as something other than an integral part of the mainstream curriculum.

Ironically, I believe that the widespread adoption of computer technologies and the growth in sophistication and popularity of the web are key factors such that interest in the field is less than it might be. The domain of the expert end user is expanding as is the encroachment of the IT industry. The confluence of the increasing sophistication of software tools, the number and variety of such tools which can be used to accomplish seemingly any task, and the growing complexity and specialization of computer science have led to a 'perfect storm' of demotivational influence(s).

Many students who might have been eager to enter the field a decade ago may very well consider computer science today as either unapproachable or irrelevant to their ambitions. To describe the current situation in terms of the familiar metaphor of a carrot on a stick dangling (forever out of reach) just in front of a hesitant donkey, it's as if the same carrot is now attached to the end of much longer stick and the donkey is already sated. All of this means it's very important that we acknowledge these changes and carefully consider how we introduce students to the field, beginning with programming and basic theory. It seems self-evident that the situation has changed as the field has grown and matured and the pervasiveness of technology has expanded to a level that may not have been foreseeable when the field was in its infancy.

Fortunately, and perhaps not entirely coincidentally, many of the innovations in educational theory involve the adoption and exploration of computer technologies, as is the case with all the methodologies discussed here. Educational theory has advanced past the earliest efforts to incorporate technology from the computer assisted learning that typified research in the 80s and early 90s, which focused on replacing instructor led education or supplementing it with stand-alone applications which simulated didactic instruction. More recently the emphasis has shifted toward utilizing technology as a resource to allow for the construction of novel educational solutions. Sometimes this is quite literally the case; that is, that technology is utilized as a basic building block, such as the MIT Media Lab's Crickets project<sup>1 2</sup>, which led to the commercial development of the LEGO Mindstorm robotics kits, both of which are programmable building blocks used to create academic or artistic projects and are flexible enough to be suitable for any number of different areas of interest.

Of course, this idea need not be so literal. I contend that we can consider CS itself to be an immensely grander expression of what is at first an apparently simple idea, from which we can construct a universe of solutions to an infinite number of problems. Let's look at each of these methodologies within the context of computer science, starting with constructionism. Computer technologies, more specifically software, programming languages, and design principles, are a virtually limitless constructionist toolkit as proposed by Papert.

---

<sup>1</sup> Pico Cricket. [www.picocricket.com](http://picocricket.com) <<http://picocricket.com>>

<sup>2</sup> Life Long Kindergarten. <<http://llk.media.mit.edu/projects.php?id=1942>>

## *Constructionism and Discovery Learning*

Piaget described Constructivism as being the process whereby students constructed their own unique systems of knowing. ...Seymour Papert, a student of Piaget, expanded on this to describe Constructionism in terms of helping the student produce constructions that others can see and critique.<sup>3</sup>

The above description of constructionism suggests that the approach is particularly well suited to software development. One of the tenets of constructionism is that there is value in the use of innovative technologies to explore personally meaningful concepts. Programmers build things that represent their ideas and which can be shared, critiqued, and refined by others. Programming doesn't teach us what to think, but how to think in new ways. In other words, it's not so much a collection of information as it is a way to explore problems which are for one reason or another important to the developer. That perspective promises not only to help us to be more adaptable learners and more expressive communicators but more introspective participants. This is the fundamental idea that inspired Papert's classic constructionism and still drives related work almost two decades later.

Let's again return to Papert's original analogy to natural language. In the same way that learning a second language helps us to appreciate the variation and diversity in the way that people communicate, it also frequently encourages us to make connections with new people, to travel, to increase our participation in the world, and expand our horizons.

To place this firmly in the context of computer science and its relevance to education, I'd like to consider this short passage from the (2004) book "What is Thought?" by Eric Baum. In the book, Baum, who holds a PhD in physics and has pursued a career as a computer scientist and AI researcher working on machine learning algorithms, proposes that the mind

---

<sup>3</sup> Straker, David. Constructionism and Constructivism. [changingminds.org. <http://changingminds.org/explanations/research/philosophies/constructionism.htm>](http://changingminds.org/explanations/research/philosophies/constructionism.htm)

is equivalent to a computer program that has been designed by evolution to exploit an underlying compact structure of the world. Of course, much of Baum's argument is well outside the scope of this paper. However, I'd like to discuss a specific example that suggests the advantage of learning to apply the ideas typically associated with computer science to a range of problems that have real world applications.

Baum describes the general class of problems which are collectively defined as block stacking problems and gives the following example:

[A person] is presented with 4 stacks of colored blocks. The last 3 stacks taken together contain the same number of blocks of each color as the first stack. The solver controls a hand that can lift blocks off of the last 3 stacks and place them on the last 3 stacks but cannot disturb the first stack. The hand can hold at most 1 block at a time. The goal is to consolidate the blocks from the last three stacks onto one stack in the same order the blocks are presented in the first (48).

Next, the author discusses the manner in which people intuitively solve this type of problem:

A person looking at these problems can in short order produce an algorithm capable of solving arbitrary block-stacking problems rapidly, by exploiting the structure of the problem. ... You, a human being, bring to bear understanding of topology, geometry, and goals. You had, I expect, modules in your mind that know about these concepts before I presented you with these block-stacking problems... A trained computer scientist has had another module in her mind: a module for recursion. This allows her to construct a yet more efficient solution to the problem. That is, this module exploits the structure of

these block-stacking problems to output an algorithm that solves them very rapidly (48).

As suggested by this quote, Baum argues, at least in part, that the mind consists of components or 'modules' that we use to process and understand the problems we're presented with, and generate solutions. The author further stipulates that the only way for this to work effectively is if these modules function by identifying and exploiting underlying patterns inherent in the problems themselves, where the problem domain is the world in which we all live. Again, as suggested in the quote, these modules can be learned and improved through practice and experience and when sufficiently mastered can be called into action to solve new problems or revisit existing problems to improve our solutions.

While Baum's arguments are theoretical, even controversial, consider this quote from Matt MacLaurin, a computer programmer and Principal Program Manager with the Creative Systems Group at Microsoft Research, who promotes a similar idea, taking a much more commonsensical approach.

Programming is good for you. Even if you don't program for a living, programming stretches your mind in ways that are very beneficial.

Think about it. Even if you're just working on something boring like a calculator program, your brain has to simulate what the computer is going to do so that you can construct your plan correctly. When something goes wrong, you again simulate many different possibilities mentally to form a theory about the bug - and to correct it.

Simulation itself is a very interesting special case of programming. Building a sim is like a multidimensional form of writing a story.

You're not writing one story; you're construction [sic] a story space in which an [sic] huge number of different stories can happen.

- story: keep track of who did what when

- sim: keep track of who can do what and what new possibilities will emerge if they do it

Learning to simulate complex systems in your head is great practice for a lot of jobs - like running non-profit organizations, effecting social change, or running a car dealership.

And simulation, as we know, is just a polite word for gaming.

Programming - or, more precisely, the mental facility developed by programming - is a life skill that should be accessible to everyone.<sup>4</sup>

MacLaurin is the project lead responsible for a product titled Kodu<sup>5</sup>, which I would describe as a programming environment, featuring its own distinct visual programming language, wrapped in a game, and geared toward young children. He speaks to the value of programming as a uniquely important and rewarding form of expression, and one that may be singularly beneficial for addressing any number of problems beyond those select areas to which it is typically applied. More to the point, as it relates to this discussion, MacLaurin seems to argue in this passage that programming is especially valuable for the reason that it bridges the logical conciseness and precision that is typically associated with the operation of computers, with the rich expressiveness of storytelling. What he's promoting is the value of programming as a constructionist toolkit.

---

<sup>4</sup> MacLaurin, Matt. *Programming: Like Karate for Your Brain*. [mattmac.spaces.live.com/blog/](http://mattmac.spaces.live.com/blog/). <<http://mattmac.spaces.live.com/blog/cns!63689DB3042B28B9!199.entry>>

<sup>5</sup> Creative Systems Group, Microsoft Research. *Kodu*. [research.microsoft.com](http://research.microsoft.com/en-us/projects/kodu/). <<http://research.microsoft.com/en-us/projects/kodu/>>

Traditional introductory programming exercises often do not involve shared discovery and collaboration. Owning and sharing of expertise is fundamental to the sort of discovery learning which is at the heart of constructionism. Too frequently students are asked to perform short arbitrary exercises based on an incomplete understanding of concepts and too small a space of time to confer ownership. Furthermore, because all students are asked to complete the same assignment, the idea of sharing is irrelevant if not wholly inappropriate and is commonly discouraged in order to avoid outright copying. Discovery learning is often contrasted with instructor-oriented learning. The essential idea of the former is that people learn best when discovering for themselves. The fundamental issue then is how do we provide enough of a framework to structure learning so that we can make predictable and steady progress? A constructionist toolkit provides that framework.

While promoting the constructionist methodology I do want to briefly acknowledge what seems to me to be a key potential concern related to the approach. In the case of verbal communication, the challenge is in finding the right words to describe an important concept. With constructionism the challenge instead involves finding a way to utilize the available technology. The risk then is in putting the learner in the position where they have committed themselves to a personally meaningful project, which they cannot express to their satisfaction because of a lack of understanding of the tools they are expected to use. There must be a sufficient amount of time, and the experience open-ended enough, that participants feel free to explore the available resources in order to reach an understanding of how to work with the tools to express themselves adequately.

Pairing something new about which learners may be unsure, e.g. a new technology, with something familiar or personally significant, a strongly held value for example, promises to be both reassuring and motivational, and could entice participants in a constructionist activity to explore ideas which may be intellectually important but which they have been reluctant to pursue otherwise. However, provided with an insufficient amount of time to adequately discover how to utilize the tools, the experience may have just the opposite

effect. That is to say that precisely because a learner has contributed something personally meaningful, an inability to complete the intended project or convey the narrative and underlying idea may reinforce whatever bias or misgivings the he or she brought to the experience, with the result that the participant may be less motivated and even averse to attempting to use the unfamiliar element in the future.

This may help to explain why we are frequently reluctant to fully incorporate new technologies into established practice, despite often having evidence of substantial benefit.

In summary, while I believe that constructionist methodology is potentially rewarding and well suited to CS education, the concept alone is not enough to guarantee success. It is commonly understood that not all constructionist toolkits are created equal, i.e. they do not all offer the same opportunities for learning and exploration. I contend that when determining whether a toolkit is suitable for use as a constructionist resource, time must be considered an integral part of the environment and carefully factored into the decision making process. I propose the qualification that the most promising environment may be no better than the poorest if participants are not afforded a sufficient amount of time to discover and utilize the richness of the toolkit.

I discuss this idea more in Section 3, which is dedicated to characterizing the sort of tool that will allow us to carry these ideas into the realm of computer science education.

### *Technological Literacy, Fluency and Relevancy*

While learning the alphabet is required to write a poem, it is not enough. In the same spirit, knowing how to use some software packages is not enough to become technologically fluent. As stated by the Committee on Information Technology Literacy in 1999, "the

'skills' approach lacks 'staying power'". ...I am suggesting that technological literacy is a fundamental stepping stone toward technological fluency, but not a goal in and by itself. I am also proposing that, regardless of the age of the children and the challenge to create developmentally appropriate curriculum and software, the goal should be to promote technological fluency and not merely technological literacy (Bers 2008a, 112).

If we are thinking about computer science education, it may be that we think in terms of programming languages, operating systems, and development environments and frameworks, rather than strictly end-user oriented software applications. With that small change, the gist of the above quote is applicable to CS education. We can still argue that the skills based approach is not enough.

I contend that the ability to program represents a particularly important goal in the pursuit of true technological fluency; one that affords us greater access to the powerful, flexible, and expressive class of tools that are computer technologies. Although coding has not typically been considered a requirement of technological fluency, maybe it should be. As we have already seen, Papert describes technological fluency in terms of natural language. It's programming that allows us to creatively express ourselves by forming new constructs the way we do with words in a natural language. When we program, we make use of a language with a specific syntax to build larger more complex grammatical structures from basic components. In no other way do we interact with computer technologies at this level, as is consistent with Papert's prototypical ideal. Limiting ourselves to using only existing applications and services might be considered analogous to writing an original story or describing a new idea exclusively in terms of shared anecdotes or existing literature. When talking about technological fluency if we limit ourselves solely to software written by someone else for us, we cannot ever exceed the limitations of what's already been done. For example, we might utilize a math program to work out specific examples of problems that

are already well understood, assuming of course that support for the relevant class of problem was included by the developer. But we can't tackle entirely new classes of problems.

Let's further explore the analogy between technological and natural language fluency. Learning a foreign language can be very satisfying and rewarding. On the other hand, when first attempting to come to grips with a new language, it can be disheartening to be put in a position such that one must struggle to express a value or an idea that is personally meaningful but finding oneself unable to do so for want of finding the right words. In the same way, it can be frustrating for new students to be asked to communicate using a new tool or technology only to feel that they are unable to express themselves because they don't understand how to utilize the tool effectively. One final point before we turn our attention away from this analogy. It seems reasonable to assume that technological fluency, much like fluency in a language, is only gained as a result of frequent, repeated exposure over an extended period of time or a lengthy immersive experience, and may require continued practice to achieve and maintain. This emphasizes the necessity of making a long term commitment if methodologies such as those being discussed here are going to make a meaningful contribution as something more than a supplement to traditional classroom instruction.

In the paper "Civic identities, online technologies: from designing civic curriculum to supporting civic experiences" the author discusses Zora, an interactive online community geared toward adolescents which she describes as a tool that functions as both "process and product". Ignoring the specific tool, which is not especially relevant to this discussion, I want to focus on what it means to use a tool as both process and product.

The interactive nature of technology-based tools can allow certain of these platforms to function as both the subject and the platform via which ideas related to the project are explored. This is a powerful aspect of any sufficiently flexible and open-ended environment

and supports the prospect of learning about learning itself, which is key to technological fluency. Again, this benefit is not limited to any single tool, but is inherent in the design of many technologically sophisticated environments. It's this sort of tool that we may want to work toward developing in order to apply these concepts to computer science education. This is the subject of the next section of the paper.

We might characterize the past quarter century starting with the mainstream adoption of the internet and the web and the integration of computer technologies into virtually every facet of modern life, as a great shared knowledge infrastructure leading us collectively toward a greater technological fluency. With this newfound knowledge has come the realization that in order for technology to lead to the theorized benefits in achievement related to learning and education and the resulting improvements to society, culture, and community, the development of technology for its own sake is simply not enough. It is the application of technology to other practical purposes that must be the ultimate goal. Why is this important to this discussion?

First, it brings computer science to bear on a much broader set of problems. In so doing, it provides us with valuable insights into how we might meet the challenges of our new and ever evolving world in all its varied complexity.

Secondly, it means that individually we can each tailor the use of technology to our own unique requirements, interests, tendencies, and capabilities. It allows for each of us to make use of these technologies in a way that is not only personally meaningful, but personally expressive. That is, technological fluency is the characterization of technology as a flexible tool that can be wielded in any number of ways and put to use toward the accomplishment of any number of divergent goals by the people and communities who both explore new technologies and adopt inherited technology to better meet their needs.

Effective use of technology is not so much a matter of memorizing an endless series of seemingly unrelated tasks, applications, and frameworks, but understanding and making use of it in service of work that is relevant to our individual and collective goals. The shift in focus from learning facts to learning how to learn is a pivotal one with regard to the distinction between technological literacy and technological fluency.

Readiness to learn and self-confidence in our own abilities are not new ideas. However, although there is a general consensus about the importance of "readiness" in traditional areas of study, such as literacy and mathematics, it is not clear what it means to achieve early readiness in areas that are more closely related to technological fluency. I am proposing that readiness is intimately associated with the ability to engage in the design and implementation of a personally meaningful project.

[Technological fluency] involves mastering not only technological skills and concepts but also the ability to learn new ways of using computers in a creative and personally meaningful way. During the process of using the technology in a creative way, people are also likely to develop new ways of thinking; therefore the computer's role goes far beyond being an instrumental machine (Bers 2008b, 142).

Closely related to the concept of technological fluency is relevancy. Fluency is defined as the ability to interface with technologies efficiently. Relevancy on the other hand is the motivating influence providing the opportunity to assign meaning to new ideas and experiences, allowing us to achieve the goal of fluency. If fluency is the objective then relevancy is the function. In other words an activity that is not seen as relevant to the context of a learner's interests is unlikely to be pursued independently, and it is this sort of independent exploration that is at the heart of discovery learning. This begs the question, How then do we improve or increase the perceived relevance of computer science?

If we do in fact agree on the importance of fluency, and familiarity, as both instrumental to the learning process and crucial to realizing the potential value of what is learned, then we need to introduce students to foundational topics in computer science and do so in a way that builds confidence and piques their interest by affording them the opportunity to do work they find rewarding.

### *Collective Intelligence and Distributed Cognition*

Consider these two different descriptions of distributed cognition:

Distributed Cognition — the ability to interact meaningfully with tools that expand our mental capacities (Jenkins 37).

Distributed cognition is not simply about technologies; it is also about tapping social institutions and practices or remote experts whose knowledge may be useful in solving a particular problem. According to this understanding, expertise comes in many shapes and sizes (both human and non-human). Experts can be expert practitioners, who can be consulted through such technologies as video conferencing, instant messaging, or email; some knowledge can emerge from technologies such as calculators, spread sheets, and expert systems; new insights can originate from the teacher or students or both. The key is having expertise somewhere within the distributed learning environment and making sure students understand how to access and deploy it (Jenkins 37).

As I have described it, distributed cognition is the idea that knowledge and learning exist not exclusively in the minds of individuals but also in the tools used in the learning process. In this context, the old adage of using the right tool for the job takes on additional significance. Following from constructionist methodology, the tools used in the study of computer science should have meaning and value, but we must also incorporate tools that allow us to establish connections among participants within the local community, and ideally beyond. These tools need to be applicable to the broader context beyond the classroom so that they help us to further our knowledge by acting as a bidirectional gateway leading from the classroom community out to a world of ideas and back again. Ideally the environment itself serves as a conduit facilitating knowledge sharing and collaboration while encouraging growth of the community, leading new members from outside the community in, each of whom brings with them their own ideas based on their unique experiences, fresh perspective, and perhaps even new complementary tools and technologies.

Take the simple example of a class discussion on a wiki. Participants are contributing to a community effort in such a way that each student's ideas and learning are impacted by and hopefully benefit from the work of others. There is ample opportunity for discussion and collaboration. What's more, the discussion can be made available to the much broader community of the web, where it can continue to be developed and refined. In this example, the wiki is a part of the distributed framework that facilitates the sort of knowledge sharing and collaboration that are key to the idea of collective intelligence.

A resourceful student is no longer one who personally possesses a wide palette of resources and information from which to choose, but rather, one who is able to successfully navigate an already abundant and continually changing world of information (Jenkins 49).

This quotation underscores a fundamental change in the nature of education itself. For any sufficiently complex system, simply presenting the body of knowledge becomes an

overwhelming burden for educators and students alike. Communicating or delivering a wealth of information might require more time than is available or leave too little time for the sort of experimentation and self-directed discovery necessary to make sense of it. The existence of a superabundant amount of information raises the question, how do we allow students to develop the working knowledge they require to be successful? Collection and distribution of information is no longer the fundamental challenge. Instead, students must be prepared to tolerate the deluge of information, much of which is frequently outdated or otherwise inaccurate, and often conflicting; they must learn to assimilate new information and thrive within what is an increasingly rich, but also demanding new intellectual reality.

As has already been described, constructivism is an educational theory which supposes that people gain knowledge and learn from their experiences by constructing an internal representation of the world around them. The existing representation forms a framework which can be extended to accommodate new information that leads to a better understanding of the world. If we accept the notion that we learn by constructing mental models and then extending that framework as a way of expanding what we know and comprehending new ideas, then we need to develop tools and techniques that help us to accomplish this. It's not enough to simply be surrounded by knowledge. Effective use of technology is not so much a matter of memorizing an endless series of seemingly unrelated tasks, applications, and frameworks, but understanding and making use of technology in service of work that is far more relevant to our individual and collective goals. The shift in focus from learning facts to learning how to learn is a pivotal one with regards to the distinction between technological literacy and technological fluency.

Learning increasingly means learning to negotiate information. This is an issue for education more generally but it's something that's a fundamental concern for computer science, and for the types of work done in this field, more specifically. Tools that facilitate growth and development of skills by inviting participants to contribute to the collective knowledge of the group and reinvest in the environment itself, encourage discussion and collaboration

naturally. Furthermore, these environments require compromise, and an appreciation of the responsibilities inherent in any distributed, collective effort.

When thinking about how these ideas of collective intelligence and distributed cognition can be put to use most effectively in computer science education, it is clear that we need to seek out tools to allow for truly collaborative work in the classroom. Rather than the often typical solitary programming assignments for which collaboration is almost always expressly forbidden out of a concern for cheating, we need to encourage students to understand not only their work but the work of others in order to contribute to and benefit from a genuinely collective intelligence. Students should be able to explore their ideas and assignments in the open. Ideally, we need a tool that ensures everyone's work is related enough to make cross-pollination of ideas feasible while at the same time alleviating concerns about cheating by removing the incentives to cheat while encouraging and rewarding original, and genuinely creative thought, and work that enriches a shared platform for all participants. Ideally class work should be a true community effort. This allows students to experience collaborative development much earlier in their careers as computer scientists than what is typical currently, thereby allowing them to hone the skills they'll need to contribute to larger scale projects later in their academic and professional lives.

This changeover from learning as an individual to learning as a social process, which is fundamental to the idea of collective intelligence, is consistent with the rapid growth in the use of the internet, and more generally, the increasingly connected world that sometimes invites distributed collaboration, and often demands it. The web is perhaps an ideal platform for this sort of work. Content creators can draw on the rich collection of information available online in all of its varied forms, fusing the existing collected work of the community with their own original material to produce distinctly new compositions. As for the field of computer science, a lot of difficult problems have made their way to the web. Whereas related work was dominated by content issues almost exclusively through the 1990s (i.e. the problem of migrating content to and accessing it from the web including

issues related to presentation, design, and the structure of content), innovation today is all about standards-based open development, interoperable services for everything from storage to distributed processing, data sharing, harnessing the collective intelligence that is the global user community, and analyzing and repurposing publicly available information.

When evaluating the potential of the web as a platform for collaborative learning, it's helpful perhaps to remind ourselves of its design goals. Originally, the web was intended to facilitate collaborative work among possibly distant groups in innovative ways. Keep in mind that the idea of distance is not necessarily limited to geographical distance; there is temporal distance, and ideological distance as well. This goal, dating back nearly two decades now, describes almost exactly the challenge and the benefit of collaboration inherent in any environment capable of supporting collective intelligence and distributed cognition. In other words, these ideas are integral to the platform that is the web, by design. This begs the question, might we be able to capitalize on the promise of the web and realize its potential to help us collaborate more effectively in the field? In other words, can we take advantage of the web to build a stronger academic community, enabling us to become better thinkers, colleagues, mentors, educators, and learners?

At the beginning of this section I said "Computer technologies, more specifically software, programming languages, and design principles, are a virtually limitless constructionist toolkit as proposed by Papert," and that may be true. But 'software development' is too broad a concept to describe a workable toolkit. What's required is a framework that allows us to apply these ideas in a very practical way to the challenge of teaching introductory computer science. I've suggested that the web may be a good foundation.

First we don't have to wonder if the web is engaging. It is has, for more than two decades now enjoyed phenomenal growth and attracted considerable and widespread interest, and particularly, among those people who may be inherently drawn to the field.

Secondly, not only is the web popular but it is legitimately relevant to computer science and represents a tremendously valuable resource for the community. Today, the web is becoming a distributed platform for building applications with the elegance of a modern framework and the capabilities of a service-oriented architecture; one that is already here, widely used, and global in scale. Furthermore, many familiar CS problems are making their way to the web, which suggests that it may be a useful platform for vetting our ideas related to these problems. For example, the emergence of open standards has led to the accessibility of large data sets, and the opportunity to confront the real world problems associated with accessing that data over a highly distributed application.

In summary, there's an important duality here that is fundamental to the applicability of all of these methodologies as they relate to the field of computer science education. They help us to understand both how to involve new students and how to continually engage them as active participants in their own education. Computer technologies are not only a tool that must be understood but also a toolkit for exploring any number of ideas. As was said when discussing relevancy, they are both the product and the process.

In the next section I will describe one tool in particular, representative of a general class, that encompasses all the methodologies discussed in this section and which may be appropriate for adapting these concepts to the realm of computer science education.

**Section 3: In which I introduce an original project that may be used to adapt these methodologies to computer science education, and discuss key characteristics of a general class of tools that might be suitable for this purpose.**

I've developed a web-based platform, called Ode (pronounced oh-dee)<sup>6</sup>, which is intended to take advantage of the appeal and utility of the web, and designed both to serve as a

---

<sup>6</sup> Reed, Robert. *Ode is Simple*. ode-is-simple.com. <<http://ode-is-simple.com/>>

suitable introduction to programming and also to bridge the gap between what students must learn early on in their studies and what they might imagine they want to do with that knowledge.

Ode is an extensible personal publishing platform, which is flexible enough to be used for any number of unique applications, but without modification is similar to a weblog or wiki. That is, an app designed to dynamically generate a website from individual posts, which are collectively the content of the site. The presentation of this content is determined by one or more themes, which dictate the overall look and layout. This sort of application speaks to the original design goals of the web.

In this section I'll briefly revisit each of the educational concepts discussed in terms of the characteristics of Ode that make it well suited to serve as a platform for adapting these methodologies to CS.

I want to emphasize that it is the characteristics of the application and not this specific tool itself that I want to highlight. Any tool sharing these attributes may also make for a suitable toolkit. Furthermore, it should be noted that this is by no means a definitive description of every appropriate app. As is nearly always the case, there may be any number of suitable alternatives.

### *Constructionism*

Constructionism is a learning theory that supposes people learn best when they create literal constructions to model the internal representations of their ideas to share with others. I've made the point that all of programming is essentially constructionism in the sense that programmers build things that represent their ideas and which can be shared, critiqued and refined by others. But true constructionism requires that learners be allowed to explore

personally meaningful projects over a sufficiently long period of time to confer a sense of ownership, i.e. a commitment to the project and intimate familiarity with the toolset.

When we think about constructionism the goal is to provide rich, expressive toolkits so that students can explore a variety of ideas and build interesting, engaging projects that are capable of conveying the imagined concepts convincingly.

This is the intended goal of Ode; to serve as a platform; accessible, general purpose and open-ended enough to allow students to make the leap from what is learned to putting that new knowledge to use, channeling it into rewarding, personally meaningful directions. This is well in line with the constructionist methodology.

### *Technological Fluency*

The concept of technological fluency dictates that we introduce students to foundational topics in computer science and do so in a way that builds confidence and piques their interest by affording them the opportunity to do work they find rewarding.

Ode takes advantage of functionality provided by the operating system and other applications. For example, under Ode, in addition to being a content creation tool, the participant's preferred text editor, in combination with CSS, HTML, and Perl, also functions as a powerful design and development environment. File management, the editing environment, browsing the web, media creation tools, etc. - all of the concepts with which a computer user might be expected to already be familiar come into play but take on new expressive power. This increases the relevancy of the tool as well as the learner's comfort and confidence encouraging participation by lowering the barrier to initial exploration and experimentation. Additionally, Ode is useful for many of the activities that already occupy our time such as creating and sharing presentations, note taking, and discussion. Any

platform that allows participants to use already familiar technologies in new ways encourages and supports the sort of technological fluency discussed.

### *Collective Intelligence and Distributed Cognition*

Ode serves as a shared project with each student contributing their own part within a classroom setting. This situation improves the overall utility of the project and allows students, individually and collectively, to become more involved with the platform. The key idea here is that contributions can be measured in terms of their uniqueness and genuine value.

The project is intended to expand or elevate introductory programming, which is frequently a solitary exercise, to the social context of the web and collaborative classroom work in a fashion that is reminiscent of open source development, an approach intended to improve learning and lead to greater achievement in the short term, the formidable first foray into the rewarding but intellectually challenging world of computer science.

Ode takes advantage of the web as a tool that fits within the model of collective intelligence, promoting new methods for more effective communication and collaboration. As the field of computer science has become more complex, there is a tendency toward specialization which continually leads toward still greater complexity and divergent interests. These niche communities develop their own languages, methodologies, and patterns and styles of communication which can become very insular. It may become difficult for colleagues, even within a single department, to carry on meaningful conversations, much less collaborate between project groups. The web can be utilized as a platform to bridge between these communities.

The web is identified by both the collection of technologies that describe how it functions and the contributions of its users. Clearly, it is a good example of a technology that marries the knowledge of an individual with the collective intelligence of the group and is inseparable from the technology itself. This is the very definition of distributed cognition.

Any project or platform that featuring these or similar attributes may be an appropriate tool for adapting the methodologies and concepts which have been the focus of this survey to the context of computer science education.