Informatics Education Using Nothing But a Browser

Chris Piech, *piech@cs.stanford.edu*
Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Eric Roberts, *eroberts@cs.stanford.edu*
Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Abstract

Despite the extraordinary growth in the volume and variety of material available on the World Wide Web, the Internet revolution has had surprisingly little effect on the delivery of informatics education. In this paper, we present an entirely web-based programming environment called StanfordKarel that introduces students to the fundamental techniques of algorithmic problem solving. We then describe a broader project at Stanford to make educational resources freely available to anyone with access to a web browser—even if that browser is on a mobile phone. In the fullness of time, we expect these resources to cover the entire core of the informatics curriculum. By developing this collection of tools, we hope to give people throughout the world access to the knowledge and skills they need to succeed in the 21st century.

Keywords

Computer and information science education; JavaScript; Java; online IDEs; HTML

INTRODUCTION

Over the last decade, the dizzying expansion of the online universe and the growing sophistication of web browsers have turned the Internet into the greatest repository of information in history. At the same time, the increased availability of mobile devices has brought the resources of the Internet to many more people throughout the world. In 2005, *The Economist* ran a cover story describing mobile phones as the next “killer app” for the developing world, citing a growing body of evidence “that the mobile phone is the technology with the greatest impact on development” [Economist2005]. Since that time, the computational power of mobile phones has increased substantially, making them even more important to economic growth in developing and newly developed nations.

For reasons that we discuss later in the paper, informatics education has been slow to take advantage of the enormous potential that the Internet offers. At Stanford, we have recently initiated a project that seeks to address this problem by creating a entirely browser-based collection of educational materials, pedagogical tools, and interactive applications to support the teaching of computer science. The first of these applications is a programming environment for Karel the Robot [Pattis81], which offers, in the words of its designer Richard Pattis, “a gentle introduction to the art of programming.”

Over the next year, we intend to expand those resources so that they cover the entire core of our undergraduate program. In contrast to existing web-based packages for informatics education, our tools require nothing but a browser. Educators and students will not need to install any additional software, download any special-purpose packages, or use a development environment not provided by our website. Our tools, moreover, run on the client side, which reduces the cost of distributing computation that so often plagues client-server designs. Our project also encompasses the development of authoring tools that simplify the process of creating web-accessible educational content for free distribution.

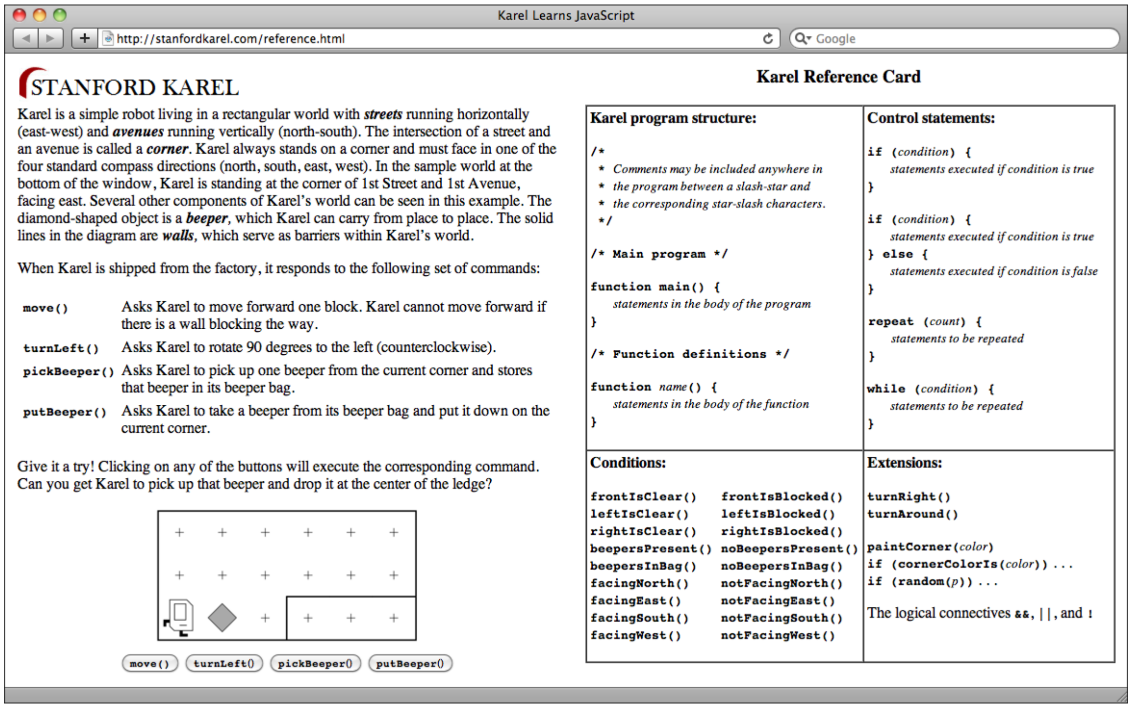
KAREL AS AN EXEMPLAR OF THE NOTHING-BUT-A-BROWSER IDEA

Before we describe our more general vision of browser-based educational tools, it is useful to begin with a specific example of how deploying resources on the Internet can make powerful educational resources available to a broad audience. Partly because simplicity is the key criterion behind its design and partly because it is furthest along in the development process, the best example is the StanfordKarel website, which makes a delightful icon of Stanford computer science curriculum accessible to anyone with a browser, even if that browser is just a smart phone.

The history of Karel the Robot at Stanford

Ever since the late 1970s, students in Stanford's introductory course in computer science have been welcomed to the field by Karel the Robot, named after the Czech playwright Karel Čapek who gave the word *robot* to the English language. Despite the appearance of recent, more sophisticated introductory environments such as Alice and Scratch (and even more powerful extensions to Karel), we have happily continued to use a version based closely on Richard Pattis's original conception precisely because it is simple. As shown in Figure 1 at the bottom of the page, it is possible to combine a narrative introduction to Karel's world and the entire reference documentation in a single browser window. The classroom presentation of Karel is complete in two 50-minute lectures that allow students to master fundamental programming concepts without having to navigate the complexity of a modern programming language. In our experience, the time we devote to these lectures is easily repaid by an increase in student understanding that allows us to move more quickly through later material.

The biggest challenge that students face in using Karel does not come from learning the structure of Karel itself but from the relatively complex process of getting Karel up and running on their computers. Since 2001, when we adopted Java as the programming language for the introductory course, Stanford has used an implementation of Karel that runs inside Eclipse, which is the leading cross-platform programming environment for Java. Although doing so



STANFORD KAREL

Karel is a simple robot living in a rectangular world with ***streets*** running horizontally (east-west) and ***avenues*** running vertically (north-south). The intersection of a street and an avenue is called a ***corner***. Karel always stands on a corner and must face in one of the four standard compass directions (north, south, east, west). In the sample world at the bottom of the window, Karel is standing at the corner of 1st Street and 1st Avenue, facing east. Several other components of Karel's world can be seen in this example. The diamond-shaped object is a ***beeper***, which Karel can carry from place to place. The solid lines in the diagram are ***walls***, which serve as barriers within Karel's world.

When Karel is shipped from the factory, it responds to the following set of commands:

`move()`	Asks Karel to move forward one block. Karel cannot move forward if there is a wall blocking the way.
`turnLeft()`	Asks Karel to rotate 90 degrees to the left (counterclockwise).
`pickBeeper()`	Asks Karel to pick up one beeper from the current corner and stores that beeper in its beeper bag.
`putBeeper()`	Asks Karel to take a beeper from its beeper bag and put it down on the current corner.

Give it a try! Clicking on any of the buttons will execute the corresponding command. Can you get Karel to pick up that beeper and drop it at the center of the ledge?

Karel Reference Card

Karel program structure:

```
/*
 * Comments may be included anywhere in
 * the program between a slash-star and
 * the corresponding star-slash characters.
 */

/* Main program */

function main() {
    statements in the body of the program
}

/* Function definitions */

function name() {
    statements in the body of the function
}
```

Control statements:

```
if (condition) {
    statements executed if condition is true
}

if (condition) {
    statements executed if condition is true
} else {
    statements executed if condition is false
}

repeat (count) {
    statements to be repeated
}

while (condition) {
    statements to be repeated
}
```

Conditions:

```
frontIsClear()    frontIsBlocked()
leftIsClear()     leftIsBlocked()
rightIsClear()    rightIsBlocked()
beepersPresent()  noBeepersPresent()
beepersInBag()    noBeepersInBag()
facingNorth()     notFacingNorth()
facingEast()      notFacingEast()
facingSouth()     notFacingSouth()
facingWest()      notFacingWest()
```

Extensions:

```
turnRight()
turnAround()

paintCorner(color)
if (cornerColorIs(color)) ...
if (random(p)) ...
```

The logical connectives `&&`, `||`, and `!`

Figure 1: StanfordKarel Reference Card

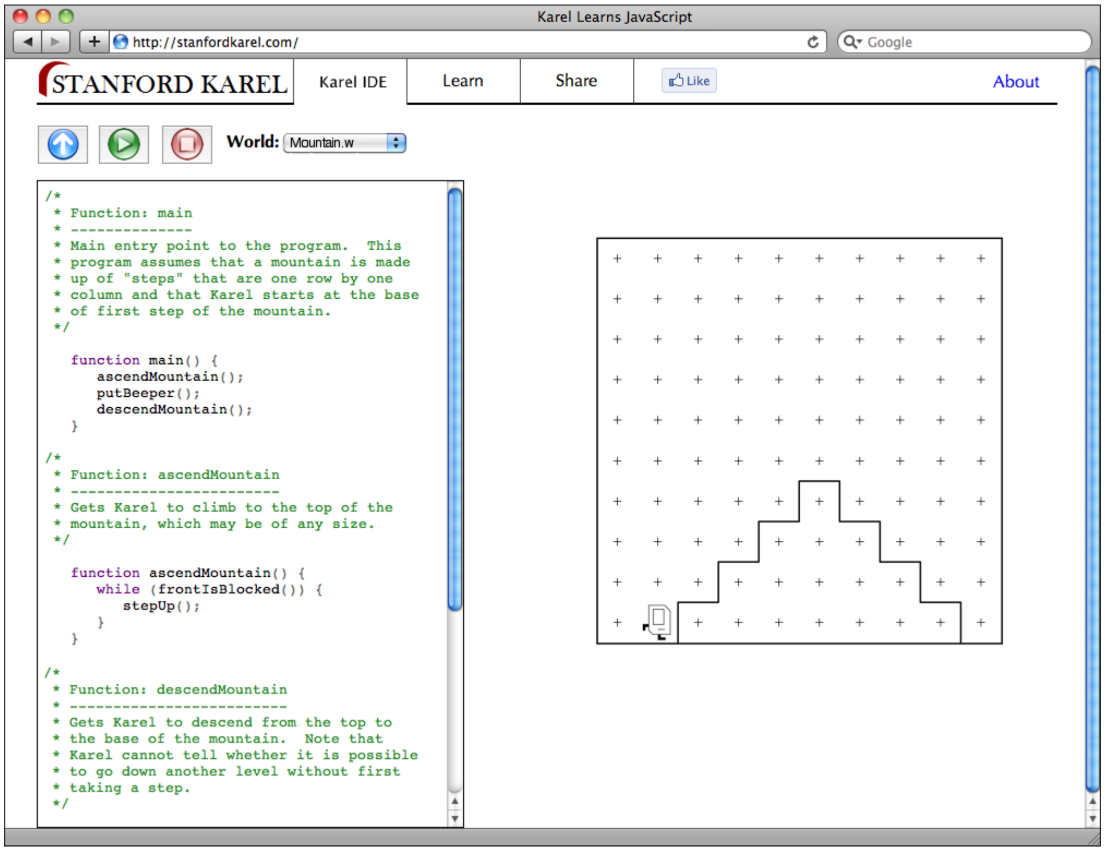
means that students learn how to use the Eclipse framework in the relatively simple world that Karel provides, this strategy forces students to complete the following process before they can run their first Karel program:

1. Download, install, and configure Stanford's distribution of Eclipse.
2. Download a starter project from the course website.
3. Import the project into Eclipse.
4. Learn enough about Eclipse to edit, run, and debug their programs.

Every step in this process is fraught with complexity for beginning students, primarily because Eclipse is an industrial-strength programming environment. While the large number of teaching assistants for our introductory curriculum makes it possible to help students who get stuck at Stanford, there is no support for individuals who are trying to learn this material independently, either through the website for the class or by downloading the lectures available through the Stanford Education Everywhere initiative (`http://see.stanford.edu`). Understandably, that complexity creates a barrier that renders this material inaccessible to much of its potential audience.

Creating a web-based version of Karel

As a prototype for the nothing-but-a-browser vision, we have released a web-based Karel implementation designed to remove all barriers to entry for students learning to program. StanfordKarel features a simple integrated development environment (IDE), an interactive E-Learning course, and an easy-to-use framework that enables students to share their work.



World: Mountain.w

```
/*
 * Function: main
 * --------------
 * Main entry point to the program.  This
 * program assumes that a mountain is made
 * up of "steps" that are one row by one
 * column and that Karel starts at the base
 * of first step of the mountain.
 */

   function main() {
      ascendMountain();
      putBeeper();
      descendMountain();
   }

/*
 * Function: ascendMountain
 * ------------------------
 * Gets Karel to climb to the top of the
 * mountain, which may be of any size.
 */

   function ascendMountain() {
      while (frontIsBlocked()) {
         stepUp();
      }
   }

/*
 * Function: descendMountain
 * -------------------------
 * Gets Karel to descend from the top to
 * the base of the mountain.  Note that
 * Karel cannot tell whether it is possible
 * to go down another level without first
 * taking a step.
 */
```

Figure 2: Screenshot of StanfordKarel

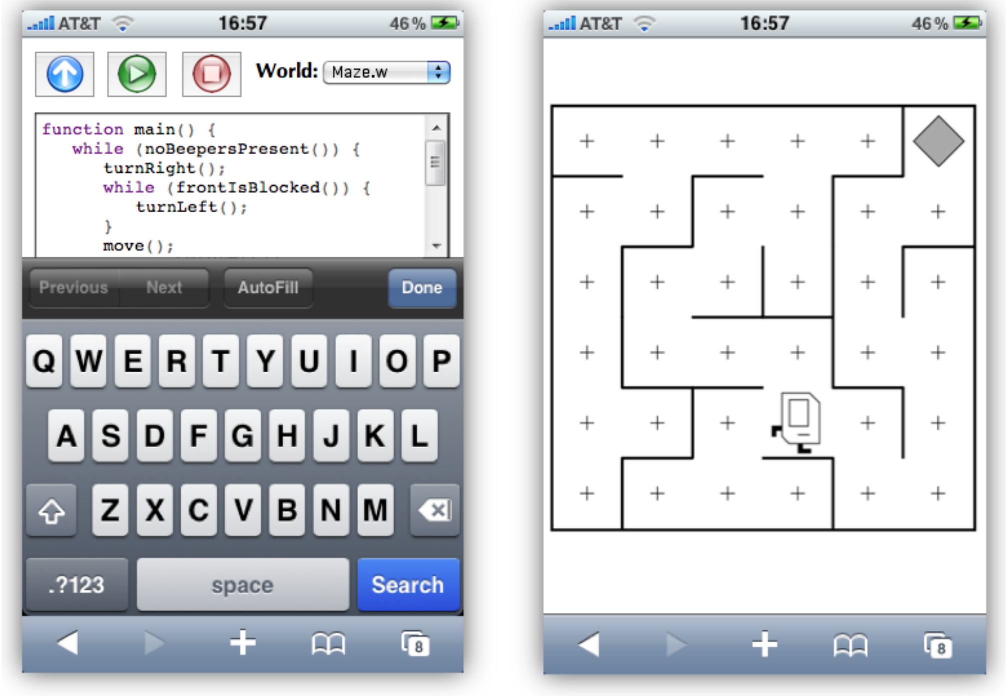
Our goal in developing StanfordKarel was to design the simplest possible IDE that still supports the set of features essential to write and execute a Karel program. When students load the website they are presented with a page that looks much like the display in Figure 2. The page contains a syntax-directed editor in which students can compose and edit their programs, buttons to control program execution, and a graphical display of Karel's world. The contents of Figure 2 show a session in progress, in which the student has written a program that teaches Karel to climb a mountain represented by the stair-step walls illustrated in the map at the right of the figure. To run the program, all the student has to do is hit the green run button, which sends Karel through the steps of the program displayed in the editor. By displaying the editor and map of Karel's world in a single visual space, the student gets to follow the code as the program execution proceeds.

The StanfordKarel IDE presents a simple user interface, requires no installation, and runs on any platform with a browser capable of displaying HTML and JavaScript. By implementing the IDE in the standard tools used to create client-side applications on the web, we gain sufficient platform independence to run Karel programs from any web-capable phone. Figure 3, for example, shows Karel running on an Apple iPhone.

As the article from *The Economist* cited in the introduction makes clear, many people in developing and newly developed nations have access to a mobile phone but limited or no access to a desktop computer. Requiring adopters to download and install a desktop-based IDE makes it far more difficult for those people to participate in the increasingly technological international economy. Making educational technology available on the web holds forth the promise of a more level playing field of economic opportunity.

Embedded learning resources

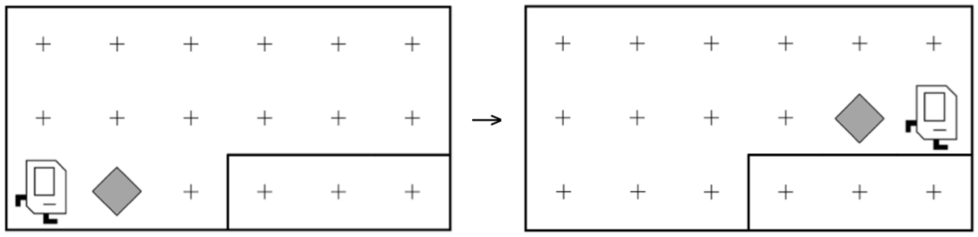
Although making development environments accessible on the web is a necessary step toward our goal of expanding the audience for informatics education, it is by no means sufficient.



```
function main() {
   while (noBeepersPresent()) {
      turnRight();
      while (frontIsBlocked()) {
         turnLeft();
      }
      move();
```

Figure 3: StanfordKarel running on the iPhone's Safari Browser

Creating a comprehensive repository of information requires making other materials available on the web as well. The StanfordKarel website, for example, includes an interactive tutorial that introduces students to the Karel programming language and the underlying principles of algorithmic problem solving. That tutorial is based on the printed materials we have used for many years at Stanford [Roberts05] but takes advantage of the power available to web-based applications. Given that we now have a functional web-based Karel IDE, we are able to blur the lines between the context in which students learn programming concepts and the context in which they write their code. The existence of the StanfordKarel application makes it possible to embed interactive demonstrations directly into the online tutorial. This facility is illustrated by the problem shown at the bottom of Figure 1, in which students are challenged to have Karel pick up the beeper and drop it at the center of the ledge. The buttons on the page in Figure 1 are active. As a result, students can practice entering the sequence of commands that transforms the original state of the world into the desired final state, as follows:



Students learn best when they are able to interact with the material. This principle is especially true in informatics where the experience of experimenting with code and observing results plays an important role in the process of understanding how computers respond to different programs. By including interactive exercises throughout the learning portion of the website, we encourage students to write, test, and debug Karel programs of gradually increasing complexity. In the process, those students learn the essential concepts without having to pay the high conceptual cost of learning to use a sophisticated desktop IDE.

Promoting code sharing

In addition to building an IDE and an interactive learning environment, we have also implemented a simple mechanism that encourages students to share their programs with other users and to learn from programs that other students have shared in this way. One of the three buttons on the Karel IDE is a blue deploy button. When a student clicks this button, a copy of the code in the editor window is stored on the StanfordKarel server. In return, that student receives a link to a page containing their program that can then be shared with others.

Uploading the program to the server also makes that program visible to other users of the StanfordKarel site. As a result, students can see the creative applications that their peers have made along with the code that makes those applications run. Students can then build their own extensions to those programs in a way that creates a decentralized learning community on the web. This approach has proven enormously successful for MIT's Scratch environment [Moloney10], despite the fact that the Scratch environment requires users to register and download additional software. The fact that using StanfordKarel requires nothing but a browser should make it even easier to exploit the "viral" capabilities of web-based applications. Early signs certainly point in that direction. With no public announcement whatever, people somehow found the StanfordKarel website and started sharing programs. In less than two weeks after putting up our own test site, StanfordKarel had already received endorsements from over 100 people who clicked on the thumbs-up icon on the StanfordKarel home page.

ACADEMIC BARRIERS TO WEB-BASED EDUCATIONAL STRATEGIES

As we came to appreciate how much value we could obtain by putting StanfordKarel on the web, one of the questions that occurred to us was why it had taken so long to adopt this strategy, which has such clear advantages. In the introduction, we note that informatics education has in many ways been slow to adopt web-based technologies, especially when compared to commercial services. One explanation for this collective failure is that academic institutions are conservative and therefore move too slowly to adopt the latest technologies. While there is some truth in that analysis, it is also possible that the challenges academic institutions currently face with respect to informatics education come less from being too *late* than from being too *early*.

To understand this seemingly paradoxical claim, it is important to remember that educators were in fact quick to recognize the potential of web-based technologies. Several papers from the 1990s argue that the web completely transforms the nature of informatics education. One paper from 1999, for example, asserts that the web represents a fundamental paradigm shift, going so far as to punctuate that assessment with an exclamation point in the title [Boroni99]. Taking much the same line, the *Curriculum 2001* recommendations from the ACM and IEEE-CS proposed an entirely web-based model as a viable organizing principle for the undergraduate curriculum [ACM01]. The enthusiasm was certainly there.

Historically, the problem is not that the informatics education community failed to embrace web technology in a timely fashion but rather that it embraced the *wrong* technology. For the most part, educators decided—on the basis of what seemed like powerful evidence in the early years—to focus on Java and the applet paradigm as the strategy for creating interactive web content [Roberts04, ACM06]. That strategy, however, did not succeed. By the early 2000s, Java applets had been abandoned by the commercial world in favor of the now ubiquitous blend of JavaScript, HTML, CSS, and related technologies that power today's web-based applications. The reasons behind the failure of the applet paradigm are clearly identified in the following article [Srinivas01] from *Java World:*

> Java applets fueled Java's initial growth. The ability to download code over the network and run it on a variety of desktops offering a rich user interaction proved quite compelling. However, Java's Write Once, Run Anywhere (WORA) promise soon became strained as browsers began to bloat and several incompatibilities emerged that were caused by the Java language itself.

By the time that the failure of the applet paradigm was unambiguously recognized, educators found themselves in a difficult position. Many people had invested considerable time and energy in development efforts that eventually came to nothing. Quite naturally, many of those people were reticent to invest similar levels of effort using a different paradigm for fear that that work would eventually suffer the same fate. To make matters worse, most educators regarded the commercially successful technology—represented most iconically by the JavaScript language itself—to be wildly inefficient, aesthetically horrible, and pedagogically disastrous. As JavaScript expert Douglas Crockford reports

> JavaScript is a language with more than its share of bad parts. It went from non-existence to global adoption in an alarmingly short period of time. It never had an interval in the lab when it could be tried out and polished. It went straight into Netscape Navigator 2 just as it was, and it was very rough. When Java applets failed, JavaScript became the "Language of the Web" by default. JavaScript's popularity is almost completely independent of its qualities as a programming language. [Crockford08]

Despite the grain of truth behind such criticism, we cannot as a community afford to ignore the realities of the technological marketplace. JavaScript has won the technological battle and is pretty much the only game in town. We must learn to play in that game or abandon the enormous potential that web technologies provide. And in doing so, we can take heart from some further thoughts that Crockford has to offer on the subject:

> Fortunately, JavaScript has some extraordinarily good parts. In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders. The best nature of JavaScript is so effectively hidden that for many years the prevailing opinion of JavaScript was that it was an unsightly, incompetent toy. My intention here is to expose the goodness in JavaScript, an outstanding, dynamic programming language.

WHERE DO WE GO FROM HERE

At present, the StanfordKarel website is best regarded as a prototype, offering an existence proof that it is possible to design effective educational software that runs entirely in a browser. Over the next year, our research group at Stanford proposes to extend that technology in the following ways:

1. Publish a collection of web-based animations that enable students to visualize how complex algorithms work. We have already been able to create several of these animations by automatically translating existing Java code into JavaScript, as described in a talk at Google Atlanta [Roberts10].
2. Extend the technology used in the StanfordKarel website to create a similar resource for teaching JavaScript itself at the introductory level. Making this technology work for a more sophisticated programming environment is a much more complex and ambitious task, but our experience to date has convinced us that (1) it is entirely possible to make such a system work and (2) that the payoff of doing so will be enormous.
3. Refine our existing set of tools for creating and maintaining JavaScript content. As Douglas Crockford observes, it is possible—by making the right engineering decisions and building the appropriate scaffolding—to create an "elegant subset [of JavaScript that] is vastly superior to the language as a whole, being more reliable, readable, and maintainable."
4. Expand our collection of materials so that it covers the fundamental core concepts of informatics education, bringing that material to the widest possible audience.

CONCLUSION

In this paper, we have described our experience with StanfordKarel as an exemplar of a completely web-based tool for informatics education available to anyone with a web browser. Using the system requires no installation, no additional software, and no specialized expertise, thereby bringing it within the reach of what has become a large fraction of the world's population. We have also sought to argue why it is essential for the informatics education community to put aside its prejudices and adopt the technologies that have unambiguously won the battle for primacy in the development of interactive content. Finally, we have outlined an ambitious program to expand this project to cover the core of a typical undergraduate curriculum.

Although we recognize that this work remains at an early stage, we believe it is important to get feedback from those parts of the world that will be most affected by the increased accessibility that our web-based vision enables. Talking with people in developing and newly developed countries is essential to the eventual success of this undertaking. By working together we can realize the potential of free, democratically available informatics education that has thus far been only a dream.

REFERENCES

[ACM01] ACM/IEEE-CS Joint Task Force. *Computing Curricula 2001.* December 2001. http://www.acm.org/education/curric_vols/cc2001.pdf

[ACM06] ACM Java Task Force. *Project Rationale.* 25 August 2006. http://jtf.acm.org/rationale/

[Boroni98] Christopher Boroni, Frances Goosey, Michael Grinder, and Rockford Ross. A paradigm shift! The Internet, the Web, browsers, Java and the future of computer science education. *SIGCSE Bulletin*, March 1998. http://doi.acm.org/10.1145/274790.273181

[Brin06] David Brin. Why Johnny can't code. *Salon,* 14 Sep 2006. http://www.salon.com/technology/feature/2006/09/14/basic

[Crockford08] Douglas Crockford. *JavaScript: The Good Parts.* Sebastopol, CA, USA: O'Reilly, May 2008. http://eleventyone.done.hu/OReilly.JavaScript.The.Good.Parts.May.2008.pdf

[Davies11] Stephen Davies, Jennifer Polack-Wahl, and Karen Anewalt. A snapshot of current practices in teaching the introductory programming sequence. *Proceedings of the Forty-second SIGCSE Technical Symposium on Computer Science Education.* Dallas, TX, USA, March 2011. http://doi.acm.org/10.1145/364447.364552

[Economist05] *The Economist.* The real digital divide: Encouraging the spread of mobile phones is the most sensible and effective response to the digital divide. 10 March 2005. http://www.economist.com/node/3742817?story_id=3742817

[GalEzer98] Judith Gal-Ezer and David Harel. What (else) should CS educators know? *Communications of the ACM,* September 1998. http://doi.acm.org/10.1145/285070.285085

[Maloney10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The Scratch programming language and environment. *ACM Transactions on Computing Education,* November 2010. http://doi.acm.org/10.1145/1868358.1868363

[Pattis81] Richard Pattis. *Karel The Robot: A Gentle Introduction to the Art of Programming.* John Wiley & Sons, 1981.

[Reed01] David Reed. Rethinking CS0 with JavaScript. *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education.* Charlotte, NC, USA, February 2001. http://doi.acm.org/10.1145/364447.364552

[Roberts04] Eric Roberts. The dream of a common language: The search for simplicity and stability in computer science education. *Proceedings of the Thirty-fifth SIGCSE Technical Symposium on Computer Science Education*. Norfolk, VA, USA, March 2004. http://doi.acm.org/10.1145/971300.971343

[Roberts05] Eric Roberts. *Karel the Robot Learns Java.* Stanford University, Stanford, CA, USA, September 2005. http://cs.stanford.edu/~eroberts/karel-the-robot-learns-java.pdf

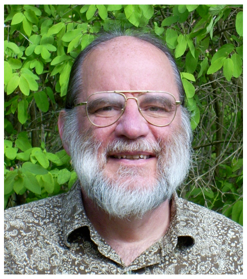
[Roberts10] Eric Roberts. Converting Java into JavaScript. Google Atlanta, Atlanta, GA, USA, October 2010. http://cs.stanford.edu/~eroberts/talks/Google/JavaIntoJavaScript.ppt

[Srinivas01] Raghavan Srinivas. Java Web Start to the rescue. *Java World*, July 2001. http://ww.javaworld.com/javaworld/ jw-07-2001/jw-0706-webstart.html

[Zachary03] Joseph Zachary and Peter Jensen. Exploiting value-added content in an online course: Introducing programming concepts via HTML and JavaScript. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. Reno, NV, USA, February 2003. http://doi.acm.org/10.1145/611892.612016

Biography

Starting in the fall of 2011, **Chris Piech** will be a Ph.D. student in Computer Science studying the application of techniques from artificial intelligence to computer science education. He holds a B.S. degree in Computer Science from Stanford University.



Eric Roberts is Professor of Computer Science at Stanford University. He received his Ph.D. in Applied Mathematics from Harvard University in 1980 and taught at Wellesley College in Massachusetts before joining the Stanford faculty in 1990. He was the principal author of the ACM/IEEE-CS *Curriculum 2001* report and the organizer of the ACM Java Task Force. He is past chair of the ACM Education Board and a member of IFIP Working Group 3.2 (Informatics and ICT in Higher Education).