DOI:10.1145/3633585

Christopher Hoadley and Sara Vogel

Opinion

Autocorrect Is Not: People Are Multilingual and Computer Science Should Be Too

Considering the interconnection of computing and human languages.

Computer science has a language problem—and we are not alluding to programming languages. Many prevalent, flawed views about natural human language are limiting who is in computer science and what people can accomplish with the technology we build.

To start, computer science centers around the English language, and that produces technologies that work poorly for many people. As Manuel Pérez-Quiñones[a] points out, when developers make assumptions about English as the default language, navigating digital device interfaces can be frustrating, even for a professional computer scientist fluent in English such as Pérez-Quiñones. Poor multilingual or character-encoding support, incorrect cultural norms baked into software, and so on—these challenges confront users all over the world. Language-neutral software becomes buggy or unusable if someone inputs non-alphabetic Unicode characters (for example, Chinese ideograms), a right-to-left language, or even uses unexpected punctuation. And in the world of speech recognition, voice assistants often struggle with accented speech, proper names, and many other aspects of natural human speech.

But we believe these English-centric problems are symptomatic of a larger issue in CS: People in general, and our field specifically, think about language in narrow ways. When designers and developers think about language, it might just be as a target for localization or regionalization of software. Or, if we are multilingual, we may feel valued because our skills are commercially useful in global business dealings. We may also think about different human languages as a problem to solve in natural language processing or in information processing. We might tag prose in html files, build speech recognition or user interfaces for some global languages, and perhaps nod approvingly if a résumé crosses our desk listing languages spoken beyond English. But the typical ways we conceive of language in our work are narrow and contradict what linguists are finding about how people communicate.

Language Evolution

Language is emergent, ever-changing, and dynamic. Nation-states and technologies may have codified and attempted to standardize the lexicons, phonologies, and syntaxes of human languages—such as Hindi, Mandarin, Spanish, English, and many others, but in fact, sociolinguists underscore what people actually do with language goes far beyond what the rulebooks say. People said to speak the same language (for example, an English speaker in Louisiana and one in Scotland) may be unintelligible to each other, while people said to be speakers of different languages (for example, a Portuguese speaker in

a See https://bit.ly/48HDFOu

IMAGE BY ANDRIJ BORYS ASSOCIATES. USING SHUTTERSTOCK

Brazil and a Spanish speaker in Argentina) may find they can communicate well enough without "learning another language." The "English" used by teenagers is not the same as the "English" used by their grandparents. Recent sociolinguistics research[b] has called attention to this fact, arguing that languages are social and political categories rather than linguistic absolutes. People dynamically draw on words, grammars, and sounds they know—as well as multimodal elements like gestures, media references, clothing, and technology—to communicate their intended messages in particular contexts. Sociolinguists call this process *translanguaging*.

Language is messy and useful regardless of whether it conforms to what a grammarian may have taught in school, and if we put "academic" or "standard" English (or any other language) on a pedestal, the natural outcome of this closed-mindedness is we can exclude people who are not already part of dominant groups. Translanguaging, on the other hand, recognizes that everyone, everywhere, is using a mix of ways of communicating based on their own repertoire, and that oftentimes this crosses the boundaries of official languages with names and dictionaries. When multilingual people use words from languages other than the one the software is set to recognize, their input gets autocorrected or flagged as errors; this extends to nonstandard use of a single language, and it is even worse for people who naturally communicate in ways that defy the normative "one language at a time" model. In computer science, we need to deemphasize interfaces and natural language processing that relies on tidy official languages and correctly handle how language speakers innovate on language all the time. One reason we fail to do this is simplification; it is easier to build systems if we pretend there are a fixed set of named languages, and it certainly matches the way dictionaries, schools, and grammar textbooks are written.

But there is a deeper, and more insidious reason we pretend language is neat and tidy. Language can be used to divide groups of people in ways that keep some powerful and some powerless. Languages such as English gain prestige because their speakers have political and economic power—not because they are any better at helping people express themselves and communicate.

English, especially the kind spoken by American or British middle-class white university graduates, is in many ways a price of entry to certain groups in the community of computer scientists, whether conversing on platforms such as stackoverflow.com or participating in various workplaces. Certainly, other languages are present in different contexts; a company in Germany may embrace a mix of German and English in its work, or an IT company in Bangalore may have a dozen or more languages

b See https://bit.ly/48vWcxK

commonly used in the workplace. But people who do not speak high-status languages will tend to be marginalized in or excluded from those spaces. Whether it is a Black American woman whose Baltimore accent is perceived as "threatening" or a "disability"[c] by a manager at a tech company, or an Indian programmer whose Hindi or English reflect accents other than a middle-class education, as in all aspects of society, people may be labeled or treated as unsuitable for being programmers. Language can also be a gatekeeper to CS for young people who hope to crack into the field. For example, in the U.S., education systems hold schools accountable for the English learning of multilingual K–12 students; those students may have to prove their English competency before they can take "honors" or "enrichment" classes such as CS.

While some people argue that assimilating to the dominant languages in computing (English and the programming languages of the day) is important to build common ground, we disagree. There is no reason, for instance, a compiler should reject code that includes comments in unexpected character sets, or programmers should not be able to use programming keywords from a variety of human languages. The point is, enforced English-centric monolingualism is so common we do not even notice it, and this should change.

Given the frustration, errors, and exclusion that English dominance has caused in our field, what can we do? First, we can invest in social and technical means to increase inclusion. On the social side, we can recognize language diversity and multilingualism are the norm around the world. It is estimated there are more than 7,000 languages spoken globally. Worldwide, it is more common than not for individuals to be multilingual. We must embrace speakers of many languages in CS settings, and we must stop emphasizing English as a prerequisite for participating in computing, both for users and developers. English-only or English-dominant tech settings should immediately raise questions about who is missing and what language is being excluded. On the technical side, we might design and test new models for more inclusive programming environments. For example, the Scratch programming environment from MIT allows children to select among dozens of languages to relabel programming blocks; we would advocate people should even be able to mix and match keywords from multiple human languages on those code blocks, or relabel blocks with language of their choosing.

By equipping people with a new way to express ideas through code, might CS education promote metalinguistic awareness?

Second, computer scientists, uniquely, can help invent new ways for technology to support communication and expression that were not possible before digital technology. Computer scientists communicate through code itself. As Donald Knuth described in defining the need for "literate programming,"[d] code is not merely a set of incantations to produce behavior from a computer, but a powerful medium to share certain types of ideas for other people to read and build on. Like human languages, code conveys nuance, style, and can be awkward or elegant, legible or obtuse. Like formal mathematical notation, code is a way of precisely describing concepts or propositions that is complementary to the ways we use natural human language. When we think of code as a language we can use to communicate with other computer scientists, we recognize it brings us the power to express our ideas in ways that are not only precise, but that can come to life through their execution. In our research,[e] we have seen a teacher who used the Scratch programming environment to help young multilingual people in an English-as-a-new-language course because it offered another way for them to express themselves—through a mix of home languages, multimedia, and code in addition to English—in other words, a new kind of translanguaging. Research on human multilingualism indicates bilingual learners develop a "metalinguistic awareness,"[f] that is, they can make language an object of thought and reflection, which is linked to enduring advantages bilingual learners have in later schooling. By equipping people with a new way to express ideas through code, might CS education promote metalinguistic awareness?

Conclusion

Thinking more about how we as computer scientists communicate and refine our ideas, we start to see ways that coding is and is not like natural human languages, and both the similarities and differences open up possibilities. As computational power for natural language processing expands, we can consider new models for coding itself that might be more flexible and evolutionary than fixed syntax and semantics—more conversational, less like a solution to an equation (see for example, the work on natural programming). And if we embrace what linguists teach us about multilingualism—that we should accept the wonderful diversity and evolution of language rather than putting language in a straightjacket—we can truly open CS as a field and our technologies themselves to the great, big, messy, and useful possibilities. C

c See https://bit.ly/3TMMbHH

d See https://bit.ly/3RcrLq5

e See https://bit.ly/3ROeB1t

f See https://bit.ly/3TFI7sS

Christopher Hoadley (tophe@buffalo.edu) is a professor in learning and instruction and a professor in computer science and engineering at the University at Buffalo, Buffalo, NY, USA.

Sara Vogel (Sara.Vogel@cuny.edu) is the computer integrated teacher education research director at the City University of New York, NY, USA.