

Rob Reed  
rob @ ode-is-simple.com  
April 6, 2010 (originally written early 2009)  
Ode and computer science

I want to address what, in my estimation, is now, and arguably has been for nearly a decade, the most fundamental problem in computer science. Namely, a lack of widespread interest in the field as a course of study. Continued progress is leading us to any number of truly difficult problems and our best hope for tackling any of them is recruiting as many capable people to the cause as we possibly can. Accomplishing this will require that we address two complex and difficult issues.

First we must rethink the ways in which we collaborate, and make a commitment to revitalizing the sense of community within and beyond academic departments.

Secondly, we must bring bright, exuberant, talented, and dedicated new people into the field.

I've chosen to focus on the latter issue both because I believe it is the more approachable of the two, and the more immediate need. Furthermore, it is reasonable to assume that we can begin to change the nature of the community and improve collaboration simply by involving more people, who bring with them new perspective, ideas, and ambitions, which are the result of their own unique collections of experiences. I contend that there is much more at stake here than enrollment numbers and short term stability in academic departments. More to the point, short term gains will continue to remain out of reach until we consider the underlying issues.

There are several such issues, all of which must be addressed:

1. We have not characterized the situation properly, in the sense that we have underestimated the scale of the problem and failed to identify its cause.
2. In part because of this mischaracterization, we have not done what's necessary to appeal to new students. This involves not only promoting computer science, but also helping prospective students to see the relevance of the field to their interests and ambitions, and to assuage any concerns they may have about their background, qualifications, and prospects for eventual success.
3. Finally we have failed to adjust the methods used to introduce students to programming and other fundamental CS topics in response to significant changes in the field, and perhaps more importantly, radical changes in their lives, expectations, and relationship(s) with computer technologies.

In this short proposal I'll consider each of these issues briefly before introducing my project, which is relevant to all of them.

The fact that enrollment numbers are roughly half of what they were a decade ago, and a quarter of the historical highs toward the end of the 1980s, only hints at the current problem. It's somewhat misguided to think in terms of figures from 10 years ago; regarding them as if they represented a momentous goal. There has been a lot of change in the past decade, much of it progress toward problems which are a great deal more complex, and requiring more, not the same amount of, participation. We need to concern ourselves with involving a number of people sufficient to meet the needs of the field given the scope of what we're trying to accomplish now, unless we're interested in only filling seats in a classroom. Achieving head counts seen in the past may be a notable milestone, but should not be considered the goal unless it can be determined somehow that it is in fact relevant.

Why belabor this point? Because establishing that we need to be thinking in terms of new goals implies that we need to pursue new and innovative strategies rather than adopting a cyclic view of the situation, and patiently, perhaps wistfully, waiting for high tide to return. It could be that the reality is more comparable to the linear pattern of geologic time with distinct and unique periods that cannot be expected

to reoccur. To be clear, I do not believe that there is sufficient evidence that this downturn is self correcting.

Whereas once it was a widely held opinion that there was a causal relationship between the so-called 'dot-com collapse' of the late 90s and the current enrollment crisis, it seems increasingly likely that this relationship is correlational at best, if not coincidental. As evidence of this, the web, which was of course at the center of the collapse, has rebounded in recent years, rising to play an increasingly important economic, social, and cultural role. Today, the web is in many ways helping to redefine life in the modern world. For example, from an economic standpoint, the importance of the web today dwarves that of the web of 10 years ago. This is the result of many factors, including:

- Continued penetration of internet and especially broadband connectivity
- Growth and development of increasingly compelling web based services, including the emergence of the social networking phenomenon
- The influence of standards which are leading toward the realization of many of the design goals of the long anticipated 'semantic web' (Various microformats are already proving to be incredibly powerful.)

The capability to discover, utilize, and repurpose data in sophisticated new ways is likely to further expand the web's importance. I have said quite a bit about the web because I believe it's crucial to this discussion. I'll elaborate on this idea more throughout the proposal.

Demand has increased for both information technology and programming jobs, after withstanding uncertain economic times, the supposed threat of competition due to global outsourcing, and other challenges. Despite these significant and encouraging happenings, we have not seen a corresponding positive correction in the numbers of new students interested in computer science.

Finally, it's likely that the stability we have known in the industry over the past half decade or so will not continue forever unabated. A slumping US economy is only one reminder that there is a limited window of opportunity when overwhelming confidence in the field might lead students to discover computer science on their own. It is simply irresponsible to leave the future of the discipline up to the fates.

So, how then can we involve a new generation of students in the field, if we are willing to admit that the situation may not simply remedy itself? I contend that we need to answer two important questions.

1. What must we do to inspire an interest in CS? This is an especially important issue for our field, because unlike other disciplines, we should anticipate that new students may have little to no relevant prior knowledge. (I'll discuss this point further).
2. How can we provide the sort of academic environment that students just being introduced to the field will require to achieve the confidence and the expertise necessary to make some meaningful contribution?

Computer Science, or basic introduction to programming for that matter, has never been established as a core part of the elementary or secondary school curriculum in the same way that other sciences, math, and social studies are considered to be integral. Rather, it is typically grouped with other electives, such as arts and music instruction, if it is offered at all. Unfortunately, even the best of these programs often do not receive adequate resources. Rather than simply bemoaning this fact, it's important that we acknowledge and understand the current situation so that we can act appropriately to improve upon it.

First, effectively mounting a CS program requires a unique set of capabilities and resources, specific to and unlike any other subject. From staff to infrastructure, putting all of the pieces into place may not be an

easy proposition even for the best of schools. The challenges involved with implementing such a program should not be underestimated.

Of course we should not fail to consider the decisions of students themselves and the motivation behind those decisions. After all, if this were simply a matter of schools undervaluing the importance of programming and computer science, we might expect students to find their way to the field at the first opportunity, and that certainly does not seem to be happening.

College-bound students are expected to complete a demanding slate of courses followed by a battery of tests, consisting of not only the SAT and ACT, along with subject exams, but also standardized tests intended to satisfy minimum competency requirements in core subject areas. Because standardized testing differs widely from state to state, I'll highlight the Regents exams in New York state (where I attended high school) to illustrate my point.

In New York students planning to continue on to post-secondary education must pass a series of as many as 8 exams to qualify for a Regents diploma. Of course this is in addition to earning a passing grade for coursework in each of the subject areas based on an assessment which is largely determined by the school and instructor.

The Regents exams include: Integrated Algebra, Global History and Geography, US History and Government, English, and at least one of Earth Science, Biology, Chemistry, or Physics. In order to qualify for an 'Advanced Regents Diploma' students must pass an additional science and math test, and oral and written foreign language examinations.

I would argue that having completed 4 years of studies which emphasize these subjects almost exclusively, followed by the exams, it is not unreasonable to assume that students will feel ready to identify their academic strengths and weaknesses. Of course Computer Science is not included among this collection of subjects, and, as a result, may be overlooked by students who will soon be entering college and making critical decisions about their academic futures. The surest decision a new college student may be able to make regarding computer science is that she is not familiar with the subject, and so may not be well suited for the field.

Outreach programs, (i.e. programs designed to introduce students to Computer Science before college) may benefit a limited number of students from among select groups targeted for participation. But, the very nature of these types of programs suggest several significant limitations.

First, they do not address the immediate need. There are many students entering college now who may be interested in the field, but cannot benefit from an outreach program that we might start today. In fact there are hundreds of millions of people currently involved in post-secondary education, well over ten million in the US alone, and we should not simply cast them aside.

Such programs cannot be considered solutions to the general problem. They are by design limited to small groups of students, and this is inconsistent with the scope of the problem. They presuppose a willingness and an ability on the part of elementary and secondary schools to make effective use of such programs, which simply may not be the reality given the many academic and financial pressures these schools operate under (some of which I have already discussed).

Finally, it is a mistake to limit ourselves to a small number of students who happen to come from just the right background. This is the unfortunate situation we find ourselves in today. We cannot predict where we will find receptive students, and so it's a disservice to everyone involved not to cultivate an interest in CS as broadly as possible. It is important to consider also that these sorts of efforts may unintentionally further existing inequities. In the United States for example, academic achievement varies greatly from state to state. We should be wary of mistaking opportunity and advantage for talent and dedication, and in so doing worsen what is already a regrettable situation.

The responsibility lies with us to provide all interested and capable students with the opportunity to pursue an education in the field. In this way we might hope do our part to insure equal opportunity for all new students regardless of background, and also to avail ourselves of the greatest number of prospective students. If we do not presuppose an exposure to computer science, we can welcome all new students who express an interest in the field.

What I have just discussed does not completely account for the current situation. After all, Computer Science has not suddenly fallen out of favor. As I have already stated, the subject has never been an integral part of the curriculum at the elementary and secondary levels. What has changed in the past decade is the nature of the relationship between people and computer technologies.

Whereas once pursuing an interest in computers necessitated learning about the low level operation of the machine and programming, putting students on a path toward Computer Science, there are now many more paths to choose from. If we assume for a moment, and just for the sake of argument, that there are as many people interested in computers and related topics as there were a decade ago then we might expect to see a decline in interest in the field as a result of some of those prospective students choosing to pursue other options. Furthermore, the mere existence of alternatives changes the perceived value of a traditional education in Computer Science.

This is not unlike the situation that many media industries have found themselves in over roughly the same period. The music and film industries, for example, have been vocal about declining revenues, which they attribute to piracy but may be better understood by considering the changing nature of the broader media and entertainment landscape. The fact is that there are more entertainment options now than there ever have been. The moviegoing public is also the tv-watching, video-game playing, web-surfing, social-networking, music listening, and world-traveling public. Let's take a closer look at video games as an example.

The gaming industry has generated more than \$6.6 billion in the first five months of 2008 alone, and it's estimated that the revenue will reach more than \$23 billion this year. The average cost of a movie ticket in the United States as of July 2008 is \$7.08. That's 3,248,587,570 movie tickets. Of course, all of the money being spent on video game consoles, games and accessories does not come at the expense of the film industry. But, it's clear that this represents a significant amount of loss to competition. Money that's being spent on games is money that's not being spent on movies and music.

It is either naive or dishonest to suggest that the only conceivable explanation for decline in sales is piracy. The emergence of increased and varied competition, including entirely new entertainment options, fundamentally changes the nature of the entertainment business. And it's not a simple matter of sharing the same audience among a larger and more diverse collection of options. Though the options are all competing, they are not equivalent, and the differences may alter the perceived value of each.

Of course change is neither inherently good or bad. In the case of the media industry, new technologies such as HDTV, the Blu-ray optical disc format, and new portable media devices along with the emergence and maturation of new industries, such as interactive media (e.g. the video game industry), and new distribution models (e.g. online distribution) are certainly disruptive, but have potential to be tremendously rewarding. On the other hand, consumers, after consideration of the new options, may be unwilling to make the same choices when it comes to allocating the limited amount of time and money they have at their disposal, which does not change as a direct consequence of increased choice.

Simply stated, change brings with it exciting new opportunities but often demands new strategies in order to ensure continued success. I contend that this describes precisely the situation the field of computer science finds itself in. It would be a mistake not to see it as a tremendous opportunity, and to do everything we can to take advantage of it. It's an obvious statement that computer technologies have never had more of an impact on all of our lives. It's not unreasonable to assume that the average college undergrad is in contact with computers, computer networks, and many inherent and complex problems which fall within the domain of computer science as a necessary part of their daily routine. This wasn't the

case even 5 years ago. An important factor contributing to this increased mainstream acceptance of computers has been the internet. More specifically, I want to talk about the web.

Although other applications and services are perhaps no less important, it is the web that has incited our collective imagination over the past several years. Social networking for example (including weblogs, photo and video sharing services, mapping services, and others) have sparked an interest, not only in the services themselves, but in the possibilities of what can be accomplished. The web has grown to become increasingly sophisticated due in part to the emergence of standards for everything from design to development methodologies, and data formats, resulting from the evolutionary progress which has taken place over the past several years. Of course a number of high-profile, and popular sites from Google and Amazon to Twitter and Facebook (among many others) have helped popularize the web.

Ironically, I believe that the widespread adoption of computer technologies and the growth in sophistication and popularity of the web are key factors contributing to a lack of interest in Computer Science as a course of study. 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 seemingly be used to accomplish 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 highly motivated to enter the field a decade ago may consider computer science today as unapproachable and almost irrelevant to their ambitions. To describe the current situation in terms of the familiar metaphor of a carrot on a stick dangling 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 satiated. All of this means it's very important that we carefully consider how we introduce students to the field, beginning with programming and basic theory. It's crucially important that we get this part right. There are two important objectives that we must achieve in order to be successful.

First, as I've already discussed, there's a disconnect between the work students imagine they may want to do with a CS education and what they're able to accomplish early in their academic career(s). We need to bridge this gap by providing projects that can help them recognize the relevance of what they're learning to work that is intrinsically motivational. The job of bringing new students to the field is as much a matter of encouragement and inspiration as it is education.

Of course, instruction is critically important, and we do need to do a better job with teaching, especially introductory material. As the field has matured and expanded to encompass a greater number of divergent topics, there is more to learn, and so more that must be taught. Of course we have the same amount of time to do it. Given increasing demands to cover a greater number of more complex topics, it is easy to make the mistake of dismissing formative introductory material as basic. To do so is a disservice to everyone involved; new students are left unprepared for the challenging work ahead of them, and as a result many choose not to continue on after their initial exposure, or opt not to do advanced work beyond a four year degree. Students who do choose to stay in the field are not adequately prepared make their way into progressively more advanced courses where their participation threatens to slow progress to the point that it is not possible to spend a sufficient amount of time on more complex topics. The effect of this accumulates, artificially limiting the potential of students, cutting short academic careers, and even impacting the productivity and culture of an academic department.

The limited, uninspired work that often characterizes introductory programming courses sends the wrong message to new students, suggesting that they're unprepared to face more difficult work that lies ahead of them and leaving them feeling insecure about their abilities as new programmers. An ideal solution would introduce students to foundational topics and do so in a way that builds confidence and piques their interest by affording them the opportunity to do work they find rewarding.

I claim that the web is a perfect bridge between the world of CS and mainstream computer technologies.

First, we don't have to wonder if the web is engaging. It is experiencing phenomenal growth and attracting a lot of interest generally, and among those people who we might hope to attract 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.

Let's briefly consider Twitter as an example. The idea of the service is very simple. To quote twitter.com:

Twitter is a service for friends, family, and co-workers to communicate and stay connected through the exchange of quick, frequent answers to one simple question: What are you doing?

Twitter's core technology is a device agnostic message routing system with rudimentary social networking features. By accepting messages from sms, web, mobile web, instant message, or from third party API projects, Twitter makes it easy for folks to stay connected.

Because this doesn't quite describe the service, I'll add that it allows for posting short messages (140 characters or less), which are publicly visible and collected together on a webpage, what might be called a micro-blog. Another important aspect of the service is that it allows friends and complete strangers alike to follow or subscribe to each other's posts (called "tweets" in Twitter-speak). Posts from members I'm following appear on my page along with my own posts. This is important for exposing relationships between members, referral, and discoverability. So, the service is essentially social IM meets weblogging.

What is apparently such a simple idea has been touted as 'the next killer app', a fundamentally new form of communication, and a powerful tool for marketing and promotion, collaboration, customer relationship management, news, organizing and facilitating large scale events, and even national politics. The service is or has been used by The New York Times, members of the United States congress, democratic and republican presidential candidates, NASA, and many other organizations and tens of millions of active members.

Along the way Twitter, whose founders do not have a background in CS, has struggled with topics that any computer scientist would recognize as both challenging and critically important, from trust and security, to data mining and machine learning, to scalability and architecture. In regards to the scalability issues, they've struggled at times to keep the service online and accessible. After hiring administrators, managers, and programmers they've recently taken the step of hiring a chief scientist to help them address some of these problems.

Of course it's no surprise to Google, Amazon, Microsoft and other large players that there are hard science problems which must be solved if we are to realize the potential of the web. However, for a decade or so Computer Science has failed to fully embrace the web, which has contributed to a perceived lack in relevance of the field. As the problems scale, literally and figuratively, we begin to see the relevancy gap shrink as Computer Science is pulled into the fray. But we need to do more to push or promote this connection.

Beyond appealing to new students, there are other advantages to be realized by embracing the web. As the field has become more complex, there is a tendency toward specialization which has led in turn to more complexity and divergent interests. These niche communities develop their own languages,

methodologies, patterns and styles of communication which can become very insular. At some point it becomes difficult for colleagues, even within a single department, to carry on meaningful conversations, much less collaborate, between project groups. We can take advantage of the web to establish new methods for more effective communication and collaboration.

It's helpful perhaps to remind ourselves of the design goals of the web. Originally, it was intended to be an application for working collaboratively among possibly distant groups in innovative ways. Keep in mind that the idea of distance does not necessarily involve geographical distance. There is temporal distance, and ideological distance as well.

This begs the question, how do we capitalize on the popularity and importance of the web to involve more people in CS, and realize the potential of the web to help us collaborate more effectively? In other words, how do we take advantage of the web to build a stronger academic community, enabling us to become better thinkers, colleagues, mentors, educators, and learners? This question brings me to the end of a lengthy discussion of what is a complicated situation and brings me to my project, which is intended to address all of the issues just described.

I've developed a web-based project which is intended to take advantage of the appeal and utility of the web, and designed both to serve as a 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.

The project 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.

Though the web today might be described as a consumer-oriented media space, collaboration is the key concept that drives its growth and development. In fact the web is typically identified not by the collection of technologies that describe how it functions, but by the contributions of its users (i.e. its content). In recognition of this, the project is intended to be more than a vehicle for exploring topics related to programming, but also an application that can be used to facilitate communication and collaboration.

There are a quite a few open source weblog packages and wikis available. The project I am proposing is unique in that it emphasizes key introductory concepts (e.g. flow control, data structures) that a new student would be expected to be responsible for as part of a first programming course. The entire project is written using only these basic concepts. As such, new students can be expected to understand the application in its entirety. Moreover because it's extensible, it allows for literally unlimited creative freedom for students to explore their own interests, and can be used to bridge into more advanced topics. For example, because this is a web publishing platform, there are any number of difficult challenges involved related to communications protocols, distribution, inter-process communication, addressing, etc.

Additionally, the project is intended to support some of the activities which occupy our time (e.g. creating and sharing presentations, note taking, and discussion). As I have already suggested, it is not enough to attract new students, we must provide them with tools to help them accomplish their goals, engage them in a continuous dialog, and encourage their participation.

The application is written in Perl. I chose Perl for several reasons. First because it is a mature and stable language and particularly adept at text processing, which describes the bulk of what we need it to do. It is well established as a web programming language but general purpose and adaptable enough to be used for other types of applications as well.

The fact that Perl is a high level language allows us to emphasize higher level concepts from logic and structure to design, documentation, and even architecture. I contend that this is a better place to start

than focusing on low level issues because it will allow students to practice conceptualizing the whole of a project and anticipate problems with design and implementation, from security to efficiency and scalability, even as new programmers. I believe that this will help students appreciate the importance of low-level issues and prepare them for the sort of work that awaits them should they choose to continue on in the field, e.g. architecture, data structures, algorithms, computational theory, and advanced work that builds on these topics. In other words we can help students to think and to reason like a programmer early on, without getting mired in low level issues, which experienced programmers understand almost intuitively. This is important to establishing the relevance of what students learn early in their education both within the academic community and beyond. Within the community they will be better able to comprehend and participate in the discussions and the work going on around them. Beyond the department, they will be able to think about how to solve the sort of practical problems that they encounter in their other work. For students who choose not to continue with CS, this sort of introduction to programming should provide a better understanding of computer technologies which may play an important role in whatever they choose to do. An intro course featuring a high level language like Perl should allow students to apply their knowledge of programming to their work.

As for Perl's shortcomings, that it is a permissive language, has a tendency to result in unstructured programs, that it is overly idiomatic; these weaknesses can be overcome simply by adhering to good coding practices. There's certainly nothing about Perl that prevents code from being well-written. The source code for the project should be as explicit and accessible as possible as opposed to being terse and overly idiomatic, except where the idioms are more efficient. Wherever idioms are preferred, liberal annotations should be used so that people who are unfamiliar with the eccentricities of the language can follow along.