Criticizing and Modernizing Computing Curriculum: The Case of the Web and the Social Issues Courses

Randy Connolly
Dept. Computer Science & Information Systems
Mount Royal University
4825 Mount Royal Gate SW, Calgary, AB,
Canada T3E 6K6
403-440-6061
rconnolly@mtroyal.ca

ABSTRACT
Computing education has faced a variety of ongoing and critical self-examinations over the past 15 years. This paper provides a set of critiques and alternative teaching approaches for two vital but under-reported computing knowledge areas: web development and computing ethics/social issues. It concludes with a claim that these two knowledge areas can also provide an important way to integrate the often-heterogeneous knowledge areas in the computing curriculum.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer & Information Science Education – computer science education.

General Terms
Design, Experimentation.

Keywords
Education, Ethics, Social Issues, Web Development

1. INTRODUCTION
Computing education has faced a variety of ongoing and critical self-examinations of different aspects of the curriculum over the past 15 years. This crisis in computing enrollments was no doubt the most important motivation for this inward turn in computing education research. Another, perhaps somewhat less visible but no less important reason was the finer articulation of computing as an academic discipline: computer science is now joined by information systems, computer engineering, software engineering, and, since 2005, information technology. Each of the computing disciplines have overlapping areas of concern; some of these common topic areas (e.g., first-year programming) have been rigorously examined in the education literature as a consequence. Other common topic areas have not been as adequately examined. This paper provides a set of critiques and alternative teaching approaches for two vital topic areas that have been under-reported in the computing education literature: web development and the ethics/social issues knowledge areas. This paper both restates and provides a capsule summary of this author’s more detailed papers on these two subjects [4,5,6,7].

2. TEACHING THE WEB
In the last decade, the web has been transformed into an ever-present information retrieval mechanism as well as the principal platform for hosting software applications. Somewhat inexplicably, however, while the web has experienced this remarkable transformation, there has been a certain dormancy to the way that the web is being taught in the computing curriculum. While web topics are certainly a recommended component of a variety of ACM curricula reports, it is, for instance, still a peripheral topic within computer science. In the CS 2008 and CS 2013 curricula almost all of the topics recognizable as pertaining to the field of web development are marked as elective topics [1,2]. Similarly, there is a relative dearth of research in the computer education literature which is somewhat surprising given the ostensive importance of web technology in the real world of software development. Each year SIGCSE and ITiCSE have multiple papers on teaching beginning programming, databases, and other curricula areas, but only fairly rarely will a paper on teaching web development appear.

In fact, looking at all the papers presented at SIGCSE and ITiCSE in the last decade, only about 1.5% of them have pertained at all to the teaching of web topics. Since 2005 the percentage is even lower; there have been only 17 papers during that time about web topics – and many of those were only peripherally focused on teaching web development. As a point of comparison, in the same five-year time span, there were 65 papers on game-related topics in those two conferences. While clearly this represents the discipline’s interest in trying to increase enrollments in computing via the appeal of games, it certainly is not at all indicative of the job market computing graduates will face, one in which arguably a majority of software development jobs are broadly within the web context [15].

Most educators would no doubt agree that there is a lot of material that needs to be shoe-horned into a typical four-year computer program. Given the unruly and shifting number of web standards and practices, it is not surprising that “there does not seem to be a consensus about where in the curriculum, and at what detail, to introduce this material” [22]. In a literature review on teaching web topics [6], by far the most common way of teaching the web course in the reported literature is to try to fit all
the material within a single upper-level course. Not coincidentally, it is in papers reporting an All-the-Web-in-One-Course (AWOC) approach that one finds most of the complaints about the difficulty of teaching the web course [6].

Given the oft-stated worries about the breadth of material needing to be taught in a web course, this author strongly believes that the AWOC approach very much needs to be retired. Back in the late 1960s/early 1970s, a math program might have had a single course in programming in Fortran or a business program might have a single course on data processing in Cobol, but eventually it was recognized that a body of knowledge as complex as programming requires multiple courses to teach the material properly. Web development should by now be in a similar state.

The principal reason why the AWOC approach is no longer appropriate is due to the peculiar combination of change and persistence that characterizes the web development world. The web environment experiences something analogous to paradigm shifts, in that web development experiences cycles of short periods of substantial change followed by periods of relative stability (see Table 1). The key point is that these are not exactly paradigm shifts in the Kuhnian sense since a previous web technology isn’t always being replaced and replaced (though, for instance, with CGI and ASP it was); more often the next set of important technologies or approaches are being used “on top” of, or in addition to, the previous ones, meaning that knowledge and expertise in the previous paradigm is still required.

<table>
<thead>
<tr>
<th>Principal technology context/layer</th>
<th>Approx. Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML + CGI</td>
<td>1995-1998</td>
</tr>
<tr>
<td>CSS (simple) + Javascript (simple) + ASP/JSP</td>
<td>1999-2002</td>
</tr>
<tr>
<td>Semantic Web Standards + PHP/ASP.NET</td>
<td>2003-2006</td>
</tr>
<tr>
<td>AJAX + REST/JSON services</td>
<td>2006-2009</td>
</tr>
<tr>
<td>Frameworks + Platforms (WordPress,JQuery,Sharepoint,etc)</td>
<td>2009-</td>
</tr>
</tbody>
</table>

Table 1. Web development paradigms

There are indications that the web development environment is currently at the start of a new cycle of change and, as such, new topics need to be integrated into how we teach the web. For those stuck in an AWOC model, it will become increasingly difficult to keep those courses comprehensive and relevant. As the title of another paper with a similar theme suggests, those courses will increasingly seem like they are “party like it’s 1999” [4].

So what needs to continue to be taught in any web course and what new topics need to be added in? The remainder of this section describes those web topics divided into two subcategories: material that is prescribed, however briefly, for example, in the ACM CS 2008 report, and material that is not mentioned in the ACM report, but which, nonetheless, needs to be covered in contemporary and future web courses.

2.1 Web Topics in CS 2008

The obvious beginning point of any web stream is with the way the web works (e.g., the various protocols, the hardware infrastructure) and with HTML itself. While HTML is relatively trivial to learn, it is important to cover it in a way that is consonant with contemporary best practice usage. For reasons of accessibility, maintainability, content management system (CMS) integration, and search-engine optimization (SEO), HTML markup today is semantically-structured so as to separate content from presentation [see 4 and 6 for a more in-depth discussion of these topic areas].

For these same reasons, CSS is now an essential part of web development. CSS coverage in reported AWOC courses is often quite minimal [6]. While basic text formatting in CSS is indeed quite straightforward, real world CSS, which is commonly used as well for positioning and layout, is notoriously difficult to master due to browser bugs, incompatibilities, and non-obvious CSS box model interactions. To complicate matters further, the CSS landscape is undergoing a period of transition, with both CSS3 and CSS frameworks like 960 adding to the layers of CSS knowledge necessary for contemporary and future practitioners.

Javascript is another key web development technology. The type of Javascript that can be covered in just a few lectures (rollovers, form data validation, browser sniffing) was reasonably close to what was needed professionally in the late 1990s. Since the “discovery” of XMLHttpRequest and the subsequent flourishing of new user interface coding and asynchronous communication with web services, Javascript coding has become simultaneously crucial to contemporary web development and significantly more complicated. Analogous to the case with CSS, this type of Javascript programming is very difficult to learn.

Another key part of learning real-world web development is the server-side environment. Potentially this is a very large topic and has its own difficulties from a teaching perspective. There are a number of different competing technologies (PHP, ASP.NET, JSP, Ruby on Rails) which all accomplish the same thing: interacting with server resources and programmatically generating HTML, CSS, and Javascript that is returned to the browser. Server-side development also has a number of substantial additional topics, such as the HTTP protocol, SQL and database-access APIs, replicating database changes across data servers, local and distributed transactions, maintaining state (via cookies, sessions, querystrings, and form elements) across requests (and, in a web farm, across processes), internationalization, architecting web infrastructures for scalability, and enterprise design patterns, which are almost impossible to fit into a AWOC course.

Digital media and information architecture and usability is another vital area of real-world web practice. This area is often under-represented in most reported web courses. Usability in general is a very large topic and will likely be covered in a computer science program’s HCI courses. Nonetheless, usability in the web context does require its own unique topics which may not make it into the typical HCI course. These topics include: the unique factors affecting web site usability, the different ways of organizing and structuring web content, the development and articulation of web conventions, designing web navigation systems, and an overview of visual design patterns. These usability topics rarely make it into an AWOC course.

2.2 Web Topics Not in CS 2008

While most of the reported web courses cover some portion of the preceding topics, the remaining topics that need to be taught in
any future web courses are almost never covered. The first of these topics is that of web frameworks and APIs. One of the key features of the contemporary web landscape is that many organizations are no longer creating their web infrastructure from scratch but are using single (or integrating multiple) already existing open-source and/or proprietary web frameworks [10, 24]. Complex content management systems such as Drupal, Joomla, DotNetNuke, or Microsoft SharePoint, blogging systems such as WordPress or Blogger, web forums such as phpBB or vBulletin, e-commerce systems such as osCommerce or Magento, and common business practice portals and services such as SugarCRM, Salesforce, or Exact are often used as the main framework for an organization’s public or private web presence.

In other words, the platform for current and future web development is expanding significantly beyond LAMP (Linux, Apache, MySQL, and PHP) or WISA (Windows, IIS, SQL Server, and ASP.NET). Contemporary web development is increasingly more about building new solutions that involve customizing or leveraging APIs and/or extension frameworks (which may themselves be built on LAMP or WISA, so knowledge of them is still required), such as the Facebook API, Sharepoint, Drupal, WordPress, Zend, JQuery, and ASP.NET MVC. The scale of something like the Facebook API usage or the number of sites built on WordPress (perhaps as many as 10% of all significant sites on the entire web [10]) is often invisible to the end user, but is something that needs to be made completely visible to our computer science students. For the students, it is important to learn about the existence of these existing frameworks because the future of web systems will increasingly lie in their use, integration, and customization.

Another important part of the web systems knowledge area that is generally missing from computer science education is that of hosting and deployment. For practical reasons, students generally learn web development using their own computer or a lab computer as if it was a web server. Ultimately, however, a web site needs to be deployed on a public web server. Students thus need to learn about the advantages and disadvantages of the main web server platforms as well as third-party hosting environments. Hosting topics such as web gardens, web farms, load balancing, data center redundancy and replication, as well as server configurations for scalability also need to be covered if a computer science graduate is going to have a realistic sense of the contemporary web world.

Finally, it is important for students to learn about web service consumption and integration, especially RESTful services (though SOAP services should be taught as well). With the broad interest in the asynchronous consumption of server data at the browser using Javascript (generally referred to as AJAX) and due to the relatively easy availability of a wide-range of RESTful services, a new style of web development known as the mashup has become increasingly common [13]. Consuming these services typically involves learning XML parsing, XPath search expressions, as well as more conceptual issues such as mediation techniques between heterogeneous data sources [5].

3. TEACHING ETHICS

One of the many breakthroughs in the teaching of computer science over the past two decades has been the relatively widespread recognition of the importance of social and professional (SPI) issues in the education of computing professionals. A recent survey of American universities found 88% of computer science programs included ethics in their computer science curricula [21]. Another survey that also included non-American universities found a full 95% of their respondents included SPI content in their computer science programs [11].

Increasingly over the past decade this concern over social and ethical issues has been dominated by a very specific analytic approach, namely, the articulation of the impacts of information and computing technology (ICT) and the ethical evaluation of those impacts. That is, students are expected to apply existing moral theories to construct guidelines or even rules to correct or prevent the wrongs caused by a particular technology. There appears to now be a standardized approach to teaching SPI. First the students are provided with at least two substantive ethical theories, generally utilitarianism and a Kantian deontology. These two forms of ethical evaluation are then used to evaluate the impacts caused by computer technology in a paradigmatic impact area such as privacy, intellectual property, security, and access to information in order to both articulate and to ethically evaluate the effect various computing technologies have had on those areas.

The appeal of this approach for computer science faculty is not hard to see. It is attractive to us because it is so algorithmic. Most computer science faculty achieved their position through their knowledge and research in traditional computer science topics, and, as a consequence, the SPI course is often not in a computer science professor’s primary knowledge area. As Grodzinsky has noted, the “many gray areas of computer ethics are often frightening … to professors who are worried about how to answer things of which they themselves are unsure.” [12]

Despite these attractions, both this algorithmic approach and the understanding of the relationship between society and technology that it is grounded upon has some real limitations and may in fact give our students an impoverished understanding of the social issues of computation. It will argue that the way we teach this course needs to move away from the preoccupation with the ethical evaluation of ICT impacts and instead emphasize the social context aspects of the Social and Professional Issues knowledge area. In particular, the teaching of this material needs to integrate the decades-old insights of researchers in the philosophy, history, and sociology of technology which emphasizes the complex interaction and co-construction between the social environment and any given technology and the resulting radical uncertainty of technological change.

3.1 What’s Wrong with Impacts?

Perhaps the first step in recognizing the shortcomings of the ethical impacts approach is to realize the central flaw in the articulate social impacts step in our SPI courses. This flaw is predicated on what seems an obvious and common-sense belief, namely, the belief that technology is simply a tool available for us to achieve our ends. This belief encourages us to examine computer technology in a means-ends manner; that is, the SPI researcher identifies and observes what affect the means is going to have on the social environment. It generally assumes that the means are by and large clear and unproblematic and will always work in the same way for all people at all times. While the
impacts approach sees ICT as a tool, it also sees it as a very special type of tool that can have large-scale impacts on society and/or the people using it. That is, while ICT, like a hammer, is just a tool, its special general-purpose nature means it has far-reaching effects outside its tool domain, akin to a hammer that changes the weather or weakens the dollar every time it strikes a nail. This approach to technology is generally called technological deterministic by those who study the history, philosophy, or sociology of technology [16]. In this approach, technological change is treated as very much the independent variable in societal change. According to this view, technological inventions—especially key ones like the printing press, the steam engine, the computer, the internet, and social networking—have transformed the world and thus new technologies need to be subjected to analysis to understand the wide-ranging transformation they have had on us and the world.

It is understandable why computer professionals find technological determinism attractive. After all, we are the people that are helping to invent some of these new technologies; it feeds our clear desire to be socially relevant [19] and to believe that we computer geeks are actually the driver of social change, and not politicians, business people, or celebrities. This view is so widespread among computer professionals that, for instance, this author’s students and fellow department members find it difficult to believe that most current historians and sociologists of technology firmly reject technological determinism as being theoretically inconsistent as well as empirically under-supported. As one recent historian has noted, sweeping accounts “about machines that shape society remain popular, but they clash with the research of most professional historians of technology” [17].

The academic field of science, technology and society (STS) studies that began in the 1960s has time and time again found that when examined carefully, most technologies rarely have had the effect that was expected and that the reason for this phenomenon is that “new technologies are shaped by social conditions, prices, traditions, popular attitudes, interest groups, class differences, and government policy” [16]. Notice the direction of agency in this quote: it is technology that is being shaped or impacted by society, not the reverse.

Most technological deterministic impact prognosticators do their work by looking at the functional capabilities of a given technology and then imagining the impact of those functions. For instance, internet search engines clearly make it easier to find knowledge; what then will the impact of increased knowledge? Household technologies make it quicker to do housework; what will be the social impact of all that spare time? Antilock disc brakes make it less likely to skid and get into accidents; what will be the social impact of fewer accidents? In all these cases—and practically any other set of prognostications and impact evaluations than begin from an unquestioned belief that the functional capabilities of a technology (i.e., the means) do what is promised (i.e., achieve their ends)—the expected social impacts ended up being wildly wrong because the prognosticators believed in a naïve technological determinism.

For instance, the introduction of household technology did not end up creating, in the words of Ruth Schwartz Cowan, less work for mother, but in fact more work because of a series of social changes that could not have been predicted if one limited one’s analysis just to the functional capabilities of the household technologies. As Cowan demonstrated [8], household technologies created more housework due to changing expectations of what constitutes cleanliness (e.g., clothes changed daily instead of weekly), new unexpected technologies enabled by the technology (e.g., wall-to-wall carpets were unknown before vacuums), and the gradual displacement of household work done by external agents (e.g., laundry services, maids, nannies) to housewives partly as a consequence of household technologies and partly due to exogenous changes in the social and economic realm.

The first step then we should take in our Social and Professional Issues course is to communicate how rarely technologies achieve their exact promise, and indeed, how many do the opposite. This so-called Revenge Effect is well-documented [23] and yet this author was unable to find it discussed at all in existing computer ethics textbooks or in published accounts of this course. As well, an equally important step we need to make in the teaching of the SPI course is to reject naïve technological determinism and help students understand the complex agency issues in the relationship between technology and society. One way to achieve this goal would be by beginning the SPI course with examples and readings in how certain vital technologies had little impact on some societies, or on how certain technologies were strongly modified and differently adapted in different cultures and countries. This more historically-nuanced (and significantly more empirically accurate) approach is what is generally called social constructivism [18]. In this approach, one looks at how technologies are researched, invented, financed, developed, adopted, marketed, and propagated within a very complex system generally referred to as society. If one carefully examines a given technology within the social system in which it is embedded, it becomes extremely difficult to maintain a belief in technological determinism. Instead one sees technologies much more strongly “impacted” by society rather than vice versa. Thus, the SPI course should integrate the historically-grounded insights of the STS research community. In other words, the SPI course should look more like a historical sociology course and a lot less like a philosophic ethics course.

3.2 The Importance of Uncertainty

The reason why revenge effects occur is due to the fact that “socio-technological transformation is a highly complex process which involves many uncertainties” [14]. While uncertainty is a key concept in many fields, within moral philosophy in general, and computer ethics in particular, it is underappreciated. This is an important problem because the substantive moral theories (such as deontology and utilitarianism) that are the bedrock of the usual computer ethics course require relatively clear and unambiguous information about effects in order to make judgments [9]. Typical problems or dilemmas for which macro-ethical approaches are applied are most often done in a context of complete knowledge (if you do action X, then Y people will be harmed, but Z people will be benefited). This is appealing for computer scientists, who often work with problems modeled by idealized abstractions for which complete knowledge is possible. The ethics of technology, by contrast, should be recognized as residing in a context of at least partial uncertainty or ambiguity. Furthermore, the degree of uncertainty is greater for emerging technologies, and the more complex the technology, the more
uncertain we are as to the developmental trajectory of a technology [18].

There are many places where uncertainty intersects with the lifecycle of a technology. For the evaluating agent \( x \) (the professor or the student or the developer or the journalist) examining emerging technology \( y \) at time \( t \), the agent must have knowledge of the development trajectory in order to morally evaluate it. Unfortunately, as we have already seen, we very often cannot know the actual development trajectory and if we hypothesize one based purely on its functional capabilities, we more often than not will be woefully wrong. Thus, in reality, we need to recognize that technology \( y \) has multiple trajectories \((o_1, o_2, o_3, \ldots)\) and that it might be more important to attempt to understand which trajectories are more likely (by unpacking the web of interests and agents) than applying a prescriptive ethical judgment on a single trajectory.

4. CONCLUSION
It has been this paper’s contention that both the web and social issues courses need to be modernized in terms of their curricula. As well, both of these courses can play an important role in the overall development of a computing student. Adams has noted that one of the key limitations with computer education and the many specialized knowledge competencies it tries to engender is that “students struggle to put all the pieces together” [3]. Humphrey similarly noted this lacuna not just in undergraduate but in graduate computer science students as well; even graduate students “usually have little or no experiences in designing, implementing, and evaluating, large-scale software systems for complex, dynamic, and heterogeneous environments” [13]. That is, it is not enough to teach each technology and concept in isolation; “students must understand how these technologies relate to each other” [3]. This theme has also been raised by other authors in their reflections on what needs to be improved in the way computing is taught [20]. These two sometimes-neglected knowledge areas can provide the students with a taste of that needed integration and complexity. They provide an experience that White and Weinberg have called for: integrative topics that needed integration and complexity. They provide an experience knowledge areas can provide the students with a taste of that way computing is taught [20]. These two sometimes-neglected to each other” [3]. This theme has also been raised by other is isolation; “students must understand how these technologies relate is, it is not enough to teach each technology and concept in implementation, and evaluating, large-scale software systems for students “usually have little or no experiences in designing, implementing, and evaluating, large-scale software systems for complex, dynamic, and heterogeneous environments” [13]. That complex, dynamic, and heterogeneous environments” [13]. That is, it is not enough to teach each technology and concept in isolation; “students must understand how these technologies relate to each other” [3]. This theme has also been raised by other authors in their reflections on what needs to be improved in the way computing is taught [20]. These two sometimes-neglected knowledge areas can provide the students with a taste of that needed integration and complexity. They provide an experience that White and Weinberg have called for: integrative topics that needed integration and complexity. They provide an experience knowledge areas can provide the students with a taste of that way computing is taught [20]. These two sometimes-neglected to each other” [3]. This theme has also been raised by other is isolation; “students must understand how these technologies relate

5. REFERENCES