Scaling Computer Science Education to Education on Scaling in Computer Science

Mehran Sahami

Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043

sahami@google.com

1   Introduction

In the past few years, two interacting trends in industrial computing have helped change the landscape of how software is constructed and delivered. First, the ubiquity of networking has helped to forge a more distributed architecture for software, including both multi-tier and cluster-based computing. Additionally, more and more applications are being delivered over the network rather than running as stand-alone desktop applications. The confluence of these trends has helped to create a new generation of applications which are created as scalable, composable services rather than single monolithic applications. Training the next generation of Computer Scientists for success in such an environment requires both redesigning and retooling computing curriculum to address these trends.

Traditionally, Computer Science education has served the dual purpose of providing students with the basic problem solving skills and algorithmic understanding to address *how* to build programs, while also providing some (mostly "siloed") coverage of particular topics (compilers, operating systems, artificial intelligence, etc.) regarding *what* is being built. Often due to resource considerations (or simply existing momentum), computing is relegated to something an individual does on one's own computer. This model is quickly becoming antiquated as many applications now involve processing at multiple levels (which may or may not be resident on the same machine), deal with data volumes that cannot reside on a single local disk, or simply make use of services that are remotely provided (and developed).

Examples of this trend are plentiful and growing. Some of the more well-known include the Google search engine (where data volumes and processing power far exceed the capacity of a single machine), Salesforce.com which has taken a leading role in popularizing the on-demand model of software delivery over the network, and Microsoft (often considered a stalwart of the desktop) which has recently touted the importance of the software services model. The time has come for the academy to embrace this trend.

2   Industrial Needs

Given the industrial trend toward software services, the challenge for many employers in the software industry is to find enough engineers with sufficient awareness and understanding of the additional complexities of such a software model. These challenges include issues such as the robustness of software to component failures, the security vulnerabilities of publicly available services in a networked environment, and sufficient testing of services to provide adequate reliability and scalability. These issues, while of intrinsic importance, often are not adequately integrated into computing curricula.

Moreover, as applications move beyond a single desktop machine and move more into the realm of distributed systems, students require a more explicit understanding of competing large-scale application architectures. For example, students could greatly benefit from instruction in and comparison of common N-tier architectures, models for distributed applications (where an application is composed of multiple processes running on different machines), and composite applications (where an application is composed of multiple distributed components). This instruction should not simply be relegated to an *Operating Systems* course, but would be served well by being integrated in a number of courses giving students experience with each model in the most appropriate context.

Indeed, there is much benefit to be gained simply by changing the wording around concepts that students already learn. In a traditional program, rather than simply reading data from a file, a program can interact with a "data service" to obtain the data that it will process. The data service is simply an interface over some repository, whether it simply be a file, which students may see as early as CS1, or a DBMS which may be introduced in a later class. The earlier students begin to think about the multiple logical components that applications may be comprised of, the earlier they can understand the issues of software development within a *Web of Services* and the challenges and opportunities that are consequently afforded.

3   Transforming Computing Education

Transforming computing education to help bring the software services model more to the fore provides several challenges. Not surprisingly, the most significant challenge is a retooling of computing curricula to provide a much

more integrated approach to Computer Science instruction from the beginning. While students in CS1 and CS2 would still need to learn basic programming skills, it is easy to envision projects based on building larger-scale applications where individual assignments are based on constructing different service components of the overall system. Such a model also lends itself much more directly to unit testing of components, identifying the robustness of components in the face of dependent service failures, and working on group projects in which multiple students must coordinate to build all the component services of a larger application.

In classes later in the curriculum, which tend to be focused on specific topics, the topical discussion can take place in a broader context. For example, courses on databases can much more seamlessly discuss data services, where critical real-world issues such as replication, federation, and optimization for scale can be discussed along with more traditional topics such as functional dependencies and entity-relation modeling. Courses on human-computer interaction can implement and contrast different user interfaces to the *same* underlying application logic if students' understanding of display/interface services are easily abstracted from other program logic. And such abstractions are easily motivated by applications that students may already commonly use—such as email, which may be easily described as having a data service (email repository), which is modified by application components (SMTP and potentially other application servers), and are accessed through display services (thick clients, such as Microsoft Outlook, or thin clients delivered through a web browser, such as Google Gmail). Indeed, it would be expected that students would develop components in multiple different categories of services at various points throughout their undergraduate program.

Part of having students more immersed in a software services model also requires much more attention to and instruction in the design and development of appropriate APIs for services. Curriculum in this area can involve courses aimed explicitly at examining the advantages and disadvantages of many new and existing APIs in order to focus on the underlying trade-offs that are made at a more abstract level. Such analysis also provides an opportunity to make issues discussed in theory classes (computational complexity, relational theory, orderings, mappings, etc.) more grounded in actual software design trade-offs.

Furthermore, this provides the opportunity in large-scale project courses (such as capstone classes) to have students work on building much more realistically scalable services dealing with large data repositories, large-scale computation, or both. And such projects can be more easily composed with existing real-world software systems (e.g., building a complex bioinformatics user-interface that actually interacts with the PubMed database, or designing data analysis algorithms that can be run on web-scale data by running as parallelized services on a cluster of machines).

While it is impossible to enumerate all the curricular issues that would arise, several themes would likely permeate the curriculum, including the discussion of various software services architectures, more integration of topics (via component services) in each course, greater emphasis on testing and security, more analysis of design trade-offs and the creation of good APIs, and a focus on system (not just *program*) decomposability for scale and robustness.

4 Strategies and Stakeholders

As with many grand plans for overhauling an existing system, perhaps the greatest inhibitor is simply the existing momentum in the way things are currently done. Moreover, by taking a more integrative approach toward computing education, it will require much more interaction between faculty (from potentially very different areas) than is traditionally expected in course development. Thus, rather than require each school to separately develop its own integrated curriculum (which is unlikely to lead to widespread conversion), the opportunity exists to have a group of committed educators with a variety of backgrounds work together to create one or more sample curricula (including class handouts and textbooks, assignments, coding infrastructure, etc.) that could be more easily adopted by other schools that may not have the resources to produce such a set of materials on their own.

Clearly, academics are not the only stakeholders in such an endeavor. Members of industry who are looking for particular skills in the next generation of software engineers need to be deeply involved in helping to shape curricular topics of industrial interest and providing real-world examples of the sorts of issues students are likely to face in working with software services.

There is also the issue of computing infrastructure, for if students are to develop large-scale computing applications, they need to have appropriate hardware and software infrastructure. While some schools may have the resources to make clusters of machines available to students, there will always be schools without adequate computing capabilities or student projects that will potentially stretch what resources may exist. This situation creates the potential for both governmental and industrial support of computing infrastructure (for example, large, shared clusters of machines) that can be simultaneously leveraged by a number of schools for both teaching and research purposes.

While the challenges in revamping Computer Science education to meet the trend in software services are great, the risks of not adequately training the next generation of software engineers to understand issues of scale, security, and robustness in an ever-more networked computing environment are even greater.