Vinton G. Cerf

DOI:10.1145/2534706.2534708

Software at Scale

I have been thinking about the implications of the increasing number of programmable devices and appliances that surround us. For one thing, they all require software to

function. Who will write all that software? Of course, much of the software will simply be copies of the same code, so it is not as if we will need to write distinct software for each of the 50 billion devices. On the other hand, some of these mobile devices will be general-purpose platforms, such as those carried by many of the world's population. These support hundreds of thousands of special-purpose applications.

Thirty-five years ago, a similar kind of software proliferation was triggered by the development of VisiCalc by Dan Bricklin and Robert Frankston. This spreadsheet program transformed the Apple II desktop computer from a hobbyist platform to a business tool platform. Literally hundreds of thousands, if not millions, of people learned to write spreadsheet programs. One did not have to be a professional programmer or have a degree in computer science or electrical engineering to code a spreadsheet. Moreover, some of these spreadsheet programs were and are large and extremely complex. VisiCalc was overtaken by more powerful spreadsheet programming platforms but the principal point regarding the large population of spreadsheet programmers is still valid. Almost anyone could learn to write simple spreadsheets.

Mobile applications fall into a similar category although they typically require somewhat more sophistication even for relatively simple cases. Programming environments take a good deal of the complexity away. There are literally dozens of programming tools, software development kits, and mobile simulators

available to aid in the development of a mobile application. The same may be said for Web-based programs that rely on browsers to interpret and execute application code or to interact with servers in the cloud. Without trivializing the effort required to program mobile, tablet, and laptop applications, it seems fair to observe that many of these programs have been and are being produced by self-taught programmers or at least those who may not have taken formal degrees in software-related disciplines.

Much of the software that underlies these conveniently programmed applications is deeply dependent on experienced, skilled, and often formally educated software developers. The so-called “warehouse computing” of the cloud relies on some of the most sophisticated operating system, application platform, and networking software ever developed. The operating systems supporting mobile devices, laptops, or desktops are similarly the products of sophisticated programming by extremely talented and experienced software developers.

I have written in the past about the notion of loosely coupled systems and the important role that standards play in their operation. The same principle is at work here. The underlying operating systems, software development kits, and networking facilities rely on standard protocols and application programming interfaces to hide complexity and allow programmers to focus on the application and not so much on all the infrastructure that must work for their applications to function. It has been observed that it is easy to write software

that is hard to use and hard to write software that is easy to use. The same notion applies to the software development environments enabling a remarkable range of users to produce their own software. The most enabling software development environments represent considerable programming effort.

Interestingly, this line of reasoning leads me to believe that everyone would benefit from exposure to some form of programming if only to experience the kind of thinking that goes into making a programmable device do something. This is one reason ACM is so active in promoting computer science as a course equal in stature to physics, chemistry, biology, and mathematics in secondary schools. It should be permitted to fulfill STEM requirements in middle and high school curricula. The intent is not to turn everyone into professional programmers. Rather, it is to create familiarity with the concepts of programming so the software-based tools and platforms that surround us can be applied at need. Interested readers are referred to code.org for ideas along these lines.

I am reminded of a prediction made in the 1930s that by the 1950s, every citizen living in the U.S. would have to become a telephone operator to handle the rapidly growing use of the telephone. In one sense, the prediction was correct. The invention of direct distance dialing made all of us operators!

The increased use of programmable devices in daily living creates demand for software skills ranging from those needed for relatively simple, ad hoc applications to extremely sophisticated and large-scale systems design and implementation. Many of the latter will demand in-depth and formal training and perhaps even certification for applications involving risk to life and limb. ACM is right to pursue its strong advocacy for computer science in school curricula.

Vinton G. Cerf, ACM PRESIDENT