Public Understanding of Software Engineering

Anthony Finkelstein
Department of Computer Science
City University
Northampton Square
London EC1V OHB {acwf@cs.city.ac.uk}

Summary

This article outlines the issues presented by the public understanding of software engineering. By public understanding I mean the appreciation or knowledge of a discipline by people in general, other than those with a specialised education or training in the area. The article concludes with a programme of action that addresses public understanding of software engineering and can form the basis for a debate on this topic

Motivation

Though we have a well developed subject, emerging professional standards and institutions and growing debate and discussion on educational issues, virtually no attention has been paid to the public understanding of software engineering. I believe this is an important omission which should be rectified. Below I provide some grounds for this assertion.

Funding for research and education in software engineering is to a large extent dependent on the public purse. Though the level of science and technology funding is conditioned by larger social and political considerations the share devoted to software engineering as against other disciplines is dependent on the public understanding of software engineering, the general perception of the value of our work and the societal importance of the research we conduct. This has broad implications as software engineering provides a justification for basic research in computer science and mathematics. In all these respects the understanding of software engineering in the broader science and technology community is particularly pertinent.

Attracting able students to our discipline depends on the general level of understanding among appropriately educated young people and, critically, among their peers, parents and teachers.

Remuneration and indeed employment opportunities for software engineers in business and in public employ depends on the perception of the value of the skills and capacities of software engineers, and hence on a general understanding of software engineering, in business and commerce.

The price that the public is willing to pay for software is closely linked to public understanding of software engineering. In other words the price that software engineering commands in the market place is conditional upon the perceived contribution of software engineering to the value of a product, which is in turn heavily dependent on the level of understanding of the development process and the skills and resources it requires. This becomes particularly important where software engineering significantly impacts the cost structure of software.

Software engineers directly interact with a broad range of people during the construction of software systems (gathering requirements and so on). For the most part these are people (office workers, military personnel, medical staff, and so on) whose work will be affected by the deployment of software or who otherwise have a stake in the development process. The nature of the working relationship will be directly impacted by the understanding these people have of the role of the software engineer, their knowledge and professional status. Indirectly, of course, software engineers interact with a broad public through the software they produce.

Software engineers build systems which are placed in the public domain and whose effective operation is a matter of general public concern. Setting aside the question of safety critical software, the administration of health care, public finance, law and order, social welfare and defence involve software intensive systems. The operation of these systems opens the work of software engineers to public scrutiny. To what extent is this informed scrutiny?

The professionalisation of software engineering, that is the establishment of rigorous standards equivalent to those to which other engineering specialisations such as civil engineering are subject, requires public recognition and possibly legislative control over matters such as qualification and professional accreditation.

A number of important matters concerning software are currently the subject of public debate (privacy, safety, regulatory control, computer misuse, security and so on). Software engineers are in a particular position to comment and contribute to the debate on these issues. The significance which will be attached to this contribution is dependent upon the public view of their "locus standi". More ephemerally the general esteem in which software engineers are held and their standing in society depends on a public understanding and respect for what they do.

Lastly, natural scientists and some engineers have appreciated the overwhelming importance of public understanding of science and technology, and have made very considerable efforts to develop it. As science and technology have become increasingly specialised and, it is argued, difficult for lay access, so the strength of the case for devoting effort to this matter has increased. There is a general point to be made about the civilising and cultural values inherent in the natural sciences which could tenuously be extended to software engineering.

What should a public understanding of software engineering consist of?

So, what exactly do we want the public to understand about software engineering? The following are general matters which must be understood before any specific technical issues of concern in software engineering can be understood. I state them baldly though, clearly, this is not the form in which they are best made accessible

- Developing software with some assurance that it behaves in the manner you wish it to behave is difficult.

- Unlike mechanical systems, very small mistakes can have major consequences; mistakes can lie dormant for long periods since it is impossible to test everything.

- The task of developing software is complicated by scale: software systems can be very large, they are often produced by large teams whose work must be coordinated.

- Software is not just computer programs: most of the work of software engineers is concerned with requirements, design, test and other documentation.

- Software engineers have developed tools, techniques and methods which can help in the process of developing software.

- Sound software engineering builds upon computer science and engineering practice in other disciplines.

- Software engineers are in short supply and there are many more software challenges than software engineers available to respond to them; most software is built without using known already well-established best practice.

- There are professional and educational standards for software engineers which can provide some assurance of competence.

- Over and above issues of competence, there are codes of conduct for professional software engineers which indicate the standards of responsible behaviour you can expect from them; the professional bodies to which they belong can be expected to take steps to enforce them.

- Software engineering is intellectually demanding and presents many interesting challenges both in relation to technology and its mathematical underpinnings; however, many of the problems of software engineering stem from the difficulty of establishing what is needed and are to a great extent human problems, not technological problems.

Anybody with a familiarity with software engineering education will know that though these issues can be easily expressed, they are far from readily absorbed. It is difficult to provide the colour and immediacy which will render them comprehensible and memorable to a general audience. I am aware that the list given above is an individual one. Perhaps an informed debate on what a public understanding of software engineering should consist of is a prerequisite for the actions discussed below.

What can we do?

Clearly there are some steps which we ought to take as individuals, institutions and as a profession.

Many of the conventional mechanisms for achieving an improved public understanding such as public lectures and contributions to the fora established for the public dissemination of scientific and technological information can and should be used. More specifically we should not allow the public face of computing to be dominated by the "sexy" technologies in place of the more "workaday" but, I would contend, more significant areas of software engineering.

Issues that concern software engineering which arise in the media ought to receive a considered, and preferably coordinated, professional response which recognises our responsibility for educating the public. Opportunities for "popular" journalism should be seized and treated with seriousness and responsibility.

We ought to be commenting on and contributing to general curriculum development in science, technology, mathematics and transferable personal skills at all levels in the educational process.

In recruiting for education establishments and in contacts with schools we should attach greater importance to communicating an understanding of software engineering to our mutual benefit over and above the narrower merits of advocating particular courses or institutions.

In our daily lives we meet people outside the software engineering profession - friends, family and others. Strange though it may seem these people form an important public for information about software engineering. We should not abrogate our responsibility for explaining what we do to them. If we are unable to explain our subject and its concerns to those in our immediate circle, and who are presumably reasonably receptive, the chances of us being able to communicate more widely are slim.

This clearly extends to people we meet in the work setting. The responsibility to educate them about software engineering and our professional responsibilities, skills and capacities borders on an ethical imperative. There is a tendency, more or less marked in many organisations to treat each encounter with "a member of the public" in their role as user, domain expert or whatever as a one-off. The focus becomes to make the particular encounter as economical, in terms of time and effort, as possible. Of course, it may be true that the relation between this particular member of the public and the organisation or individual software engineer will be limited but, the same member of the public may

participate in many projects and with many other software engineers in the course of their work. A local economy of the form "we will not explain what we do and why we are here because it will take too long and is not worth it for these relatively unimportant participants" may prove false when aggregated across software engineering as a whole.

For those of us who work in universities it is particularly important that we take on the task of developing the understanding of colleagues in the other science and technology fields (physicists, chemists, mechanical engineers, and so on). This is in some ways easier than educating the general public as we can expect a higher degree of sophistication and familiarity with issues of this form. It is at the same time more difficult as they tend to ask tough questions about methodology, the status of knowledge in software engineering and our educational practice! This article does not discuss the thorny question of the relation between software engineering and academic computer science in general, for an excellent discussion of this see Berry (1992).

Among the most important "public" audiences to which we speak are politicians. For the most part relations between software engineers and the political arena are coloured not so much by the politicians' ignorance of of science, technology and engineering (they are truly representative of the general public in this regard) but rather a profound ignorance of politics among software engineers. If we are to communicate effectively on matters of professional concern, a prerequisite is that we develop a sophisticated understanding of the political environment uncoloured by partisan prejudice or fashionable cynicism. Understanding and respect go hand-in-hand and can only be achieved if they are mutual. Politicians are very busy and have many competing demands on their attention; thus it is helpful to establish that an understanding of software engineering confers mutual benefits and that we can, as a profession, contribute to troublesome public policy issues such as military systems procurement.

Obviously, given the importance that I attach to a greater public understanding of software engineering, I regard it as critical that students are trained how to communicate technical information in a range of media and to a broad range of audiences. Such training should form a component of the transferrable personal skills element of courses at all levels.

Our efforts to help the public to understand software engineering should be informed by an accurate picture of the current extent of that understanding and of the means by which communication can be most effectively achieved. In this regard we can probably pick up on the work that has been done in other areas of science and technology. However, it may be necessary to undertake a programme of surveys and monitoring to support this.

Resources for public understanding (lecture and display material, examples, interactive demonstrations) are expensive to create and require much effort. Often they are used once (for a public lecture, open day or visit) and discarded. Feedback is not incorporated and the effort is not shared more widely. We need to develop mechanisms and opportunities for sharing resources and experience.

Once there is a broad understanding of software engineering among the general public it is necessary to ensure that this understanding remains in line with current practice. This demands continuing effort.

Conclusion

The agenda for software engineering is very full and the problems and challenges of public understanding may seem a distraction. I hope the arguments presented above have persuaded you otherwise. If we fail to educate the public about software engineering the public will develop a picture of our subject and profession without us. They will draw on casual encounters, advertisers, the popular press and other professions and their view will be incoherent and possibly inaccurate or

damaging. This view will be difficult to change: a little attention paid now may avert this, to our long-term benefit.

References

Berry, D. (1992); Academic Legitimacy of the Software Engineering Discipline; Software Engineering Institute Technical ReportCMU/SEI-92-TR-34.