Chapter 16
Software Engineering Ethics Education:
Incorporating Critical Pedagogy into Student Outreach Projects

Gada Kadoda
University of Khartoum, Sudan

ABSTRACT

The difficulties inherent in the nature of software as an intangible object pose problems for specifying its needs, predicting overall behavior or impact on users, and therefore on defining the ethical questions that are involved in software development. Whereas software engineering drew from older engineering disciplines for process and practice development, culminating in the IEEE/ACM Professional Code in 1999, the topic of Software Engineering Ethics is entwined with Computer Science, and developments in Computer and Information Ethics. Contemporary issues in engineering ethics such as globalization have raised questions for software engineers about computer crime, civil liberties, open access, digital divide, etc. Similarly, computer-related ethics is becoming increasingly important for engineering ethics because of the dominance of computers in modern engineering practice. This is not to say that software engineers should consider everything, but the diversity of ethical issues presents a challenge to the approach of accumulating resources that many ethicists maintain can be overcome by developing critical thinking skills as part of technical training courses. This chapter explores critical pedagogies in the context of student outreach activities such as service learning projects and considers their potential in broadening software engineering ethics education. The practical emphasis in critical pedagogy can allow students to link specific software design decisions and ethical positions, which can perhaps transform both student and teacher into persons more curious about their individual contribution to the public good and more conscious of their agency to change the conditions around them. After all, they share with everyone else a basic human desire to survive and flourish.

DOI: 10.4018/978-1-4666-8130-9.ch016

INTRODUCTION

As a discipline, software engineering grew out of computer science in response to the "software crisis" of the 1960s that was characterized by the growth in complexity or criticality of computer applications and the problems of software projects going over budget and time. The need arose to identify processes and methods that could "engineer" software in similar ways as material objects like buildings or cars. Pioneering works of Dijkstra(1968) and Parnas (1972) on programming among others laid out the foundations of development methodologies and early models such as Waterfall and Spiral that borrow from engineering and project management typical practices as requirements, design, construction, risk management, etc. The debates on whether software can be "engineered" are mainly concerned with the difficulties in matching the preciseness of measurements that are found in traditional engineering work (used to describe structures, electronic or mechanical devices). In turn, this drove research on areas such as software metrics and effort estimation that are concerned with defining qualities and quantities for measuring software and predicting cost and time of software development projects (Fenton & Pfleeger, 1997). These difficulties, inherent in the nature of software as an intangible object, also pose similar problems for specifying its needs, predicting overall behavior or impact on users, and therefore on defining the ethical questions that are involved in software development. Although the position of software engineering as an engineering discipline is still controversial (Shaw, 1990; McConnell, 1999; Parr, 2013) and the problems of software projects going over budget and time persist, research on approaches for more rigorous and disciplined practices is still evolving (Schmidt, 2013). These debates, however, are not the focus of this chapter, but the origins of software engineering in computer science and its "transition" into an engineering discipline that are relevant to the development of its professional obligations and ethical codes, as well as teaching approaches.

The fact that software engineering matured in computer science departments rather than in engineering schools, some researchers argue, led to an emphasis on moral or legal abuses committed with a computer in its approach to ethics. However, ethical considerations in software engineering have been evolving over the past two decades from focus on customer and employer to look at societal implications of computer systems. Although it took a century for ethical codes for the medical profession and decades for engineering to consider the social context, the rapid pace of technological advancement and their ubiquitous nature bring new ethical issues more frequently into the lexicon of computing ethicists. Contemporary issues in general engineering ethics such as globalization have raised questions for software engineers about computer crime, civil liberties, open access, digital divide, etc. Computer-related ethics is also becoming increasingly important for engineering ethics because of the dominance of computers in modern engineering practice. In the early 1990s, a different emphasis within computer ethics was advocated by Donald Gotterbarn (1991) who believed that computer ethics should be seen as a professional ethics devoted to the development and advancement of standards of good practice and codes of conduct for computing professionals. He headed the joint task force of the IEEE and ACM that created the code of ethics for software engineers in the late 1990s. The code lays out 8 principal obligations of software engineers to society, client, employer, colleagues, and the profession in the processes they follow and judgments they make to develop and maintain their products (Gotterbarn, 1997).

In class, software engineering ethics, often overlooked, highlights issues of confidentiality, competence, intellectual property, and computer misuse, and introduces the ACM/IEEE Code of