Chapter 7.8
Trusting Computers Through Trusting Humans:
Software Verification in a Safety-Critical Information System

Alison Adam
University of Salford, UK

Paul Spedding
University of Salford, UK

ABSTRACT

This article considers the question of how we may trust automatically generated program code. The code walkthroughs and inspections of software engineering mimic the ways that mathematicians go about assuring themselves that a mathematical proof is true. Mathematicians have difficulty accepting a computer generated proof because they cannot go through the social processes of trusting its construction. Similarly, those involved in accepting a proof of a computer system or computer generated code cannot go through their traditional processes of trust. The process of software verification is bound up in software quality assurance procedures, which are themselves subject to commercial pressures. Quality standards, including military standards, have procedures for human trust designed into them. An action research case study of an avionics system within a military aircraft company illustrates these points, where the software quality assurance (SQA) procedures were incommensurable with the use of automatically generated code.

INTRODUCTION

They have computers, and they may have other weapons of mass destruction. Janet Reno, former US Attorney General

In this article our aim is to develop a theoretical framework with which to analyse a case study where one of the authors was involved, acting as an action researcher in the quality as-

surance procedures of a safety-critical system. This involved the production of software for aeroplane flight systems. An interesting tension arose between the automatically generated code of the software system (i.e., 'auto-code'—produced automatically by a computer, using CASE [Computer Aided Software Engineering] tools from a high level design) and the requirement of the quality assurance process which had built into it the requirement for human understanding and trust of the code produced.

The developers of the system in the case study designed it around auto-code—computer generated software, free from 'human' error, although not proved correct in the mathematical sense, and cheaper and quicker to produce than traditional program code. They looked to means of verifying the correctness of their system through standard software quality assurance (SQA) procedures. However, ultimately, they were unable to bring themselves to reconcile their verification procedures with automatically generated code. Some of the reason for this was that trust in human verification was built into (or inscribed into [Akrich, 1992]) the standards and quality assurance procedures which they were obliged to follow in building the system. Despite their formally couched descriptions, the standards and verification procedures were completely reliant on human verification at every step. However these 'human trust' procedures were incompatible with the automated production of software in ways we show below. The end result was not failure in the traditional sense but a failure to resolve incommensurable procedures; one set relying on human trust, one set on computer trust.

Our research question is therefore: How may we understand what happens when software designers are asked to trust the design of a system, based on automatically generated program code, when the SQA procedures and military standards to which they must adhere demand walkthroughs and code inspections which are impossible to achieve with auto-code?

The theoretical framework we use to form our analysis of the case study is drawn from the links we make between the social nature of mathematical proof, the need to achieve trust in system verification, the ways in which we achieve trust in the online world, the methods of software engineering, and within that, the software quality movement and the related highly influential domain of military standards.

In the following section we briefly outline the social nature of mathematical proof. The next section discusses the debate over system verification which encapsulates many of the ideas of mathematical proof and how such proofs can be trusted by other mathematicians. The article proceeds to consider 'computer mediated' trust, briefly detailing how trust has been reified and represented in computer systems to date, mainly in relation to the commercial interests of e-commerce and information security. Trust is particularly pertinent in the world of safety-critical systems, where failure is not just inconvenient and financially damaging, although commercial pressures are still evident here, but where lives can be lost. The model of trust criticised by e-commerce critics is more similar to the type of trust we describe in relation to safety-critical systems, than one might, at first, expect. Understandably, we would like to put faith in a system which has been mathematically proved to be correct. However computer generated proofs, proofs about correctness of computer software, and automatically generated code are not necessarily understandable or amenable to inspection by people, even by experts. The question then arises of whether we can bring ourselves to trust computer generated proofs or code, when even a competent mathematician, logician,or expert programmer cannot readily understand them.

Following this, we describe the evolution of software development standards and the SQA movement. We argue that the development of quality assurance discourse involves processes of designing human ways of trusting mathematical evidence into standardisation and SQA. Military