I recently conducted an e-mail survey among the Development Managers’ Discussion Group in Microsoft at Redmond, asking for reports on interesting ways that assertions are now being used in Microsoft programming practice. In this talk, I will summarise the results of the survey, and speculate on ways in which assertions can contribute further advantage in speeding the development and maintenance of mass-market software, and the improvement of its quality. Although my examples draw on my experience in Microsoft, I expect that my experience generalises to other software Companies, and I am sure that many readers and developers of Open Source software are equally familiar with the benefits of assertions.

Assertions have long been the topic of my academic research into the theory of programming; they are now a wide-spread feature of Microsoft coding and testing practice. In a modern release of Windows, around one percent of the lines of code are assertions, and in a few products like Office, this figure rises to about ten percent. This suggests that there is considerable scope for improvement of software engineering standards, simply by transferring best practice from one product team to another.

Assertions play many related roles in Microsoft programming practice. Their primary purpose is to instrument code with test probes that will detect errors as close as possible to their place of occurrence. They are also used for program documentation, to assist in the further development of high-volume legacy code for delivery in later releases. They are beginning to be used to guide optimising compilation, and to suppress spurious warnings from a program analysis tool. Assertion failure is used to classify multiple occurrences of the same defect, and so ensure that it is cleared only once. Finally, assertions are now increasingly shipped to customers, to reduce the risk of crashes on the customer’s machine.

My early research into programming theory was motivated by the hope that assertions would be used to specify the criterion of total correctness of a program, and to prove that the program implements its specification, without risk of error. Ironically, these are two purposes for which they are almost never used.