CHAPTER 13

Strategies for Testing LINT

> Don't blame the Compiler.
>
> —David A. Spuler: *C++ and C Debugging, Testing, and Code Reliability*

IN THE PREVIOUS CHAPTERS WE have encountered here and there hints for testing individual functions. Without meaningful tests to ensure the quality of our package, all of our work would be for naught, for on what else are we to base our confidence in the reliability of our functions? Therefore, we are now going to give our full attention to this important topic, and to this end we ask two questions that every software developer should ask:

1. How can we be certain that our software functions behave according to their specifications, which in our case means first of all that they are mathematically correct?
2. How can we achieve stability and reliability in the functioning of our software?

Although these two questions are closely related, they are actually concerned with two different problem areas. A function can be mathematically incorrect, for example if the underlying algorithm has been incorrectly implemented, yet it can reliably and stably reproduce this error and consistently give the same false output for a given input. On the other hand, functions that apparently return correct results can be plagued by other sorts of errors, for example an overflow of the length of a vector or the use of incorrectly initialized variables, leading to undefined behavior that remains undetected due to favorable (or should we rather say unfavorable?) test conditions.

We thus must be concerned with both of these aspects and institute development and test methods that can provide us sufficient trust in both the correctness and reliability of our programs. There are numerous publications that discuss the significance and consequences of these wide-ranging requirements for the entire software development process and delve deeply into the issues of software quality. Considered attention to this topic has found expression not least in the international trend to institute the ISO 9000 standard in software

production. In this regard one no longer speaks merely of “testing” or “quality assurance,” but instead one hears talk of “quality management” or “total quality management,” which in part are simply the result of effective marketing, but which nonetheless cast the issue in the proper light, namely, to consider the process of software creation in its multifaceted entirety and thereby improve it. The frequently employed expression “software engineering” cannot blind us to the fact that this process, as it relates to predictability and precision, as a rule can scarcely compete with the classical discipline of engineering.

The comparison may be characterized aptly by the following joke: A mechanical engineer, an electrical engineer, and a software engineer have decided to take an automobile trip together. They seat themselves in the car, but it refuses to start. The mechanical engineer says at once, “The problem is with the motor. The injection nozzle is clogged.” “Nonsense,” retorts the electrical engineer. “The electronics are to blame. The ignition system has certainly failed.” Whereupon the software engineer makes the following suggestion: “Let’s all get out of the car and climb back in. Perhaps then it will start.”

Without pursuing the further conversations and adventures of the three intrepid engineers, let us proceed to consider some of the options that were implemented in the creation and testing of the FLINT/C package. Above all, the following references were consulted, which do not exhaust the reader with abstract considerations and guidelines but get down to concrete assistance in solving concrete problems, without in the process losing sight of the big picture.[1] Each of these books contains numerous references to further important literature on this topic:

- [Dene] is a standard work that deals with the entire process of software development. The book contains many methodological pointers based on the practical experience of the author as well as many clear and useful examples. The theme of testing is attacked again and again in connection with the various phases of programming and system integration, where the conceptual and methodological fundamentals are discussed together with the practical point of view, all in conjunction with a thoroughly worked out example project.
- [Harb] contains a complete description of the programming language C and the C standard library, and it gives many valuable pointers and comments on the prescriptions of the ISO standard. This is an indispensable reference work to be consulted at every turn.
- [Hatt] goes into great detail on the creation of security-critical software systems in C. Typical experience and sources of error are demonstrated

[1] The titles named here represent the author’s personal, subjective selection. There are many other books and publications that could as well have been listed here but that have been omitted for lack of space and time.