Extended Abstract

The attention of practitioners and proponents of the use of formal methods has for a long time been focused on relatively small and complex applications. The hardware domain lends itself well to this and it has therefore been the target of some of the most significant applications of formal methods. The software applications that have typically been considered are for safety-critical systems though there have also been some experiments in the specification of parts of large subsystems, such as CICS.

This restricted focus is understandable while formal methods are evolving and practical considerations limit the size of the system that can be specified and verified. However, it has the unfortunate effect of leaving most of the work in the software industry outside its purview. Whether formal methods were intended for use with such systems or not, large programs are being developed and maintained, both system software and a variety of applications that are business-critical and mission-critical.

To set a context for the discussion, consider a single large software system containing around 5M lines of program written using one or more languages (e.g. Cobol, Pl/1, C, C++ or Java) to operate with a particular data-base system (e.g. IMS, DB2, or Oracle) and some middle-ware (e.g. CICS or Tuxedo), all executing on one or more platforms. The front-end executes on a different platform and is written in a different language (e.g. PowerBuilder or Java). The program code has been written at different times, from 1—25 years ago.

Such a software system must be maintained through the following kinds of activities:

- **Remedial**: correction of errors discovered during use;
- **Adaptive**: making changes to cater to changes in the operating environment;
- **Enhancing**: adding new features or capabilities; and
- **Improving**: making the software more robust and easier to maintain.

Studies show that correction of errors forms a smaller proportion (around 25%) of the maintenance work than (b), (c) and (d). Thus, even if formal methods are used to improve the process of software development, there is urgent need to examine where formal methods can help to make the improvements and adaptation of software more error-free. It does not take long to conclude that a large software system cannot be considered to be ‘complete’ when it is first commissioned. In fact, because of the frequency with which requirements change, it may never stay ‘complete’ for very long.

It is estimated that the cost of software maintenance amounts to 90% of the life-cycle cost of a software system. Improvements in software development methods can help to reduce the need for, and therefore the cost of, making remedial improvements (i.e. bug fixing) but similar methods are urgently needed for the remaining 75% of the maintenance work.

In this talk, I will examine some areas of software maintenance that can benefit from the use of formal methods. The choice of the method will vary with the nature of the problem to be solved. The most direct approach is therefore to have a toolbox of techniques to be used as needed. However, the techniques will often make use of similar methods of analysis so it is possible to consider an integrated toolset for maintenance.

We will first consider ways of understanding the functions and operations of the software system in terms of models, internal information and control flow, input and output, and relating these to the business rules that are being implemented. We will then examine how the results of this analysis can be used to improve different steps of the maintenance process, from making changes and enhancements to testing the system.