

Software review of a self-calibrating instrument

R. Barker and G. Parkin, NPL*

Abstract

We describe a case study performed jointly with NMI to validate a self-calibrating instrument, a commercial LCR meter. The aim of the validation was to gain assurance of the traceability of the instrument measurement. The case study was divided between NMI reviewing the hardware (described elsewhere by G. Rietveld, NMI) and NPL reviewing the software.

The software validation was based on the Software Support for Metrology Best Practice Guide No.1 *Validation of Software in Measurement Systems*; this guide would recommend a wide range of validation techniques because of the complexity of the flow of the measurement data through the instrument software. Indeed, there are different flows of measurement data depending on whether the data is being used for instrument calibration or for a user measurement. Nevertheless, the software validation proceeded using techniques that only required the use of general-purpose tools.

The hardware and software review teams worked together at many stages: there were requests and suggestions passed back and forth, and there were joint meetings at which we discussed our findings and so came up with areas for further work. There would be requests for clarification to see if an obscure point in the software (for example) could be resolved by reference to the hardware. Similarly, there were suggestions from the hardware side that some feature was particularly crucial (or easy to get wrong) and so the implementation of that feature in the software should be investigated. In particular, after an initial review, both teams drew up a list of questions that needed to be answered by the review. These questions addressed issues in the operation of the instrument that needed to be answered to be assured of correct, traceable measurement outputs.

The main approach to software validation was code review together with a limited amount of static analysis and re-implementation of one aspect of the functionality. We were supplied with the code and some documentation; but we had no access to compilers for the code, so it was not possible to independently compile and test the code. Even the code review and static analysis was hampered by lack of knowledge of the compiler options – the code contains many pre-processor switches, and (without knowledge of the compiler options) we could not know what code was actually included in the actual software for the instrument.

The approach taken to analyse the software was determined by the information available. In view of the lack of documentation of the software specification and structure, the only way to analyse the software was to reconstruct the purpose and structure of the software from the code itself – a process known as reverse engineering. Given the objective of the software validation, the code review and the reverse engineering focused on the part of the code for calibration and measurement. The reverse engineering was guided by input from the analysis

of the instrument hardware and by our general knowledge of calibration and measurement software. The software review team built up a picture of the software design and functionality, which was tested and refined by discussions with the hardware review team. We are happy that we have a reasonably accurate view of the intended functionality but any false assumptions we have made will lead to misleading findings from the software validation.

The static analysis was performed using a general-purpose tool (splint) that was used to identify variables and functions that were not used or not initialized/declared. To get the static analysis tool on the instrument code involved lots of work selecting options to the tool and finding appropriate setting for pre-processor options in the code. The code review also used general-purpose tool to navigate the code: a text editor (vim) combined with a tool to index objects in the different source files (ctags). One question from the hardware team involved the uncertainty calculations in the user manual. The software team was able to re-implement the calculations in a different language and found that there was an error in the equations given in the manual.

The conclusions of the case study were that, although there were no actual errors in the software processing identified by the reviewers, the (lack of) quality of the software and the apparent deficiencies in the software development process did not give us sufficient assurance in the instrument's measurement processes. The lessons from the case study were that general-purpose software tools are useful in review and analysis of instrument software and that analysis of the software can aid the validation of the instrument hardware. Software tools can improve the quality of tools during the software development process especially if the software is designed to take advantage of the tools.