MEASUREMENT STANDARDS LABORATORY OF NEW ZEALAND

SOFTWARE DEVELOPMENT GUIDELINES

A supplement to the MSL Quality Manual

May 21, 2020

Text in this colour is awaiting approval by the Quality Council.

Text in this colour is deprecated and awaiting approval to be deleted.

Measurement Quality Council
Measurement Standards Laboratory of New Zealand

Contents

1 Introduction

MSL relies heavily on software to control experiments and for acquiring, processing and presenting data. Most technical sections develop bespoke solutions that suit their needs.

The quality system imposes requirements on software used to carry out test and calibration work, but does not set expectations about software quality. To satisfy the quality system, software must demonstrably meet the requirements of the task, which will be verified during validation of a technical procedure. The quality system also requires the integrity of both software and measurement data to be strictly controlled.

These are rather modest requirements for assets that represent a very significant investment of MSL's time to develop and maintain. Perhaps unsurprisingly, problems arising from insufficient maintenance of software are rife within the Laboratory.

Much MSL software consists of 'legacy' programs that are used occasionally. Specification of requirements, documentation and software testing are often lacking, making maintenance difficult. Some sections have invested heavily in types of software that are inherently difficult to maintain at the level required for calibration work: spreadsheets being a good example. Spreadsheets are convenient for quick work, but notorious for hiding errors and renowned for allowing inadvertent changes to creep in.

Another problem is the platform required to run legacy software. IT systems evolve much faster than MSL would wish, so older hardware, and older software tools (e.g., compliers) need to be operated after they would have retired elsewhere. In many cases, these older systems are retained because not enough is known to allow migration.

This document has been prepared to present techniques that can improve the reliability and maintainability of software. While recognising that MSL has substantially different requirements to most commercial and even scientific organisations, and that a variety of software development systems are in use, there are general practices that can improve software quality. There are also useful tools available.

This guide recommends a structured approach to software development. The objective is not merely to produce an executable application, it is also to provide: a clear statement of the requirements, an explanation of the design to meet those requirements and a suite of tests that verifies the software's performance against requirements. It is desirable that any part of a software project can be examined and understood by someone other than the author. This facilitates re-use of software and maintenance.

We recommend that software development be undertaken in active consultation among members of the technical section, or wider MSL. As in most quality system activities, the quality of one individual's work will benefit from a review by someone else. Professional developers use regular reviews to both detect errors and improve the quality of their work. These reviews occur throughout the development cycle, not just at the end. MSL should do this too.

2 Style and Structure

"If you don't know where you're going, any road will take you there."
– George Harrison

Software should be developed with end-users and future owners in mind – it should also be designed to be tested and maintained easily. This requires consideration of how software is written.

2.1 Style

When writing, consideration should always be given to stylistic conventions: be consistent in your use of a simple, clear, style. A worthy goal is to write software that is *self-explanatory*. This can be achieved by appropriate use of: names, data structures, programming structures and idioms, layout and comments.

Names Careful naming is important. Names for variables, functions and classes, etc, should be

- easily remembered
- concise
- descriptive (for readers)
- consistent (with names of similar entities)
- easy to chose

Languages such as Visual BASIC, Python, C++, etc, and some end-user tools (e.g., MathCAD) allow considerable freedom to choose names.

Data structures All programming languages support a few basic data structures (e.g., arrays), some allow *ad hoc* structures to be defined for particular uses (e.g., object-oriented structures). Good design of data structures is important, because it can make instructions that manipulate data easier to read, understand and maintain.

Programming structures and idioms Authors in languages like English, etc, use syntactical rules and common idioms to write statements that can be easily understood. The same applies to programming. Each language has its own syntax, features and quirks, but there will always be good, widely-recognised, ways of describing common tasks, which can be regarded as 'best-practice'.

All programming languages have a few basic control structures (e.g., `FOR-NEXT` loops, `IF-THEN` statements, etc). While the logic of these structures is simple and universal, the way they are deployed is usually language-dependent. There are subtleties in how the language works that make certain idioms preferable. Using recognised language idioms both improves code quality and makes it easier to read and maintain.

Layout The way that code appears on the page (screen) is important. Often source code will work fine no matter how it is laid out, but consistent layout makes a huge difference to readability. Once again, languages tend to have different conventions. Python even incorporates layout in the syntax of its block statements, in a deliberate effort to make code easier to read.

Comments Software comments are something of a necessary evil: they should be used sparingly. It is better to revise code, considering appropriate naming, layout and idioms, because doing so may make a comment unnecessary.

While changes are made to code over time, comments are often overlooked. So, comments can drift with respect to the corresponding code base. Later, it may become unclear just how accurate a comment is. Writing code that does not need many comments is a better strategy. Only details that cannot be inferred from a clear coding style should be added as (preferably) short local comments.

3 Maintainable code

"Maintainable code is more important than clever code"
– Guido van Rossum

4 Next section

"No problem is too small or too trivial if we can really do something about it."
– Richard Feynman

A References

[1] Brian W. Kernighan and Rob Pike, *The Practice of Programming*, (Addison-Wesley Professional Computer Series, 1999).

[2] Steve Maguire, *Writing Solid Code*, (Mircosoft Press, 1993).

[3] Brian W. Kernighan and P.J. Plauger, *The Elements of Programming Style*, (McGraw-Hill, 1974).

[4] Diomidis Spinellis, *Code reading: the open source perspective*, (Addison-Wesley, 2003); (additional information available on the books website)

[5] Diomidis Spinellis, *Code quality: the open source perspective*, (Addison-Wesley, 2006); (additional information available on the books website)

[6] Richard Blaustein, *Eliminating Spreadsheet Risks: A Guide for Internal Auditors to Regain Control and Limit Exposure*, Internal Auditor Magazine, 2009; online

[7] Greg Wilson, D A Aruliah, *et al*, *Best Practices for Scientific Computing*, PLOS Biology, 12(1), 2014; DOI: 10.1371/journal.pbio.1001745

[8] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, *Good enough practices in scientific computing*, PLoS Comput Biol, **13**(6), June 2017, DOI: 10.1371/journal.pcbi.1005510.

[9] Marian Petre and Greg Wilson, *Code Review For and By Scientists*, in: Katz D, editor., Proc. Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE 2), 2014; ArXiv:1407.5648, September 7, 2014; online

[10] Working Towards Sustainable Software for Science: Practice and Experiences, workshop presentations available online