

AN ANALYSIS OF SOFTWARE QUALITY AND MAINTAINABILITY
METRICS WITH AN APPLICATION TO A LONGITUDINAL
STUDY OF THE LINUX KERNEL

LAWRENCE GRAY THOMAS

November 18, 2009

The vast majority of any non-trivial program's life cycle will be spent in the maintenance phase. The degree to which the software is readily maintainable has enormous impact on the cost of the maintenance, and can also impact the overall quality of the software.

Coupling between software modules has been shown to be deleterious to both software quality and maintainability. Common coupling (two or more modules sharing a global variable, or GV) is a particularly strong (and therefore undesirable) form of coupling. We examined 83 production (stable) versions of the Linux kernel. Previous studies have determined that the size of the Linux kernel (as measured in lines of code) grows *linearly* and the number of instances of common coupling grows *exponentially* with respect to version number. We found that the size of the Linux kernel and the number of instances of common coupling both grow *linearly* with respect to release date, because Linux kernels are released at irregular intervals.

We also measured correlations between common coupling and Oman's maintainability index as well as two of Halstead's Software Science metrics (Volume and Effort), and McCabe's Cyclomatic Complexity metric. The general validity of these metrics has long been questioned. We performed a longitudinal study on three distinct series of the Linux kernel, in an attempt to see if these metrics were valid within multiple consecutive versions of the same program. Of all the metrics we examined, only Lines of Code maintained a significant linear correlation with the number of instances of common coupling in all cases.

Rather than analyzing the code in only the `/kernel/` subdirectory as most previous studies have, we built complete, executable kernels, and analyzed all of the source code used to build each kernel. Doing so required us to develop an algorithm to select consistent kernel configurations across all versions of the kernel we analyzed.

We used a variety of open-source software tools (and one proprietary CASE tool), and built several specialized tools to link the existing tools, resulting in an end-to-end solution for the analysis of the various software metrics we gathered. Newer tools cannot be used to build the older versions of the kernel, and some of the code had to be modified in order to successfully build the kernel. We carefully documented these changes to assist future researchers.