On the Nonmaintainability of Open-Source Software

Stephen R. Schach
Dept. of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN
srs@vuse.vanderbilt.edu

A. Jefferson Offutt
Dept. of Information and Software Engineering, George Mason University, Fairfax, VA
ofut@ise.gmu.edu

A major strength of open-source software is that the source code is open to scrutiny by anyone who chooses to examine it. Accordingly, it is reasonable to assume that the quality of open-source software will be higher than that of closed-source software. After all, closed-source software is examined by only a limited number of individuals, all of whom are paid to do so. It seems equally reasonable to conclude that open-source software is superior to closed-source software in other ways as well, including maintainability. Again, the argument is that the scrutiny by a large number of volunteers leads to a better product.

On the other hand, the fact that open-source software is a product of an amorphous group of individuals, rather than a hierarchical development team, means that there is no single person who is in charge of an open-source software product. As a result, modifications can be made to an individual module that could have a deleterious effect on the maintainability of the open-source software product as a whole. An example of this is the introduction of common coupling into an open-source software product.

The *coupling* between two units of a software product is a measure of the degree of interaction between those units [1–3] and, hence, of the dependency between the units. Modules X and Y are common (global) coupled if X and Y share references to the same global variable. It has been shown [4] that coupling is related to fault-proneness. Coupling has not yet been explicitly shown to be related to maintainability. On the other hand, there is as yet no precise definition of maintainability, and therefore there are no generally accepted metrics for maintainability. Nevertheless, if a module is fault-prone then it will have to undergo repeated maintenance, and the resulting frequent changes are likely to compromise its maintainability. Furthermore, these frequent changes will not always be restricted to the fault-prone module itself; it is not uncommon to have to modify more than one module to fix a single fault. Thus, the fault-proneness of one module can adversely affect the maintainability of a number of other modules. In other words, it is easy to believe that common coupling can have a deleterious effect on maintainability.

Common coupling has another disadvantage, namely, the number of instances of common coupling between a module M and the rest of the product can increase even if no change whatsoever is made to M. This phenomenon is termed *clandestine common coupling* [5].

We downloaded 365 versions of Linux. For each version in turn, we examined the 17 kernel modules and counted the number of lines of code in each module. Then we counted the number of instances of common (global) coupling between each of the kernel modules and all the other modules in that version of Linux. We obtained two primary results.

First, we found a linear dependency between lines of code and version number that is significant at the 99.99% level; 95.1% of that dependency can be explained by the three effects:

version number, module, and their interaction [6]. In other words, the number of lines of code in each kernel module increases linearly with version number, and no additional variables are needed to explain this increase; it is an inherent feature of successive versions of Linux.

This result should come as no surprise. After all, successive versions of Linux provide additional functionality. One would expect this increased functionality to be achieved by both inserting additional code into existing modules and adding new modules. That the size of the kernel grows only linearly could be an indication that the kernel modules are well designed; only a small amount of additional code needs to be inserted to interface the kernel with modified existing modules and new modules that provide the additional functionality.

Second, we found that the number of instances of common coupling grows exponentially with version number [6]. This result, too, is significant at the 99.99% level. In this case, 94.6% of the observed growth can be explained by the three effects: version number, module, and their interaction. That is, the exponential growth in common coupling is again an inherent feature of successive versions of Linux.

We previously related common coupling to fault-proneness. Consequently, combining our two results reveals a worrying trend. Even though the number of lines of code in the kernel grows only linearly, the number of instances of common coupling between each kernel module and all the other Linux modules grows exponentially. To see why this is disturbing, suppose that every statement added to a kernel module were a call to another module. Because the number of lines of code grows only linearly, the number of new instances of coupling induced by these calls even in this extreme case can grow only linearly. However, as previously explained, common coupling can increase even when a module does not change. This is the mechanism by which the common coupling increases exponentially, even though the number of lines of code increases only linearly.

Common coupling was introduced into Linux from the very beginning, and the nature of common coupling led to an exponential growth in the number of instances in successive versions of Linux. There is no reason to suppose that this growth will be slowed in the future unless Linux is completely restructured with a bare minimum of common coupling. It could be argued that this restructuring of a huge product will mean that the further development of Linux will have to be postponed for many months until the restructuring has been completed. On the other hand, if this restructuring is not performed, it seems inevitable that, at some future date, the dependencies between modules induced by common coupling will render Linux extremely hard to maintain. It will then be exceedingly hard to change one part of Linux without inducing a regression fault (an apparently unrelated fault) elsewhere in the product. The only alternative will then be to restructure what by that time will be an even larger software product.

Linux is surely the best-known example of open-source software. The operating system has been featured in numerous television programs and in a multitude of articles in newspapers and magazines. The widespread publicity given to Linux may be a consequence of "Microsoft bashing." But whatever the reason, the name "Linux" is fast becoming a household word. As a consequence of the visibility of Linux, it is reasonable to believe that the open-source operating system has had more than its fair share of scrutiny by open-source software enthusiasts. Despite this attention, it appears that the maintainability of Linux is low, and will get worse in the future.

If Linux, the "poster child" for the open-source software movement, is essentially nonmaintainable, then it seems likely that many other open-source software products are also nonmaintainable. Unfortunately, at this time we have no data to back up our claim. However, we have built a CASE tool to automate the counting of instances of common coupling. We are

about to apply this tool to a variety of open-source software products to determine whether indeed there is an unacceptably large incidence of common coupling in almost all open-source software, with the resulting deleterious influence on maintainability.

ACKNOWLEDGMENT

The work of Stephen R. Schach and A. Jefferson Offutt was supported in part by the National Science Foundation under grant number CCR–0097056.

REFERENCES

[1] Stevens, W. P, Myers, G.J., and Constantine, L. L. Structured Design, IBM Systems Journal, 1974, 13 (2), 115–139.

[2] Schach, S. R. Object-Oriented and Classical Software Engineering, 5th Edition. WCB/McGraw-Hill, Boston, 2002, 181–189, 426.

[3] Offutt, J., Harrold, M. J., and Kolte, P. A software metric system for module coupling. Journal of Systems and Software, 1993, 20 (3), 295–308.

[4] Briand, L. C., Daly, J., Porter, V., and Wüst, J. A comprehensive empirical validation of design measures for object-oriented systems, in Proceedings of the 5th International Software Metrics Symposium, (Bethesda, MD, November 1998), 246-257.

[5] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, A. J. Clandestine common coupling. Computer Science Technical Report 01–02, Vanderbilt University, Nashville, TN, June 2001.

[6] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, A. J. Maintainability of the Linux kernel. IEE Proceedings—Software. To appear.