

Categorization of Common Coupling in Kernel Based Software

Liguo Yu
Assistant Professor
Tennessee Tech University, Box 5101
Cookeville, TN 38505
(001) 931-372-6484
yul@csc.tntech.edu

Srini Ramaswamy
Associate Professor
Tennessee Tech University, Box 5101
Cookeville, TN 38505
(001) 931-372-3691
srini@csc.tntech.edu

ABSTRACT

Common coupling is an important factor that needs to be considered in software design. It affects software dependency via the definition-use relationship of global variables. Common coupling can arise in all types of software; here we focus on issues specific to kernel-based software. In a previous paper, we described a categorization of common coupling and used it to study the maintainability of the Linux operating system. In this paper, we present a detailed description of this categorization, prove its completeness, and suggest further applications. We hope that, by this approach, we can make it easier for others to use our categorization to measure the maintainability of other kernel-based software.

Categories and Subject Descriptors

D.2.2.d [Modules and interfaces], D.2.7.g [Maintainability], D.2.8 [Metrics/ Measurement], D.2.10.h [Quality analysis and evaluations].

General Terms

Design.

Keywords

Modularity, dependencies, common coupling, definition-use analysis, kernel-based software.

1. INTRODUCTION

Two modules are called *common coupled* if they access the same global variable. In a previous study [1], we presented a categorization of common coupling based on definitions and uses of shared variables in kernel and nonkernel modules. An instance of a variable is either a *definition* (a change to the value of the variable) or a *use* (a utilization of the value of the variable). Changes to a definition during maintenance will affect uses, whereas changes to a use cannot affect other uses. In that previous

work, we split common coupling into five categories, as described in Section 5.2. Category-4 and category-5 are more undesirable than category-1, category-2, and category-3 from the viewpoint of maintainability. Using this categorization, we studied common coupling in the Linux kernel and showed that Linux has an excess of instances of undesirable category-4 and category-5 of global variables, which will make it extremely difficult to maintain Linux in the future.

In this paper, we describe our categorization in more detail and prove its completeness. In section 2, we review software dependency. Section 3 describes the definition-use relationship of variables. Section 4 reviews kernel based-software. Section 5 describes in more detail the definitions of our new categorization of common coupling. Section 6 proves the completeness of our categorization. Our conclusions appear in Section 7.

2. SOFTWARE DEPENDENCY

Coupling measures the dependency between the modules [2]. Coupling is necessary for software design. A good software system should have high cohesion within each module and low coupling between modules. Coupling between modules strengthens the dependencies of one module on others and increases the possibility that changes in one module may affect other modules. This relation has two effects on software faults. First, one fault in a module may transfer to another module via coupling. Second, an inappropriate change in one module may cause a fault in other modules.

Coupling is related to maintenance in two ways. First, coupling is the degree of dependency between modules, so a high level of coupling means a high degree of dependency. Strong dependency is an indication of high complexity. It is generally accepted that a high degree of complexity means a high level of maintenance difficulty.

As redefined by Offutt et al. [3], modules A and B are common coupled if A and B share references to the same global variable. Common coupling has always been considered as a strong coupling and been used with caution. Common coupling possesses an unfortunate property that the number of instances of common coupling between a module M and the other modules can change drastically, even if module M itself never changes [4]. In kernel-based software this makes the kernel more difficult to maintain. For example, the value of a global variable that is used

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

in a kernel module may be changed in a nonkernel module, which may affect the use of this global variable in the kernel.

3. DEFINITION-USE

Each occurrence of a variable in source code is either a definition of the variable or use of the variable. A *definition* of a variable x is a statement that assigns a value to x . The most common forms of definition are assignments to x such as $x = 1$. The *use* of a variable x is a statement that references the value of x , such as $y = x+9$. From the creation of a variable to the destruction of that variable, each time the variable is invoked, it is either assigned a new value or its present value is used.

4. KERNEL-BASED SOFTWARE

In almost all operating systems, there is a set of kernel modules that are architecture- and driver-independent and are included in all installations. The kernel is also the core part of kernel–nonkernel structured software such as database management systems [5]. One example is the Valentina database kernel [6], the heart and brain of all Valentina products. In almost all video game systems, there is a kernel, which provides the interface between the various pieces of hardware, allowing the video game programmers to write code using common software libraries and tools [7]. In our study, software that is comprised of a kernel together with optional nonkernel modules is referred to as kernel-based software. In addition, many (but not all) software product lines [8] may include certain core assets that are part of every product generated from that product line. These products also have a kernel–nonkernel structure.

5. CATEGORIZATION OF COMMON COUPLING

5.1 Traditional Definition-Use Analysis of Common Coupling

As described in Section 3, in definition-use analysis, each instance of a variable is labeled as either a definition or a use of that variable. Evaluating the potential effects of common coupling requires an analysis of the definition-use relationship of the variables. Generally, it is considered that a product is more difficult to maintain if there are more definitions than uses. Suppose global variable gv is found in two products:

- (1) Product 1 has 10 definitions of gv , and 10 uses of gv .
- (2) Product 2 has 1 definition of gv , and 10 uses of gv .

It is usually considered that product 1 is more difficult to maintain than product 2, because in product 1, there are 10 places that change the value of gv and that may affect other modules, whereas in product 2, there is only one such place.

Now, consider a special situation of case (1):

- (3) Two modules $M1$ and $M2$ in product 1 access gv , there are 10 definitions in $M1$ and 10 uses in $M2$.

If we are only interested in the maintainability of $M1$, in situation (3), $M1$ will be considered easy to maintain, because there is no use of gv in $M1$, and no definition can affect $M1$. This shows that, if we consider the maintenance of a small number of modules,

like the kernel in kernel-based software, we need a more precise categorization.

5.2 Definition-Use Analysis of Common Coupling in Kernel-Based Software

Because kernel modules are included in all installations, we are more interested in the maintainability of the kernel modules. Definitions and uses of global variables have different effects on maintenance. This becomes more complicated for kernel-based software. In our previous study [1], we created a refinement of the traditional notion of common coupling. We divided common coupling in kernel-based software into five categories, as described below. Because uses in nonkernel modules cannot affect kernel modules, we do not include them in our following analysis.

Category 1: A global variable defined in kernel modules but not used in any kernel modules.

Category 2: A global variable defined in one kernel module and used in one or more kernel modules, but not defined in any nonkernel modules.

Category 3: A global variable defined in more than one kernel module, and used in one or more kernel modules, but not defined in any nonkernel modules.

Category 4: A global variable defined in one or more nonkernel modules and used in one or more kernel modules, but not defined in any kernel modules.

Category 5: A global variable defined in one or more nonkernel modules and defined and used in one or more kernel modules.

Table 1 shows the definitions and uses of the five categories of global variable in both kernel and nonkernel modules. They are represented with one of the three notations: required, forbidden, or optional. For example, a category-1 global variable is required to be defined in one or more kernel modules but is forbidden to be used in a kernel module. In addition to these requirements, a category-1 global variable is optional to be defined and/or used in nonkernel modules. As proved in section 6, this five-way categorization of common coupling includes all the situations. Here we suggest the procedures to categorize global variables.

- (1) Identify the global variables that appear in kernel and ignore all the other global variables that appear only in nonkernel but not in kernel.
- (2) For each identified global variable, find the modules it appears.
- (3) For each instance of a global variable, determine whether it is a definition or use.
- (4) Finally, for every global variable, we have the following information: number of kernel modules has definition, number of kernel modules has use, number of nonkernel modules has definition.
- (5) Based on this information, each global variable is classified to one of the five categories according to Table 1.

Table 1. Definitions and uses of the five categories of global variable

	Kernel modules		Nonkernel modules	
	Definition	Use	Definition	Use
Category 1	Required in one or more modules	Forbidden	Optional	Optional
Category 2	Required in exactly one module	Required in one or more modules	Forbidden	Optional
Category 3	Required in more than one module	Required in one or more modules	Forbidden	Optional
Category 4	Forbidden	Required in one or more modules	Required in one or more modules	Optional
Category 5	Required in one or more modules	Required in one or more modules	Required in one or more modules	Optional

5.2.1 Category-1 Global Variables

A category-1 global variable is defined in kernel modules but has no kernel uses. A category-1 global variable could be defined in nonkernel modules, but this is not important from the viewpoint of kernel maintainability. Because there is no use in kernel module, these definitions cannot affect kernel module. Category-1 global variables can probably be considered the least undesirable with respect to maintainability of the kernel. There is no use of global variable in kernel modules, so a change to a category-1 variable cannot affect another kernel module via direct definition-use relation of this variable, which means that category-1 global variables have minimal impact on the maintainability of kernels.

5.2.2 Category-2 Global Variables

A category-2 global variable is defined in one kernel module but not defined in any nonkernel module. It is used in one or more kernel modules. As with category-1, a modification to a category-2 global variable in a nonkernel module cannot affect a kernel module via direct definition-use relation of this variable because there are no definitions of category-2 global variables in nonkernel modules. However, a change to the one kernel module that defines the variable can affect the kernel module that uses it.

5.2.3 Category-3 Global Variables

A category-3 global variable is defined in more than one kernel module but not defined in any nonkernel module, and is also used in one or more kernel modules. As with category-2, a modification to a category-3 global variable in a nonkernel module cannot affect a kernel module via direct definition-use relation of this variable because there are no definitions of category-3 global variables in nonkernel modules. However, a change to more than one kernel module that defines the variable can affect the kernel module that uses it.

5.2.4 Category-4 Global Variables

A category-4 global variable is defined in one or more nonkernel modules, and used in one or more kernel modules but not defined in any kernel modules. A kernel module that uses a category-4 global variable is not vulnerable to modifications in a kernel module, because this no definition in a kernel module. But, a

category-4 global variable is vulnerable to modifications to that global variable in a nonkernel module that defines the variable.

5.2.5 Category-5 Global Variables

A category-5 global variable is defined in one or more nonkernel modules, defined in one or more kernel modules, and used in one or more kernel modules. A kernel module that contains a category-5 global variable is vulnerable to modifications to both a kernel module and a nonkernel module in which that global variable is defined.

For symmetry, it may be better to combine category-2 with category-3 and have four categories. However, our categorization is based not only on the theoretical analysis of the definition-use relation of kernel-based software, but also on the case studies of real-world projects. Our study of the Linux kernel [1] showed that category-2 occurs more frequently than other categories in practice. Therefore, in our categorization we separated category-2 global variables from category 3. In applications of this categorization, the decision whether to combine or separate category 2 and category 3 depends on the product and the research objective.

Based on our categorization, we can see that category-1 is the least undesirable global variable, because changes to any module cannot affect the kernel via a direct definition-use relationship. Category 2 and category 3 are both preferable to category-4 and category-5. The former satisfy the principle of “separation of concerns,” that is, changes to nonkernel modules cannot affect kernel modules. Category-4 and category-5 global variables are the most undesirable ones. They should be limit in use.

6. COMPLETENESS OF THE CATEGORIZATION

Our five-way categorization of common coupling in kernel-based software is based on the safety of kernel modules. Here we prove the completeness of our categorization.

An instance of global variable gv (say) can be either a definition or a use of that variable. With respect to the kernel modules, gv can be categorized in one of three ways:

A1. gv is defined but not used in kernel modules;

A2. gv is used but not defined in kernel modules; or

A3. gv is both defined and used in kernel modules.

(Because we are concerned with the maintainability of the kernel, we ignore global variables that are neither defined nor used in the kernel.)

With respect to the nonkernel modules, gv can be categorized in one of four ways:

B1. gv is defined but not used in nonkernel modules;

B2. gv is used but not defined in nonkernel modules;

B3. gv is both defined and used in nonkernel modules; or

B4. gv is neither defined nor used in nonkernel modules.

Considering gv with respect to both kernel and nonkernel modules, this yields 12 possible combinations:

A1–B1: This combination cannot occur. It makes no sense to have a global variable that is never used.

A1–B2: This combination corresponds to our Category 1, a global variable that is defined in kernel modules but is not used in any kernel modules.

A1–B3: This combination also corresponds to our Category 1, a global variable that is defined in kernel modules but is not used in any kernel modules. Even though there are definitions in nonkernel modules, these definitions cannot affect the kernel modules, because there are no uses in the kernel.

A1–B4: Again, this combination cannot occur. It makes no sense to have a global variable that is never used.

A2–B1: This combination corresponds to our Category-4, a global variable that is defined in nonkernel modules and used in kernel modules.

A2–B2: This combination cannot occur. It makes no sense to have a global variable that is used but never defined.

A2–B3: This combination also corresponds to our Category-4, a global variable that is defined in nonkernel modules and used in kernel modules. The uses of the global variable in nonkernel modules do not affect the kernel modules via the direct definition-use relation.

A2–B4: Again, this combination cannot occur. It makes no sense to have a global variable that is used but never defined.

A3–B1: This combination corresponds to our Category-5, a global variable defined in both kernel and nonkernel modules and used in kernel modules.

A3–B2: This combination corresponds to our Category 2 or Category 3 (depending on the number of kernel modules defining

the variable), a global variable is defined and used in kernel modules, but there are no definitions in nonkernel modules.

A3–B3: Again, this combination corresponds to our Category 5.

A3–B4: Again, this combination corresponds to our Category 2 or Category 3.

From this case analysis, we can see that our five-way categorization of common coupling in kernel-based software is complete and includes all possible combinations.

7. CONCLUSIONS

In this paper, we extend our previous work on the categorization of common coupling and prove its completeness. We hope that this approach will make it easier for others to apply it to software development or research. In a previous paper [1], we have applied our categorization to the Linux operating system. The material in this paper can be applied to any kernel-based software products, such as other operating systems, database systems, and games systems.

8. ACKNOWLEDGMENTS

This work was sponsored in part by the National Science Foundation under grant number CCR–0097056.

9. REFERENCES

- [1] Yu, L., Schach, S. R., Chen, K., Offutt, J., “Categorization of Common Coupling and its Application to the Maintainability of the Linux Kernel”, IEEE Transactions on Software Engineering, 2004, 30(10): 694–706.
- [2] Stevens, W.P., Myers, G.J. and Constantine, L.L., “Structured Design,” IBM Systems Journal, 1974, 13(2): 38–54.
- [3] Offutt, J., Harrold, M. J. and Kolte, P., “A Software Metric System for Module Coupling,” Journal of System and Software, 1993, 20(3): 295–308.
- [4] Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z., and Offutt, A.J., “Quality Impacts of Clandestine Common Coupling,” Software Quality Journal, 2003, 11(7): 211–218.
- [5] Härden, T, “New Approaches to Object Processing in Engineering Databases,” Proceedings of the International Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September 1986: 217–217.
- [6] Paradigmasoft, (2004): <http://www.paradigmasoft.com/kernel.html>
- [7] Howstuffworks, (2004): <http://entertainment.howstuffworks.com/video-game3.htm>
- [8] Clements, P. and Northrup, L., Software Product Lines: Practices and Patterns, Addison-Wesley, New York, 2002.