On Information Hiding

May 29, 2014.

José Francisco García Juliá

jfgj1@hotmail.es

Information hiding is not programming hiding. It is the hiding of changeable information into programming modules.

Key words: Information hiding, changeable information.

Information hiding (IH) [1-6] is the only principle that guarantees the construction of successful, cost reduced, maintainable and reusable (software) systems [7]. The first goal of the software engineering (SE) [8] is also the construction of successful, cost reduced, maintainable and reusable software systems. Hence, IH is the first basic principle of the SE.

The basic idea of the IH principle is to hide the information that is likely to change. One needs to structure the system so that all changeable or arbitrary or very detailed or common-use [5] information is hidden into modules. The "solid" or lasting information is specified as interfaces that communicate the modules.

But the first question is how to do this. How to apply the IH principle. We see, following [5], that the requirements of the system are the information. The information of each requirement is evaluated as changeable, arbitrary, very detailed, common-use or lasting. The four first information categories are hidden into modules, the last one are the interfaces that communicate the modules. The lasting information not used in an interface is also hidden into stable modules.

The information of each requirement is specified in the corresponding native language (English, Spanish, French, German, etc.). From this information, we underline only the parts considered lasting. We use underline because it can be handmade when we do the interviews to the customer. If the not underlined information cannot be converted into a small module, where a small module is a piece of software small enough to be developed by one person in a limited period of time generally of one to three months, then the requirement would be divided into two or more requirements and the process (of underlining) is repeated, and so on.

The underlined information parts are the (lasting) interfaces that connect the modules, that is, the use and define lists of the modules of the Modula language of Wirth [9, 7], the input and output of the Input-Process-Output (IPO) modules of the IBM HIPO (Hierarchical IPO) chart diagrams (which are equivalents to the Warnier-Orr diagrams), or the controller and view of the Controller-Model-View (CMV) of Reenskaug (who defined it in 1979 as Model-View-Controller or MVC). The not underlined information parts are the (changeable, arbitrary, very detailed, common-use or non-interface-stable) blocks of the modules, that is, the functions, procedures, (sub)routines, processes, models, methods and data structures of the programming modules.

Therefore, from the requirements of the system, which are a separation of concerns, we obtain a structured division (tree structure) of the information of the system under consideration. This tree-structured information is enough to guarantee the construction of successful, cost reduced, maintainable and reusable (software) systems. Next, we convert each information module into a programming module using Modula [9, 7] (or its successors), or another high-level programming language (including Fortran [6]), or using several languages and connecting all together with a module interconnecting language (MIL) as MIL75 [10] (or its successors), which was the first MIL.

By design decisions, we may put in an information module, information or parts of information of one or more requirements. Hence, after the corresponding conversion, the produced programming module may be the typical sequential program or, in its place, a collection of software pieces (functions, procedures, (sub)routines, processes, models, methods and data structures), not executed sequentially but by a call.

Also, between the information modules and the corresponding programming modules, we may use a function specification language [2], a language of axioms that permit to prove certain theorems about those specifications. Then, the tree structure information modules are converted into the so-called module guide [4], which easily is converted finally into more reliable programming modules.

As the goal of the requirements phase is to establish and specify precisely what the software must do without describing how to do it, the information modules are the "what" modules and the programming modules are the "how" modules, recovering the classic WHAT-HOW (what to do and how to do it) approach for software design and construction.

What is essential is to design the software using the "what to do", written in the native language prose, and the IH principle, and next to code it with a programming language, the "how to do it".

In summary, IH is not programming hiding. It is the hiding of changeable information into programming modules.

[1] D. L. Parnas, "Information Distributions Aspects of Design Methodology", Proceedings of IFIP Congress 71, 1971, Booklet TA-3, pp. 26-30.

[2] D. L. Parnas, "A Technique for Software Module Specification with Examples", Communications of the ACM, vol. 15, no. 5, pp. 330-336, 1972.

[3] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", Communications of the ACM, vol. 15, no. 12, pp. 1053-1058, 1972.

[4] D. L. Parnas, P. C. Clements and D. M. Weiss, "The Modular Structure of Complex Systems", Proceedings of 7th International Conference on Software Engineering, March 1984, 408-417.

[5] S. D. Hester, D. L. Parnas and D. F. Utter, “Using Documentation as a Software Design Medium”, The Bell System Technical Journal, Vol. 60, No. 8, October 1981.

[6] D. L. Parnas, “The Secret History of Information Hiding”, pp. 399-409, M. Broy, E. Denert (Eds.): Software Pioneers, Springer-Verlag, Berlin, Heidelberg 2002.

[7] José Francisco García Juliá, “Information Hiding and Modula”, viXra: 1306.0213 [Data Structures and Algorithms].
http://vixra.org/abs/1306.0213

[8] D. L. Parnas, “Software Engineering Programs Are Not Computer Science Programs”, November/December 1999 IEEE Software.

[9] N. Wirth, “Modula: A language for modular multiprogramming”, Institut für Informatik, ETH Zürich, 1976.

[10] F. DeRemer, H. Kron, “Programming-in-the large versus programming-in-the-small”, Proceedings of the International Conference on Reliable Software. Los Angeles, California. ACM, pp. 114-121, April 1975.