Revealing the Secrets of David Parnas

H. Conrad Cunningham

Department of Computer and Information Science
University of Mississippi

March 7, 2014

Those of us in the fast-changing field of computing often dismiss anything written more than five years ago as obsolete. Yet several decades-old papers by David L. Parnas [1, 4, 5, 6, 7, 8] are as timely as those published in recent issues of the top journals. Parnas articulates the timeless software design concepts known as information hiding and abstract interfaces.

Most programmers would describe a module as a unit of code such as a subroutine or class. Parnas focuses on the programmers rather than the programs. He defines a *module* as "a work assignment given to a programmer or group of programmers" as a part of a larger software development project [7]. His goals are to enable programmers to develop each module independently, change one module without affecting other modules, and comprehend the overall system by examining one module at a time [5].

Programmers often design a software system by breaking the required processing into steps and making each step a module. Instead, Parnas uses information hiding to decompose the system into modules that satisfy his goals (2); each module keeps its own secreta design decision about some aspect of the system (e.g., choice of a data structure). A modules design decision can change but none of the other modules should be affected. If some aspect is unlikely to change, the design can distribute this knowledge across several modules and the interfaces among them.

Most programmers would define a module by listing the names, parameters, and return values–the *operation signatures*–of its subprograms. Parnas again focuses on the programmers rather than the programs. He defines the "interface between two programs" to consist "of the set of assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his own program." [4] In addition to an operation signature, these assumptions must specify the restrictions on data passed to the operation, the effect of the operation, and exceptions to the normal processing that may arise.

The interface of an information-hiding module must enable programmers to replace one implementation of the module by another without affecting other modules. This is called an abstract interface because it represents the assumptions common to all implementations of the module [1, 7]. It reveals the module's unchanging aspects but obscures aspects that may vary among implementations.

Information hiding and abstract interfaces underlie object-oriented programming, the predominant contemporary approach to programming. But programmers often oversimplify these concepts as merely hiding data representations inside classes [9]. A secret of a well-designed module is more than hidden data. It is any aspect that can change as the system evolves: processing algorithms used, hardware devices accessed, other modules present, and specific functional requirements supported [4, 5, 8].

Although 40 years have passed since Parnas first wrote about information hiding, programmers can still learn much from studying his papers carefully and can improve their software designs by applying his ideas systematically.

Acknowledgements

The author and collaborators published some of these ideas previously [2, 3].

References

[1] Kathryn Henninger Britton, R. Alan Parker, and David L. Parnas. A procedure for designing abstract interfaces for device interface modules. In *Proceedings of the 5th International Conference on Software Engineering*, pages 195–204, March 1981.

> This classic paper by Parnas and his colleagues builds on the concept of information hiding [5]. It describes the concept of an abstract interface as a mechanism for hiding the details of a low-level interface from the other modules of the software system. It uses a two-phase design method for such modules.

[2] H. Conrad Cunningham, Pallavi Tadepalli, and Yi Liu. Secrets, hot spots, and generalization: Preparing students to design software families. *Journal of Computing Sciences in Colleges*, 20(6):118–124, June 2005.

[3] H. Conrad Cunningham, Cuihua Zhang, and Yi Liu. Keeping secrets within a family: Rediscovering Parnas. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, pages 712–718. CSREA Press, June 2004.

[4] D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, SE-11(3):259–266, March 1985.

[5] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

This classic paper by Parnas is the first publication of the design concept known as information hiding. The key idea is to design software modules around a single design decision that may change over time or among versions of the system. The module hides this design decision, known as the secret of the module, behind an interface that is unlikely to change.

[6] David L. Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.

This classic paper by Parnas introduces the design concept known as program (or software) families, often called software product lines today. This means that the designers of a software system should seek to design the system to exist in multiple versions from the beginning. New versions may be created for different customers or platforms or perhaps just over time as improvements are made to the original system.

[7] David L. Parnas. Some software engineering principles. In *Infotech State of the Art Report on Structured Analysis and Design*, page 10 pages. Infotech International, 1978. Reprinted in Software Fundamentals: Collected Papers by David L. Parnas, Daniel M. Hoffman and David M. Weiss, editors, Addison Wesley, 2001.

[8] David L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, SE-5(1):128–138, March 1979.

[9] David M. Weiss. Introduction: On the criteria to be used in decomposing systems into modules. In Daniel M. Hoffman and David M. Weiss, editors, *Software Fundamentals: Collected Papers by David L. Parnas*, pages 143–144. Addison-Wesley, 2001.