
Abstract
RS: Radical solution
A radical solution to the problem of <efficiently breaking software into modules> is described, based on <information hiding>. In comparison with <existing approaches based on control flow> it offers <improved changeability, interface cleanliness, and comprehensibility>, which have been demonstrated in preliminary tests, but it leaves a number of side effects to be addressed including <a potential for decreased performance>. Strategies are suggested for addressing these side effects.

Why RS?
It could also be reasonably argued that this paper is an “enhancement” rather than radical if it could be established that it extends an existing method in an incremental fashion. The main support for this line of reasoning is that Parnas’s contribution is a new modularization method, and that it improves on existing modularization methods (in the context of the larger question, “What is the most efficient way to design?”) However, this argument overlooks the fact that previous modularization methods focused on control flow, whereas Parnas’s focuses on data dependencies/flow; in the context of the time, when functional languages were virtually all that existed, Parnas’s work represented a radical departure from the thinking of the time.

Question - [Method/means of development]
<problem definition>
What is a better way of efficiently breaking software into modules?

Results - [Procedure / technique]
<solution strategy>
A new way of doing development is to identify modules on the basis of information hiding.

Validation – [Example]
<existing normal solutions>
The old way of doing development was to identify modules on the basis of control flow through functions.

<advantages>
The new way offers improved changeability, interface cleanliness, and comprehensibility, as demonstrated through a small example (KWIC).

<list of side effects>
The new way may generate an implementation with lower performance (though the author offers something like C macros as a solution to the problem).