of layers, with each layer representing one or several levels in the Uses Relation. Grouping adjacent levels in Uses helps to simplify the representation and allows for cases where there are small loops in the relation. One guideline in performing such a grouping is that programs at one layer should execute approximately 10 times as quickly and 10 times as often as programs in the next layer above it (Courtois 1977).

A system that has a hierarchical Uses Structure can be built one or a few layers at a time. These layers are sometimes known as "levels of abstraction," but this is a misnomer. Because the components are individual programs, not whole modules, they do not necessarily abstract from (hide) anything.

Often a large software system has too many programs to make the description of the Uses Relation among programs easily understandable. In such cases, the Uses Relation may be formed on aggregations of programs, such as modules, classes, or packages. Such aggregated descriptions lose important information but help to present the "big picture." For example, one can sometimes form a Uses Relation on Information Hiding Modules, but unless all programs in a module are on the same level of the programmatic Uses hierarchy, important information is lost.

In some projects, the Uses Relation for a system is not fully determined until the system is implemented, because the developers determine what programs they will use as the implementation proceeds. The architects of the system may, however, create an "Allowed-to-Use" Relation at design time that constrains the developers' choices. Henceforth, we will not distinguish between "Uses" and "Allowed-to-Use."

A well-defined Uses Structure will create proper subsets of the system and can be used to drive iterative or incremental development cycles.

CONCERNS SATISFIED: Producibility and ecosystem.

The Process Structures

COMPONENTS AND RELATION: The Information Hiding Module Structures and the Uses Structures are static structures that exist at design and code time. We now turn to a runtime structure. The components that participate in the Process Structure are Processes. Processes are runtime sequences of events that are controlled by programs (Dijkstra 1968). Each program executes as part of one or many Processes. The sequence of events in one Process proceed independently of the sequence of events in another Process, except where the Processes synchronize with each other, such as when one Process waits for a signal or a message from the other. Processes are allocated resources, including memory and processor time, by support systems. A system may contain a fixed number of Processes, or it may create and destroy Processes while running. Note that *threads* implemented in operating systems such as Linux and Windows fall under this definition of Processes. Processes are the components of several distinct relations. Some examples follow.