Definitions of Modularity

compiled by

Richard P. Gabriel

For the Retrospective on Modularity hold at AOSD 2011 in Porto de Galinhas, Brazil, I asked a number of researchers and practitioners of software development and programming languages to provide their definitions of the term "modularity." My exact question was as follows:

What's your definition of "modularity." If possible, I would like you to think of how you would like it to be defined today, with everything you now know about the concept, and with the intention of making as broad yet accurate a definition as you can.

One way to think about this is that there might be specialized definitions for modularity in programming languages (such as a definition that would delineate a module system, for example) and another for software systems and perhaps another for biological mechanisms such as a genetic module, etc. And your definition should be a "super-definition," as it were, that would subsume all of these.

The following little sections contain the responses I got.

Michael Jackson

"Modularity": The property of being structured in modules.
"Module": A part of an artifact designed to be separately constructed and understood.

Lodewijk Bergmans

Here's my two cents:
I would define modularity (wording should be sharpened, I do have a more detailed/precise version of this) as:

A property of a specific view of a system (artifical or natural), where the system can be decomposed into components that have a form of independence—with respect to some properties.

NB1: Independence of components means that one component does not rely on other components with respect to some property: for example, it can be physically isolated, it may be totally self-sustained, it may make its own decisions, one can do some reasoning about its' behavior independent of other components, it can be compiled separately,….

But: being independent with respect to some property, does not necessarily mean being independent in all other respects! For example, the component may not 'die' without others, will never receive inputs, may relay on other components to achieve its goals, etc.

NB2: All meaningful modular systems require some interactions among the components [cf. near decomposability of Herbert Simon]; and this requires interfaces (in the most general sense of the word), but IMO that does not strictly belong to the definition of modularity.

NB3: Perhaps less common in modularity definitions is the emphasis that it is a property that is *specific for certain views* upon that system. But I think that it is important, and in fact strongly related to AOP, that what is modular in one view (e.g. an aspect-aware model/program), is not modular at all in another view (a woven model / generated byte code).

William Harrison

Before defining modularity, I would have to ask whether the term is to be applied to a system, as commonly done, or whether it is a property of induced on a system relative to a point-of-view, as I actually think is more "correct."

Depending on the answer I have two different definitions. From the first point of view (a property of a system) I think I would, in greatest generality, define modularity as "Modularity is the property by which the elements of a system can be treated as equivalent or interchangeable with respect to some set of properties, called the modulus."

But I prefer the second point of view (a property induced by a view) and define it as "Modularity is the degree to which a point of view about equivalence of properties partitions the elements of a system into equivalence classes with respect to those properties."

I refer to the mathematician's notion of the modulus, whether in the common "modulo" function of in the more abstract group-theoretic sense with complex moduli.

I think the general definition specializes to a number of uses, and in the case of software it highlights the fact (of opinion) that modularity is relative to the things we consider to be independently substitutable.

John Lamping

Modularity is a property of a system that allows it to be altered to handle altered requirements by making relatively local changes, and that allows several, individually useful, changes to be usefully combined.

Gregor Kiczales

Modularity is a property of a representation of a system of interest such that:

- The entire representation of particular issues or concerns is locally contiguous.
- And the interaction of each locally represented concern with other distant concerns is described by clearly represented interfaces.

Where locally contiguous is defined with respect to the purpose of the modularity.

In design disciplines we usually speak of modularity as satisfying human readers, in such cases locally contiguous relates to a rendering on screen or paper of the representation. But even in our own discipline that is not the only purpose of modularity. Consider tools that rearrange system software to make the software contiguous on a disk drive—this is causing the executable representation of the software to have better physical modularity with respect to expected paging behavior. Modularity in biological systems is similar: the modularity can be in terms of spatial locality (as in organs and body parts), but can also be in other terms such as networks that communicate via specific enzymes.

In designed systems of any significant complexity modularity is inherently a tradeoff. For any given modularity, there will always be some task or activity for which that modularity is less than ideal. As a simple example, a layered architecture may help to design and understand distinct kinds of functionality in a network architecture; but it will impede understanding end-to-end performance of specific network services.

The main contribution of our community was to establish once and for all that not only does software development inherently require different modularities, but also that those different modularities do not, in general, bear a hierarchical relationship to one another. Put another way, different granularities of hierarchy is not a sufficient way of organizing all desired modularities. Instead, some desired modularities have a crosscutting relationship to one another. Hierarchy is dead as a unifying framework.

In this we were part of a larger rebellion against the strictures of hierarchy. The term *crosscutting* is not only more common in our own field today than it was ten years ago, it is also more common in the popular discourse.

The secondary contribution of our community was to develop a range of mechanisms that support crosscutting (as opposed to hierarchically related) representations of a software system. One way these mechanisms can be categorized is along an axis from tool-like to programming-language like. To pick just a few examples, at the tool-like end is Mylyn, which allows programmers to work with task-based modularities of a system. In the middle of this spectrum are systems like HyperJ. At the language-like end are languages like AspectJ. Mylyn, HyperJ, and AspectJ, which are all general purpose; another categorization of mechanisms is on an axis from general purpose to concern specific to domain-specific. Concern specific examples include Demeter traversals and RG, domain specific examples include D.

NOTE: To understand the above it is important to understand crosscutting is a symmetric relationship, two concerns crosscut each other with respect to a given representation. Even when one concern and/or its representation may be treated as primary, crosscutting itself is a symmetric relationship.

Harold Ossher

First a narrower definition than you asked for, specific to software systems:

Definition 1: Many modern software systems—those for which modularity matter—are large and highly complex. When a software engineer needs to perform a task (e.g., fix a bug, enhance the functionality, change a business rule, create a variant), it is important that s/he not have to examine and understand the entire system; if that is even possible, it would be prohibitively time-consuming. Instead, s/he must be able to focus on a "relevant subset" of the system, whose size is more closely related to the size of the task than to the size of the system. Modularity is any approach that supports this focus, in such a way that the software engineer has confidence that the relevant subset contains all that needs to be examined, and that changes that are correct with respect to that subset will be correct with respect to the entire system (i.e., will not have unexpected interactions with elements of the system outside the subset).

Though phrased in terms of software, I believe that this applies to any system in which changes are engineered rather than occurring spontaneously. I had some real trouble thinking of a generalization that handles both kinds of change. Here's one attempt:

Definition 2: Modularity is the property that realistic localized changes in a system have localized effects. The definition of "realistic localized change" depends on the kind of change:

- For an engineered change, it is a change that occurs in one or a small number of places and is determined to be locally correct, i.e., by examining nearby elements (e.g., a change to code that is determined to preserve the interface of its enclosing module).
- For a non-engineered change (spontaneous, caused by an outside agent without design, etc), it is a change that can reasonably occur (e.g., a supporting girder expanding by a millimeter when the weather is hot is a realistic change, whereas part of the girder suddenly melting is not).
-

Modularity is not just present or absent, there are degrees of modularity. The greater the extent to which this property is true (i..e, the more (kinds of) localized changes have localized effect), the greater the modularity.

I'm not thrilled with this—and certainly it is not very formal, but I'm trying to capture some of the nuances of the SE definition in the more general situation.

While writing the above, though, the following suddenly struck me, and I like it best so far:

Definition 3: Modularity is the extent to which localized changes in a system have localized effects.

(I put "in a system" in the definition because it provides the context for defining "localized," "changes," and "effects." I certainly consider the universe, biological organisms, etc. to be systems in this context.)

In elaborating "extent," one can consider the various cases above, e.g.,

- All changes having localized effect is the maximum extent (unrealistic)
- All engineered changes that are locally correct having localized effect is a good extent, and an interesting point

in the modularity spectrum

- All small perturbations having localized effect is a good extent, and another interesting point in the modularity spectrum
- etc

Mary Shaw

Modularity (n) from “modular” +”ity”
“ity” is a suffix used to form abstract nouns expressing state or condition so modularity is the state of being modular

Modular (adj)
related to, or based on, a module or modules

This leaves only the problem of defining module. :-)

Mira Mezini

Modularity: Principles and mechanisms for structuring complex spaces of units in manageable chunks with the goal of facilitating some task involving that space, whereby the chunks can themselves be subject of further structuring.

Richard P. Gabriel

Modularity is the compartmentalization of information that gives rise to behavior and information that achieve a coherent purpose in a (running) system—living or artificial.