# Loosely Coupled Yet Highly Cohesive

Steven Bucksbaum, February 5, 2011

## Where Change Does Not Mean Broken

Strive for low coupling and high cohesion: What does that even mean? Coupling means that something is depending on something else, multiple things are tied together. Cohesion means (in terms of information passing between modules) ... well, maybe the person asking the question heard something about it in high school chemistry and can recall it has something to do with sticking together. Maybe they don't know at all.

**Technical Examples**

**C#:** Use reflection, dynamically search for an assembly and then dynamically instantiating an object in that assembly. The assembly can change (code changes made) without recompiling any of the dependent assemblies. Heavy reliance on reflection in C# will noticeably slow down the program.
**Web Services:** Returning an XML string. An attribute can be added to the XML string with out "breaking" the calling client who has to interrogate the returning XML string.
**C++:** Dynamically Linking to a DLL**.** Not often done, but the option is there.
**COM:** Microsoft's Common Object Model (COM) has a medium level of both cohesion and coupling. The calling program can dynamically link to the interface but is must understand the both the interface the data exchange.
**CORBA:** Common Object Request Broker Architecture (CORBA) allows distributed objects to interoperate. A competing technology to Microsoft's COM and DCOM

**Seek High Cohesion**:

**Coincidental cohesion (worst)** is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used functions).
**Logical cohesion** is when parts of a module are grouped because they logically are categorized to do the same thing, even if they are different by nature (e.g. grouping all I/O handling routines).
**Temporal cohesion** is when parts of a module are grouped by when they are processed - the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).
**Procedural cohesion** is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).
**Communicational cohesion** is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information).
**Sequential cohesion** is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data).
**Functional cohesion (best)** is when parts of a module are grouped because they all contribute to a single well-defined task of the module

**Seek Low or Loose Coupling:**

**Content coupling (high)** is when one module modifies or relies on the internal workings of another module (e.g. accessing local data of another module). Therefore changing the way the second module produces data (location, type, timing) will lead to changing the dependent module.
**Common coupling** is when two modules share the same global data (e.g. a global variable). Changing the shared resource implies changing all the modules using it.
**External coupling** occurs when two modules share an externally imposed data format, communication protocol, or device interface.
**Control coupling** is one module controlling the logic of another, by passing it information on what to do (e.g. passing a what-to-do flag).

**Stamp coupling (Data-structured coupling)** is when modules share a composite data structure and use only a part of it, possibly a different part (e.g. passing a whole record to a function which only needs one field of it). This may lead to changing the way a module reads a record because a field, which the module doesn't need, has been modified.
**Data coupling** is when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data which are shared (e.g. passing an integer to a function which computes a square root).
**Message coupling (low)** is the loosest type of coupling. Modules are not dependent on each other, instead they use a public interface to exchange parameter-less messages (or events, see Message passing).
**No coupling** [is when] modules do not communicate at all with one another.