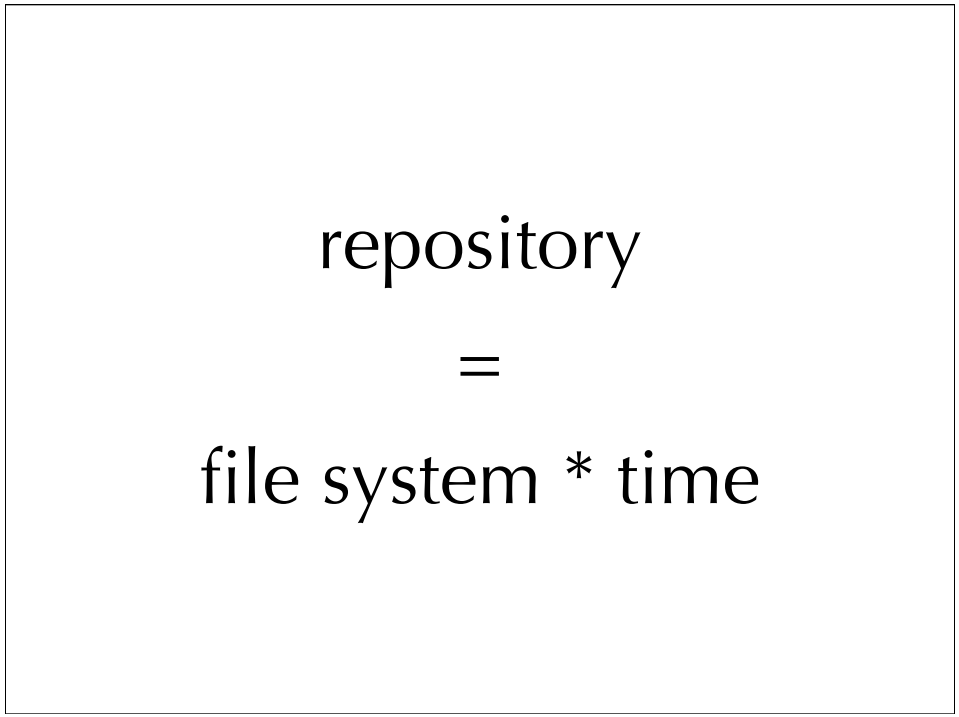
<George Carlin>

http://www.youtube.com/watch?v=MvgN5gCuLac
0:00-1:45

Disclaimer and Warning: The rest of the video is funny. But this is George Carlin, so be prepared for language unbecoming a college professor in the classroom.

This is, at its simplest, what source control is all about.

Thanks to Allyn, Nick, and Danny for sharing their expertise — and students' point of view! They have offered their services to answer SVN questions as well. Simply use the course mailing list:



But there is more. We speak not only of source control but also **version control**.

A source control system maintains a **historical record** of what you have done over time.

Finally, version control systems offer one last essential feature for software engineers: a way for developers to **work together without getting in each others' way.**

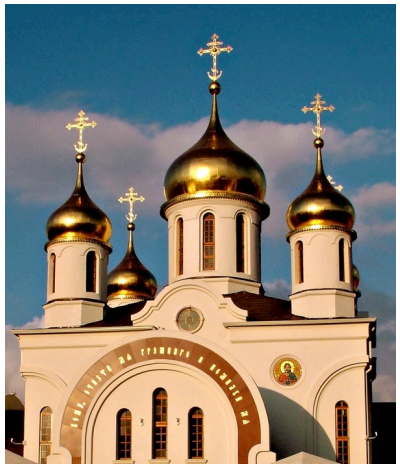
a whole industry
built on keeping an
eye on your stuff

SVN ... CVS ... SourceSafe ... Perforce ... Vault

Arch ... DARCS ... Mercurial ... Git

??

checkout - edit - checkin



edit - merge - commit

checkout – edit – checkin: conservative, traditional strategy ... most knowledge for the team

edit – merge – commit: looser, freer strategy preferred by many developers ... rely on merge tools

When learning, conservative strategies are usually helpful. They allow you to build habits and expertise with the maximum amount of support and the minimum amount of risk. Later, when you have good habits and skills, you can work in whatever style you like best.

I suggest that your project teams use SVN with a **checkout – edit – checkin** approach.

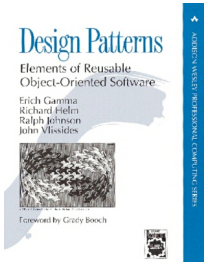
But how can we know?

example:

duplication

heuristic: don’t repeat yourself (DRY)
AKA: say it once and only once

handbook...



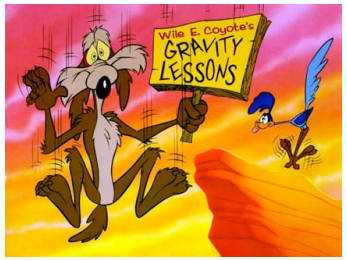
?

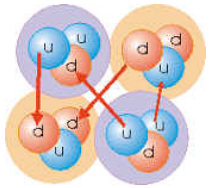
...science

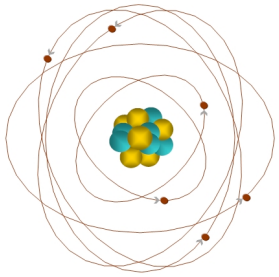
The goal of a handbook for software engineers is to codify practitioners' knowledge.

We also need to codify the science that explains why architectures and patterns work.

four basic forces







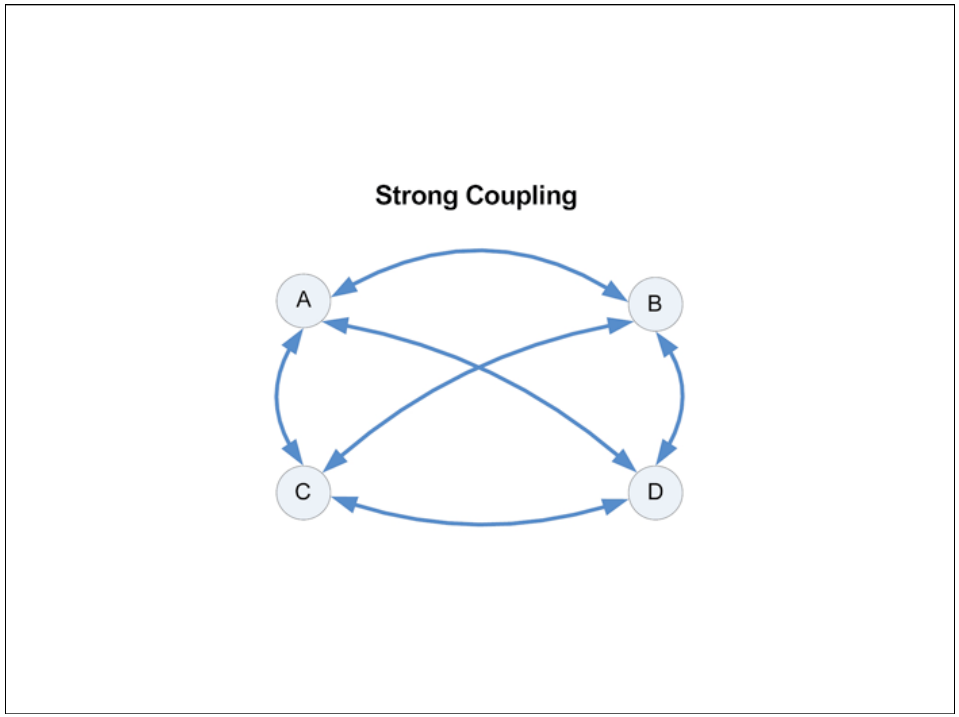
Matter is bound by four fundamental forces: electromagnetism, strong nuclear force, weak nuclear force, and gravity.

What binds software?

coupling

Coupling is a force binding one module to another.
When modules refer to one another, they are coupled.

http://en.wikipedia.org/wiki/Coupling_(computer_science)

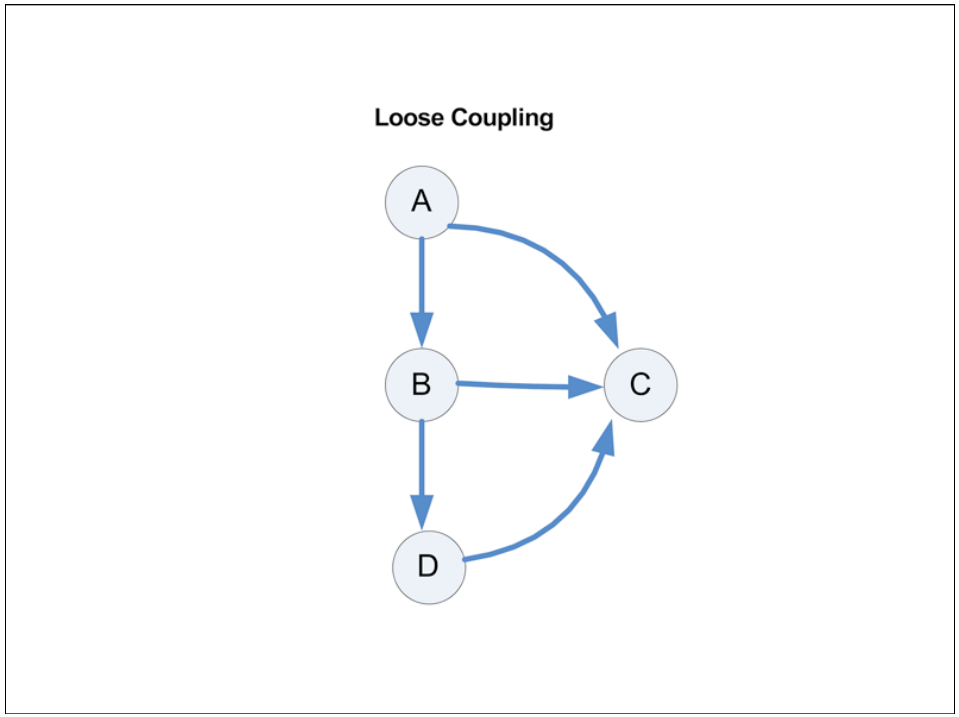


Strong coupling means that many modules refer to each other.

This design is the maximal scenario: 12 module references, out of 12 possible.

What does this mean about implementing a module, or modifying one?

Image courtesy of Intel: http://software.intel.com/file/4043



Loose coupling means that many modules refer to each other.

This design is the less-tightly coupled: 5 module references, out of 12 possible.

Image courtesy of Intel: http://software.intel.com/file/4044

coupling
is a **negative** force
in software

Coupling implies shared knowledge, which hampers creation, modification, and understanding.

In general, we try to **minimize** coupling.

types of coupling

content

common

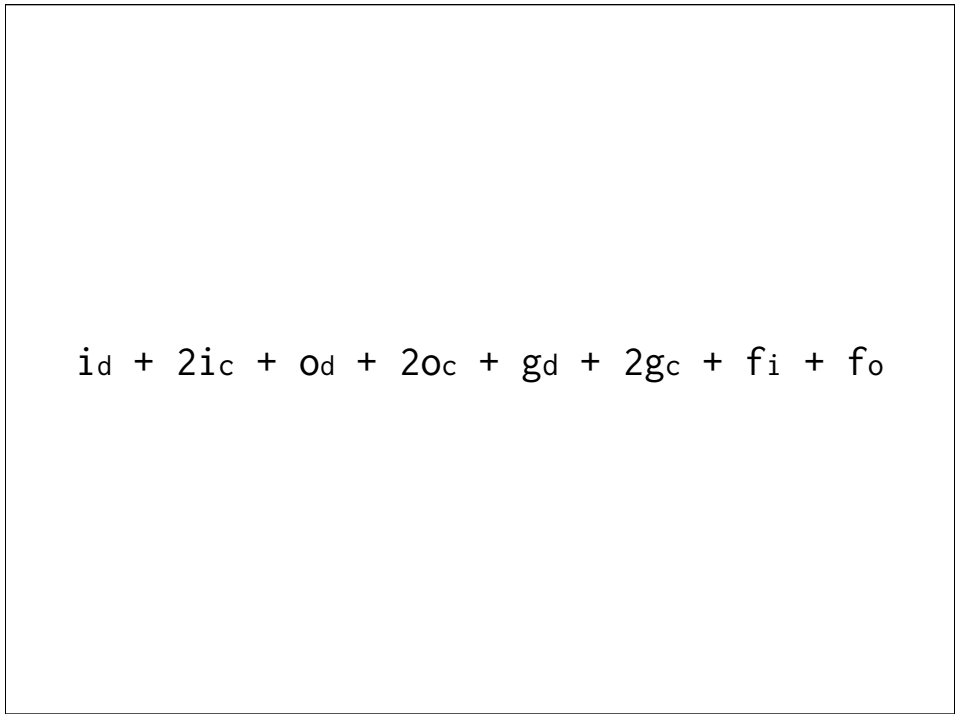
external

control

data

invocation

content	local data, behavior	insidious
common	global data	insidious
external	imposed data format, interface, protocol	insidious
control	one module controls another's behavior	insidious
data	parameters and return values	controllable
invocation	calling a function, sending a message	minimal



an ad hoc metric of coupling

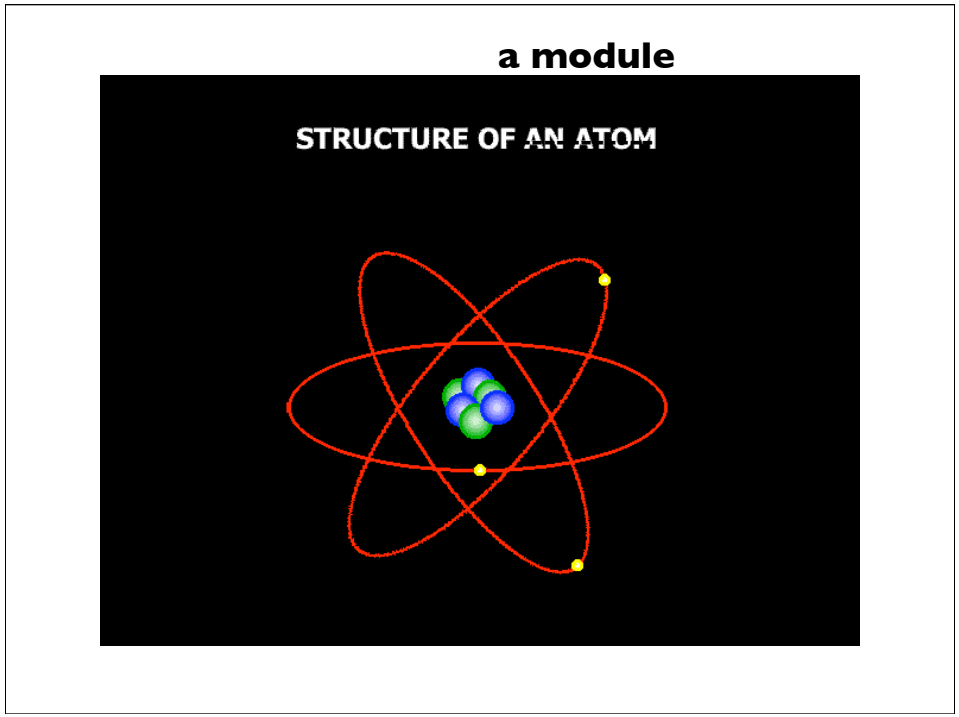
1 – 1/c = coupling ratio, in the range [0.5 to 1.0)

We need better empirical measures of coupling, both magnitude and value.

cohesion

Cohesion is the force binding the elements of a single module.
When modules refer to one another, they are coupled.

http://en.wikipedia.org/wiki/Cohesion_(computer_science)



High cohesion means that the elements of a module — data and functions — refer frequently to one another. Each function uses many of the data elements in the module, and the functions call one another to perform their tasks.

Cohesion is similar to **intra-module** coupling. It is akin to the strong nuclear force.

Image courtesy of: http://www.nuceng.ca/igna/images/Sbb_structure.gif

cohesion

is a **positive** force

in software

Cohesion implies shared purpose, which enhances modification and understanding.

In general, we try to **maximize** coupling.

types of cohesion

coincidental

logical

temporal

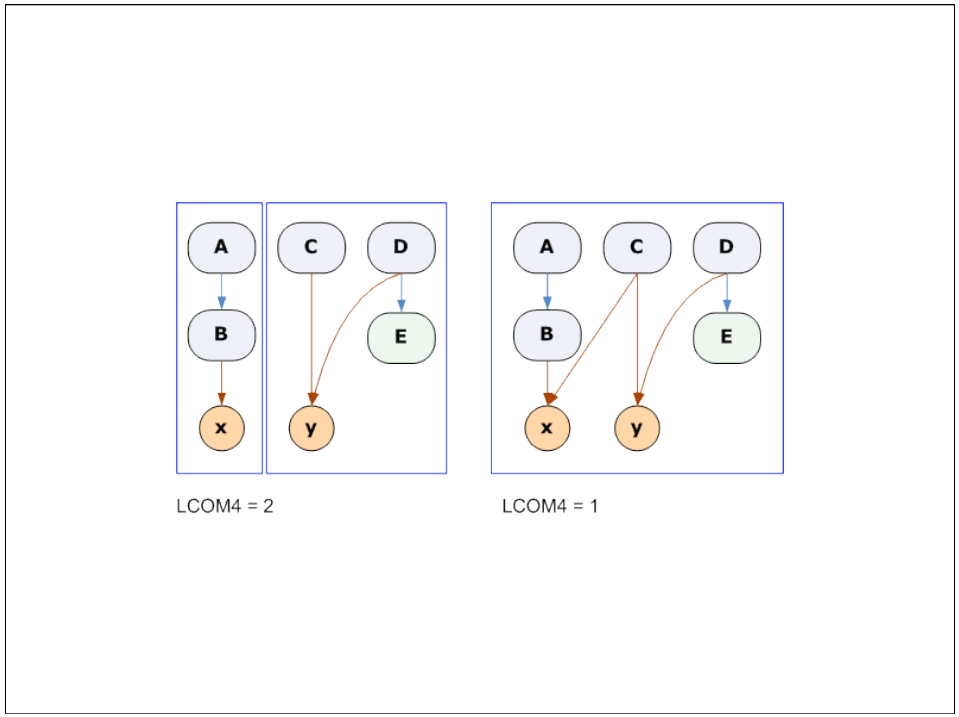
procedural

communication

sequential

functional

coincidental	arbitrary	insidious
logical	by type, not function	insidious
temporal	by when used	?
procedural	by usage in algorithm	?
communication	operate on same data	
sequential	functions are like pipes	
functional	contribute to a single well-defined task	



a simple graph-theoretic metric of cohesion: connected components

a more complex metric would look at the percentage of total possible connections

Again: We need better empirical measures of coupling, both magnitude and value.

coupling and
cohesion are
complements

Increasing cohesion often decreases coupling, and vice versa.

The Law of Demeter

A method M of an object O may invoke only the methods of:

- O
- M's parameters
- any objects created within M
- O's instance variables

A law, or a strong desire?
A useful goal.

dates of interest

10/26/09

10/27/09

10/29/09

Monday: project designs are due (or: iteration 2 is due)

Tuesday: discuss designs in class ... informal presentations

Thursday: midterm exam