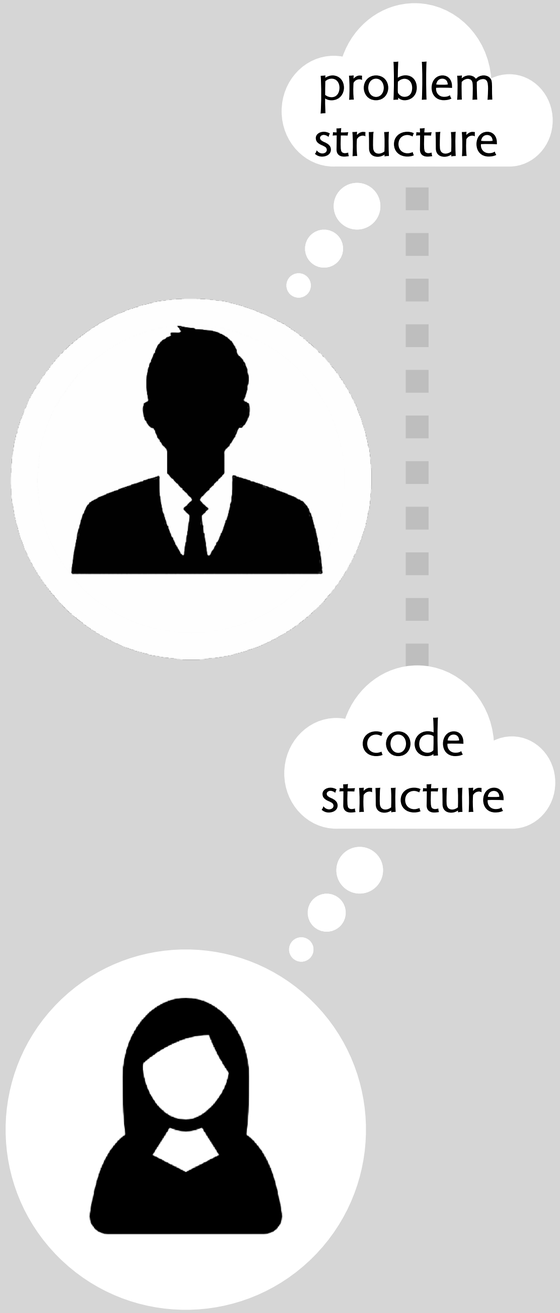
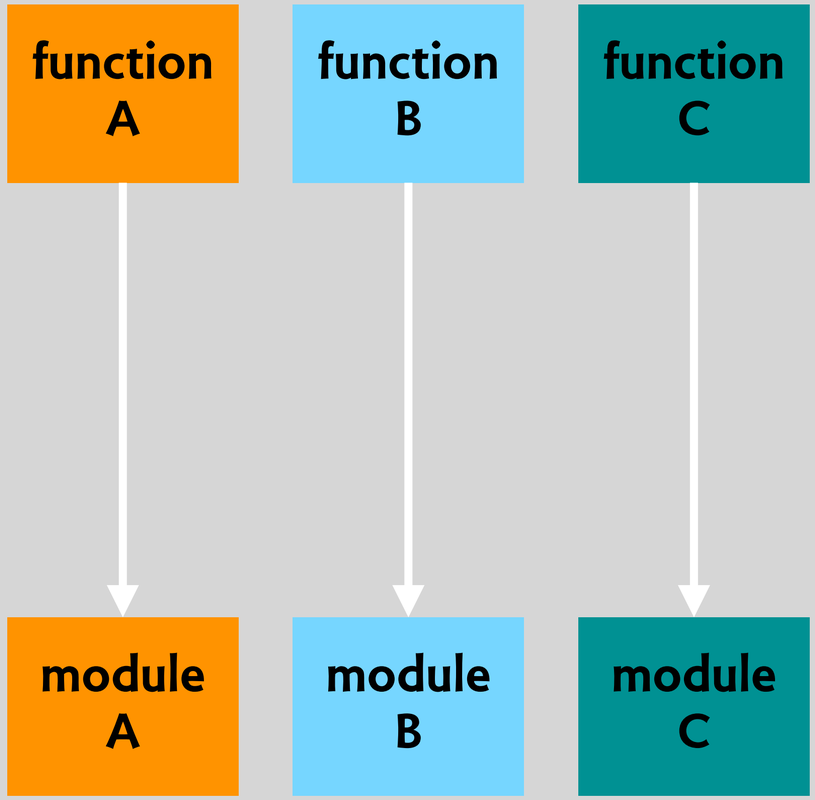
modularity in software

Daniel Jackson · MIT CSAIL/EECS · April 2026

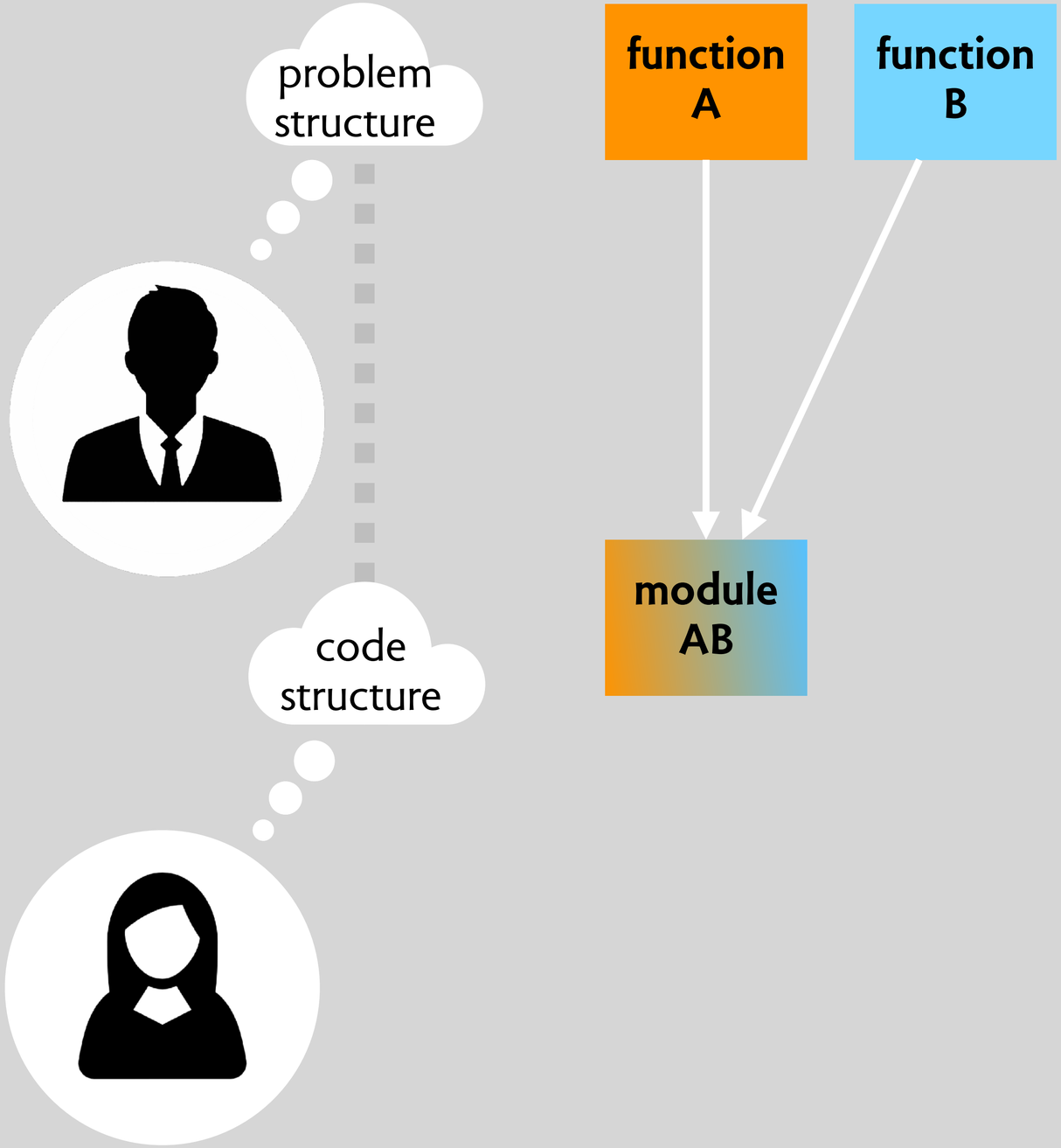
what is modularity?





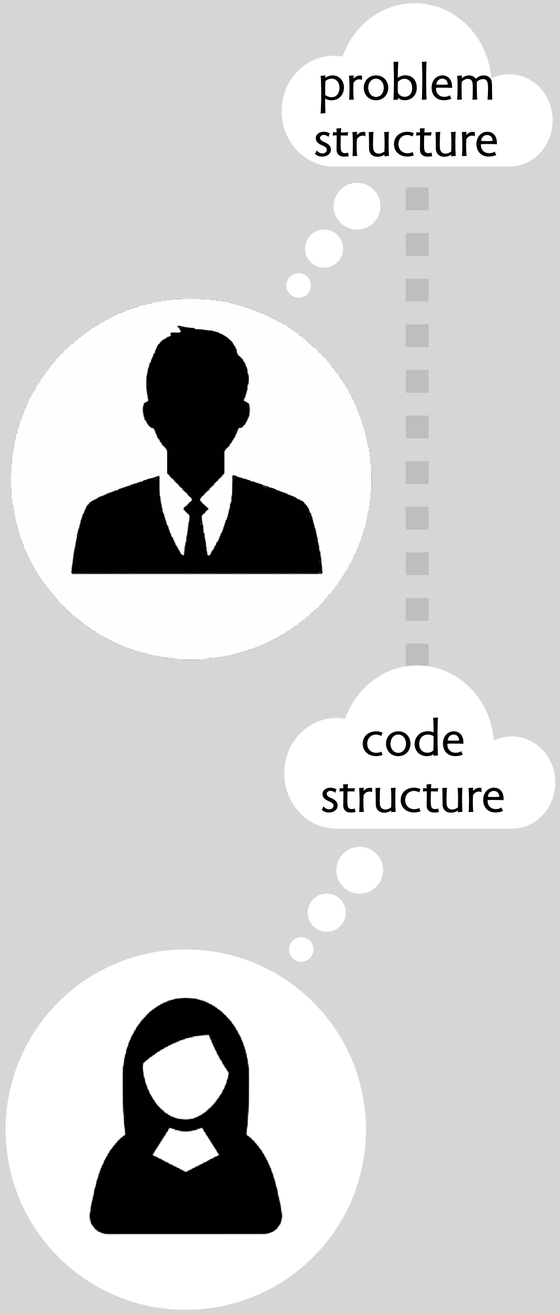
why?

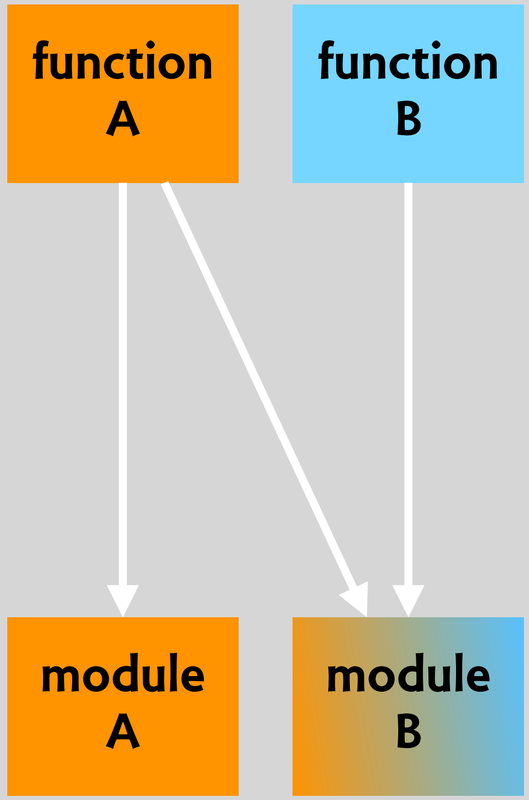
independent *coding of modules*
reuse *of modules for familiar functions*
localization *of changes to functions*



conflation

failure to "separate concerns"
independent coding lost
reusability lost
needless complexity





fragmentation

failure to "encapsulate"
localization lost

why OOP isn't modular

user registers with name and password

user logs in

user posts a blog article

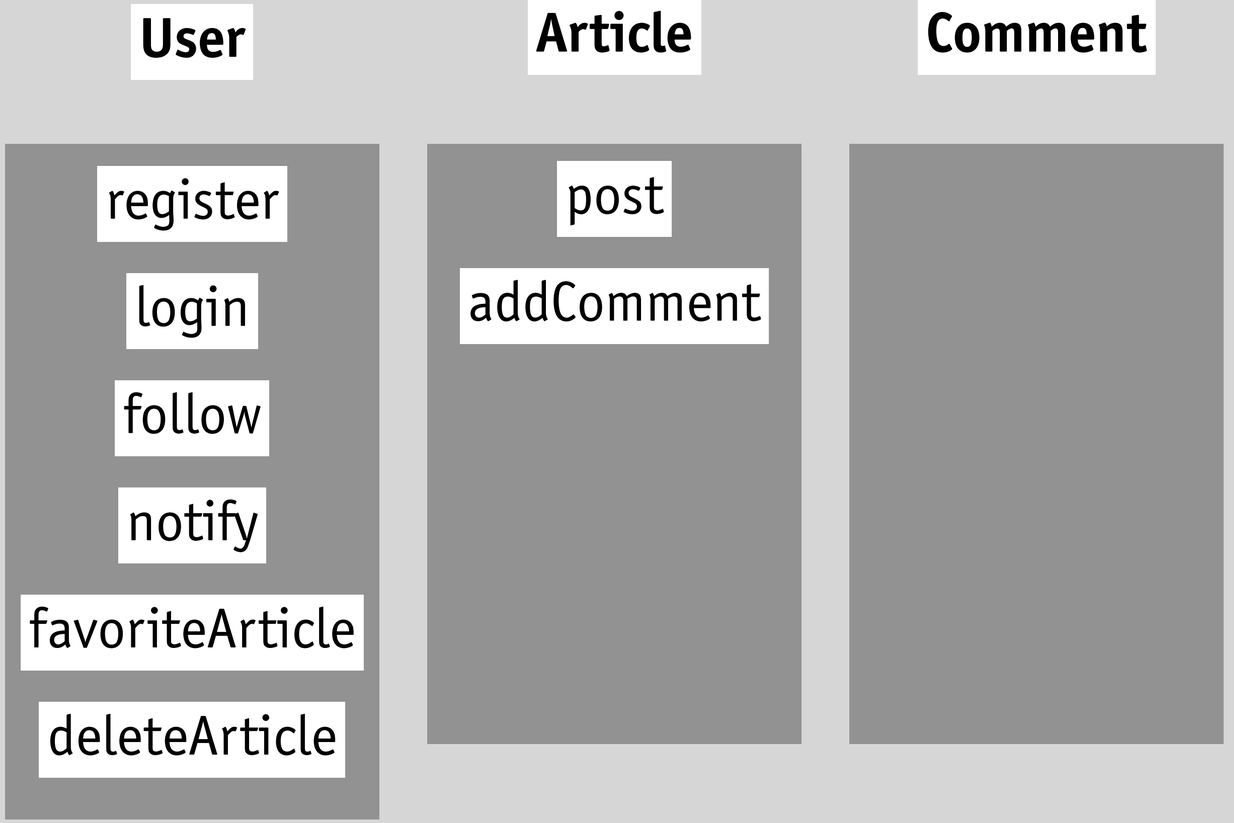
user follows another user

user comments on an article

user is notified about comment on their article

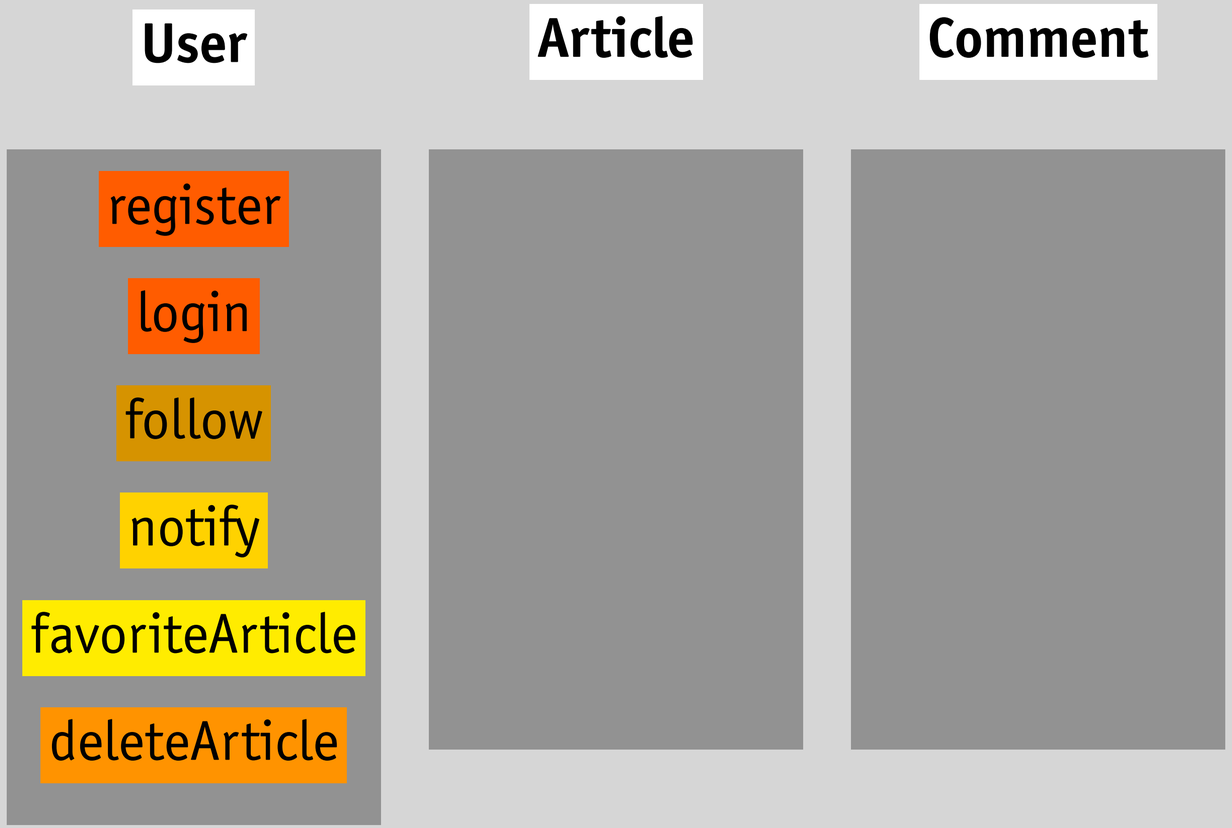
user favorites an article

user deletes an article



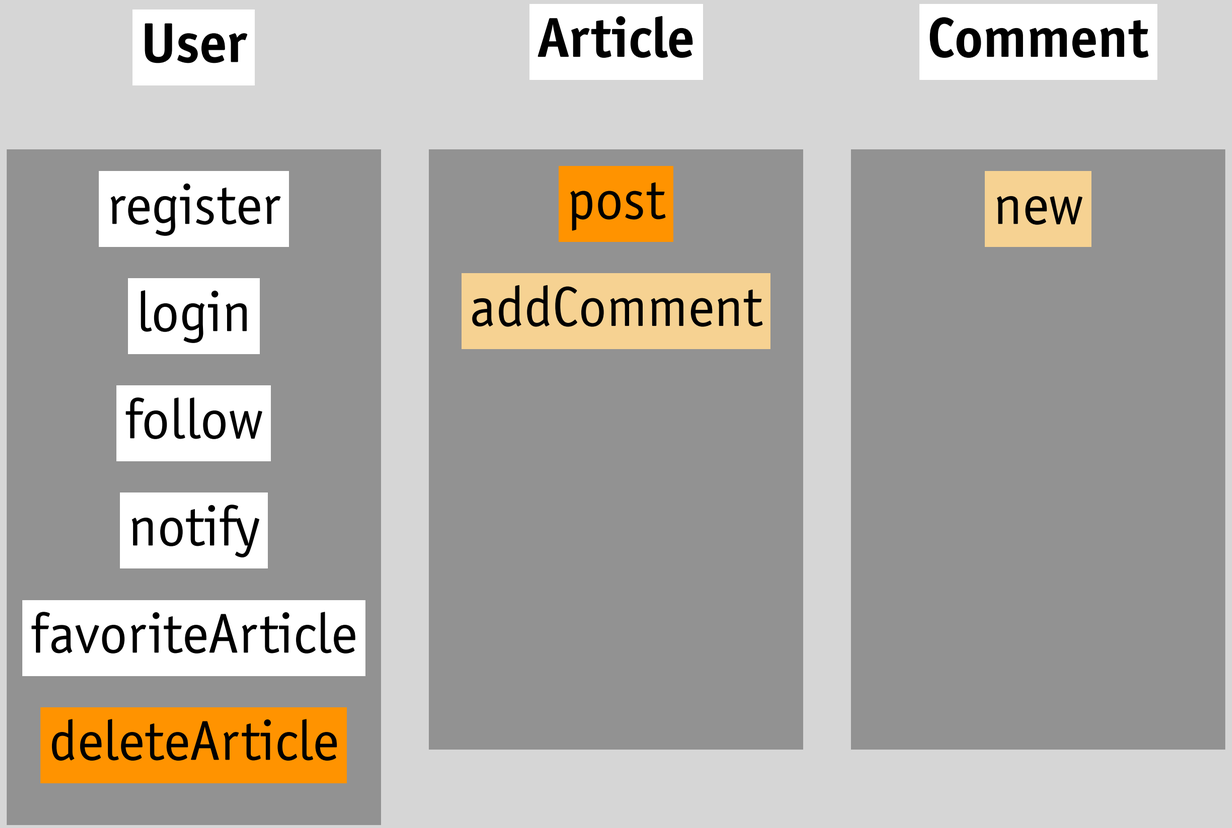
conflation

failure to "separate concerns"
independent coding lost
reusability lost
needless complexity



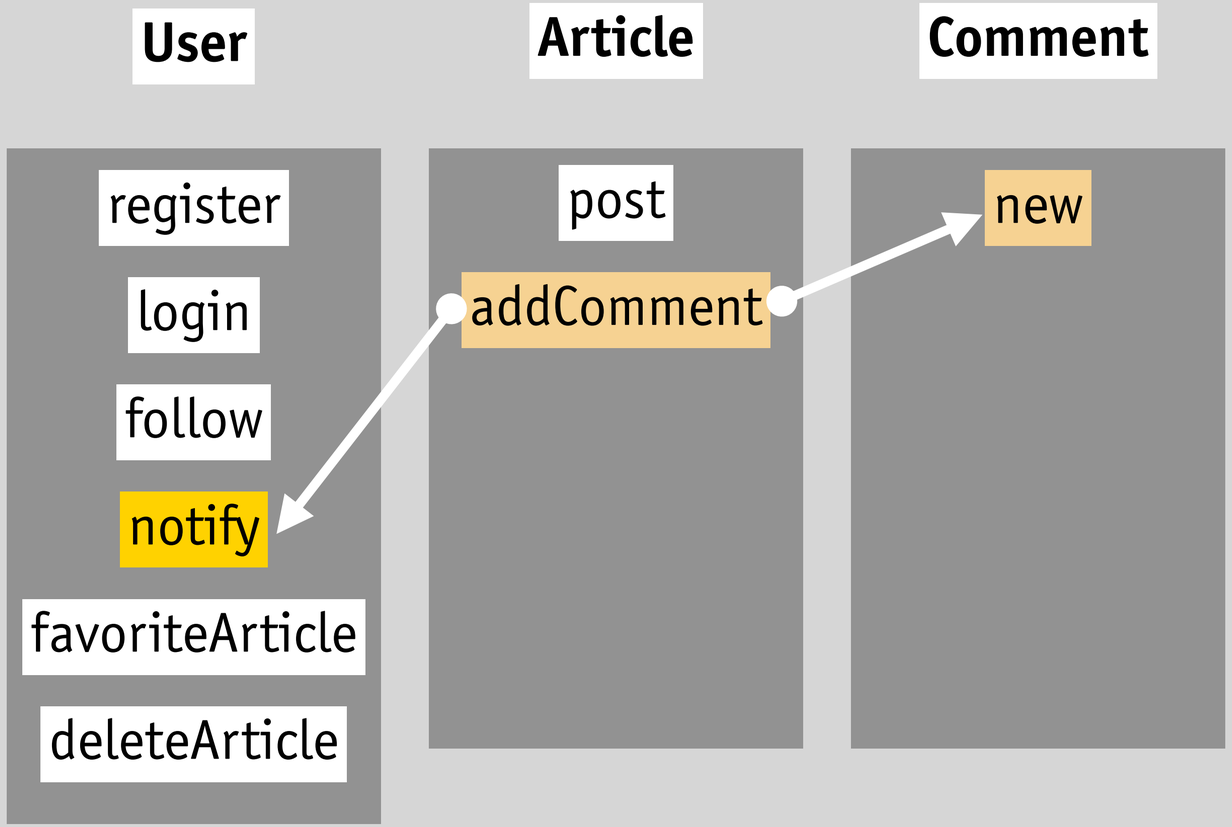
fragmentation

failure to "encapsulate"
localization lost

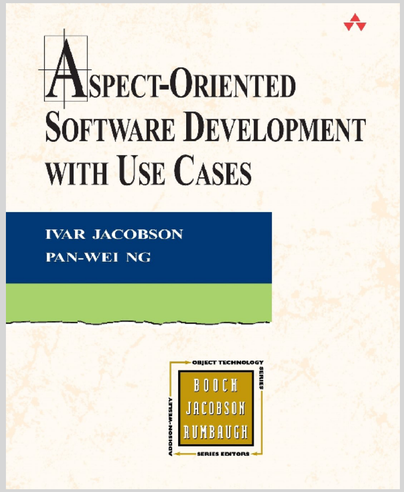


coupling

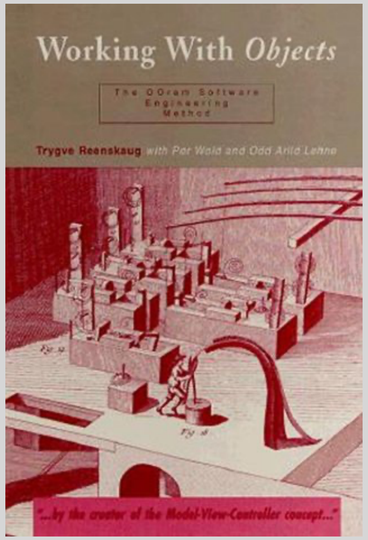
calls between modules
make them interdependent



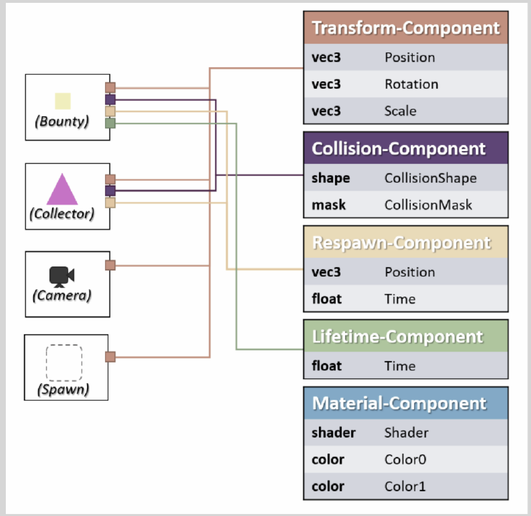
a long history of fixes for OOP's conflation



Aspect-oriented programming
Kiczales et al (1997)



Role-oriented programming
Reenskaug et al (1983)



Entity-component system
Scott Bilas et al (2002)

concepts:
strong modularity

concept structuring

Authenticating

register

login

logout

Posting

addPost

delPost

Commenting

addComment

delComment

Favoriting

favorite

unfavorite

Notifying

register

notify

Following

follow

unfollow

concepts
offer actions
maintain state
objects & rels

syncs
externalize
coordination
& decouple

when Commenting.addComment (comment, post)

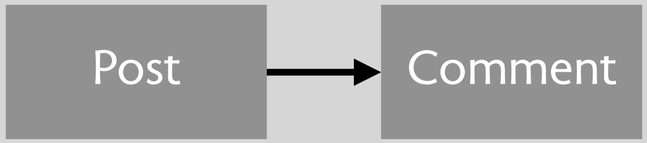
where Posting.isAuthor (post): (user)

then Notifying.notify (user, ...)

surprise

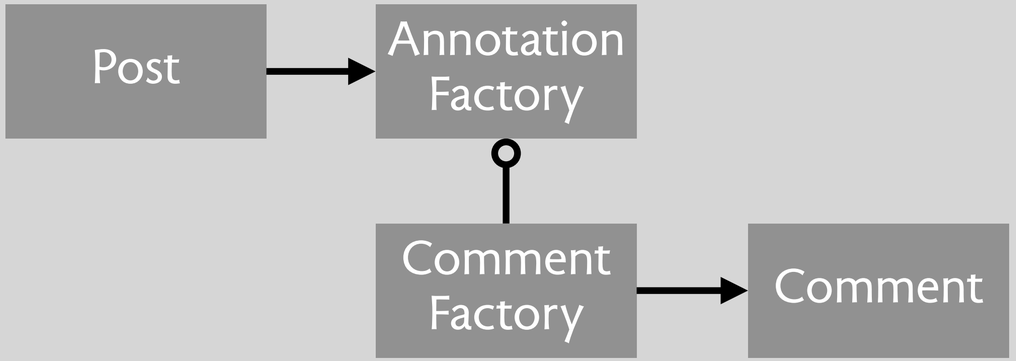
the wrong kind of modularity can damage legibility

a common take on modularity
few dependencies
minimal interfaces

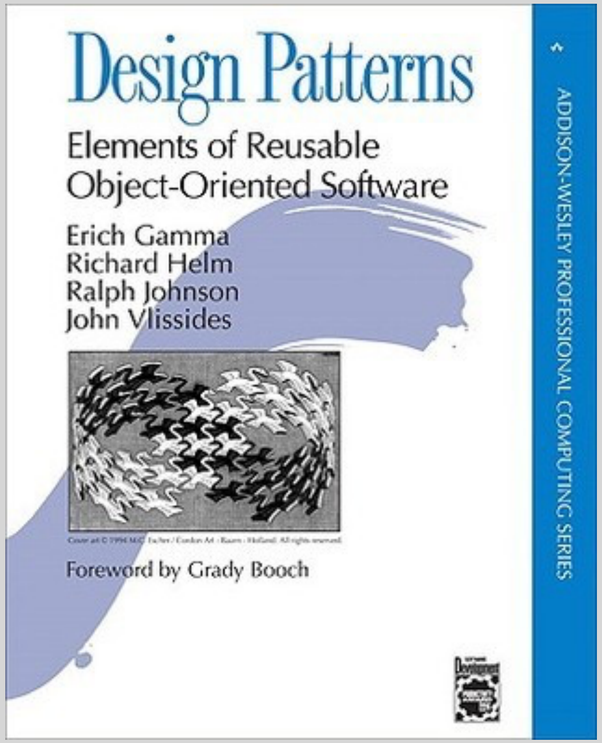


code design patterns
eliminate dependencies
introduce minimal interfaces

example: Factory
eliminate dependencies
introduce minimal interfaces



outcome: may be worse
Post uncoupled from Comment
but code is much harder to read



Relating Run-Time and Compile-Time Structures

An object-oriented program's run-time structure often bears little resemblance to its code structure. The code structure is frozen at compile-time; it consists of classes in fixed inheritance relationships. A program's run-time structure consists of rapidly changing networks of communicating objects. In fact, the two structures are largely independent. Trying to understand one from the other is like trying to understand the dynamism of living ecosystems from the static taxonomy of plants and animals, and vice versa.

With such disparity between a program's run-time and compile-time structures, it's clear that code won't reveal everything about how a system will work. The system's run-time structure must be imposed more by the designer than the language. The relationships between objects and their types must be designed with great care, because they determine how good or bad the run-time structure is.

problem
language

code
language

function

when a post is deleted,
its associated comments
are deleted too

module

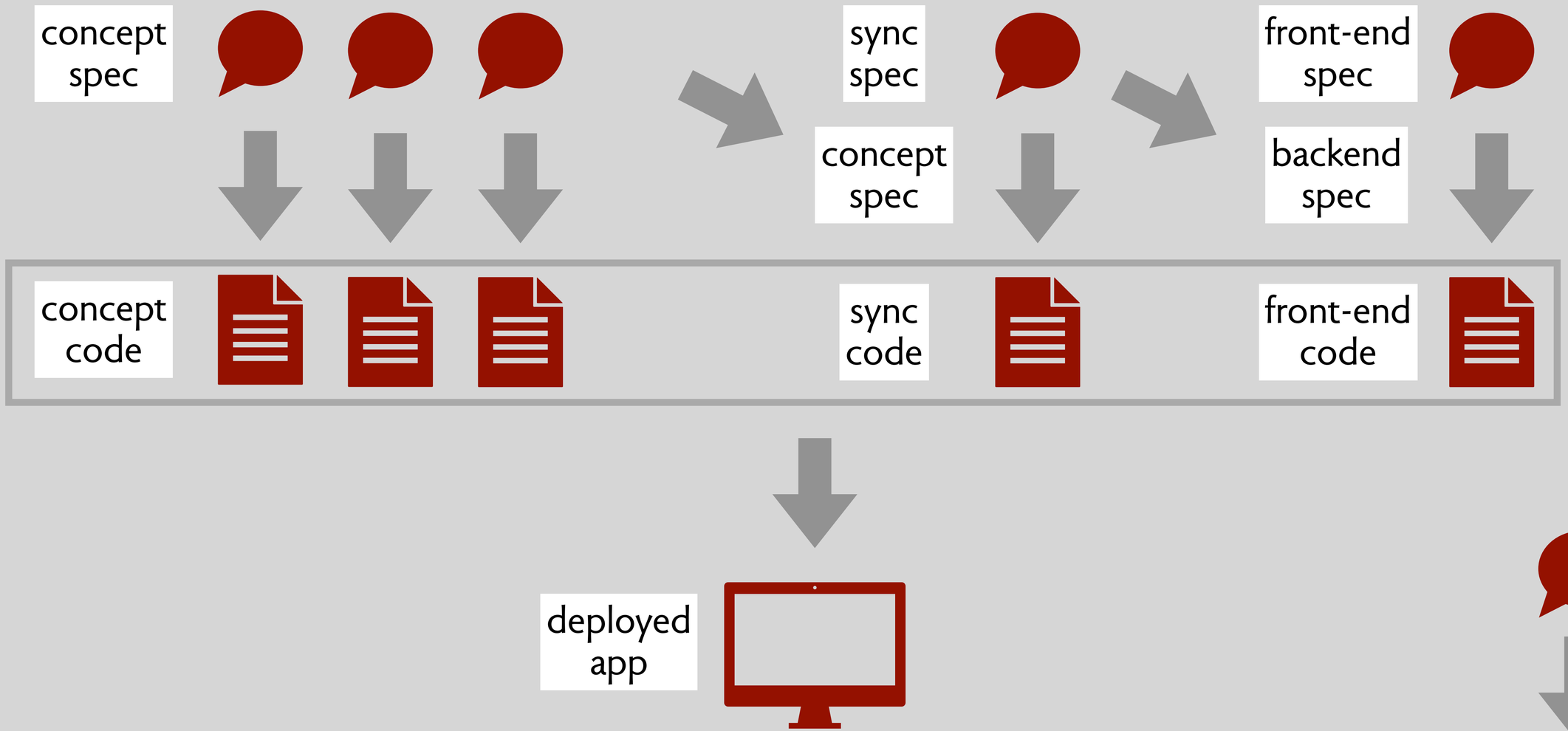
when: Posting.delete (p)
where: Commenting.isTarget (c, p)
then Commenting.delete (c)

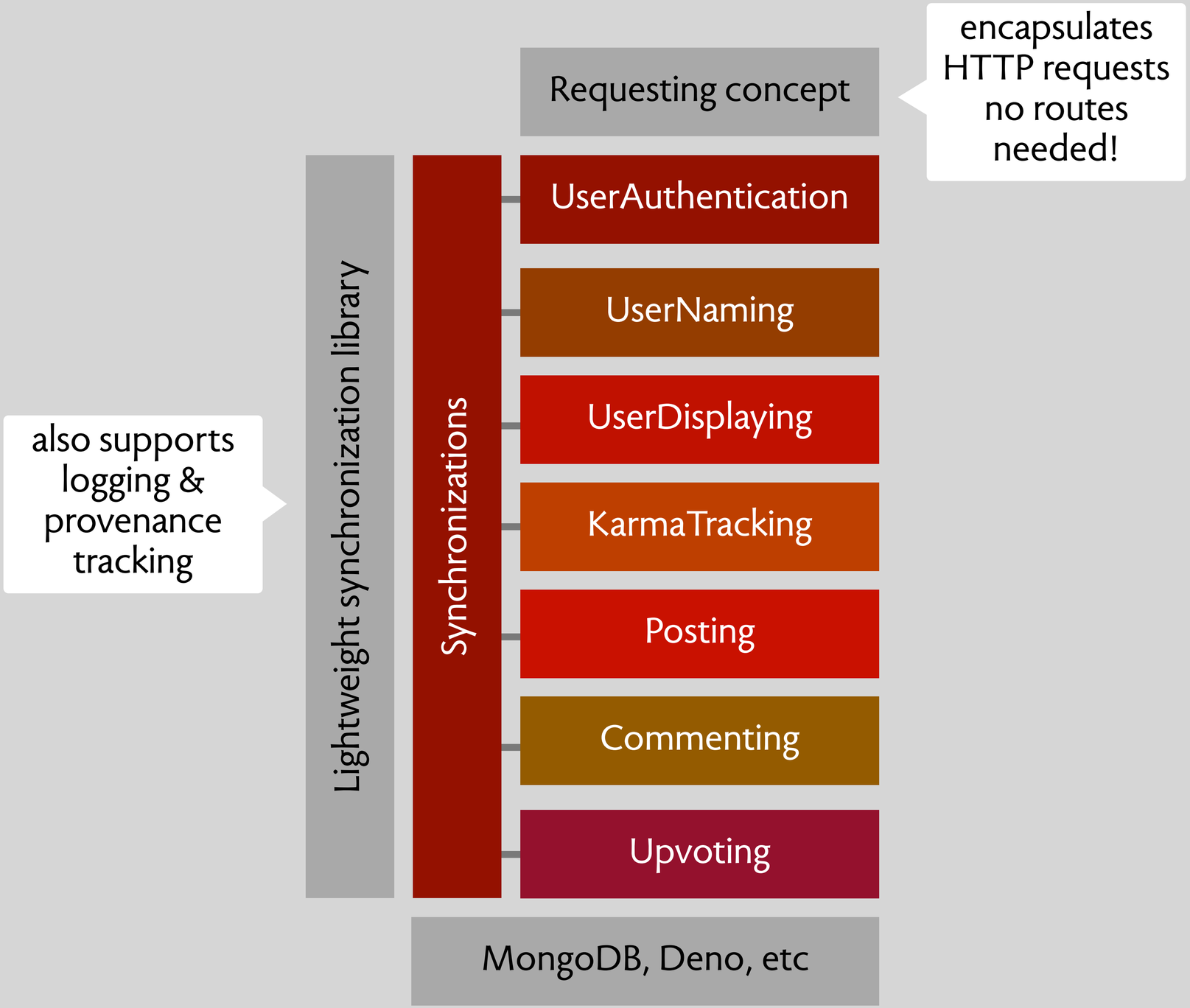
legibility

code is expressed in the
language of the problem domain

generating code

exploiting concept modularity to generate code




Requesting concept
encapsulates HTTP requests no routes needed!
Lightweight synchronization library
Synchronizations
UserAuthentication
UserNaming
UserDisplaying
KarmaTracking
Posting
Commenting
Upvoting
also supports logging & provenance tracking
MongoDB, Deno, etc