an intro to concept design for coding

Daniel Jackson · MIT EECS/CSAIL · Sundai Club · Aug 10, 2025

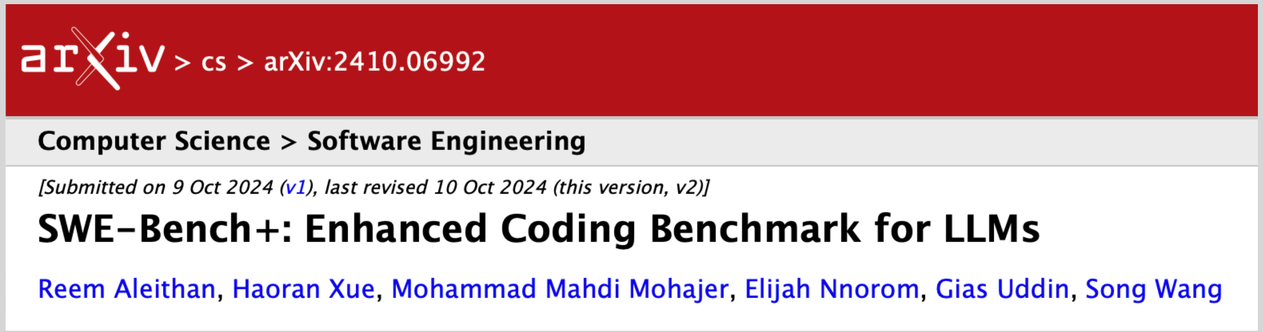
how good are LLMs at coding?

a benchmark and its analysis

a benchmark for realistic coding problems

2,294 issue/pull request pairs from 12 Python repos

best LLM resolves 65% of issues



follow-up study at York University

33% of good patches "cheated": code appears in issue

31% of patches deemed correct by incomplete tests

94% issues were present before training cutoff

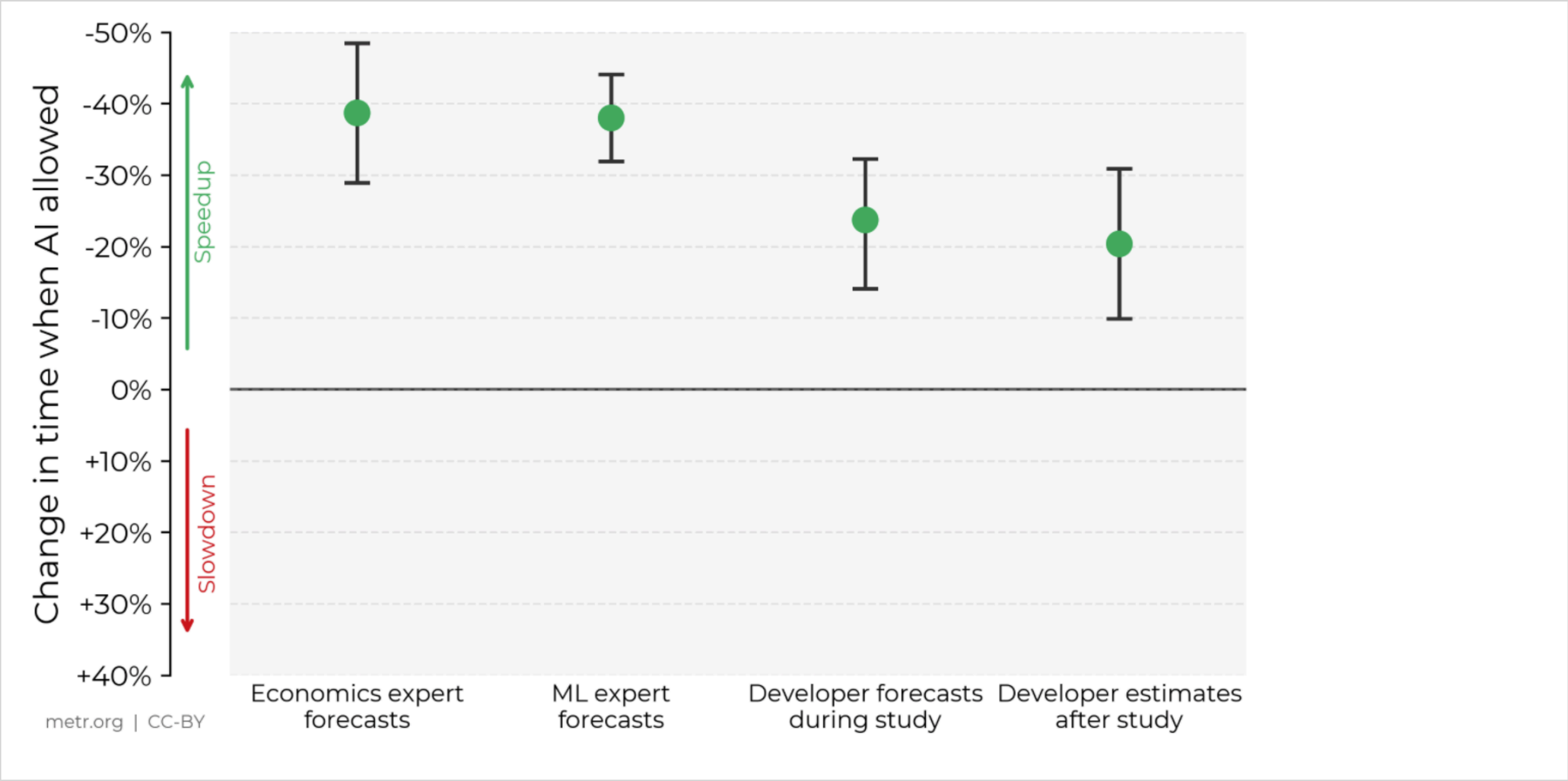
with all this, resolution rate for GPT-4 falls to 0.55%

in short: LLM-based coding assistants

often suggest code that doesn't work

and breaks existing functionality

how much does AI speedup skilled developers on real codebases?



METR study (10 July 2025)
Randomized control trial
16 developers on 246 tasks

Developers qualitatively note LLM tooling performs worse in more complex environments. One developer says “**it also made some weird changes in other parts of the code** that cost me time to find and remove […] My feeling is the refactoring necessary for this PR was “too big” [and genAI] introduced as many errors as it fixed.” Another developer comments that one prompt “failed to properly apply the edits and **started editing random other parts** of the file,” and that these failures seemed to be heavily related to “the size of a single file it is attempting to perform edits on”.

might LLMs inspire
us to rethink software?

LLMs: the canary in the mine

LLMs expose deep flaws in how we build software
software is brittle, poorly organized & needlessly complex
LLMs fail for the same reasons as human programmers

some mistakes we make
focus on code at expense of design
underestimating cost of technical debt

opportunity to rethink our approach
better not only for LLMs but for humans too!

the problem

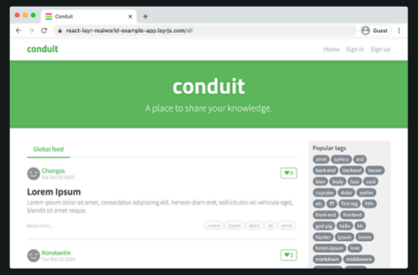
incrementality & integrity
small changes are hard to make
and risk breaking everything

the solution

modularity
each feature in its own place
changes don't propagate

transparency
clear where to find something
code corresponds to behavior

RealWorld, a full-stack benchmark



The mother of all demo apps

See how the exact same application is built using different libraries and frameworks.

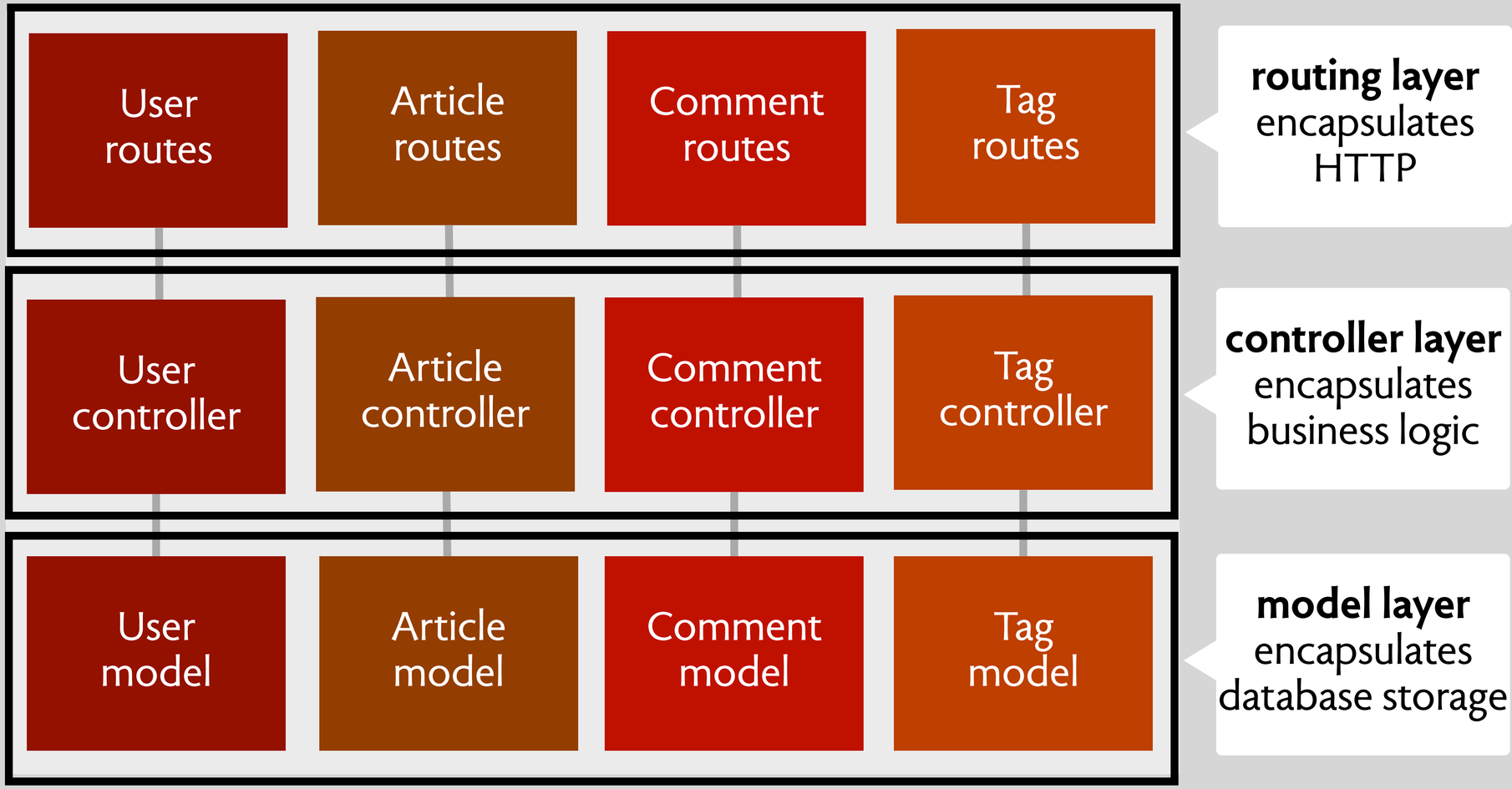
Frontend | Backend | Fullstack

LANGUAGES

All
TypeScript
JavaScript
Kotlin
ClojureScript
Elm
PureScript
Rust
C#
Dart
Mint
Swift
ReScript

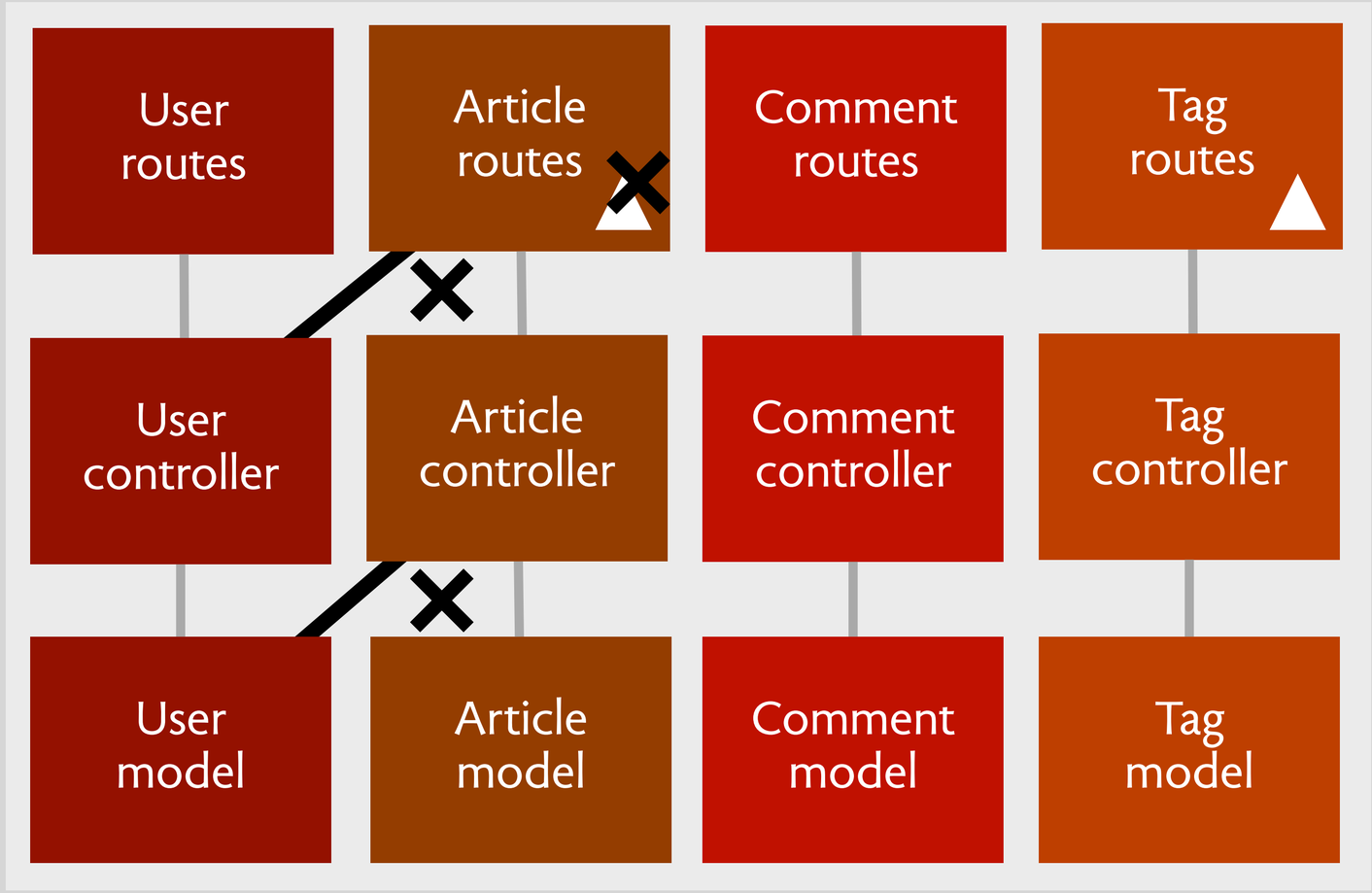
Android Native MOBILE coding-blocks-archives/Conduit_Android_Kotlin	Kotlin
Android Native + Retrofit + Jetpack MOBILE Marvel999/Conduit-Android-kotlin	Kotlin
Angular khaledosman/angular-realworld-example-app	TypeScript
Angular AndyT2503/angular-conduit-signals	TypeScript
Angular iancharlesdouglas/ng-realworld-ssr	TypeScript
Angular + NgRx + Nx	TypeScript

a typical architecture



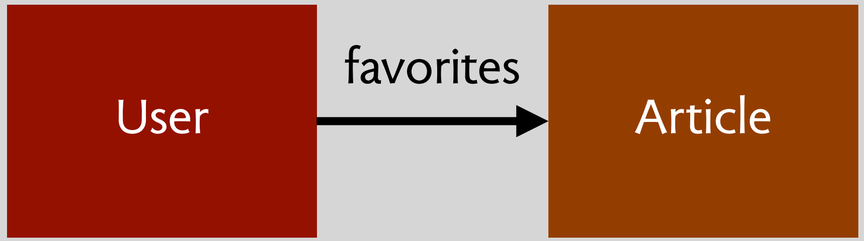
design rules

modularity
of services



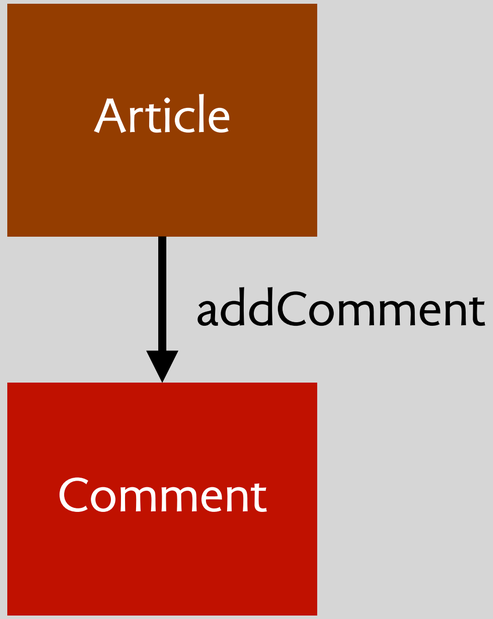
transparency
each increment has a clear home

why are design rules broken? OOP!



many features involve >1 object
eg, favorites relates Users to Articles

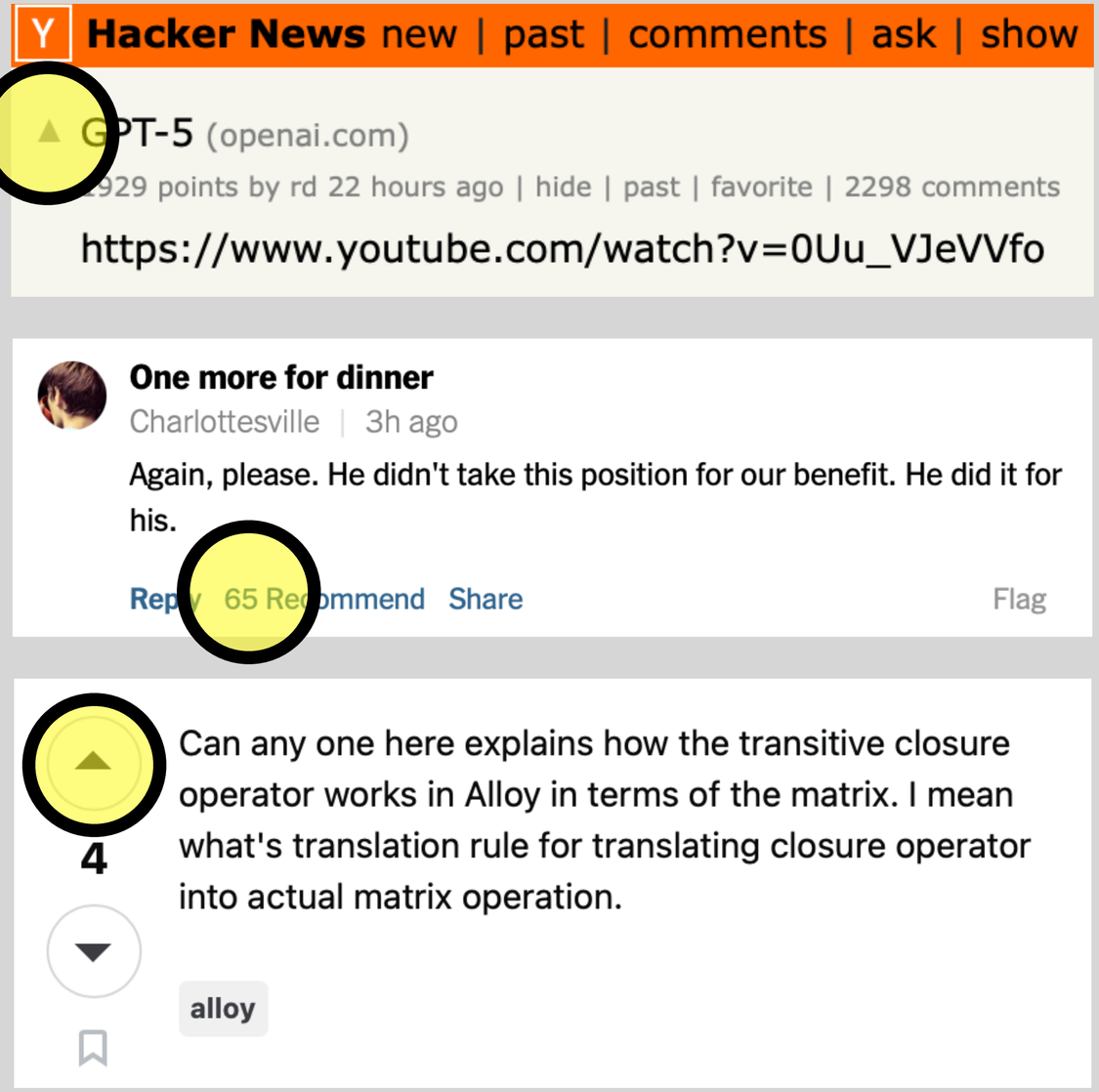
objects conflate features
authentication, profiles, following are all in User



OOP encourages dependencies
eg, addComment is method of Article

concepts:
a new modularity

a sample concept: Upvote



what's a concept?

a coherent **unit** of behavior

user-facing (a behavioral pattern)

a nano **service** (a backend API)

reusable & **familiar**

designed, coded and explained **independently**

defining a concept

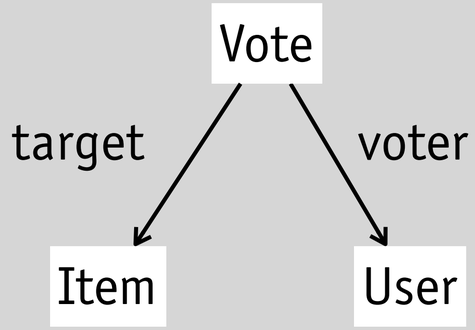
concept Upvote [User, Item]

purpose rank items by popularity

principle after series of votes of items, the items can be ranked by their number of votes

state
a set of Votes with
  a voter User
  a target Item

actions
upvote (u: User, i: Item)
unvote (u: User, i: Item)

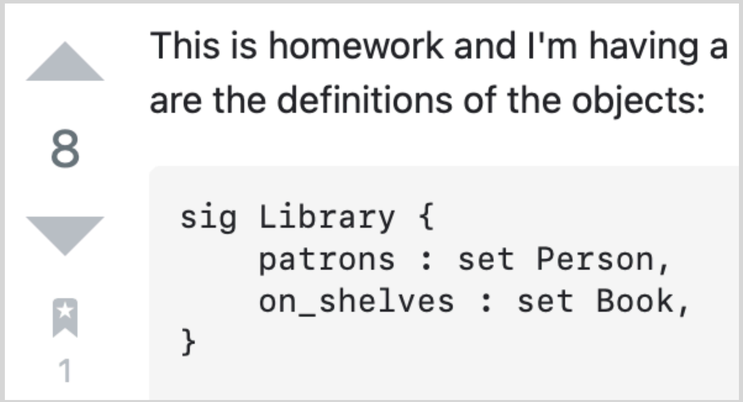


similar UIs, different concepts

concept Upvote

purpose rank items by popularity

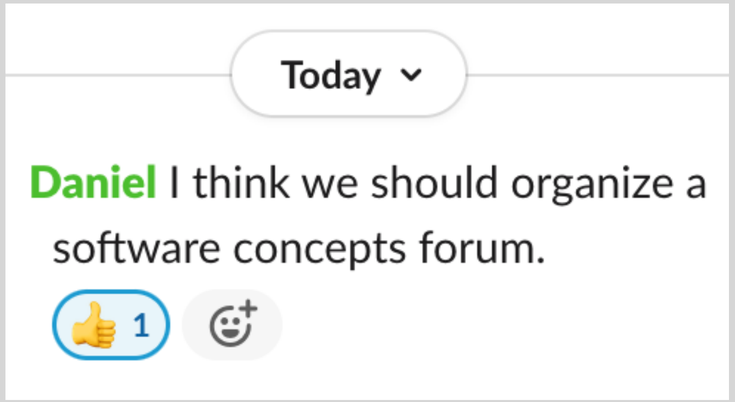
principle after series of votes of items, the items can be ranked by their number of votes



concept Reaction

purpose send reactions to author

principle when user selects reaction, it's shown to the author (often in aggregated form)



concept Recommendation

purpose use prior likes to recommend

principle user's likes lead to ranking of kinds of items, determining which items are recommended

extreme decoupling (xd): syncs & polymorphism

how to ensure concepts are independent?

no calls from one to another

no assumptions about external **types**

suppose we want to notify authors when their posts are upvoted

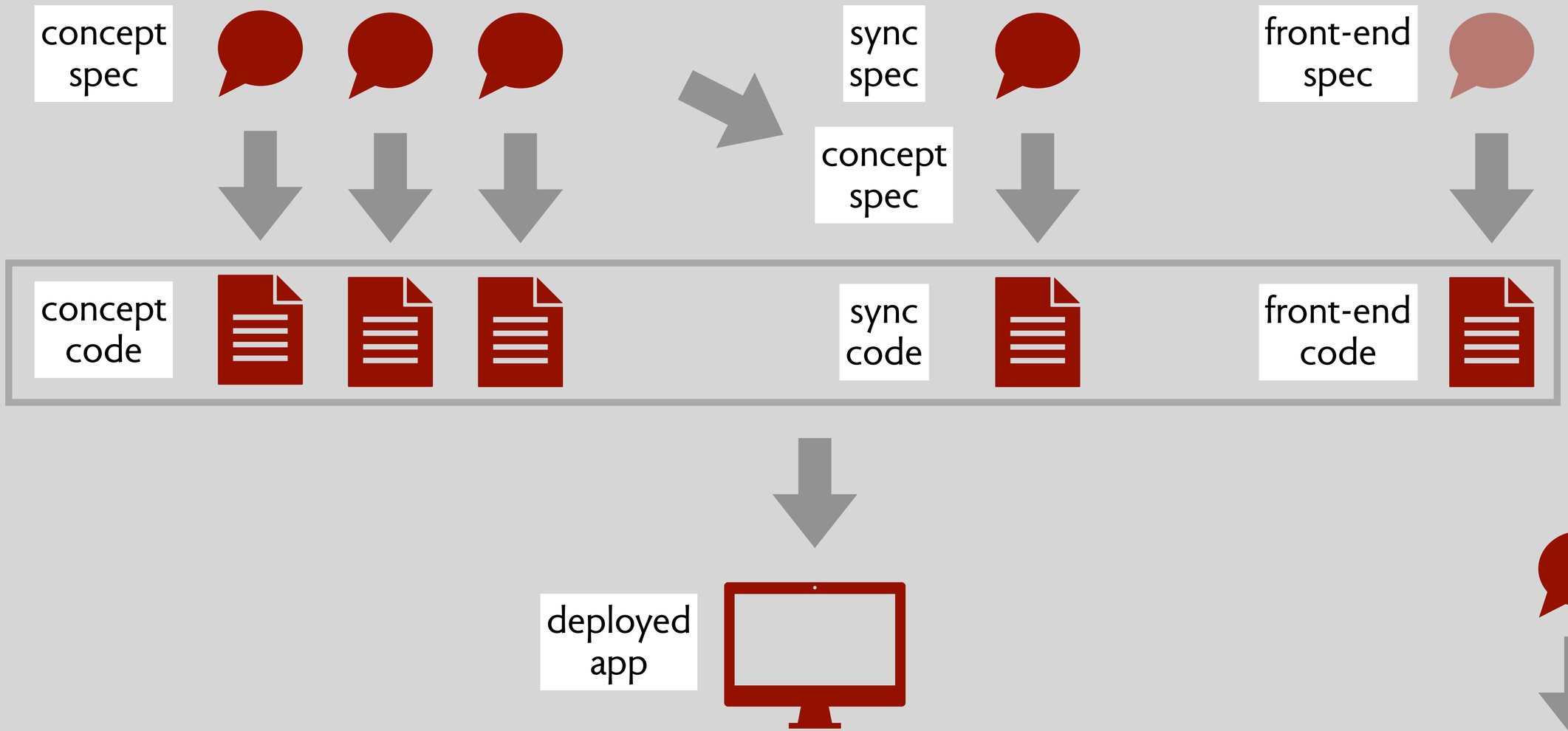
we have Upvote.upvote (…) and Notification.notify (…)

sync NotifyAuthorOnUpvote
when Upvote.upvote (item)
where author of item is user in Post concept
then Notification.notify (user, item + " upvoted")

concept Upvote [User, Item]

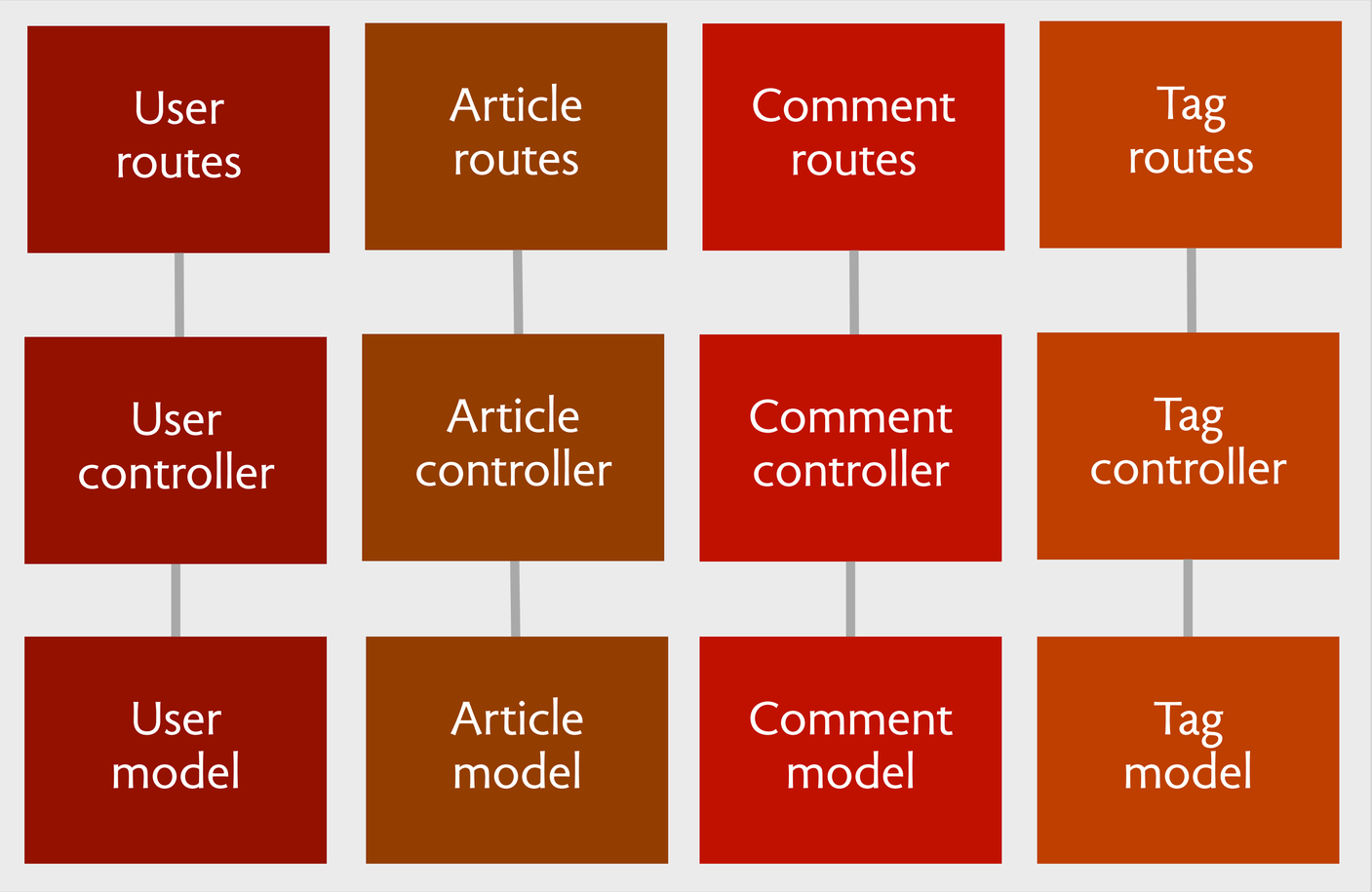
state
a set of Votes with
  a voter User
  a target Item

exploiting concept modularity to generate code



concepts aren't objects

back to RealWorld



using a concept design lens, some problems become evident

they don't cover concepts that lack objects (eg, following, favoriting)

each stack doesn't fully encapsulate its functionality

comparing object (class) to concept

	functionality contained	objects involved	genericity of references	modularity summary
Tag Object	tag format, but not adding or deleting (which is in Article object)	just tag objects	none: reference to Tag in Article is explicit	not really modular
Tag Concept	all functions associated with tags	tags and items tagged	fully polymorphic: tagged item can be any type	fully modular & reusable

a modularity exercise

can you design a concept for URL shortening (like tinyurl, eg)?

concept UrlShortening
purpose shorter or more memorable way to link
principle after create generates a short url, lookup will return the original url
state
    a set of Shortenings with
        a targetUrl String
        a shortUrl String
actions
    create (shortUrlBase, targetUrl: String): (shortUrl: String)
        pick any shortUrl of the form shortUrlBase/foo that has not been used
        return it and create a shortening for it
    lookup (shortUrl: String): (targetUrl: String)
        requires some shortening with shortUrl
        effect returns targetUrl corresponding to it
    delete (shortUrl: String)
        requires some shortening with shortUrl
        effect removes the shortening

purpose

principle

actions

state

what other features might a URL shortening service offer?

user-provided short url suffix

expire short after some time

track uses of short url

require authenticated users

...

a more minimal shortening concept

concept UrlShortening
purpose shorter or more memorable way to link
principle after create generates a short url, lookup will return the original url
state
    a set of Shortenings with
        a targetUrl String
        a shortUrl String
actions
    register (shortUrlSuffix, shortUrlBase, targetUrl: String): (shortUrl: String)
    lookup (shortUrl: String): (targetUrl: String)
    delete (shortUrl: String)

a concept for generating nonces

concept NonceGenerator [Context]
purpose generate unique strings within a context
principle each generate returns a string not returned before for that context
state
    a set of Contexts with
        a used set of Strings
actions
    generate (context: Context) : (nonce: String)
        return a nonce that is not already used by this context
        (one possible implementation is to just increment a counter,
        and the used set is then implicit in the current counter value)

a concept for expiring resources

concept ExpiringResource [Resource]
purpose expire resources automatically to manage costs
principle after setting expiry for a resource, the system will expire it after given time
state
    a set of Resources with
        an expiry DateTime
actions
    setExpiry (resource: Resource, seconds: Int)
        associate with resource an expiry that is now plus seconds
        if resource already has an expiry, override it
    system expireResource () : (resource: Resource)
        requires expiry of resource is before the current time
        forget resource and its expiry

a concept for tracking visits

concept WebAnalytics
purpose track visits to links
principle after a series of visits to various urls, can see stats on which and when visited
state
    a urls set of String // check this
    a set of Visits with
        a url string
        a DateTime
        an ip String // originating IP address of request
actions
    register (url: String)
    visit (url: String, date: DateTime, ip: String)

some syncs for set ups

```
sync generateNonce
when Web.request (method: "shortenUrl", shortUrlBase)
then NonceGenerator.generate (context: shortUrlBase)

sync registerShort
when Web.request (method: "shortenUrl", targetUrl, shortUrlBase)
  NonceGenerator.generate (): (nonce)
then UrlShortening.register (shortUrlSuffix: nonce, shortUrlBase, targetUrl)

sync registerAnalytics
when UrlShortening.register (): (shortUrl)
then WebAnalytics.register (url: shortUrl)

sync setExpiry
when UrlShortening.register (): (shortUrl)
then ExpiringResource.setExpiry (resource: shortUrl, seconds: 3600)
```

syncs for lookup and expiry

```
sync lookupSync1
when
    Web.request (method = "get", url)
then
    UrlShortening.lookup  (shortUrl: url)

sync lookupSync2
when
    Web.request (method = "get", url)
    UrlShortening.lookup  (): (targetUrl)
then
    Web.redirect (targetUrl)
```

```
sync deleteOnExpiry
when
    ExpiringResource.expireResource () : (resource: shortUrl)
then
    UrlShortening.delete  (shortUrl)
```

conclusions

top predictions you should never make

1. LLMs won't ever be able to do that

2. X is too corrupt to get elected

3. The stock market is going to crash this year

so why fix the structure of software?

better structure will amplify the benefits of advances in LLMs
as LLMs get better, our approach will be even more effective

better structure enables scaling
to the max that current LLMs can support

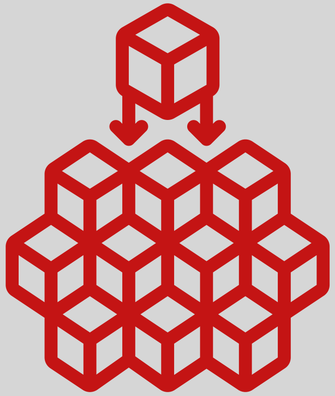
better structure reduces costs
fewer tokens in LLM context saves money and time

better structure improves security
can audit a module and ensure it doesn't get modified

the benefits concepts bring

initial motivations

better UX
clarity & power



modularity
in design & code

a design language
bridging roles too

a place for design
concept-specific issues

what may matter as much or more

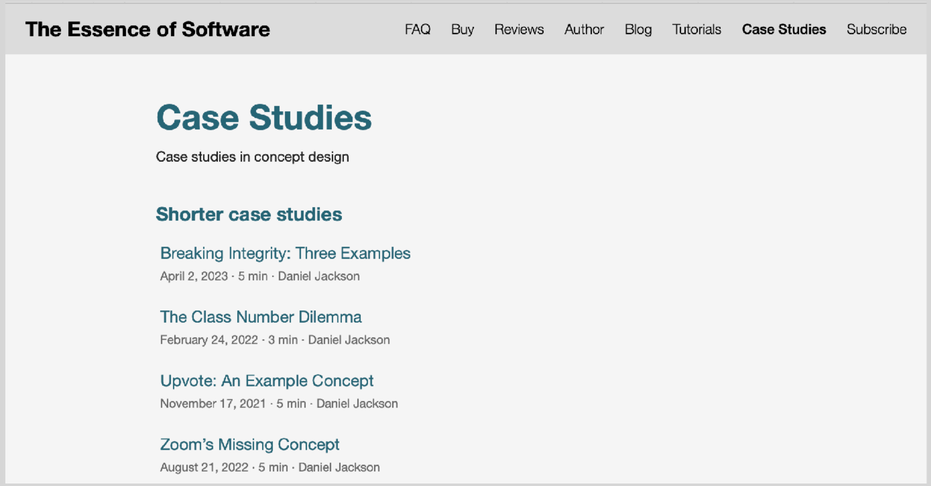
for more information

What You See Is What It Does:
A Structural Pattern for Legible Software

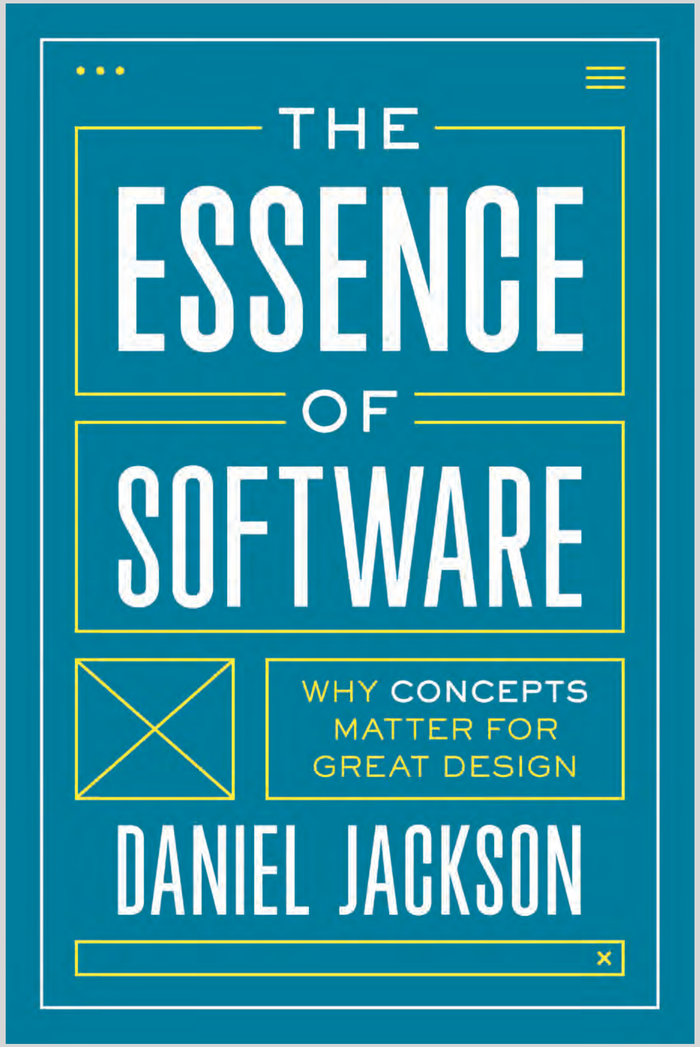
Eagon Meng
eagon@mit.edu
MIT CSAIL
Cambridge, MA, USA

Daniel Jackson
dnj@mit.edu
MIT CSAIL
Cambridge, MA, USA

paper appearing any day now



website: essenceofsoftware.com



book about concept design