how to become a great software designer
Putin Threatens Takeover of Western Countries

By Anton Troianovsky

Washington Post

Russian Leader Facing Generation-Defining Economic Crisis

I have no doubt that these sanctions would have been implemented no matter what," Mr. Putin said in televised remarks on Thursday, arguing that his interdiction in Ukraine served merely as a pretext for the West to try to wreck Russia's economy. "Just as we overcome these difficulties in years past, we will overcome them now, too."

But the sanctions imposed in the past two weeks since the invasion, which has come with multinationals companies and has cost more than $5 billion in cash payments. The economic impact will be significant, and it will affect the lives of millions of people. Continued on page 3.

/NEW YORK TIMES

photos by Brendan Hoffman

Grief in Kyiv

The Ukraine flag is raised over a memorial at the site of the Lychakiv Cemetery in Lviv along with a fellow soldier, Ivan Koverznev.

likely will not follow suit because they have already lost trillions of dollars in stock market value. The S&P 500, down 3.7% thus far in 2022, has fallen more than 20% from its record highs in January. Continued on page 2.

/FOSSIL CREMATION}

Grief in Kyiv

The Ukraine flag is raised over a memorial at the site of the Lychakiv Cemetery in Lviv along with a fellow soldier, Ivan Koverznev.

likely will not follow suit because they have already lost trillions of dollars in stock market value. The S&P 500, down 3.7% thus far in 2022, has fallen more than 20% from its record highs in January. Continued on page 2.
how to build software?
two problems to solve
functional structures
functional structures

implementation structures
is it usable? effective? fit for purpose?

functional structures

implementation structures
functional structures

is it usable? effective? fit for purpose?

is it robust? performant? maintainable?

implementation structures
functional structures

implementation structures

software design

is it usable? effective? fit for purpose?

is it robust? performant? maintainable?
functional structures

is it usable? effective? fit for purpose?
software design

is it robust? performant? maintainable?
software engineering

implementation structures
The rôle of a formal functional specification is simply to act as a logical firewall between two completely different concerns… The **pleasantness problem** concerns the question whether a system meeting such-and-such a formal functional specification would satisfy our needs, meet our expectations and fulfill our hopes. The **correctness problem** concerns the question whether a given design meets such-and-such a formal functional specification.

Dijkstra, EWD952 (1986)
Conceptual integrity is the most important consideration in system design.

Fred Brooks, Mythical Man Month (1975)
I am more convinced than ever. Conceptual integrity is central to product quality.

Fred Brooks, Mythical Man Month (1975)

Conceptual integrity is the most important consideration in system design.

Conceptual integrity is the most important consideration in system design.

Fred Brooks, Mythical Man Month (1975)

I am more convinced than ever. Conceptual integrity is central to product quality.


The essence of a software entity is a construct of interlocking concepts... I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it...

No Silver Bullet (1986)
When you go to design a house, you talk to an architect first, not an engineer. Why is this?
When you go to design a house, you talk to an architect *first*, not an engineer. Why is this?

Because the criteria for what makes a good building fall outside the domain of engineering.

*A Software Design Manifesto*

*Mitchell Kapor, 1996*

paraphrasing slightly
When you go to design a house you talk to an architect first, not an engineer. Why is this?

Because the criteria for what makes a good building fall outside the domain of engineering.

Similarly, in computer programs, the selection of the various components and elements of the application must be driven by the conditions of use. How is this to be done? By software designers.

A Software Design Manifesto
Mitchell Kapor, 1996
paraphrasing slightly
what makes great designers?
inspiration = talent + process
design by inspiration

inspiration = talent + process

the design thinking process

empathize  define  ideate  prototype  test
inspiration = talent + process

the design thinking process

empathize → define → ideate → prototype → test

problem at hand defines context
design by inspiration

inspiration = talent + process

the design thinking process

empathize → define → ideate → prototype → test

problem at hand defines context

is there any great design that is the result of a great design process?
design by expertise

expertise = patterns + design moves

familiar structures    familiar steps
The well-kept secret behind the most successful designs is that they are less new than they appear. Invariably, it turns out that, in some form or other, the designer has done the same thing before. — AFD
design by expertise

**expertise = patterns + design moves**

familiar structures    familiar steps

The well-kept secret behind the most successful designs is that they are less new than they appear. Invariably, it turns out that, in some form or other, the designer has done the same thing before. — AFD
What are the elements of software design?
Jackson structured programming ([wikipedia.org](https://en.wikipedia.org/wiki/Jackson_programming))

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

**danielnicholas** 63 days ago [-]

If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing context-free structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized input and output.

- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] [https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...](https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...)

**ob-nix** 63 days ago [-]

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

**CraigJPerry** 63 days ago [-]

This is referenced(1) as a core inspiration in the preface to "How to Design Programs" but i never researched it further because i've found the "design recipes" approach in htdp to be pretty solid in real-life problems.
If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare’s explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I’d point to these ideas as worth knowing:

- There’s a class of programming problem that involves traversing context-free structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized input and output.

- There are some archetypal problems that, however you code, can’t be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It’s why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i’ve found the “design recipes” approach in htdp to be pretty solid in real-life problems.
Jackson structured programming (wikipedia.org) [post]

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

**danielnicholas** 63 days ago [-]

If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing context-free structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized input and output.

- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

**ob-nix** 63 days ago [-]

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

**CraigJPerry** 63 days ago [-]

This is referenced(1) as a core inspiration in the preface to "How to Design Programs" but i never researched it further because i've found the "design recipes" approach in htdp to be pretty solid in real life problems.
If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing input structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized it.
- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.
- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.
- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/soq/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i’ve found the “design recipes” approach in htdp to be pretty solid in real-life problems.
If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing a data structure. Tasks can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized input and output.

- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtrack as the standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i've found the "design recipes" approach in htdp to be pretty solid in real life problems.
If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing input structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized it.
- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.
- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.
- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtrack standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i’ve found the “design recipes” approach in htdp to be pretty solid in real-life problems.
If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized it.
- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.
- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.
- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtrack as the standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i've found the “design recipes” approach in htdp to be pretty solid in real life problems.
Jackson structured programming

You might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift.

I’d point to these ideas as worth knowing:

- There are some archetypal problems that, however you code, can’t be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It’s why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn’t heard about the method before.

If I remember correctly did the book clearly point out backtrack as the standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

This is referenced(1) as a core inspiration in the preface to “How to Design Programs” but i never researched it further because i’ve found the “design recipes” approach in htdp to be pretty solid in real-life problems.
Jackson structured programming (wikipedia.org)

You might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift

user: danielnicholas
created: 63 days ago
karma: 11

I'd point to these ideas as worth knowing:

- The archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as the standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

CraigJPerry 63 days ago

This is referenced[1] as a core inspiration in the preface to "How to Design Programs" but I never researched it further because I've found the "design recipes" approach in htdp to be pretty solid in real-life problems.
a concept has a name
a concept has a name

same concept in HackerNews, NYTimes comment section, StackOverflow, etc
A concept has a name.

Concept Upvote

Same concept in HackerNews, NYTimes comment section, StackOverflow, etc.

John
Boston | Oct. 27

To protect children? Seems far more likely it's yet one more way to extract personal information to feed the insatiable advertising machines.

1 Reply 143 Recommend Share
a concept has a purpose

**concept** Upvote

**purpose** rank items by popularity

This is homework and I'm having a hard time understanding the definitions of the objects:

```plaintext
sig Library {
    patrons : set Person,
    on_shelves : set Book,
}
```
<table>
<thead>
<tr>
<th>concept</th>
<th>Upvote</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>rank items by popularity</td>
</tr>
<tr>
<td>concept</td>
<td>Reaction</td>
</tr>
<tr>
<td>purpose</td>
<td>send reactions to author</td>
</tr>
</tbody>
</table>

This is homework and I'm having a hard time understanding the definitions of the objects:

```
sig Library {  
    patrons : set Person,  
    on_shelves : set Book,  
}  
```

Daniel: I think we should organize a software concepts forum.
This is homework and I'm having a hard time understanding the definitions of the objects:

```plaintext
sig Library {
  patrons : set Person,
  on_shelves : set Book,
}
```

Today:

Daniel: I think we should organize a software concepts forum.
concept Upvote

purpose rank items by popularity

state
votes: User -> set Vote
for: Vote -> one Item
Upvote, Downvote: set Vote
rank: Item -> one Int
a concept has a state

**concept** Upvote

**purpose** rank items by popularity

**state**
votes: User -> set Vote for: Vote -> one Item
Upvote, Downvote: set Vote rank: Item -> one Int

include in state **only** what’s needed for the concept’s own computations
**Concept** Upvote

**Purpose** rank items by popularity

**State**
- votes: User -> set Vote
- for: Vote -> one Item
- Upvote, Downvote: set Vote
- rank: Item -> one Int

Include in state **only** what's needed for the concept's own computations

- track users to prevent duplicate voting

Diagram:
- User -> votes to Vote
- Vote -> for Item
- Int: rank
- Upvote, Downvote
A concept has a state like bounded context in DDD, but even more localized.

**Concept** Upvote

**Purpose** Rank items by popularity

**State**
- Votes: User -> set Vote
  - For: Vote -> one Item
- Upvote, Downvote: set Vote
- Rank: Item -> one Int

Include in state **only** what's needed for the concept's own computations.

Track users to prevent duplicate voting.

Like bounded context in DDD, but even more localized.
a concept has actions

**concept** Upvote

**purpose** rank items by popularity

**state**
votes: User $\rightarrow$ set Vote
for: Vote $\rightarrow$ one Item
Upvote, Downvote: set Vote
rank: Item $\rightarrow$ one Int

**actions**
upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)
concept Upvote

purpose rank items by popularity

state
votes: User -> set Vote
for: Vote -> one Item
Upvote, Downvote: set Vote
rank: Item -> one Int

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

actions capture the concept behavior in full
a concept has actions

**concept** Upvote

**purpose** rank items by popularity

**state**
votes: User -> set Vote
for: Vote -> one Item
Upvote, Downvote: set Vote
rank: Item -> one Int

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

actions capture the concept
behavior in full

downvote (i: Item, u: User)
// no existing Downvote for i in u.votes
// remove any Upvote for i from u.votes
// add a Downvote for i in u.votes
// update i.rank ...
a concept has actions

**concept** Upvote

**purpose** rank items by popularity

**state**
votes: User -> set Vote
for: Vote -> one Item
Upvote, Downvote: set Vote
rank: Item -> one Int

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

actions capture the concept behavior in full

**downvote** (i: Item, u: User)
  // no existing Downvote for i in u.votes
  // remove any Upvote for i from u.votes
  // add a Downvote for i in u.votes
  // update i.rank ...

succinct specification as actions on states
VDM (1986)
Z (1992)
Larch (1993)
Alloy (2006)
**a concept catalog entry**

**concept Upvote**

**related concepts**
Recommendation, Reaction, ...

**design variants**
downvote as unvote
use age in ranking
weigh downvotes more

**typical uses**
social media posts
comments on articles
Q&A responses

**known issues**
preventing double votes
(require login, use IP address, save cookie)
saving storage space
(freeze old posts and from user info)

**often used with**
Karma, Session, ...
composing concepts by synchronization
how to extend behavior?

**concept** Upvote

**actions**
upvote \((u: \text{User}, i: \text{Item})\)
downvote \((u: \text{User}, i: \text{Item})\)
unvote \((u: \text{User}, i: \text{Item})\)
suppose I want this behavior:
you can’t downvote an item until you’ve received N upvotes on your own items
suppose I want this behavior:
you can’t downvote an item until you’ve received N upvotes on your own items

define a new concept!
a hint: not just used by Upvote
how to extend behavior?

**concept** Upvote

**actions**
- upvote \((u: \text{User}, i: \text{Item})\)
- downvote \((u: \text{User}, i: \text{Item})\)
- unvote \((u: \text{User}, i: \text{Item})\)

suppose I want this behavior:
you can’t downvote an item until you’ve received \(N\) upvotes on your own items

**define a new concept!**
a hint: not just used by Upvote

**concept** Karma

**purpose** privilege good users

**state**
- karma: \(\text{User} \rightarrow \text{one Int}\)
- contribs: \(\text{User} \rightarrow \text{set Item}\)

**actions**
- contribute \((u: \text{User}, i: \text{Item})\)
- reward \((u: \text{User}, r: \text{Int})\)
- permit \((u: \text{User}, r: \text{Int})\)
**concept** Upvote

**actions**
- upvote (u: User, i: Item)
- downvote (u: User, i: Item)
- unvote (u: User, i: Item)

**concept** Karma

**actions**
- contribute (u: User, i: Item)
- reward (i: Item, r: Int)
- permit (u: User, r: Int)
**concept** Upvote

**actions**
- upvote (u: User, i: Item)
- downvote (u: User, i: Item)
- unvote (u: User, i: Item)

**concept** Karma

**actions**
- contribute (u: User, i: Item)
- reward (i: Item, r: Int)
- permit (u: User, r: Int)

when upvote (u, i) and i in u'.contribs also reward (u', 10)
**concept** Upvote

**actions**
upvote \((u: \text{User}, i: \text{Item})\)
downvote \((u: \text{User}, i: \text{Item})\)
unvote \((u: \text{User}, i: \text{Item})\)

**when** upvote \((u, i)\)
and \(i \in u'.\text{contribs}\)
also reward \((u', 10)\)

**concept** Karma

**actions**
contribute \((u: \text{User}, i: \text{Item})\)
reward \((i: \text{Item}, r: \text{Int})\)
permit \((u: \text{User}, r: \text{Int})\)

**when** downvote \((u, i)\)
also permit \((u, 20)\)
concept Upvote

concept Karma
**concept** Upvote

**concept** Karma

contrib (Alice, post1)

contrib (Bob, post2)
A concept is Upvote.

Bob upvoted post1.

When Bob upvotes, the contributor also gets rewarded.

Contrib (Alice, post1)

Concept Karma

Contrib (Bob, post2)

Reward (Alice, 10)
concept Upvote

upvote (Bob, post1)
upvote (Carol, post1)

when upvote (u, i)
also reward (u, 10)

concept Karma

contrib (Alice, post1)
contrib (Bob, post2)

reward (Alice, 10)
**concept** Upvote

- upvote (Bob, post1)
- upvote (Carol, post1)
- downvote (Alice, post2)

**concept** Karma

- contrib (Alice, post1)
- contrib (Bob, post2)
- when upvote (u, i)
  - also reward (u, 10)
- reward (Alice, 10)
**concept** Upvote

- upvote (Bob, post1)
- upvote (Carol, post1)
- downvote (Alice, post2)

- when upvote (u, i) also reward (u, 10)
- when downvote (u, i) also permit (u, 20)

**concept** Karma

- contrib (Alice, post1)
- contrib (Bob, post2)
- reward (Alice, 10)
- permit (Alice, 20)
**concept** Upvote

- upvote (Bob, post1)
- upvote (Carol, post1)
- downvote (Alice, post2)

**when** upvote (u, i)
**also** reward (u, 10)

**concept** Karma

- contrib (Alice, post1)
- contrib (Bob, post2)

- reward (Alice, 10)
- permit (Alice, 20)

**when** downvote (u, i)
**also** permit (u, 20)

Composition uses event sync from Hoare's CSP
concept Upvote

upvote (Bob, post1)

upvote (Carol, post1)

downvote (Alice, post2)

when upvote (u, i)
also reward (u, 10)

when downvote (u, i)
also permit (u, 20)

concept Karma

contrib (Alice, post1)

contrib (Bob, post2)

when upvote (u, i)
reward (Alice, 10)

reward (Alice, 10)

permit (Alice, 20)

composition uses event sync from Hoare’s CSP

no concept coupling concepts preserve properties
design moves
mechanical analogs
three pairs of design moves
three pairs of design moves
unify
merge
split

three pairs of design moves

unify
specialize

merge
split
three pairs of design moves

1. **Merge** ↔ **Split**
2. **Unify** ↔ **Specialize**
3. **Loosen** ↔ **Tighten**
split-merge: tradeoff simplicity/flexibility
split-merge: tradeoff simplicity/flexibility

photocopier

split

printer + scanner
split-merge: tradeoff simplicity/flexibility

- split
  - photocopier
  - printer + scanner

- merge
  - emergency flashlight
  - flashlight + battery + charger
unify-specialize: tradeoff simplicity/specificity
**unify-specialize**: tradeoff simplicity/specificity

- set of wrenches
- adjustable wrench
**unify-specialize:** tradeoff simplicity/specificity

- **Set of wrenches**
  - **Adjustable wrench**
- **Macro lens**
  - **General-purpose lens**
tighten-loosen: tradeoff automation/flexibility
**tighten-loosen:** tradeoff automation/flexibility

- light pull / door lock
- airplane toilet lock
tighten-loosen: tradeoff automation/flexibility

- light pull / door lock
- airplane toilet lock
- dimmers with separate controls
- rotary dimmer switch
successful design moves in software
split: emergence of a concept in Keynote
split: emergence of a concept in Keynote

full screen toggle emerges as partial concept (c. 2010?)
split: emergence of a concept in Keynote

full screen toggle emerges as partial concept (c. 2010?)

play-in-window option now an independent concept (2021)
split: emergence of a concept in Keynote

full screen toggle emerges as partial concept (c. 2010?)

play-in-window option now an independent concept (2021)
merge: a simple & clever new shortening concept

yellkey

enter url and length of time for key to exist.

https://dnj.photo

5 minutes

generate yellkey

your key is: move.

go to www.yellkey.com/move to use.

one concept, allows familiar words as short URLs
merge: a simple & clever new shortening concept

yellkey

enter url and length of time for key to exist.

https://dnj.photo

5 minutes

generate yellkey

your key is: move.

go to www.yellkey.com/move to use.

one concept, allows familiar words as short URLs
unify: subsuming access control in MIT’s Moira
unify: subsuming access control in MIT's Moira
unify: subsuming access control in MIT’s Moira
can toggle mailing list attribute

can create admin list with no login users!

<table>
<thead>
<tr>
<th>List Name</th>
<th>dnj-play1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>none</td>
</tr>
<tr>
<td>Attributes</td>
<td>active, moira mailing list</td>
</tr>
<tr>
<td>Permissions</td>
<td>private, visible</td>
</tr>
<tr>
<td>Last Modified</td>
<td>by dnj with moira on 22-mar-2022 09:39:00</td>
</tr>
</tbody>
</table>

**Members**

<table>
<thead>
<tr>
<th>Add Member:</th>
<th>Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leave List:</td>
<td>Remove Me</td>
</tr>
</tbody>
</table>

**MIT Users**

- Daniel Jackson (dnj)
- daniel@dnj.photo

**Administrators**

<table>
<thead>
<tr>
<th>Owner: dnj-play2 (List)</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Owner:</td>
<td>Change</td>
</tr>
<tr>
<td>Add Administrator:</td>
<td>Add</td>
</tr>
<tr>
<td>Leave Owner List:</td>
<td>Remove Me</td>
</tr>
</tbody>
</table>

**MIT Users**

- Daniel Jackson (dnj)
tighten: metadata and publication status in Hugo
Tighten: metadata and publication status in Hugo

Squarespace: can schedule blog posts but not other pages, and can change pub date independently (only affects order)
tighten: metadata and publication status in Hugo

Squarespace: can schedule blog posts but not other pages, and can change pub date independently (only affects order)

Hugo: any page can have date field to schedule, just set date in future
Squarespace: can schedule blog posts but not other pages, and can change pub date independently (only affects order)

Hugo: any page can have date field to schedule, just set date in future

---
author: "Daniel Jackson"
title: "The Class Number Dilemma"
date: "2022-02-24"
description: "A Concept Example in Everyday Life"
summary: "Who knew such a simple thing could be so challenging?"
---
tighten: label and trash concepts in Gmail
tighten: label and trash concepts in Gmail
tightly label and trash concepts in Gmail

- A label
- Show messages with label hacking
tighten: label and trash concepts in Gmail

- A label
- Also implemented as a label
- Show messages with label hacking
tighten: label and trash concepts in Gmail

- Show messages with label hacking
- Also implemented as a label
- Label
- Trash
**concept** trash

**purpose** undo deletion

**structure**
trash: `set Item`

**actions**
delete (i: Item)
restore (i: Item)
empty ()
**Purpose** undo deletion

**Concept** trash

**Structure**
trash: \texttt{set Item}

**Actions**
delete \((i: \text{Item})\)
restore \((i: \text{Item})\)
empty ()

---

**Purpose** organize with overlapping

**Concept** label

**Structure**
labels: \text{Item} \rightarrow \texttt{set Label}

**Actions**
add \((i: \text{Item}, l: \text{Label})\)
remove \((i: \text{Item}, l: \text{Label})\)
find \((ls: \texttt{set Label}, \texttt{out is: set Item})\)
**concept** trash

**purpose** undo deletion

**structure**
trash: set Item

**actions**
delete (i: Item)
restore (i: Item)
empty ()

---

**concept** label

**purpose** organize with overlapping

**structure**
labels: Item -> set Label

**actions**
add (i: Item, l: Label)
remove (i: Item, l: Label)
find (ls: set Label, out is: set Item)

---

**when** delete (i)
**also** add (i, ‘trash’)
integrating these concepts is tricky

click on trash
integrating these concepts is tricky

click on trash

filter on todo label
integrating these concepts is tricky

- click on trash
- filter on todo label
integrating these concepts is tricky

click on trash

filter on todo label

filter on todo and trash
integrating these concepts is tricky
integrating these concepts is tricky

1. Click on trash

2. Filter on todo label

3. Filter on todo and trash

4. Filter on something else
a beautiful (but tricky) synergy
a beautiful (but tricky) synergy

folder sortable by volume!
a beautiful (but tricky) synergy

folder sortable by volume!

new in Lion (2011)
a beautiful (but tricky) synergy

Folder sortable by volume!

new in Lion (2011)
loosen: expert control in ProCamera

focus and exposure points aligned
loosen: expert control in ProCamera

focus and exposure points aligned

focus and exposure points separated
loosen: expert control in ProCamera

focus and exposure points aligned

focus and exposure points separated
loosen: expert control in ProCamera

focus and exposure points aligned

focus and exposure points separated
loosen: expert control in ProCamera

Focus

Exposure

focal point

exp point

focus and exposure points aligned

focus and exposure points separated

loosen
design moves in response to problems
aspect ratio in Fujifilm cameras
a lovely camera fuji x100
complex menu system: image quality setting
complex menu system: image quality setting
complex menu system: image quality setting
aspect ratio
aspect ratio
image size setting
image size setting

- L 3:2 664
- L 16:9 681
- M 3:2 707
- M 16:9 719
- M 1:1 734
- S 3:2 746

4896x3264 (16M) 664 FRAMES
image size setting
image size setting
non-standard ratio + raw?
problem #1: no non-standard ratio unless also save JPG!

raw image showing non-destructive aspect ratio crop
problem #2: very few ratio options

Fuji, give us 4:3, 5:4, and 6:7 aspect ratios on X-series cameras

636 have signed. Let's get to 1,000!

At 1,000 signatures, this petition is more likely to be featured in recommendations!
problem #2: very few ratio options

change.org

Petition details  Comments  Updates

Fuji, give us 4:3, 5:4, and 6:7 aspect ratios on X-series cameras

636 have signed. Let's get to 1,000!

At 1,000 signatures, this petition is more likely to be featured in recommendations!
aspect ratio is not a concept
merged into JPEG image size concept
so cannot be controlled independently
I call this “overloading by piggybacking”

solution: split concepts
would allow ratio change to raws without JPEGs
would avoid combinatoric explosion of options
aspect ratio is not a concept
merged into JPEG image size concept
so cannot be controlled independently
I call this “overloading by piggybacking”

solution: split concepts
would allow ratio change to raws without JPEGs
would avoid combinatoric explosion of options
message filters
in apple mail
how many ways to filter messages?
how many ways to filter messages?

search for a message
how many ways to filter messages?

- search for a message
- create a rule
how many ways to filter messages?

search for a message

create a rule

define a smart folder
how many ways to filter messages?

search for a message

create a rule

define a smart folder

search options
how many ways to filter messages?

search for a message

create a rule

define a smart folder

search options

rule options
how many ways to filter messages?

search for a message

create a rule

define a smart folder

search options

rule options

smart folder options
search, rule and smart folder
all include their own specialized concepts
incomparable features, different UIs

unify in a single message filter concept
include “create folder from search”, eg
search, rule and smart folder all include their own specialized concepts incomparable features, different UIs

unify in a single message filter concept include “create folder from search”, eg
sticky hands

in zoom
likes in facebook
are angry reactions upvotes?
are angry reactions upvotes?

**concept** Upvote

**purpose** rank items by popularity

**actions**
- upvote (u: User, i: Item)
  ...

are angry reactions upvotes?

- **concept** Upvote  
  **purpose** rank items by popularity  
  **actions**  
  upvote (u: User, i: Item)

- **concept** Reaction  
  **purpose** convey emotion to author  
  **actions**  
  reactAngry (u: User, i: Item)
are angry reactions upvotes?

concept Upvote
purpose rank items by popularity
actions
upvote (u: User, i: Item)

concept Reaction
purpose convey emotion to author
actions
reactAngry (u: User, i: Item)

unwanted sync?
a facebook split
a facebook split

Like  Comment

split

Approve  React  Comment
a facebook split
event deletion in google calendar
Despite some erroneous messages sent to this list accidentally, Kanit’s talk is happening! Please join us on Monday.
seminar announced as email to listserv with attached calendar event
seminar announced as email to listserv with attached calendar event

event installed automatically in user's calendar
HCI Seminar announced as email to listserv with attached calendar event.

Event installed automatically in user's calendar.

User deletes event from calendar.
HCI Seminar announced as email to listserv with attached calendar event.

- Event installed automatically in user's calendar.
- User deletes event from calendar.
- Cancellation email automatically sent to other invitees.
Canceling and deleting events in the Google Calendar mobile app is similar to on a desktop.

1. First, open Google Calendar.
2. Tap on the event you wish to cancel.
3. Press on the three dots in the top right corner of the event window.
4. Select Delete.
5. Tap Delete event. Google Calendar will send a cancellation email to the guests.

Mar 22, 2021

https://wpamelia.com Blog

How to Cancel an Event in Google Calendar - Amelia booking ...
a long time problem in iCal too
how to delete spam calendar events?
diagnosis?
concept calendar

purpose record engagements

actions
   createEvent (...): Event
   deleteEvent (e: Event)
   ...


diagnosis?
concept calendar
purpose record engagements
actions
   createEvent (…): Event
   deleteEvent (e: Event)
   ...

concept invitation
purpose coordinate events
actions
   accept (e: Event)
   decline (e: Event)
   ...

diagnosis?
concept calendar
purpose record engagements
actions
  createEvent (...): Event
  deleteEvent (e: Event)
  ...

concept invitation
purpose coordinate events
actions
  accept (e: Event)
  decline (e: Event)
  ...

---
diagnosis?
concept calendar
purpose record engagements
actions
  createEvent (...): Event
  deleteEvent (e: Event)
...

concept invitation
purpose coordinate events
actions
  accept (e: Event)
  decline (e: Event)
...

unwanted sync!
apple’s solution

Are you sure you want to delete this event?
Deleting this event will notify the organizer that you’re declining the event and deleting it from your calendar. You can't undo this action.

[Cancel] [Delete and Don't Notify] [Delete and Notify]

resolution to design problem
make sync optional
Are you sure you want to delete this event?
Deleting this event will notify the organizer that you’re declining the event and deleting it from your calendar. You can't undo this action.

Cancel  Delete and Don't Notify  Delete and Notify

resolution to design problem
make sync optional

loosen
takeaways
takeaways

structure your software design with concepts
inventory the concepts, identify the critical ones
see if you can describe them fully independently
then formulate interactions as synchronizations
structure your software design with concepts
inventory the concepts, identify the critical ones
see if you can describe them fully independently
then formulate interactions as synchronizations

apply design moves to explore new options
never a panacea, always a tradeoff
structure your software design with concepts
inventory the concepts, identify the critical ones
see if you can describe them fully independently
then formulate interactions as synchronizations

apply design moves to explore new options
never a panacea, always a tradeoff

software concepts as patterns
only hinted at this, but equally important
don’t reinvent the wheel!
express your design as sync of familiar concepts?
unused slides
Dropbox & the case of the disappearing files
making space: list big files and delete the ones you don’t recognize
making space: list big files and delete the ones you don’t recognize.

All our files suddenly disappeared from the Dropbox folder! Did someone delete them?

very unhappy colleagues
Someone accidentally deleted thousands of files in my company Dropbox: how can I quickly undelete them?
Ava is a party planner
Ava is a party planner

Bella is having a party
Ava is a party planner

Bella is having a party
Ava is a party planner

Bella is having a party
Ava is a party planner

Bella is having a party
Ava is a party planner

Bella is having a party
Ava is a party planner

Bella is having a party

does the name change for Ava too?
Ava is a party planner

Bella is having a party

---

**does the name change for Ava too?**

**answer: it depends**

if Ava just shares Bella Plan with Bella and Bella renamed the folder, Ava sees no change

**if Ava shared a folder containing Bella Plan then Ava does see the change**
Delete folder?

Are you sure you want to delete **Bella Plan** from the shared folder ‘Bella Party’?
Bella deletes Bella Plan from shared folder Bella Party.

Delete folder?

Are you sure you want to delete Bella Plan from the shared folder ‘Bella Party’?

Cancel   Delete

Remove shared folder?

Are you sure you want to remove the shared folder Bella Party from your Dropbox? This folder will stay shared with any existing members. You can re-add it later.

Cancel   Remove
survey of dropbox users (MIT CS undergrads)

correctly predicting behavior

- delete shared folder results in leaving
- delete shared subfolder removes it

Kelly Zhang
how many users believe Dropbox is structured
how many users believe Dropbox is structured

how Dropbox is actually structured
how many users believe Dropbox is structured

how Dropbox is actually structured
**concept** metadata

**purpose** tag items with properties for lookup

**state**
- properties: Item -> set Property
- key: Property -> one Key
- val: Property -> one Val
**concept** metadata

**purpose** tag items with properties for lookup

**state**
- properties: Item -> set Property
- key: Property -> one Key
- val: Property -> one Val

**concept** folder

**purpose** organize items into disjoint categories

**state**
- contents: Folder -> set (Folder + Item)
- root: one Folder
**concept** metadata

**purpose** tag items with properties for lookup

**state**

properties: Item -> set Property
key: Property -> one Key
val: Property -> one Val

---

**concept** folder

**purpose** organize items into disjoint categories

**state**

contents: Folder -> set (Folder + Item)
root: one Folder
**concept** metadata

**purpose** tag items with properties for lookup

**state**
- properties: Item $\rightarrow$ set Property
- key: Property $\rightarrow$ one Key
- val: Property $\rightarrow$ one Val

**concept** folder

**purpose** organize items into disjoint categories

**state**
- contents: Folder $\rightarrow$ set (Folder + Item)
- root: one Folder

**concept** unix directory

**purpose** organize items into overlapping categories

**state**
- entries: Dir $\rightarrow$ set Entry
- name: Entry $\rightarrow$ one Name
- item: Entry $\rightarrow$ one (File + Dir)
Twitter & the case of the surprised first lady
Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump
Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump

MELANIA TRUMP liked your Tweet
Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump  pic.twitter.com/XiNd2jiLUF
Nov 2, 2015: Twitter changes Favorite (Star) to Like (Heart)
Nov 2, 2015: Twitter changes Favorite (Star) to Like (Heart)

We are changing our star icon for favorites to a heart and we’ll be calling them likes... We know that at times the star could be confusing, especially to newcomers. You might like a lot of things, but not everything can be your favorite. *Twitter press release*
Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump
what concept is this?
what concept is this?

concept Upvote

purpose rank items by popularity
what concept is this?

concept Upvote
purpose rank items by popularity

concept Bookmark
purpose save items to revisit

Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump
what concept is this?

concept: Upvote
purpose: rank items by popularity

concept: Bookmark
purpose: save items to revisit

missing concept!
Andrew Yang would fine gunmakers for deaths caused by their products.

You probably know Andrew Yang wants to give every American $1,000 a month. Something you might not know: He wants to fine gun makers...
Andrew Yang would fine gunmakers for deaths caused by their products.

You probably know Andrew Yang wants to give every American $1,000 a month. Something you might not know: He wants to fine gun... bostonglobe.com

concept: Upvote

purpose: rank items by popularity
The Boston Globe @BostonGlobe 21h

Andrew Yang would fine gunmakers for deaths caused by their products.

Yang would fine gunmakers for deaths caused by their products - Th...
You probably know Andrew Yang wants to give every American $1,000 a month. Something you might not know: He wants to fine gun ...

concept Upvote
purpose rank items by popularity
Twitter adds a new concept (2018)

**Purpose:** Rank items by popularity

**Concept:** Upvote
Twitter adds a new concept (2018)

- **Concept**: Upvote
  - **Purpose**: Rank items by popularity

- **Concept**: Bookmark
  - **Purpose**: Save items to revisit
a new path
inspired by old ideas
user's model
user’s model
designer’s model
designer's model

user's model

user interprets interface
The diagram illustrates the process of converting a design model into an interface that is interpretable by users. It shows two models: one for the designer and another for the user. The designer's model is transformed into an interface that the user interprets. The interface then conveys the design to the user.
user’s model

user interprets interface

interface designer conveys design

programmer implements design

designer’s model

user interprets interface

interface designer conveys design

programmer implements design
designer's model

user's model

psychology

phone with codes
interface design

psychology

designer's model

designer's model

user's model
but what if the system just does the wrong thing?
but what if the system just does the wrong thing?

go this way instead?

user’s model

designer’s model
focus on faithful projection of design model mapping, affordance/signifiers, gulfs

theory turned into practice through heuristics
8 Golden Rules (Shneiderman)
First Principles of Interaction Design (Tognazzini)
10 Usability Heuristics (Nielsen)

result: huge improvements in usability
anyone can design a great UI
If a simple model is not explicitly or implicitly provided, users formulate their own myths about how the system works... The system has to be designed with an explicit conceptual model that is easy enough for the user to learn.

Stuart Card & Thomas Moran (1986)
should facebook concepts be desynchronized too?
should facebook concepts be desynchronized too?

concept Upvote
purpose rank items by popularity
actions
  upvote (u: User, i: Item)
  ...

…
should facebook concepts be desynchronized too?

**concept** Upvote  
**purpose** rank items by popularity  
**actions**  
  upvote (u: User, i: Item)  
  ...

**concept** Reaction  
**purpose** convey emotion to author  
**actions**  
  reactAngry (u: User, i: Item)  
  ...
Should Facebook concepts be desynchronized too?
should facebook concepts be desynchronized too?

**concept** Upvote
**purpose** rank items by popularity
**actions**
upvote (u: User, i: Item)