finding structure in software
we use structure to understand artifacts
villa designs (andrea palladio, c. 1570)
periodic table (mendeleev, c. 1870)
london underground (harry beck, 1933)
experiential structure helps you understand how it behaves not how it’s built
experiential
structure helps you understand how it behaves not how it’s built

modular
components of the structure can be understood independently
experiential
structure helps you understand how it behaves not how it’s built

modular
components of the structure can be understood independently

abstract
internal workings and structure are not shown
what are the elements of software?
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)

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 pref ace 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/sgd/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 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 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 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 was 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 structures can be solved very systematically. HTDP addresses this class, but bases code structure on input structure; JSP synthesized input structure.

- 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 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 structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized a next method.
- 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 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.
You might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift:

- 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 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 htpd to be pretty solid in real life problems.
Jackson structured programming (wikipedia.org)

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

post

^ danielnicholas 63 days ago [-]

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: algebras, functors, monads. Structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized it.

- 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 backtrack - 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.
but what’s a concept? three things it isn’t
but what’s a concept? three things it isn’t

**abstract type, class/object**

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

not limited to built-in types
encapsulate representation
defined by operations alone
but what’s a concept? three things it isn’t

**abstract type, class/object**

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

not limited to built-in types
encapsulate representation
defined by operations alone

what operations can you do on an upvote?
but what’s a concept? three things it isn’t

abstract type, class/object

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

not limited to built-in types
encapsulate representation
defined by operations alone

what operations can you do on an upvote?

corecept lattice

stagnant — natural — constant

running

temporary — reservoir

channel — canal

puddle

maritime

maar — lake — pond — tarn — pool

trickle — runnel — stream — river — rivulet — torrent

lagoon — sea
but what’s a concept? three things it isn’t

abstract type, class/object

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

not limited to built-in types
encapsulate representation
defined by operations alone

what operations can you do on an upvote?

upvotes and downvotes are votes and then what?

concept lattice

- Matrix
- multiply
- add
- invert
- equals

not limited to built-in types
class/object

concept lattice

- stagnant
- natural
- constant
- running
- reservoir
- temporary
- puddle
- maritime
- lagoon
- sea
- lagoon
- pond
- tarn
- pool
- maar
- stream
- river
- rivulet
- torrent
but what’s a concept? three things it isn’t

abstract type, class/object

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

not limited to built-in types
encapsulate representation
defined by operations alone

concept lattice

- stagnant
- natural
- constant
- running
- temporary
- reservoir
- channel
canal
- maritime
- lagoon
sea
- pond
- pool

entity in data model

- User → Vote → Post
- Upvote
- Downvote

what operations can you do on an upvote?

upvotes and downvotes are votes and then what?
but what’s a concept? three things it isn’t

abstract type, class/object

<table>
<thead>
<tr>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>invert</td>
</tr>
<tr>
<td>equals</td>
</tr>
</tbody>
</table>

don’t limited to built-in types
encapsulate representation
defined by operations alone

concept lattice

- stagnant
- natural
- constant
- running
- temporary
- reservoir
- channel
canal
- puddle
- maritime
- maar
- lake
- pond
- tarn
- pool
- trickle
- runnel
- stream
- river
- rivulet
- torrent
- lagoon
- sea

entity in data model

User -> Vote -> Post

votes for Vote

Upvote, Downvote

but concept is in the relationships, not the entities!

what operations can you do on an upvote?

upvotes and downvotes are votes and then what?
a concept has a name
A concept has a name. Same concept in HackerNews, NYTimes comment section, StackOverflow, etc.
a concept has a name

same concept in HackerNews, NYTimes comment section, StackOverflow, etc

Reader Picks

John

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 Flag
This is homework and I'm having a hard time with it. Here are the definitions of the objects:

```plaintext
sig Library {
    patrons : set Person,
    on_shelves : set Book,
}
```
a concept has a **purpose**

**concept** Upvote

**purpose** rank items by popularity

**concept** Reaction

**purpose** send reactions to author

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

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

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

👍 1 😊
A concept has a purpose.

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

**Concept: Reaction**
- Purpose: Send reactions to author

**Concept: Recommendation**
- Purpose: Use prior likes to recommend

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

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

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

👍 1😊
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
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

![Diagram]

- User
- Vote
- Item
- Upvote
- Downvote
- Int
- rank
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.

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.
**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)
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**
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)

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)
Concept: Upvote

**Design Variants**
- Downvote as unvote
- Use age in ranking
- Weigh downvotes more

**Related Concepts**
- Recommendation, Reaction, ...

**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, ...
how to compose concepts?
how to extend behavior?

**concept** Upvote

**actions**
- upvote (u: User, i: Item)
- downvote (u: User, i: Item)
- unvote (u: User, i: Item)
how to extend behavior?

concept Upvote

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

suppose I want this behavior:
you can’t downvote an item until you’ve received N upvotes on your own items
how to extend behavior?

**concept** Upvote

**actions**
upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: 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
how to extend behavior?

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: Upvote

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

Concept: Karma

Purpose: Privilege good users.

State:
- karma: User → one Int
- contribs: User → set Item

Actions:
- contribute (u: User, i: Item)
- reward (u: User, 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)
**concept** Upvote

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

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

**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)

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

**concept** Karma

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

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

concept Karma
**concept** Upvote

**concept** Karma

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

concept Upvote

concept Karma
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)

reward (Alice, 10)
upvote (Bob, post1)
upvote (Carol, post1)
downvote (Alice, post2)

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

reward (Alice, 10)

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

reward (Alice, 10)

concept Upvote

concept Karma
**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)

**when** downvote (u, i)
**also** permit (u, 20)
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)
reward (Alice, 10)
permit (Alice, 20)

composition uses event sync from Hoare’s CSP
concept Upvote

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

concept Karma

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

composition uses event sync from Hoare’s CSP

no concept coupling concepts preserve properties
so what can you do with concepts?
characterize apps and families

Facebook
characterize apps and families
characterize apps and families
characterize apps & families

- text editor
  - line
  - character set
  - markup

- word processor

- desktop publishing app

waffling about concepts and fine points of design
characterize apps & families

- text editor
  - line
  - character set
  - markup

- word processor
  - paragraph
  - format
  - style

- desktop publishing app

waffling about concepts and fine points of design
characterize apps & families

- text editor
  - line
  - character set
  - markup

- word processor
  - paragraph
  - format
  - style

- desktop publishing app
  - paragraph
  - format
  - style
  - page
  - textflow

waffling about concepts and fine points of design
explore & evaluate individual concepts
explore & evaluate individual concepts

Ava Dropbox

Bella Dropbox

Bella Party

Bella Plan

how many users believe the folder concept works
explore & evaluate individual concepts

how many users believe the folder concept works

how folders actually work (in Dropbox, Unix, Multics)
analyze how concepts fit together
analyze how concepts fit together

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

[Emoji reactions]
analyze how concepts fit together

**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)

...
analyze how concepts fit together

![Facebook reaction icons with Upvote and Reaction concepts]

**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)

...
analyze how concepts fit together

**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?
design moves
mechanical analogs
three pairs of design moves
three pairs of design moves

- merge
- split
three pairs of design moves

merge - split
unify - specialize
three pairs of design moves

merge \rightarrow split
unify \rightarrow specialize
loosen \rightarrow tighten
split-merge: tradeoff simplicity/flexibility
**split-merge**: tradeoff simplicity/flexibility
**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
**unify-specialize**: tradeoff simplicity/specificity

- **set of wrenches** → unify → **adjustable wrench**
- **macro lens** → specialize → **general-purpose lens**
tighten-loosen: tradeoff automation/flexibility
tighten-loosen: tradeoff automation/flexibility

light pull / door lock

tighten

airplane toilet lock
tighten-loosen: tradeoff automation/flexibility

light pull / door lock

dimmers with separate controls

airplane toilet lock

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)
unify: subsuming access control in MIT’s Moira
unify: subsuming access control in MIT’s Moira
unify: subsuming access control in MIT’s Moira

List
Access List
Mailing List

WebMoira List Manager: Daniel Jackson

List Name: dnj-play1
Description: none
Attributes: active, moira mailing list
Permissions: private, visible
Last Modified: by dnj with moiraws on 22-mar-2022 09:39:00

Members
Add Member:
Leave List: Remove Me

MIT Users
Daniel Jackson (dnj)
Email Addresses
daniel@dnj.photo

Administrators
Owner: dnj-play2 (List)
Change Owner:
Add Administrator:
Leave Owner List: Remove Me

MIT Users
Daniel Jackson (dnj)

Can toggle mailing list attribute
unify: subsuming access control in MIT's Moira

- can toggle mailing list attribute
- can create admin list with no login users!
tighten: label and trash concepts in Gmail
tighten: label and trash concepts in Gmail
tighten: label and trash concepts in Gmail

- Show messages with label hacking
- A label
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

- Label
- Trash

show messages with label hacking

also implemented as a label

a label
<table>
<thead>
<tr>
<th>concept</th>
<th>trash</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>undo deletion</td>
</tr>
<tr>
<td>structure</td>
<td>trash: set Item</td>
</tr>
<tr>
<td>actions</td>
<td>delete (i: Item)</td>
</tr>
<tr>
<td></td>
<td>restore (i: Item)</td>
</tr>
<tr>
<td></td>
<td>empty ()</td>
</tr>
<tr>
<td>concept</td>
<td>trash</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>purpose</td>
<td>undo deletion</td>
</tr>
<tr>
<td>structure</td>
<td>trash: set Item</td>
</tr>
<tr>
<td>actions</td>
<td>delete (i: Item)</td>
</tr>
<tr>
<td></td>
<td>restore (i: Item)</td>
</tr>
<tr>
<td></td>
<td>empty ()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>concept</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>organize with overlapping</td>
</tr>
<tr>
<td>structure</td>
<td>labels: Item -&gt; set Label</td>
</tr>
<tr>
<td>actions</td>
<td>add (i: Item, l: Label)</td>
</tr>
<tr>
<td></td>
<td>remove (i: Item, l: Label)</td>
</tr>
<tr>
<td></td>
<td>find (ls: set Label, out is: set Item)</td>
</tr>
<tr>
<td>concept</td>
<td>trash</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>purpose</td>
<td>undo deletion</td>
</tr>
<tr>
<td>structure</td>
<td>trash: set Item</td>
</tr>
<tr>
<td>actions</td>
<td>delete (i: Item)</td>
</tr>
<tr>
<td></td>
<td>restore (i: Item)</td>
</tr>
<tr>
<td></td>
<td>empty ()</td>
</tr>
</tbody>
</table>

**When delete (i)**
**also add (i, 'trash')**

<table>
<thead>
<tr>
<th>concept</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>organize with overlapping</td>
</tr>
<tr>
<td>structure</td>
<td>labels: Item -&gt; set Label</td>
</tr>
<tr>
<td>actions</td>
<td>add (i: Item, l: Label)</td>
</tr>
<tr>
<td></td>
<td>remove (i: Item, l: Label)</td>
</tr>
<tr>
<td></td>
<td>find (ls: set Label, out is: set Item)</td>
</tr>
</tbody>
</table>
integrating these concepts is tricky
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

- Click on trash
- Filter on todo label
- Filter on todo and trash
integrating these concepts is tricky

click on trash

filter on todo label

filter on todo and trash

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)
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

4896x3264 (16M)  664 FRAMES
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!
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 options

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
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
event deletion in google calendar
Despite some erroneous messages sent to this list accidentally, Kanit's talk is happening! Please join us on Monday.
HCI Seminar
Wednesday, December
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
1. Seminar announced as email to listserv with attached calendar event.
2. Event installed automatically in user's calendar.
3. User deletes event from calendar.
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)
...
**concept** calendar
**purpose** record engagements
**actions**
createEvent (...) : Event
deleteEvent (e: Event)
...

---

**concept** invitation
**purpose** coordinate events
**actions**
accept (e: Event)
decline (e: Event)
...
concept calendar
purpose record engagements
actions
  createEvent (...): Event
  deleteEvent (e: Event)
...

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

resolution to design problem
make sync optional
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

loosen
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?
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

**apply design moves to explore new options**
ever 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?

**in formal methods**
can concepts help structure & validate models?
THE ESSENCE OF SOFTWARE

WHY CONCEPTS MATTER FOR GREAT DESIGN

DANIEL JACKSON

much more in the book

essenceofsoftware.com

join the discussion about concept design!

forum.softwareconcepts.io

newsletter

essenceofsoftware.com/subscribe