the essence of software
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
“make sure the user has the right conceptual model”

“get the concepts straight and everything else will fall into place”

“but what exactly are concepts? and how do get them straight?”
this talk: just a flavor, but hopefully enough to give you a new way to think about software
puzzle #1: Dropbox & the case of the disappearing files
Someone accidentally deleted thousands of files in my company Dropbox: how can I quickly undelete them?
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’?

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.
Survey of Dropbox users (MIT CS undergrads)

Correctly predicting behavior

- Good knowledge:
  - Delete shared folder results in leaving: 60.0%
  - Delete shared subfolder removes it: 70.0%

- Average knowledge:
  - Delete shared folder results in leaving: 40.0%
  - Delete shared subfolder removes it: 60.0%

- Poor knowledge:
  - Delete shared folder results in leaving: 20.0%
  - Delete shared subfolder removes it: 40.0%
puzzle #2: 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  pic.twitter.com/XiNd2jiLUF
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*
puzzle #3: Google Calendar & the case of the mysterious cancellations
Despite some erroneous messages sent to this list accidentally, Kanit's talk is happening! Please join us on Monday.
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 new path
inspired by old ideas
what kinds of problems are these?

- not just human errors
- not bugs in the code
- not classic UI design flaws
user's model

user interprets
interface

interface designer
conveys design

programmer
implements design

designer's model

user interprets
interface

interface designer
conveys design
user's model

designer's model

psychology

interface design

software engineering
user’s model

designer’s model

classic usability research
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
designer's model

but what if the system just does the wrong thing?

goto this way instead?
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)
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)
challenges
or, why it’s not easy
what is the conceptual model of Hacker News?

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](https://wikipedia.org))

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:

- 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...](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 rectangles” approach in htdp to be pretty solid in real life problems.

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?

**concept lattice**

stagnant  | natural | constant |
---------|---------|----------|
running  | reservoir | channel |
          | trickle   | canal    |
          | runnel    | stream   |
          | river     | rivulet  |
          | torrent   |          |

puddle  | maritime |
---------|----------|
          | lagoon   |
          | sea      |
          | sea      |
          | tarn     |
          | pool     |
          | maar     |
          | lake     |
          | pond     |

**entity in data model**

User ➔ Vote ➔ Post

votes for
Upvote  | Downvote

but concept is in the relationships, not the entities!

upvotes and downvotes are votes and then what?
two more problems

where's the boundary?
if concept is in relationships too, which ones?
for upvote, are posts in the concept? users?

are concepts coupled?
eg, upvotes add to karma points
need karma points to downvote
so are these concepts dependent?
an example concept
finally!
a concept has a name

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

we learned from the GOF patterns book how essential names are for sharing expertise
This is homework and I'm having a hard time with it. Here are the definitions of the objects:

```
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 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)

**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 ...

**need unvote** in HackerNews (eg) since only high karma users can downvote

succinct specification as actions on states
- VDM (1986)
- Z (1992)
- Larch (1993)
- Alloy (2006)
a concept has an operational principle

**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)

**operational principle**
after sequence of upvote (...)’s,
if item i has more than item j,
then i.rank < j.rank

an archetypal scenario that captures the essence of how the concept fulfills its purpose

use cases are actually more like actions
a concept handbook entry

**Design Variants**
- downvote as unvote
- use age in ranking
- weigh downvotes more

**Typical Uses**
- social media posts
- comments on articles
- Q&A responses

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

**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 make an app
concept composition
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

**concept** Karma

**purpose** privilege good users

**state**
- karma: User → one Int
- contribs: User → set Item

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

**operational principle**
allow permit (u, R) if sum of rewards for u.items ≥ R
**concept** Upvote

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

**when**
- upvote (u, i)
- downvote (u, 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

- 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 (post1, 10)
- permit (Alice, 20)

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

**C.A.R. Hoare**

*Communicating Sequential Processes*

Composition uses event sync from Hoare’s CSP

No concept coupling concepts preserve properties
resolving the puzzles
concept design in action
#1: dropbox
a troubled concept
how many users believe Dropbox is structured

how Dropbox is actually structured

confusing concept
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 unix directory
purpose organize items into overlapping categories
state
  entries: Dir -> set Entry
  name: Entry -> one Name
  item: Entry -> one (File + Dir)
#2: twitter

a missing concept
what concept is this?

- **concept** Upvote
  - **purpose** rank items by popularity

- **concept** Bookmark
  - **purpose** save items to revisit

- missing concept!
Twitter adds a new concept (2018)

**Purpose**: Upvote

**Concept**: Rank items by popularity

**Purpose**: Bookmark

**Concept**: Save items to revisit
#3: Google Calendar

a bad synchronization
Concept: Calendar
Purpose: Record upcoming engagements
Actions:
- createEvent (...): Event
- deleteEvent (e: Event)
- ...

Concept: Invitation
Purpose: Coordinate event participants
Actions:
- accept (e: Event)
- decline (e: Event)
- ...

Unwanted sync!
a long time problem in iCal too
how to delete spam calendar events?

resolution to design problem
make sync optional
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)

unwanted sync?
lessons & takeaways
how we resolved the challenges

where’s the boundary?  
a concept is more like a microservice than a class, datatype or entity.  
call it a “nanoservice”?

are concepts coupled?  
no! synchronize when composing so concepts are free-standing
practical lessons

Twitter
- do you have the **right concepts?**
- inventory your app’s concepts
- identify **familiar** concepts to reuse

Dropbox
- is each concept **fit for purpose?**
- convey the **purpose**
- localize the data model

Calendar
- is the **composition** right?
- compose by **sync**
- watch for **over & under sync**
the costs of bad concept design

unclear concepts makes life less pleasant and upsets customers

a concept design flaw reported at https://googledrivesucks.com

tricky concepts cause users to limit their use of an app

Me: Have you ever encountered these Dropbox problems?
Computer scientist: No, I don’t use sharing. Too risky.

confusing concepts lead to costly mistakes

I feel really bad for the person that fat fingered a $900mm erroneous payment. Not a great career move

in re Citibank, 2020

would better concepts prevent dosage errors?
the potential benefits of good concept design

**design focus**
inform scope choices
new modularity
helps divide work

**design ideas**
record knowledge
share it with others
reuse designs

**design principles**
avoid user testing
complement code
& UI heuristics
much more in the book

THE
ESSENCE
OF
SOFTWARE

WHY CONCEPTS
MATTER FOR
GREAT DESIGN

DANIEL JACKSON

essenceofsoftware.com

join the discussion
about concept design!
forum.softwareconcepts.io