Hints and Principles for Computer System Design

Butler Lampson
Microsoft Research
Heidelberg Laureate Forum
August 27, 2015
Overview

- A 32-year update of my 1983 *Hints for Computer Systems*
- These are mostly hints, often not consistent or precise
  - Hints *suggest*—no nitpicking allowed
- **STEADY by AID**
  - **What**: Simple, Timely, Efficient, Adaptable, Dependable, Yummy
  - **How**: Approximate, Incremental, Divide & conquer, …
- The future: Engagement with the physical world

*There are three rules for writing a novel. Unfortunately, no one knows what they are.*
—Somerset Maugham

*You got to be careful if you don’t know where you’re going, because you might not get there.*
—Yogi Berra

*The quest for precision, in words or concepts or meanings, is a wild goose chase.*
—Karl Popper
What: Goals

- **Simple**
- **Timely (to market)**
- **Efficient**
- **Adaptable**
- **Dependable**
- **Yummy**

STHEADY

*More important today*

First ↔ Fast ↔ Frugal ↔ Flexible ↔ Faithful ↔ Fancy ↔ Fun

TTM ↔ speed ↔ cost ↔ change ↔ trust ↔ features ↔ coolness

[Data is not information,] Information is not knowledge, Knowledge is not wisdom, Wisdom is not truth, Truth is not beauty, Beauty is not love, Love is not music and Music is THE BEST” —Frank Zappa
How: Methods

- **Approximate**
  - Good enough
  - Loose specs
  - Lazy/speculative

- **Incremental**
  - Indirect
  - Iterate
  - Extend

- **Divide & conquer**
  - Interfaces to abstractions
  - Recursive
  - Atomic
  - Concurrent
  - Replicated

AID
Kinds of Software

- **Precise vs. approximate software**
  - Precise: Get it right
    - avionics, banks, Office
  - Approximate: Get it soon, make it cool
    - search, shopping, Twitter

- Which kind is yours?
  - One isn't better or worse than the other,
  - but they are different.

*Unless in communicating with it [a computer] one says exactly what one means, trouble is bound to result.* —Turing

*There’s no sense being exact about something if you don’t even know what you’re talking about.* —von Neumann
Coordinate Systems and Notation

Choose the right coordinate system
- Like center of mass for dynamics, or eigenvectors for matrices
- Ex: State as being vs. becoming, function as code vs. table vs. overlay

Choose a good notation
- This is why domain specific languages succeed
- Relations cover most needs for design
  - subsuming sets, functions, graphs, programs
  - with composition, transitive closure, union, intersection as primitives

A point of view is worth 80 points of IQ. —Alan Kay

Science is not there to tell us about the Universe, but to tell us how to talk about the Universe. —Niels Bohr

A good notation has a subtlety and suggestiveness which at times make it seem almost like a live teacher... and a perfect notation would be a substitute for thought. —Russell
## Coordinates: State

- **State as being vs. becoming**
  - **Being:** map from names → values
  - **Becoming:** initial state + log of updates

- **Being is the usual form**
- **Becoming is good for undo, versions and recovery**

<table>
<thead>
<tr>
<th>Example</th>
<th>Being</th>
<th>Becoming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>bitmap</td>
<td>display list</td>
</tr>
<tr>
<td>Document</td>
<td>sequence of characters</td>
<td>sequence of inserts / deletes</td>
</tr>
<tr>
<td>Database</td>
<td>table + buffer cache</td>
<td>redo-undo log</td>
</tr>
<tr>
<td>Eventual consistency</td>
<td>names → values read</td>
<td><em>any</em> subset of updates that are commutative and associative</td>
</tr>
</tbody>
</table>

*Don’t ask what it means, but rather how it is used.* —Wittgenstein  
*No matter how far down the wrong road you have gone, turn back now.* —Turkish Proverb
Coordinates: Functions

Function as code vs. table vs. overlay

- Code: execute $f(x)$ to get the result
- Table: lookup $x$ in a set of (argument, result) pairs
- Overlay: try $f_1(x)$, if undefined try $f_2(x)$, ...

<table>
<thead>
<tr>
<th>Example</th>
<th>Code</th>
<th>Table</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main memory</td>
<td>—</td>
<td>RAM</td>
<td>write buffer</td>
</tr>
<tr>
<td>Database</td>
<td>—</td>
<td>data on disk</td>
<td>buffer cache</td>
</tr>
<tr>
<td>bin for shell cmd</td>
<td>—</td>
<td>/bin directory</td>
<td>search path</td>
</tr>
<tr>
<td>Function of simple argument</td>
<td>run the code</td>
<td>precomputed results</td>
<td>saved old results</td>
</tr>
<tr>
<td>Database view</td>
<td>run the query</td>
<td>materialized view</td>
<td>incremental updates</td>
</tr>
</tbody>
</table>

If all you have is a hammer, everything looks like a nail. —A. Maslow
Write a Spec: State

- At least, write down the abstract state
  - Abstract state is *real*
  - Example: File system state is PathName → ByteArray

The purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise. —Dijkstra

Beware of bugs in the above code; I have only proved it correct, not tried it. —Knuth
At least, write down the state—Abstract state is \textit{real}

Example: File system state is PathName$\rightarrow$ByteArray

Then, write down the interface actions (APIs),

which ones are external, and what each action $\pi$ does

Example: For failures, volatile vs. persistent state

On crash, volatile $:= $ persistent

On sync, persistent $:= $ volatile

\textit{The purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.} —Dijkstra
The purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise. —Dijkstra
At least, write down the state—Abstract state is real
Example: File system state is PathName→ByteArray
Then, write down the interface actions (APIs),
which ones are external, and what each action π does
Next, write the abstraction function $F$ from code to spec
Finally, show that each action $\pi$ preserves $F$:

$$F(s) \xrightarrow{\pi} F(s')$$

Newcombe et al., How Amazon Web Services uses formal methods, *Comm ACM* 58, 4 (March 2015), pp 66-73
How: Methods

- **Approximate**
  - Good enough
  - Lazy/speculative
  - Loose specs

- **Incremental**
  - Compose (indirect, virtualize)
  - Iterate
  - Extend

- **Divide & conquer**
  - Interfaces to abstractions
  - Recursive
  - Replicated
  - Concurrent

AID
AID: Divide & Conquer

- Interfaces to abstractions: Divide by **difference**
  - Limit complexity, liberate parts. TCP/IP, file system, HTML
  - Platform/layers. OS, browser, DB. X86, internet. Math library
    - Need this to ship
  - Declarative. HTML/XML, SQL queries, schemas
    - The program you think about takes only a few steps
  - Synthesize a program from a partial spec. Excel Flashfill
    - Signal + Search → Program

*Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.* —Whitehead

*Don’t tie the hands of the implementer.* —Martin Rinard
AID: Divide & Conquer

- Interfaces: Divide by difference
- Recursive: Divide by structure. Part ~ whole
  - Quicksort, DHTs, path names. IPV6, file systems
- Replicated: Divide for redundancy, in time or space
  - Retry: End to end (TCP). Replicated state machines.
- Concurrent: Divide for performance
  - Stripe, stream, or struggle: BitTorrent, MapReduce

*If you come to a fork in the road, take it.* —Yogi Berra

*To iterate is human, to recurse divine.* —Peter Deutsch
AID: Incremental

- **Indirect**: Control name → value mapping
  - Virtualize/shim: VMs, NAT, USB, app compat, format versions
  - Network: Source route → IP addr → DNS name → service → query
  - Symbolic links, register rename, virtual methods, copy on write

- **Iterate** design, actions, components
  - **Redo**: Log, replicated state machines (state as becoming)
  - **Undo**: File system snapshots, transaction abort
  - **Scale**: Internet, clusters, I/O devices

- **Extend**: HTML, Ethernet

Any problem in computing can be solved by another level of indirection. —David Wheeler
Compatible, adj. Different. —The Devil’s Dictionary of Computing
AID: Approximate

- **Good enough.** Web, search engines, IP packets
  - Eventual consistency. DNS, Dynamo, file/email sync

- **Loose coupling:** springy flaky parts. Email, Fedwire

- **Brute force.** Overprovision, broadcast, scan, crash fast
  - Strengthen (do more than is needed). Redo log, coarse locks

- **Relax:** small steps converge to desired result
  - Routing protocols, daily builds, exponential backoff

- **Hints:** Trust, but verify

*I may be inconsistent. But not all the time.—Anonymous*
What: Goals

- Simple
- **Timely** (to market)*
- Efficient
- **Adaptable***
- Dependable
- **Yummy***

**STEADY**

- First↔Fast↔Frugal↔Flexible↔Faithful↔Fancy↔Fun

- Need tradeoffs—You can’t get *all* these good things
STEADY: Simple, Timely

Simple is important because we can’t do much

- Simple enough? I can still understand it
  - But when it evolves, only abstraction and interfaces can save me
- Simple is hard, often not rewarded—“That’s obvious.”
  - Why didn’t computer scientists invent the web?

Timely: Good enough is good enough

- The web is successful because it doesn’t have to work.
- Learn what customers really want—Iterative development

Less is more. —Browning
Everything should be as simple as possible, but no simpler. —Einstein
I’m sorry I wrote you such a long letter; I didn’t have time to write a short one. —Pascal

The best is the enemy of the good. —Voltaire
If you don’t think too good, don’t think too much. —Ted Williams
And the users exclaimed with a laugh and a taunt,
“it's just what we asked for but not what we want.” —Anonymous
STEDY: **Efficient, Adaptable**

- **Efficient** has two faces: for the implementer, for the client
  - Not unrelated: the client wants it fast and cheap enough
  - Efficient *enough*, not optimal

- **Adaptable**—Plan for success
  - Evolution/scaling: Successful systems live a long time
    - 2015 PC = 100,000 × Xerox Alto, Web grew from 100 users to $10^9$
  - Incremental update: Big things change a little at a time

> An efficient program is an exercise in logical brinkmanship. —Dijkstra

> *I see how it [the phone] works. It rings, and you have to get up.* —Degas

> That, Sir, is the good of counting. It brings everything to a certainty, which before floated in the mind indefinitely. —Samuel Johnson

> Success is never final. —Churchill (attributed)

> APL is like a diamond; Lisp is like a ball of mud. —Joel Moses
STEDDY: Dependable, Yummy

- **Dependable:** Reliable, Available, Secure
  - Reliable: Gives the right answer (safe)
  - Available: Gives the answer promptly (live)
  - Secure: Works in spite of bad guys

- Often dependable **undo** is the most important thing

- **Yummy:** Users really want it
  - Function: Spreadsheets, the web, smartphones
  - Design: Apple’s forté

*But who will watch the watchers? She'll just begin with them and buy their silence.* —Juvenal

*The unavoidable price of reliability is simplicity. It is a price which the very rich find most hard to pay.* —Tony Hoare
# The Future: What Do Computers Do?

<table>
<thead>
<tr>
<th>Simulate</th>
<th>1950-ongoing</th>
<th>nuclear weapons, payroll, protein folding, games, virtual reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect (and store)</td>
<td>1980-ongoing</td>
<td>email, airline tickets, books, movies, Bing, Virtual Earth</td>
</tr>
<tr>
<td>Engage (with physical world)</td>
<td>2010-...</td>
<td>factories, cars, robots, smart dust: <em>Embodiment</em></td>
</tr>
</tbody>
</table>

*The future ain’t what it used to be.*—Yogi Berra  
*Reality is that which, when you stop believing in it, doesn’t go away.*—Philip K. Dick
Big Trends

- Connectivity—cloud and data everywhere
- Ubiquity, invisibility: systems everywhere
- Scaling—billions of users, billions of gigabytes
- Approximation—good enough is good enough
- AI and systems are converging
- Reusable components are finally catching on
- **Uncertainty**—fundamental to engagement
- **Dependability**—critical systems have to work

_They always say time changes things, but you actually have to change them yourself._
—Andy Warhol

_You see things; and you say, ‘Why?’ But I dream of things that never were; and I say ‘Why not?’_
—Shaw
Grand Challenge: Zero Traffic Deaths

- Cars have to drive themselves
  - A pure computer science problem

- Needs
  - Computer vision
  - World models for roads and vehicles
  - *Dealing with uncertainty* about sensor inputs, vehicle performance, changing environment
  - *Dependability*

- DARPA Challenges, Google cars a start

- Huge economic impact

- Safety trumps liability

*Problems worthy of attack prove their worth by hitting back.*—Piet Hein
Dealing with Uncertainty

- Unavoidable in the physical world
  - Need good models of what’s possible, and their limits
- Unavoidable for “natural” user interfaces: speech, writing, language
  - The machine must guess; what if it guesses wrong?
- Paradigm?: Probability distributions
  - Distributions as a standard data type?
    - Parameterized over the domain (like lists). What are the operations?
  - A start: Microsoft Infer.Net, probabilistic programming

*Logic, like whiskey, loses its beneficial effect when taken in too large quantities.* —Lord Dunsany

*Do I contradict myself? Very well then I contradict myself. (I am large, I contain multitudes.)*
—Whitman
Dependable $\Rightarrow$ No Catastrophes

- A realistic way to reduce aspirations
  - Focus on what’s really important

- What’s a catastrophe? It has to be very serious
  - USS Yorktown: database failure $\rightarrow$ can’t run engines
  - Terac 25 and other medical equipment: Patients die

- Architecture: Normal vs. catastrophe mode
  - Catastrophe mode $\Rightarrow$ high assurance CCB

- Catastrophe mode requires limited goals = limited function
  - And strict bounds on complexity
    - Less than 50k lines of code? Can verify? Examples: Ironclad, FSCQ

*If you can’t make it fast and correct, make it fast.*—Luca Cardelli

*As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.*—David Parnas
Summary

**STEADY by AID**
- What: Simple, Timely, Efficient, Adaptable, Dependable, Yummy
- How: Approximate, Incremental, Divide & conquer

**If you only remember three things:**
- Keep it simple
- Interfaces to abstractions
- Write a spec

**The future: Engagement with the physical world**

*If I have seen further than others, it is because I have stood on the shoulders of giants.*  
—Schoolmen of Chartres, via Newton

*The only thing new in the world is the history you don’t know.*  
—Harry Truman

*History doesn’t repeat, but it rhymes.*  
—Mark Twain