Worse Is Better, for Better or for for Worse

@KevlinHenney
In 1990 I proposed a theory, called *Worse Is Better*, of why software would be more likely to succeed if it was developed with minimal invention.
It is far better to have an underfeatured product that is rock solid, fast, and small than one that covers what an expert would consider the complete requirements.
- **Simplicity**: The design is simple in implementation. The interface should be simple, but anything adequate will do.

- **Completeness**: The design covers only necessary situations. Completeness can be sacrificed in favor of any other quality.

- **Correctness**: The design is correct in all observable aspects.

- **Consistency**: The design is consistent as far as it goes. Consistency is less of a problem because you always choose the smallest scope for the first implementation.
Implementation characteristics are foremost:

- The implementation should be fast.
- It should be small.
- It should interoperate with the programs and tools that the expected users are already using.
- It should be bug-free, and if that requires implementing fewer features, do it.
- It should use parsimonious abstractions as long as they don’t get in the way.
#!/usr/bin/perl
#PerlInterpreter
#PerlInterpreter must be the first line of the file.
#Copyright (c) 1995, Cunningham & Cunningham, Inc.
#
#This program has been generated by the HyperPerl generator. The source hypertext can be found
#at http://c2.com/cgi/wikibase. This program belongs
#to Cunningham & Cunningham, Inc., is to be used
#only by agreement with the owner, and then only
#with the understanding that the owner cannot be
#responsible for any behaviour of the program or
#any damages that it may cause.
#--------------------------------------------------------
InitialComments

# InitialComments
print "Content-type: text/html\n\n";
$DBM = "/usr/ward/$ScriptName";
dbmopen(%db, $DBM, 0666) || &AbortScript("can't open $DBM");
$CookedInput(browse) && &HandleBrowse;
$CookedInput(edit) && &HandleEdit;
$CookedInput(copy) && &HandleEdit;
$CookedInput(links) && &HandleLinks;
$CookedInput(search) && &HandleSearch;
dbmclose (%db);
if ($ENV{REQUEST_METHOD} eq POST) {
  $CookedInput(post) && &HandlePost;
}
&DumpBinding(*CookedInput);
&DumpBinding(*old);
&DumpBinding(*ENV);
#-------------------------------------------------------- WikiInHyperPerl
I always have it in the back of my head that I want to make a slightly better C.

But getting everything to fit, top to bottom, syntax, semantics, tooling, etc., might not be possible or even worth the effort.

As it stands today, C is unreasonably effective, and I don't see that changing any time soon.

Damien Katz

http://damienkatz.net/2013/01/the_unreasonable_effectiveness_of_c.html
It Must be Beautiful
Great Equations of Modern Science
Edited by Graham Farmelo
Beautiful Code
Leading Programmers Explain How They Think
A THEORY OF OBJECTS

MARTÍN ABADI
LUCA CARDELLI
In a purist view of object-oriented methodology, dynamic dispatch is the only mechanism for taking advantage of attributes that have been forgotten by subsumption. This position is often taken on abstraction grounds: no knowledge should be obtainable about objects except by invoking their methods. In the purist approach, subsumption provides a simple and effective mechanism for hiding private attributes.
OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I'm not aware of them.

Alan Kay
One of the most pure object-oriented programming models yet defined is the Component Object Model (COM).

It enforces all of these principles rigorously.

William Cook

"On Understanding Data Abstraction, Revisited"
Lambda-calculus was the first object-oriented language (1982)
newStack =
  λ • (let items = ref(⟨⟩) •
    {
      isEmpty = λ • #items = 0,
      depth = λ • #items,
      push = λ x • items := ⟨x⟩⟨items_y | y ∈ 0...#items⟩,
      top = λ • items₀
    })
var newStack = function() {
    var items = []
    return {
        isEmpty: function() {
            return items.length === 0
        },
        depth: function() {
            return items.length
        },
        push: function(newTop) {
            items = items.unshift(newTop)
        },
        top: function() {
            return items[0]
        }
    }
}
Any application that can be written in JavaScript, will eventually be written in JavaScript.

Atwood's Law
Starting Linux
Linux version 2.6.20 (bellard@voyager) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-9)) #2 Mon Aug 8 23:51:02 CEST 2011
BIOS-provided physical RAM map:
sanitize start
sanitize bail 0
  BIOS-e801: 0000000000000000 - 000000000009f000 (usable)
  BIOS-e801: 0000000000100000 - 0000000001000000 (usable)
16MB LOWMEM available.
Zone PFN ranges:
  DMA 0 -> 4096
  Normal 4096 -> 4096
early_node_map[1] active PFN ranges
  0: 0 -> 4096
DMI not present or invalid.
Allocating PCI resources starting at 10000000 (gap: 01000000:ff000000)
Detected 3.333 MHz processor.
Built 1 zonelists. Total pages: 4064
Kernel command line: console=ttyS0 root=/dev/ram0 rw init=/sbin/init notsc=1
Initializing CPU#0
Disabling TSC...
PID hash table entries: 64 (order: 6, 256 bytes)
Console: colour dummy device 80x25
Dentry cache hash table entries: 2048 (order: 1, 8192 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory: 11956k/16384k available (1265k kernel code, 4040k reserved, 324k data, 1 24k init, 0k highmem)
virtual kernel memory layout:
  fixmap : 0xffffc000 - 0xffffffff ( 12 kB)
  vmalloc : 0xc1800000 - 0xffffffff (999 MB)

© 2011 Fabrice Bellard - News - FAQ - Technical notes
Lions' Commentary on UNIX 6th Edition
with Source Code
John Lions
Foreword by Dennis Ritchie
There have always been fairly severe size constraints on the Unix operating system and its software. Given the partially antagonistic desires for reasonable efficiency and expressive power, the size constraint has encouraged not only economy but a certain elegance of design.

Dennis Ritchie and Ken Thompson
"The UNIX Time-Sharing System", CACM
This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Doug McIlroy
The hard part isn’t writing little programs that do one thing well. The hard part is combining little programs to solve bigger problems. In McIlroy’s summary, the hard part is his second sentence: Write programs to work together.

John D Cook

http://www.johndcook.com/blog/2010/06/30/where-the-unix-philosophy-breaks-down/
Software applications do things they’re not good at for the same reason companies do things they’re not good at: to avoid transaction costs.

John D Cook

http://www.johndcook.com/blog/2010/06/30/where-the-unix-philosophy-breaks-down/
Architecture is the decisions that you wish you could get right early in a project, but that you are not necessarily more likely to get them right than any other.

Ralph Johnson
Properly gaining control of the design process tends to feel like one is *losing* control of the design process.
The classic essay on "worse is better" is either misunderstood or wrong.

Jim Waldo
Decide for yourselves.

Richard P Gabriel
Thank you and goodbye.

Hope you enjoyed the conference!