Worse Is Better, for Better or for Worse
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann, Heinz Krüger, and Reginald Quraishi

PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann, Heinz Krüger, and Reginald Quraishi

97 Things Every Programmer Should Know
Collective Wisdom from the Experts
Edited by Kevin Henney
Beautiful Code
Leading Programmers Explain How They Think

Edited by Andy Oram & Greg Wilson
There are only two kinds of languages: the ones people complain about and the ones nobody uses.

Bjarne Stroustrup
In 1990 I proposed a theory, called *Worse Is Better*...
The "good enough software" concept popularized by Yourdon.

In many senses, it's just a rationalization of what's happening in the software world: the first company hitting the market with a feature-rich product is more likely to win the battle than the careful, quality-seeking company.

Carlo Pescio

http://www.eptacom.net/pubblicazioni/pub_eng/wirth.html
"Good enough software" is rarely good enough.

Niklaus Wirth
http://www.eptacom.net/pubblicazioni/pub_eng/wirth.html
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.
The following is a characterization of the contrasting [the right thing] design philosophy:

- **Simplicity**: The design is simple [...] . Simplicity of implementation is irrelevant.

- **Completeness**: The design covers as many important situations as possible. All reasonably expected cases must be covered.

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

- **Consistency**: The design is thoroughly consistent. A design is allowed to be slightly less simple and less complete in order to avoid inconsistency. Consistency is as important as correctness.
People overvalue their knowledge and underestimate the probability of their being wrong.
What I cannot create, I do not understand.

Richard Feynman
You have to finish things — that's what you learn from, you learn by finishing things.

Neil Gaiman
Here are the characteristics of a worse-is-better software design:

- *Simplicity*: The design is simple in implementation. The interface should be simple, but anything adequate will do.
Interface

Implementation
Implementation

Interface
Implementation

Interface
Aesthetics does not affect perceived usability, but degraded usability negatively affects perceived aesthetics. In other words, usability is more important than beauty.

"Is Beautiful Usable, or Is It the Other Way Around?"
http://www.infoq.com/news/2012/05/Aesthetics-Usability
SHOWER INSTRUCTIONS

TO TURN ON WATER
ROTATE RIGHT

TO ADJUST TEMPERATURE
PRESS DOWN AND TURN

TEMPERATURE CONTROL

HOT
COLD

(Instructions in Chinese: 淋浴器提示)

Temperature control:
- HOT: Press and turn LEFT for hot water
- COLD: Turn knob to the right for cold water
Here are the characteristics of a worse-is-better software design:

- **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.
The more specific a design idea is, the greater its appeal is likely to be.
#!/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.
#
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);

---

InitialComments

---

WikiInHyperPerl
Here are the characteristics of a worse-is-better software design:

- **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.
Savings in time feel like simplicity
Implementation characteristics are foremost:

- The implementation should be fast.
- It should be small.
Milk is cheapest in BIG cartons

And small cartons of software reduce risk

Software is cheapest in lots of small cartons

Allan Kelly
Beyond Projects
http://www.slideshare.net/allankellynet/no-prokects-beyond-projects-refreshed-version
Software development does *not* have economies of scale. Development has *diseconomies of scale.*
I would therefore like to posit that computing's central challenge, "How not to make a mess of it", has not been met. On the contrary, most of our systems are much more complicated than can be considered healthy, and are too messy and chaotic to be used in comfort and confidence.

Edsger Dijkstra
Um. What's the name of the word for things not being the same always. You know, I'm sure there is one. Isn't there? There's must be a word for it... the thing that lets you know time is happening. Is there a word?

*Change.*

Oh. I was afraid of that.
Because the design that occurs first is almost never the best possible, the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design.

Fred Brooks
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.
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
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.
"After 20 years, this is still the best exposition of the workings of a 'real' operating system."

Ken Thompson

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
The classic essay on "worse is better" is either misunderstood or wrong.

Jim Waldo
Decide for yourselves.

Richard P Gabriel