the challenges of minimalism.

Hisham Muhammad

FOSDEM 2023, Brussels
2023-02-04

minimalism.

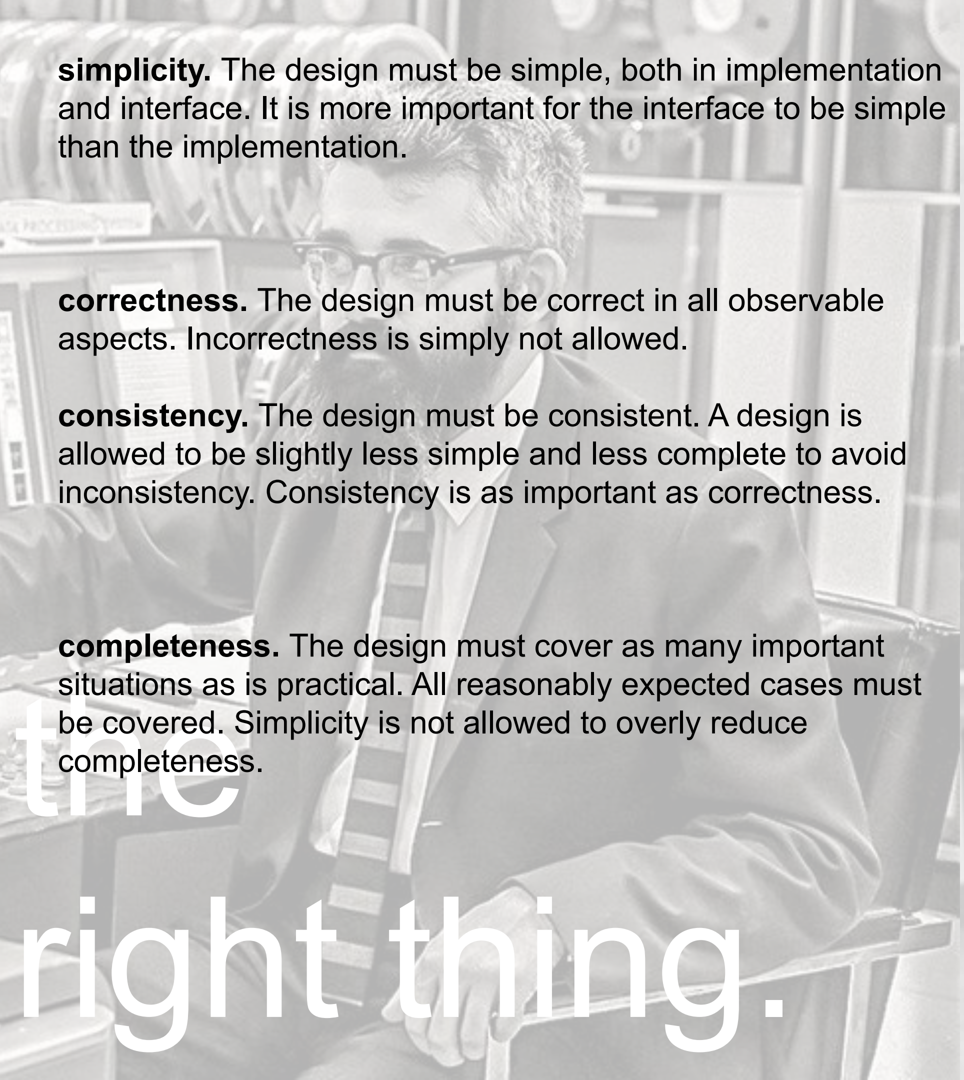


minimalism.


minimalism.


the right thing.


worse is better.



the right thing.

simplicity. The design must be simple, both in implementation and interface. It is more important for the interface to be simple than the implementation.

correctness. The design must be correct in all observable aspects. Incorrectness is simply not allowed.

consistency. The design must be consistent. A design is allowed to be slightly less simple and less complete to avoid inconsistency. Consistency is as important as correctness.

completeness. The design must cover as many important situations as is practical. All reasonably expected cases must be covered. Simplicity is not allowed to overly reduce completeness.

worse is better.

simplicity. The design must be simple, both in implementation and interface. It is more important for the implementation to be simple than the interface. Simplicity is the most important consideration in a design.

correctness. The design should be correct in all observable aspects. It is slightly better to be simple than correct.

consistency. The design must not be overly inconsistent. Consistency can be sacrificed for simplicity in some cases, but it is better to drop those parts of the design that deal with less common circumstances than to introduce either complexity or inconsistency in the implementation.

completeness. The design must cover as many important situations as is practical. All reasonably expected cases should be covered. Completeness can be sacrificed in favor of any other quality. In fact, completeness must be sacrificed whenever implementation simplicity is jeopardized. Consistency can be sacrificed to achieve completeness if simplicity is retained; especially worthless is consistency of interface.

simplicity. The design must be simple, both in implementation and interface.

It is more important for the interface to be simple than the implementation.

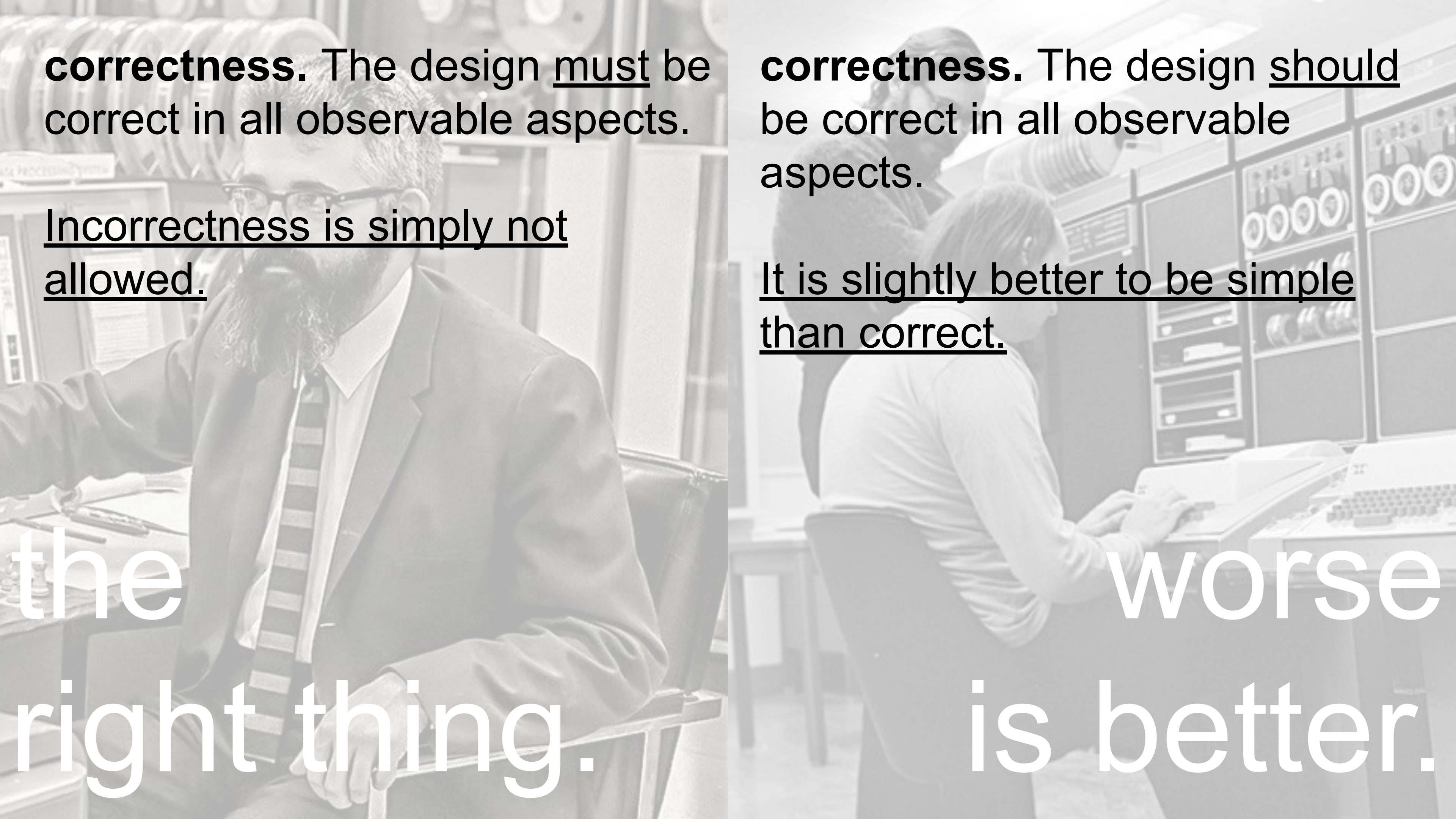
the right thing.

simplicity. The design must be simple, both in implementation and interface.

It is more important for the implementation to be simple than the interface.

Simplicity is the most important consideration in a design.

worse is better.


correctness. The design must be correct in all observable aspects.
Incorrectness is simply not allowed.
the right thing.
correctness. The design should be correct in all observable aspects.
It is slightly better to be simple than correct.
worse is better.

consistency. The design must be consistent.

A design is allowed to be slightly less simple and less complete to avoid inconsistency.

Consistency is as important as correctness.

the right thing.

consistency. The design must not be overly inconsistent.

Consistency can be sacrificed for simplicity in some cases,

but it is better to drop those parts of the design that deal with less common circumstances than to introduce either complexity or inconsistency in the implementation.

worse is better.

completeness. The design must cover as many important situations as is practical. All reasonably expected cases must be covered.

Simplicity is not allowed to overly reduce completeness.

the right thing.

completeness. The design must cover as many important situations as is practical. All reasonably expected cases should be covered.

It can be sacrificed in favor of any other quality. *It must be sacrificed if implementation simplicity is at risk.* Consistency can be sacrificed to get completeness if simplicity is retained.

worse is better.

both work.

when things go wrong.


LuaDec – a Lua decompiler
Hisham Muhammad
Lua Workshop 2005

Lua


CAUTION
RISK OF ELECTRIC SHOCK
DO NOT OPEN
WARNING: TO REDUCE THE RISK OF FIRE OR ELECTRIC SHOCK
DO NOT EXPOSE THIS APPLIANCE TO RAIN OR MOISTURE.
R14 x 4 batteries

modular.

LuaRocks

Lua

Lua

Lua

Lua


ALGUIEN LO SABE.
WHO
WATCH
WATC
ES THE
MEN
?

luarocks.fs.unix

luarocks.fs.lua

luarocks.fs.unix

luarocks.fs.lua

luarocks.fs.bsd

luarocks.fs.win32

luarocks.fs.lua

```
$ luarocks install luarocks
```

scope.

mechanisms, not policies.

when in doubt, make it extensible.

extensible url protocols.

extensible build types.

one build type
to rule (80% of) them all.

```
isham@proxy ~]luarocks config
ccept_unknown_fields = false
rch = "linux-x86_64"
ache = {
  luajit_version_checked = true

ache_fail_timeout = 86400
ache_timeout = 60
heck_certificates = false
make_generator = "Unix Makefiles"
onfig_files = {
  nearest = "/Users/hisham/.luarocks/config-5.4.lua",
  system = {
     file = "/System/Settings/luarocks/config-5.4.lua",
     found = true
  },
  user = {
     file = "/Users/hisham/.luarocks/config-5.4.lua",
     found = true
  }

onnection_timeout = 30
eploy_bin_dir = "/System/Aliens/LuaRocks/bin"
eploy_lib_dir = "/System/Aliens/LuaRocks/lib/lua/5.4"
eploy_lua_dir = "/System/Aliens/LuaRocks/share/lua/5.4"
eps_mode = "one"
isabled_servers = {}
xport_path_separator = ":"
xternal_deps_dirs = {
  "/usr/local",
  "/usr",
  "/"

xternal_deps_patterns = {
  bin = {
     "?"
  },
  include = {
     "?.h"
  },
  lib = {
     "lib?.a",
     "lib?.so",
```

```
rocks_trees = {
   {
      name = "user",
      root = "/Users/hisham/.luarocks"
   },
   {
      name = "system",
      root = "/System/Aliens/LuaRocks"
   }
}
runtime_external_deps_patterns = {
   bin = {
      "?"
   },
   include = {
      "?.h"
   },
   lib = {
      "lib?.so",
      "lib?.so.*"
   }
}
runtime_external_deps_subdirs = {
   bin = "bin",
   include = "include",
   lib = {
      "lib",
      "lib64"
   }
}
static_lib_extension = "a"
sysconfdir = "/System/Settings/luarocks"
target_cpu = "x86_64"
upload = {
   api_version = "1",
   server = "https://luarocks.org",
   tool_version = "1.0.0"
}
user_agent = "LuaRocks/3.9.1 linux-x86_64"
variables = {
   AR = "ar",
   BUNZIP2 = "bunzip2",
   CC = "gcc",
   CFLAGS = "-O2 -fPIC",
   CHMOD = "chmod",
```

ugh.

LuaRocks

zero dependencies

dog-foods optional deps

well-defined scope

minimal base, yet extensible

a large system that tries to be all things to all people :(

what happened? two things.

1

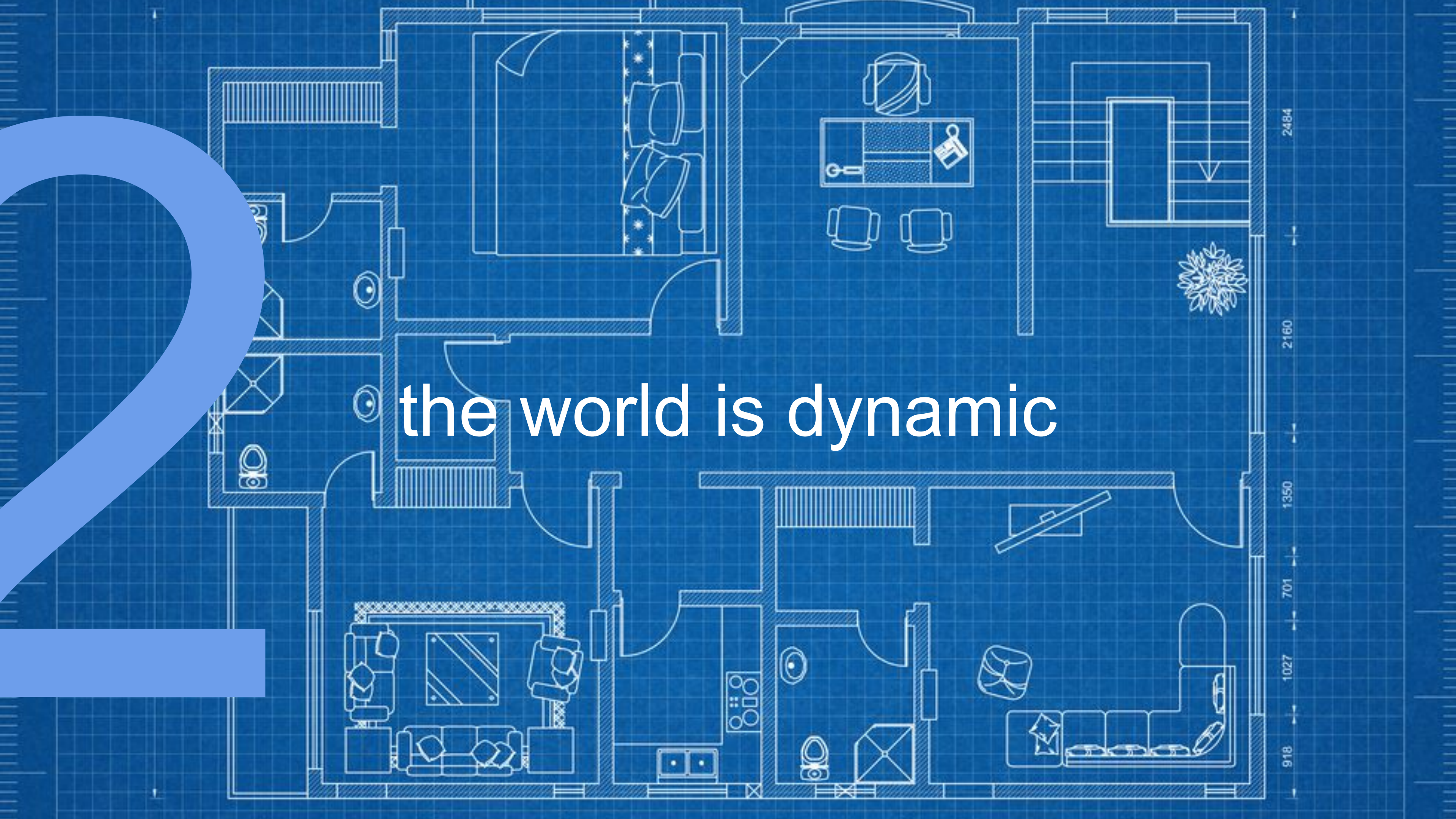
reducing complexity

≠

shifting complexity around

2

the world is dynamic



minimalistic software maintenance?

setting boundaries.

simplicity over compatibility.

"I have intentionally caricatured the worse-is-better philosophy to convince you that it is obviously a bad philosophy and that the New Jersey approach is a bad approach.

However, I believe that worse-is-better, even in its strawman form, has better survival characteristics than the-right-thing, and that the New Jersey approach when used for software is a better approach than the MIT approach."

lessons learned?

LuaRocks

zero dependencies for users

simplified scope

minimal base that is
extensible, not extended

simplicity.

correctness.

completeness.

consistency.

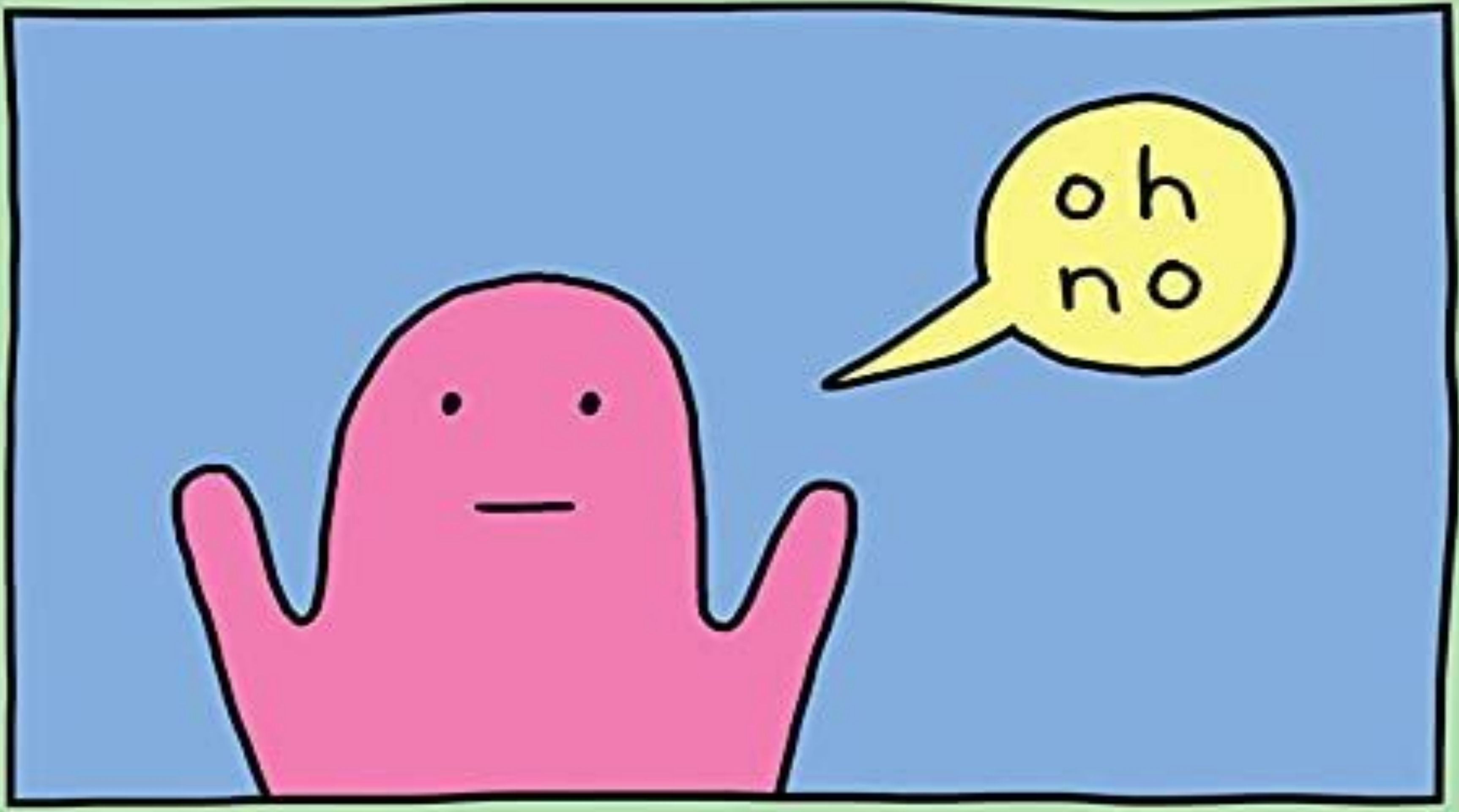
simplicity over time.

correctness over time.

completeness over time.

consistency over time.

thank you.


oh no

Taxonomy of Package Management in Programming Languages and Operating Systems

Hisham Muhammad
Kong Inc.
hisham@konghq.com

Lucas C. Villa Real
IBM Research
lucasvr@br.ibm.com

Michael Homer
Victoria University of Wellington - Wellington, New Zealand
mwh@ecs.vuw.ac.nz

Abstract

Package management is instrumental for programming languages and operating systems, and yet it is neglected by both areas as an implementation detail. For this reason, it lacks the same kind of conceptual organization: we lack terminology to classify them or to reason about their design trade-offs. In this paper, we share our experience in both OS and language-specific package manager development, categorizing families of package managers and discussing their design implications beyond particular implementations. We also identify possibilities in the still largely unexplored area of package manager interoperability.

Keywords package management, operating systems, module systems, filesystem hierarchy

1 Introduction

for `node.js` [3], a JavaScript environment. On a Mac system, the typical way to install command-line tools such as `npm` is via either Homebrew [4] or MacPorts [5], the two most popular general-purpose package managers for macOS. This is not a deliberately contrived example; it is the regular way to install development modules for a popular language in a modern platform.

The combinations of package managers change as we move to a different operating system or use a different language. Learning one's way through a new language or system, nowadays, includes learning one or more packaging environments. As a developer of modules, this includes not only using package managers but also learning to deploy code using them, which includes syntaxes for package specification formats, dependency and versioning rules and deployment conventions. Simply ignoring these environments and managing modules