Compatibilité arrière : morceaux choisis

Guillaume Munch-Maccagnoni

Inria
Équipe Gallinette, Nantes

12 mai 2026
Gallinette

Sur la compatibilité arrière

SNOYBERG

Enough with Backwards Compatibility

PUBLISHED APRIL 1, 2017

Share this

See a typo? Have a suggestion? *Edit this page on Github*

SNOYBERG

Stop breaking compatibility

PUBLISHED APRIL 1, 2018

Share this

See a typo? Have a suggestion? *Edit this page on Github*

Get new blog posts via email Email address Subscribe!

Decide on core principles against which to judge breaking changes #12

Closed

santiweight opened on Oct 31, 2021

Let's decide on some principles that guide the design of `base` and other CLC-maintained libraries, and let's write them down in a central location.

As it stands, well-thought-out and considered proposals such as Joachim Breitner's no `/=` in `Eq` proposal get accidentally hijacked by high-level discussion behind what is sufficient breakage to make a proposal untenable.

For example, within that thread alone, the proposal maker and myself both disagree on the cost of such a change. Joachim argues that the change will cause minimal issues, whereas I argue that while the breakage is relatively small, the change will cause

NOVEMBER 14, 2019

Why is the Migration to Python 3 Taking So Long?

For some companies who have already made the change years ago, it won't be an issue. However, there's a whole range of companies who won't be making the change anytime soon, for a number of reasons. What does this change mean for companies heavily utilizing the language, particularly those who may not be ready to migrate?

▲ Why Is the Migration to Python 3 Taking So Long? (stackoverflow.blog)

202 points by josep2 on Nov 14, 2019 | hide | past | favorite | 351 comments

▲ recursivecaveat on Nov 14, 2019 | next [–]

The simple reason is that there was no compelling feature to reward you for upgrading. You'd spend a tremendous amount of effort for dubious return and (until recently) a smaller ecosystem.

1. Unicode support was actually an anti-feature for most existing code. If you're writing a simple script you prefer 'garbage-in, garbage-out' unicode rather than scattering casts everywhere to watch it randomly explode when an invalid byte sneaks in. If you *did* have a big user-facing application that cared about unicode, then the conversion was incredibly painful for you because you were a real user of the old style.

2. Minor nice-to-haves like print-function, float division, and lazy ranges just hide landmines in the conversion while providing minimal benefit.

In the latest py3 versions we've finally gotten some sugar to tempt people over: asyncio, f-strings, dataclasses, and type annotations. Still not exactly compelling, but at least something to encourage the average Joe to put in all the effort.

takeda on Nov 14, 2019 | parent | next [28 more]

Scala is a Maintenance Nightmare

This post explains why Scala projects are difficult to maintain.

Scala is a powerful programming language that can make certain small teams hyper-productive.

Scala can also slow productivity by drowning teams in in code complexity or burning them in dependency hell.

Scala is famous for crazy, complex code - everyone knows about that risk factor already.

The rest of this post focuses on the maintenance burden, a less discussed Scala productivity drain.

This post generated lots of comment on Hackernews and Reddit. The Scala subreddit made some fair criticisms of the article, but the main points stand.

Li tweeted the post and empathizes with the difficulties of working in the Scala ecosystem.

Cross compiling Scala libs

Scala libraries need to be cross compiled with different Scala versions. utest v0.7.7 publishes separate JAR files for Scala 2.11, Scala 2.12, and Scala 2.13 for example.

Code

Deprecate Pervasives #1605

Merged 13 commits merged into `trunk` from `unknown repository` on Aug 27, 2018

Conversation 67 Commits 13 Checks 0 Files changed 110

ghost commented on Feb 12, 2018 • edited by ghost

Following some comments on #1010, this PR deprecates the `Stdlib.Pervasives` module in favour of just `Stdlib`.

In particular this makes the `Stdlib` module cleaner since we don't have the `Pervasives` sub-module that is immediately included.

3

c-cube commented on Feb 12, 2018 Contributor

Isn't that just going to kill retrocompatibility with any program that refers to `Pervasives` explicitely?

ghost commented on Feb 12, 2018 Author

It's only deprecated, not removed, so it should be fine.

c-cube commented on Feb 12, 2018 Contributor

Reviewers

No reviews

Assignees

No one assigned

Labels

stdlib

Projects

None yet

Milestone

No milestone

Development

Successfully merging issues.

None yet

Notifications

You're not receiving r

15 participants

```
13282     (Jacques Garrigue, report by Hugo Herbelin)
13283   - ocamlbuild: add an -ocamlmklib option to change the ocamlmklib command
13284     (Jérôme Vouillon)
13285
13286   OCaml 4.02.0 (29 Aug 2014):
13287   ---------------------------
13288
13289   (Changes that can break existing programs are marked with a "*")
13290
13291   Language features:
13292   - Attributes and extension nodes
13293     (Alain Frisch)
13294   - Generative functors (#5905)
13295     (Jacques Garrigue)
13296   * Module aliases
13297     (Jacques Garrigue)
13298   * Alternative syntax for string literals {id|...|id} (can break comments)
13299     (Alain Frisch)
13300   - Separation between read-only strings (type string) and read-write byte
13301     sequences (type bytes). Activated by command-line option -safe-string.
13302     (Damien Doligez)
13303   - #6318: Exception cases in pattern matching
13304     (Jeremy Yallop, backend by Alain Frisch)
13305   - #5584: Extensible open datatypes
13306     (Leo White)
13307
```

Immutable strings in OCaml-4.02

Why the concept is not good enough - by Gerd Stolpmann, 2014-07-04

In the upcoming release 4.02 of the OCaml programming language, the type `string` can be made immutable by a compiler switch. Although this won't be the default yet, this should be seen as the announcement of a quite disruptive change in the language. Eventually this will be the default in a future version. In this article I explain why I disagree with this particular plan, and which modifications would be better.

Of course, the fact that `string` is mutable doesn't fit well into a functional language. Nevertheless, it has been seen as acceptable for a long time, probably because the developers of OCaml did not pay much attention to strings, and felt that the benefits of a somewhat cleaner concept wouldn't outweigh the practical disadvantages of immutable strings. Apparently, this attitude changed, and we will see a new `bytes` type in OCaml-4.02. This type is accompanied by a `Bytes` module with library functions supporting it. The compiler was also extended so that `string` and `bytes` can be used interchangably by default. If, however, the `-safe-strings` switch is set on the command-line, the compiler sees `string` and `bytes` as two completely separate types.

This is a disruptive change (if enabled): Almost all code bases will need modifications in order to be compatible with the new concept. Although this will often be trivial, there are also harder cases where strings are frequently used as buffers. Before discussing that a bit more in detail, let me point out why such disruptive changes are so problematic. So far there was an implicit guarantee that your code will be compatible to new compiler versions if you stick to the well-established parts of the language and avoid experimental additions. I have in deed code that was developed

Aliasing == and != with explicit names #9080

Closed **thomasblanc** wants to merge 5 commits into `ocaml:trunk` from `thomasblanc:physical_eq`

Conversation 49 | Commits 5 | Checks 0 | Files changed 3

thomasblanc commented on Oct 30, 2019 — Contributor

I like to spend time on Stackoverflow's OCaml corner. One mistake I repetitively see from beginners is them using the `==` operator instead of `=`. (see [1] [2] [3] [4] [5] [6] and that's just the last few ones)

Of course, those newcomers usually only try to compare integers and therefore don't realize their mistake, and I pretty much always end up commenting something like "not the answer to your question but you shouldn't use `==`, or you will have a very bad surprise some day".

Unfortunately, the rise of Reason and the constant increase of newcomers only makes this problem more likely to cause serious bugs to happen in production OCaml code. That's why I'm proposing to deprecate the (much loved) `==` and `!=` operators.

Problems:

- This PR adds a new name to `Stdlib`. I know this is bad. I don't think that the name I chose would collide with existing names, but that's possible. Of all the modules in the stdlib, perhaps only `Gc` could make sense hosting those new names, I don't think it is worth creating a new module though.
- I count about 300 occurrences of `==` in trunk right now. I will change those if I get agreement in principle and a commonly accepted name.

15

`Deprecating == and != in favor of explicit names` 8b53297

Reviewers
Octac

Assignees
No one ass

Labels
None yet

Projects
None yet

Milestone
No milesto

Developm
Successfull issues.
None yet

Notificatio
You're rece

14 particip

gadmm commented on Oct 31, 2019 Contributor •••

Here is the result of running `grep -r --include \*.ml --include \*.mli -e '[a-zA-Z( ][!=]=[a-zA-Z) ]'` on opam sources: https://gitlab.com/gadmm/stdlib-experiment/raw/master/other/async_audit/phys_equal (2,16 MB). It confirms that the change can help catch improper uses, but the number of occurrences raises the question of managing the transition.

[...]

Anybody should mind that the 26000 occurrences for what is searchable publicly in opam amounts to 45% more than `Pervasives` (18000 as of Aug 2019). The deprecation of Pervasives was seen by many as a mistake, even though it was supposed to be fixable by a simple search-and-replace. It led many to wonder about the meaning of backwards compatibility and deprecation in OCaml. I would have preferred if the proponents of deprecation were also the ones to propose an assessment of the impact and a way to manage the transition.

Migrating to floatarray: experience report.

Nicolás Ojeda Bär

This post is about our use of OCaml inside LexiFi and how we managed a large refactoring of our codebase that we enterprised in order to take advantage of a new feature of OCaml.

Namely, in OCaml 4.08 a specific type `floatarray` for arrays of floating-point numbers was introduced which has a number of advantages relative to the usual type `float array` (see below for some details or this blog post to learn more about the motivation for this change).

Feedback

We want to reiterate that as with any alpha release of OCaml, we're keen to hear about bugs and performance regressions. The move to parallel OCaml may bring new debugging challenges, but it remains the case that pure OCaml programs which do not use unsafe features should absolutely never crash. We'll be taking part in the discussion on GitHub, Discuss, and Twitter.

The change in major version number (from 4.*n* to 5.*n*) may result in minor breaking changes which affect your packages, particularly if you've been allowing some deprecation warnings to slip through in the past! We'll be following up with more information about the required tweaks that may be required for packages supporting both old and new versions of OCaml, as well as with specifics on the testing infrastructure.

Survey

Semantic Versioning 2.0.0

Summary

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes
2. MINOR version when you add functionality in a backward compatible manner
3. PATCH version when you make backward compatible bug fixes

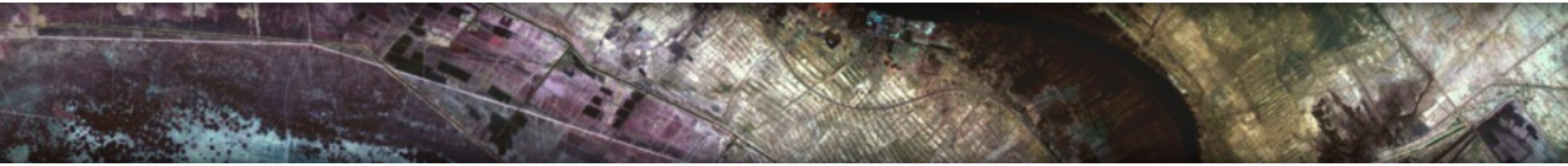
Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Introduction

In the world of software management there exists a dreaded place called “dependency hell.” The bigger your system grows and the more packages you integrate into your software, the more likely you are to find yourself, one day, in this pit of despair.

In systems with many dependencies, releasing new package versions can quickly become a nightmare. If the dependency specifications are too tight, you are in danger of version lock (the inability to upgrade a package without having to release new versions of every dependent package). If dependencies are specified too loosely, you will inevitably be bitten by version promiscuity (assuming compatibility with more future versions than is reasonable). Dependency hell is where you are when version lock and/or version promiscuity prevent you from easily and safely moving your project forward.

As a solution to this problem, we propose a simple set of rules and requirements that dictate how version



We're drowning

MARCH 10, 2022

We live in a golden age of software reuse. We've never before had such a wealth of freely available code, in so many languages, so easy to find and install.

Matthew Childs / Reuters

And yet, we're drowning. We slap together rickety rowboats and toss them out on PyPI Ocean and npm Sea, then act surprised when the changes flood in. We ignore the flood as long as we can, then patch the holes with duct tape and bilge pumps as if they can hold back the tide. They cannot.

It's a wonderful, horrible problem, and I don't know what to do about it.

Se soucier des utilisateurs

Linus Torvalds schools Lennart Poettering on the importance of users

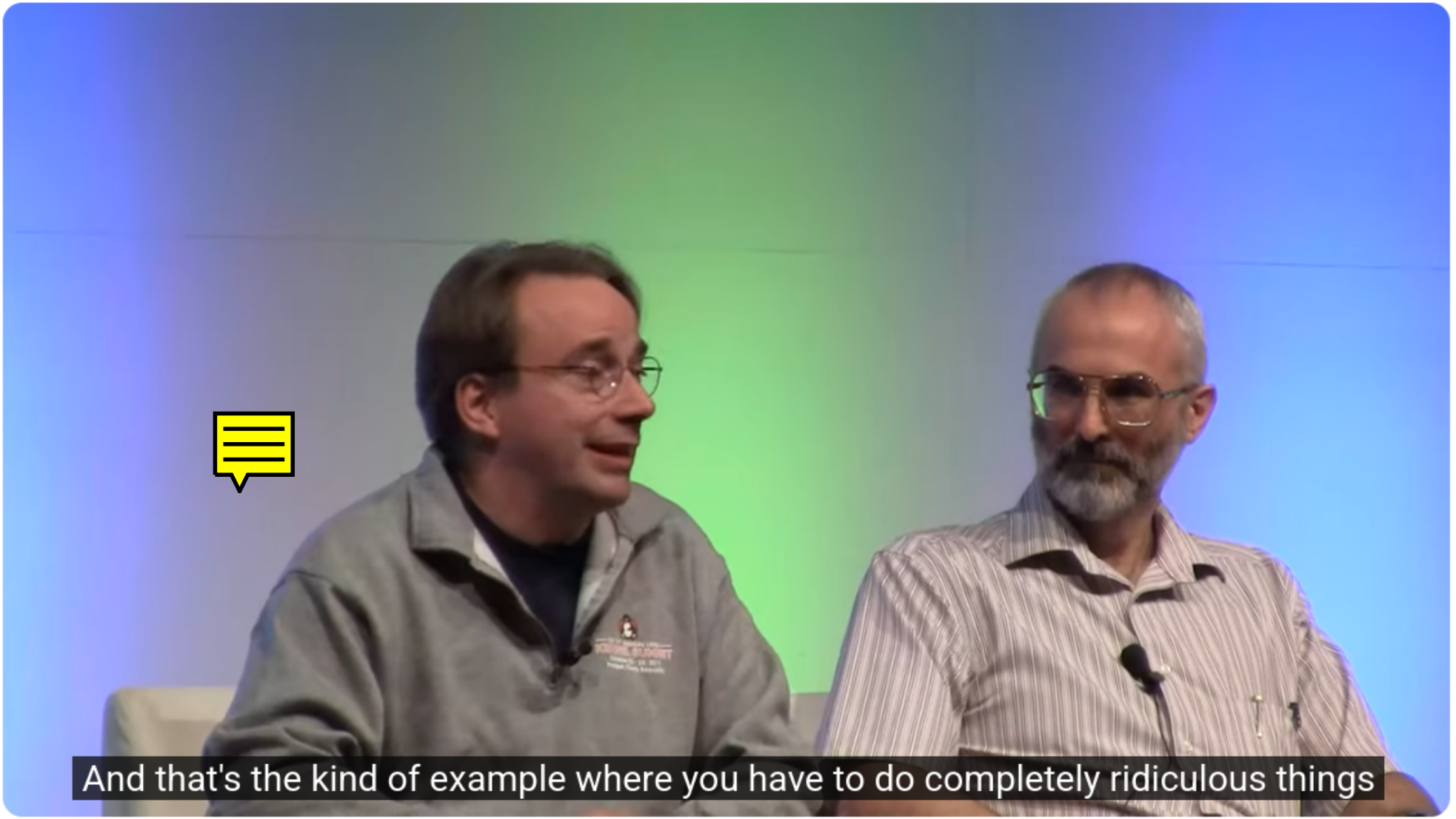
Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save



Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers
Subscribe

2.5K
Share
Save

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

From Linus Torvalds <>
Date Sun, 15 Sep 2019 15:00:06 -0700
Subject Linux 5.3

[...]

What's instructive about it is that I reverted a commit that wasn't actually buggy. In fact, it was doing exactly what it set out to do, and did it very well. [...]

[...] The reverted commit didn't change any
API's, and it didn't introduce any new bugs. But it ended up exposing another problem, and as such caused a kernel upgrade to fail for a user. So it got reverted.

The point here being that we revert based on user-reported _behavior_, not based on some "it changes the ABI" or "it caused a bug" concept.
[...]

MAY 24, 2000 *by* JOEL SPOLSKY

Strategy Letter II: Chicken and Egg Problems

CEO, NEWS

[…]

Windows 95? No problem. Nice new 32 bit API, but it still ran old 16 bit software perfectly. Microsoft obsessed about this, spending a big chunk of change testing every old program they could find with Windows 95. Jon Ross, who wrote the original version of SimCity for Windows 3.x, told me that he accidentally left a bug in SimCity where he read memory that he had just freed. Yep. It worked fine on Windows 3.x, because the memory never went anywhere. Here's the amazing part: On beta versions of Windows 95, SimCity wasn't working in testing. Microsoft tracked down the bug and *added specific code to Windows 95 that looks for SimCity*. If it finds SimCity running, it runs the memory allocator in a special mode that doesn't free memory right away. That's the kind of obsession with backward compatibility that made people willing to upgrade to Windows 95.

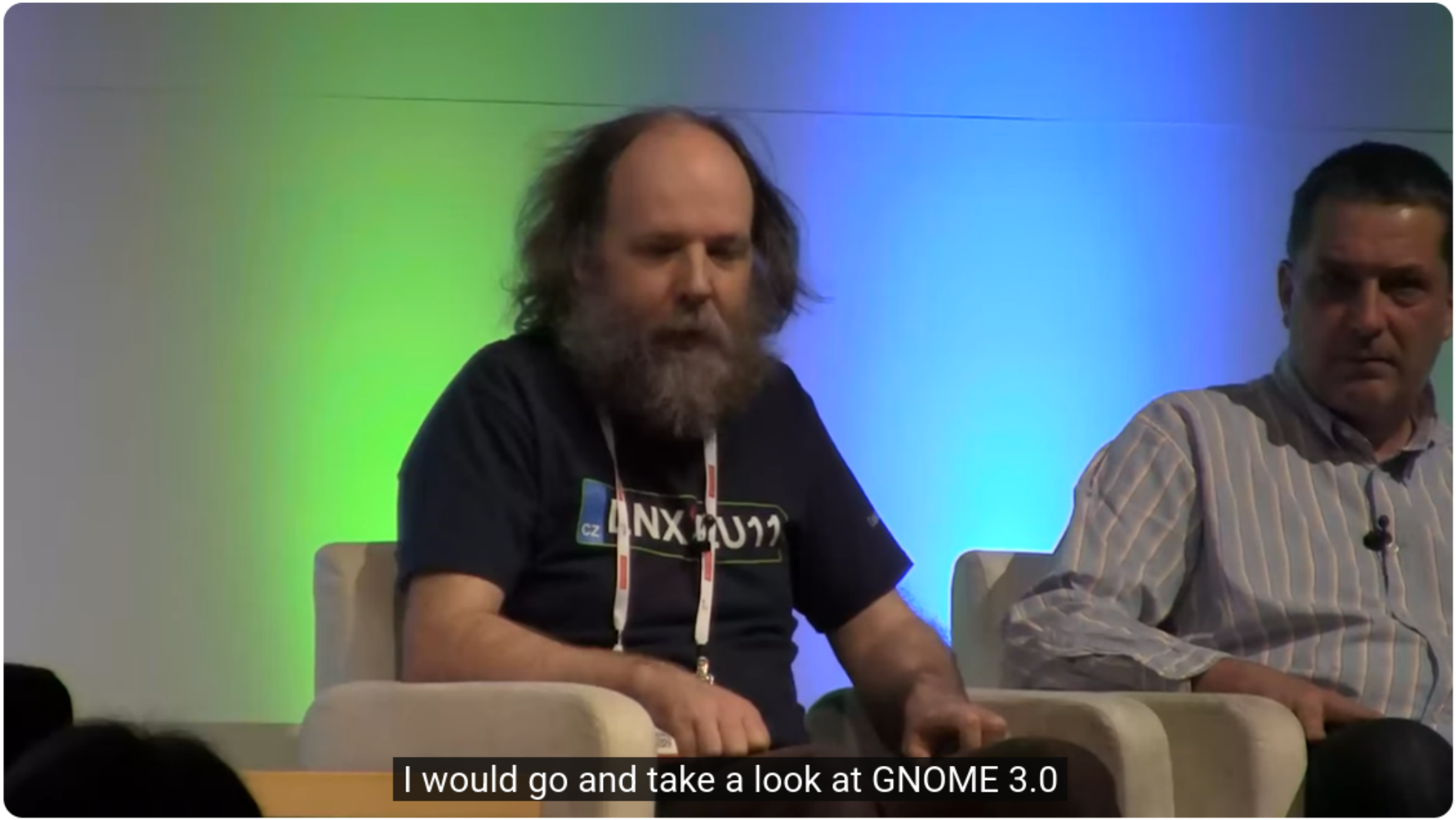
You should be starting to get some ideas about how to break the chicken and egg problem: provide a backwards compatibility mode which either delivers a truckload of chickens, or a truckload of eggs,

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers
Subscribe
2.5K
Share
Save

YouTube FR

Search



Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers
Subscribe
2.5K
Share
Save

Linus Torvalds schools Lennart Poettering on the importance of users

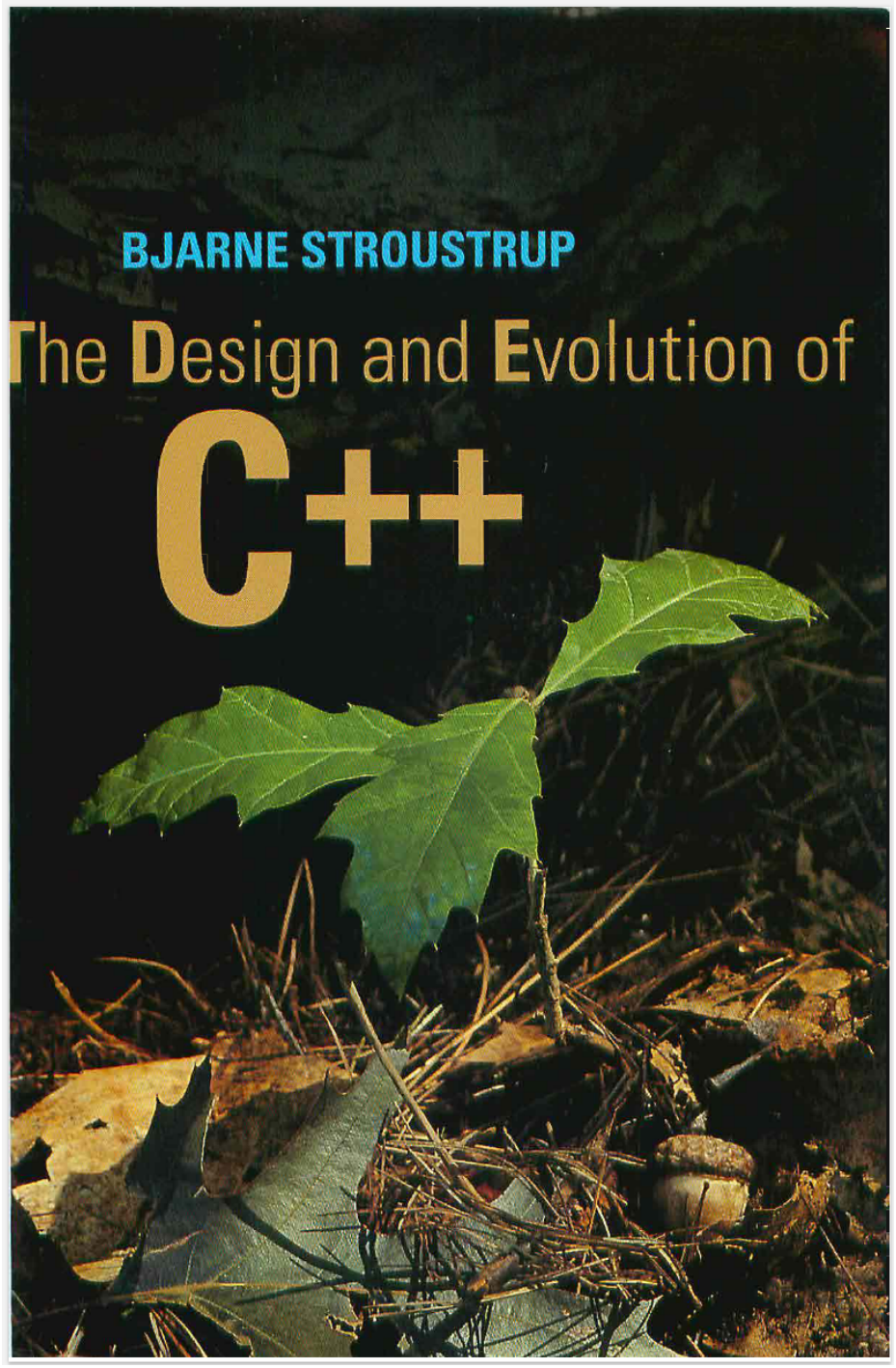
Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save


BJARNE STROUSTRUP
The Design and Evolution of
C++

sive ''systems'' like those of Plato and Kant fascinating, yet fundamentally unsatisfying in that they appear to me dangerously remote from everyday experiences and the essential peculiarities of individuals.

I find Kierkegaard's almost fanatical concern for the individual and keen psychological insights much more appealing than the grandiose schemes and concern for humanity in the abstract of Hegel or Marx. Respect for groups that doesn't include respect for individuals of those groups isn't respect at all. Many C++ design decisions have their roots in my dislike for forcing people to do things in some particular way. In history, some of the worst disasters have been caused by idealists trying to force people into ''doing what is good for them.'' Such idealism not only leads to suffering among its innocent victims, but also to delusion and corruption of the idealists applying the force. I also find idealists prone to ignore experience and experiment that inconveniently clashes with dogma or theory. Where ideals clash and sometimes even when pundits seem to agree, I prefer to provide support that gives the programmer a choice.

My preferences in literature have reinforced this unwillingness to make a decision based on theory and logic alone. In this sense, C++ owes as much to novelists and essayists such as Martin A. Hansen, Albert Camus, and George Orwell, who never saw a computer, as it does to computer scientists such as David Gries, Don Knuth, and Roger Needham. Often, when I was tempted to outlaw a feature I personally

disliked, I refrained from doing so because I did not think I had the right to force my views on others. I know that much can be achieved in a relatively short time by the energetic pursuit of logic and by ruthless condemnation of ''bad, outdated, and confused habits of thought.'' However, the human cost is often high. A high degree of tolerance and acceptance that different people do think in different ways and strongly

The Evils of Paradigms

Or

Beware of one-solution-fits-all thinking

Bjarne Stroustrup

There is a notion (popularized by the American philosopher and historian Thomas Kuhn) of progress happening through "paradigm shifts" where an old (supposedly bad) way of doing or understanding something (a "paradigm") is replaced by a new (supposedly good) paradigm. Popular examples are the shift from Newtonian physics to Einstein's universe and the shift from geocentric view of the universe to the heliocentric. In programming, some people deem imperative, object-oriented, and functional programming different paradigms. I think the very notion of a paradigm does harm to use and to design because people all too easily fall into the trap of considering only one paradigm "good" and then try to fit everything into it, discarding all aspects of alternative "paradigms" as wrong or inferior (aka "If your only tool is a hammer, everything looks like a nail").

I reject that notion, as did Kristen Nygaard, who invented object-oriented programming. Instead, I see progress on a large scale as necessarily evolutionary, rather than revolutionary, progressing by older concepts, techniques, and tools being gradually absorbed into a more general framework.

- For a century after Copernicus, calculations of planet orbits were best done using the old geocentric model (cycles and epicycles!) because the heliocentric model had not reached the maturity needed to accurately model the sky. Even when a new way of looking at things is fundamentally better, it may not yet be sufficiently mature to be useful for practical tasks.
- Einstein's model of the universe is different from Newton's, and in important cases far more accurate. However, we spend almost all of our time in Newton's universe: Relativity is useful only for relatively esoteric topics, such as GPS implementation, HEP, and astronomy. To go shopping or to fly to Jacksonville from anywhere on earth, Newton's view is sufficiently accurate and much easier for us to deal with. Even when a new way of looking at things is fundamentally better, it may actually be inferior for simple, everyday tasks.

What does this have to do with C++? C++ is built on the idea of incremental growth and the gradual replacement of older facilities with newer ones where appropriate. Examples:

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

YouTube FR

Search

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers
Subscribe
2.5K
Share
Save

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

YouTube FR

Search

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers
Subscribe
2.5K
Share
Save

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

Linus Torvalds schools Lennart Poettering on the importance of users

Felipe Contreras
529 subscribers

Subscribe

2.5K

Share

Save

Des tentatives pour
casser les choses de façon responsable

SemVer Considered Harmful

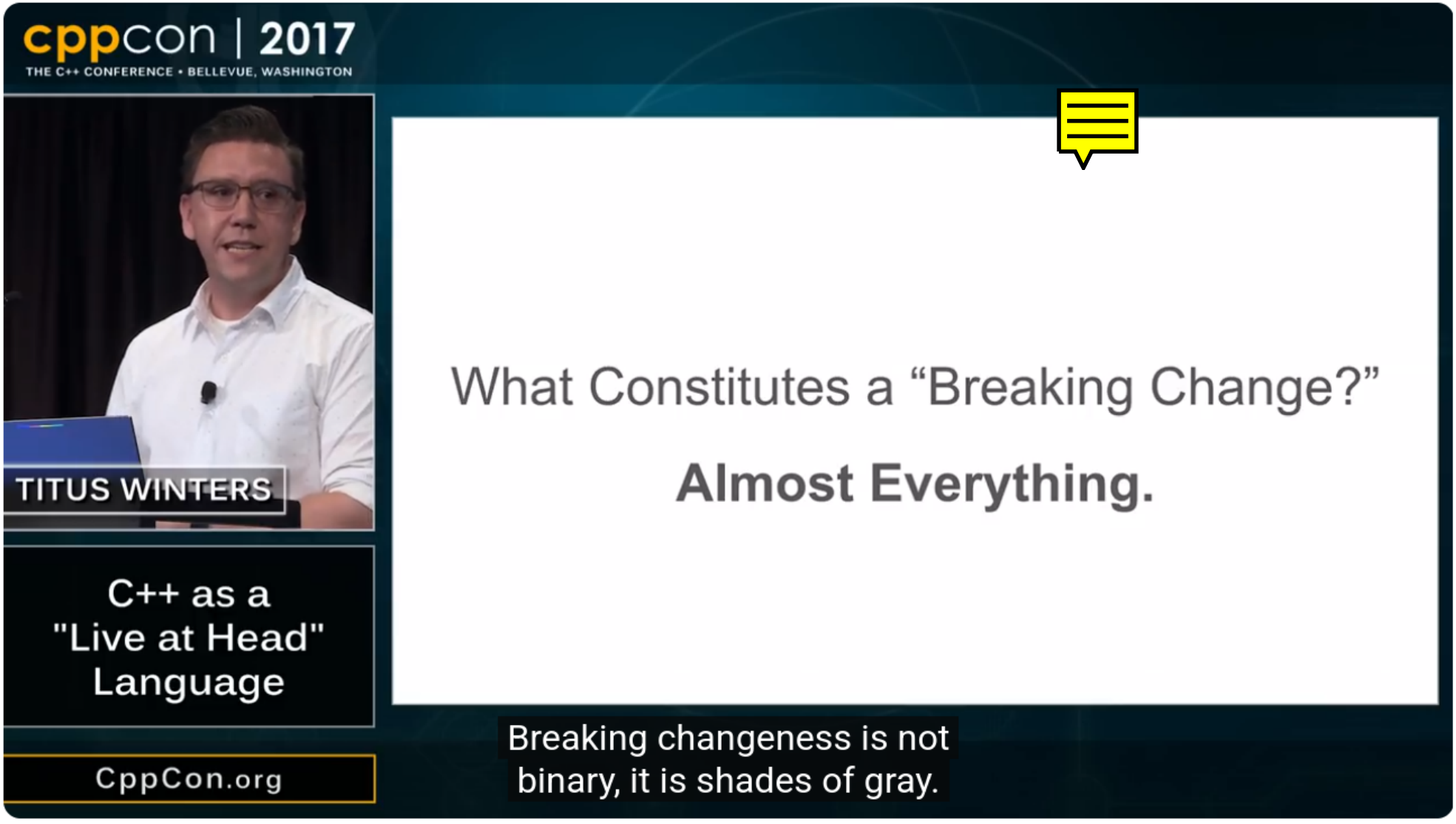
2020/07/04

In the past ten years or so, Semantic Versioning a.k.a "SemVer" has become extremely popular in the software development world. The idea is that libraries and services can convey information to users about how the application programming interface (API) of that library/package/service is evolving just using the version number. This information is conveyed through three dotted numbers that form a logical clock for totally ordering changes to the software API:

Semantic Version Numbers

```
====================== Specification ======================

Version = <Major>.<Minor>.<Patch>

Major: this number goes up when the public API breaks
Minor: this number goes up when the public API changes
Patch: this number goes up when the public API doesn't change

======================= Examples ==========================

# A Minor API change happened, safe to upgrade
1.4.5 -> 1.5.0
```

YouTube FR
Search



CppCon 2017: Titus Winters “C++ as a "Live at Head" Language”

CppCon
181K subscribers
Join
Subscribe
719
Share
Save
Thanks

Abseil Compatibility Guidelines

Abseil follows Google's **Foundational C++ Support Policy** and **OSS Library Breaking Change Policy**.

In general, we avoid making backwards incompatible changes to our C++ APIs (see below for the definition of "API"). Sometimes such changes yield benefits to our customers, in the form of better performance, easier-to-understand APIs, and/or more consistent APIs. When these benefits warrant it, we will announce these changes prominently in our commit messages as well as in the release notes for **LTS releases**. Nevertheless, we have found that at scale, **every change is potentially a breaking change** for some user. Though we take reasonable efforts to prevent this, it is possible that backwards incompatible changes go undetected and, therefore, undocumented. We apologize if this is the case and welcome feedback or **bug reports** to rectify the problem.

Non-Atomic Refactoring and Software Sustainability

Titus Winters
Google, Inc
titus@google.com

ABSTRACT

Sustainability is the ability of a project / codebase / organization to react to necessary changes over its expected lifespan. At a large enough scale, or with enough disconnect between dependencies, sustainability comes from application of both technical and non-technical approaches. On the technical side, I advocate for restraint among API providers on making arbitrary changes, and use of non-atomic refactoring techniques when more invasive changes are required; such techniques are employed in many Google projects, and in programming languages like Go and C++, to allow more flexible changes to language standards over time. On the non-technical side, I argue for a clear separation of responsibilities (providers need to do the bulk of the work for the update), as well as a growing need to document acceptable usage of an API, be it a library or programming language. In many languages, there are very few changes to an API that are provably safe without this idea: just because a users code currently works does not mean that it is supported and can be expected to continue to work indefinitely under maintenance. Taken together, these two approaches form what I believe to be a minimum set of requirements when approaching software sustainability.

CCS CONCEPTS

of the hard parts of engineering come from dealing with time: compatibility over time, dealing with changes to underlying infrastructure and dependencies, and working with legacy code or data. Fundamentally, it is a different task to produce a programming solution to a problem (that solves the current instances of the problem) vs. an engineering solution (that solves current instances, future instances that we can predict, and - through flexibility - allows updates to solve future instances we may not be able to predict).

I have described software sustainability for years in very similar terms: your code is sustainable if you are capable of updating it to respond to necessary change for the expected lifespan of the project. For programming projects (quick hacks, school assignments, one-off academic analyses), it is likely that there are no necessary changes - there probably won't be a new language version published in the days or weeks that project is alive. For longer term projects, especially successful OSS or industry projects, it is entirely possible that the code in question needs to live for years or decades. It should come as no surprise there are differences in approach between these styles of projects.

From what I've seen, there is only limited awareness of sustainability issues on the part of API providers or consumers. Few providers are clear about what to expect from their APIs, and thus consumers assume that anything that works now will work indefinitely. Speaking as someone that has been

rust-lang/rust

#78802 Implement network primitives with ideal Rust layout, not C...

127 comments 19 reviews 10 files +172 −247

faern • November 6, 2020 8 commits

github.com Ouvrir

700 · 77

GeneReddit123 AO • il y a 4 a

The summary of the change is that Rust now uses its own, native representation and APIs for an IP address and friends (e.g. netmask, socket address), both for IPv4, and IPv6, rather than relying on `libc`.

This will allow moving IP functionality from `std` to `core`, potentially improve optimization, improve `const` support of IP addresses, and some ergonomic improvements. It also further reduces Rust's dependence on `libc` in general.

However, this change comes with some risk. Many popular crates including `mio` (incorrectly and unsafely) assumed Rust always uses C reprensetation of IP addresses, and (unsafely, and in violation of Rust's API guarantees) read the underlying memory directly, rather than rely on Rust's APIs. Continuing to do so after this change **will result in undefined behavior.**

The Rust team put a lot of effort in reaching out to every incorrect crate it found, which have patched their logic and yanked the old crates due to them having a security vulnerability, forcing users to upgrade. However, if any crate was missed, or if a Rust user updates their Rust version yet (somehow) continues to use an old version of a crate, they may experience undefined behavior. The fault for it technically lies on the crate (due to using incorrect unsafe logic all along) rather than with the Rust team, but since this is a big change, the Rust team still wants to do as much as it can to make the transition smooth and safe.

Ppxlib.0.22: an update on the state of ppx

Community

jeremiedimino Maintainer

Feb 2021

To give a bit of context as to why we switched from the “forever stable” to the “upgrade the world” plan, the old ppx project was becoming technically too complex. We often had to remind ourselves how the whole thing was supposed to work. We were basically building something that was as complex as camlp4 was. Sometimes such complexity is justified, such as in the compiler or build system, but in this case it didn’t seem justified to us and we decided to go for a simpler design.

With our new plan, it will be easy for anyone to understand what we did and contribute. In the long run, if the set of people who maintain the ppx ecosystem changes, it will be easy for the new people to pick up. On the compiler side, the process for upgrading Astlib will be simple, and especially much more straightforward and easier to reason about than the process we would have ended up with the “forever stable” plan.

Finally, I’d just like to point out that upgrading the ppx world used to be painful because the ecosystem was very fragmented. As the ecosystem is becoming more unified, it is also becoming much simpler to upgrade the world, as Nathan mentioned in his post. In the long run this should benefit proprietary code bases as well.

6 Reply

Breaking PPX API changes and the package ecosystem

Ecosystem

NathanReb Jul 2025

You are raising perfectly valid concerns. When we initially decided to use this “update the world” approach, the AST was much more stable than it has been over the last few compiler releases.

The main problem here is that, though we do send patches upstream, we do not control all reverse dependencies and cannot ensure that compatible versions are released on time, we’re merely “helping out” but this doesn’t guarantee the stability of the ecosystem.

Now might be a good time to think of a new approach and a slightly more stable API. I’ve been thinking about it and I would like to submit a proposal to the community soon.

2 Reply

Les Éditions de Rust

What are Editions?

In May 2015, the release of Rust 1.0 established "stability without stagnation" as a core Rust axiom. Since then, Rust has committed to a pivotal rule: once a feature is released through stable, contributors will continue to support that feature for all future releases.

However, there are times when it's useful to make backwards-incompatible changes to the language. A common example is the introduction of a new keyword. For instance, early versions of Rust didn't feature the `async` and `await` keywords.

If Rust had suddenly introduced these new keywords, some code would have broken: `let async = 1;` would no longer work.

Rust uses **editions** to solve this problem. When there are backwards-incompatible changes, they are pushed into the next edition. Since editions are opt-in, existing crates won't use the changes unless they explicitly migrate into the new edition. For example, the latest version of Rust doesn't treat `async` as a keyword unless edition 2018 or later is chosen.

Each crate chooses its edition within its `Cargo.toml` file. When creating a new crate with Cargo, it will automatically select the newest stable edition.

Editions do not split the ecosystem

Epochs: a backward-compatible language evolution mechanism

Document #:	P1881R1
Date:	2020-01-12
Project:	Programming Language C++ Evolution Working Group Incubator (EWGI)
Reply-to:	Vittorio Romeo <vittorio.romeo@outlook.com>

Contents

Bibliographie

- Why Is the Migration to Python 3 Taking So Long? HN discussion
  https://news.ycombinator.com/item?id=21536468
- “Linus Torvalds schools Lennart Poettering on the importance of users”
  https://www.youtube.com/watch?v=Nn-SGblUhi4
- Strategy Letter II : Chicken and Egg Problems :
  https://www.joelonsoftware.com/2000/05/24/strategy-letter-ii-chicken-and-egg-problems/
- CppCon 2017 : Titus Winters “C++ as a "Live at Head" Language”
  https://www.youtube.com/watch?v=tISy7EJQPzI
- Abseil Compatibility Guidelines https ://abseil.io/about/compatibility
- Titus Winters. Non-Atomic Refactoring and Software Sustainability.
  https://dl.acm.org/doi/10.1145/3194793.3194794
- Stroustrup. The Evils of Paradigms Or Beware of one-solution-fits-all thinking.
  https://www.open-std.org/JTC1/SC22/WG21/docs/papers/2018/p0976r0.pdf
- The Rust Edition Guide – What are Editions?
  https://doc.rust-lang.org/edition-guide/editions/index.html
- Epochs : a backward-compatible language evolution mechanism [for C++]
  https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1881r1.html