20 more years of bootstrapping

SBCL – past, present, future

Christophe Rhodes

April 1, 2019

Outline

Contact details

Christophe Rhodes <xof@google.com>

Contact details

Christophe Rhodes <xof@google.com>

Contact details

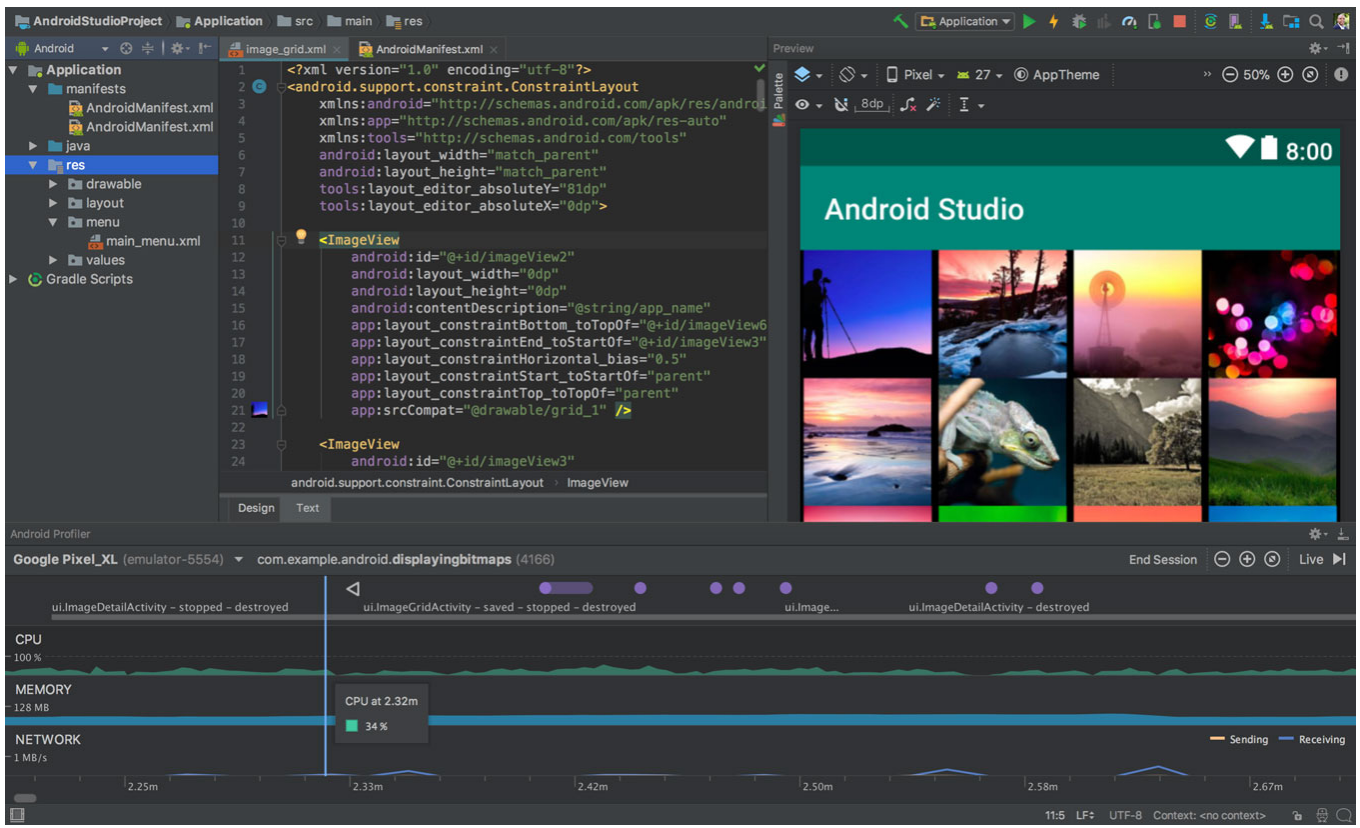
wait, what?

Currently

Software Engineer, Android Developer Tools

Currently

Software Engineer, Android Developer Tools



Previously

Senior Lecturer, Deputy Head of Department, Computing, Goldsmiths

Previously

Senior Lecturer, Deputy Head of Department, Computing, Goldsmiths

Previously[2]

Chief Scientist, Chairman, Teclo Networks

Previously[2]

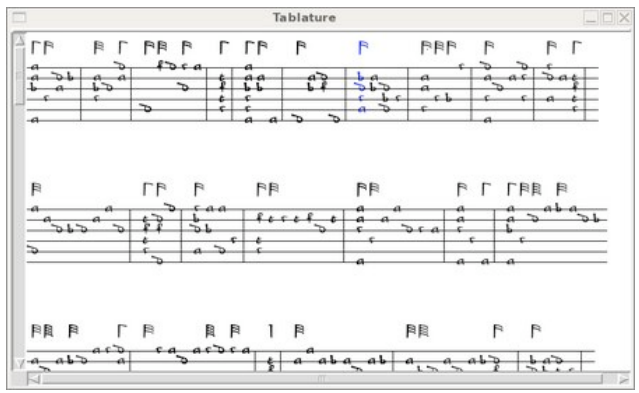
Chief Scientist, Chairman, Teclo Networks

Previously[3]

Research Associate, Computer Music

Previously[3]

Research Associate, Computer Music

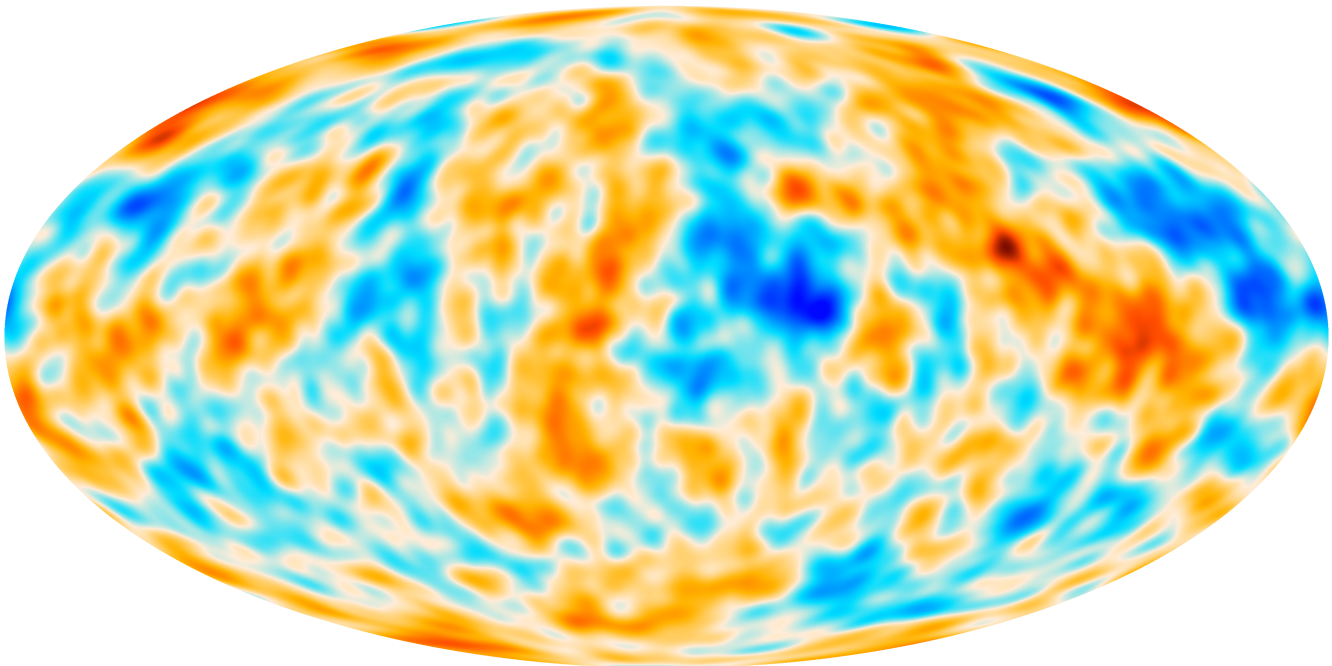


Previously[4]

PhD student, speculative cosmology

Previously[4]

PhD student, speculative cosmology



Concurrently

SBCL developer and maintainer

- first patch: April 2001
  - (now old enough to vote!)
- commit access, March 2002

Concurrently

Theme: no-one employs me to write Common Lisp

- not likely to change any time soon
- not likely to negatively affect my ability to contribute to SBCL either

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime
- ... and garbage collector

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime
- ... and garbage collector
- ... and interpreter (or three)

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime
- ... and garbage collector
- ... and interpreter (or three)
- ... and developer tools (profiler, stepper, tracer)

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime
- ... and garbage collector
- ... and interpreter (or three)
- ... and developer tools (profiler, stepper, tracer)
- ... and language extensions

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

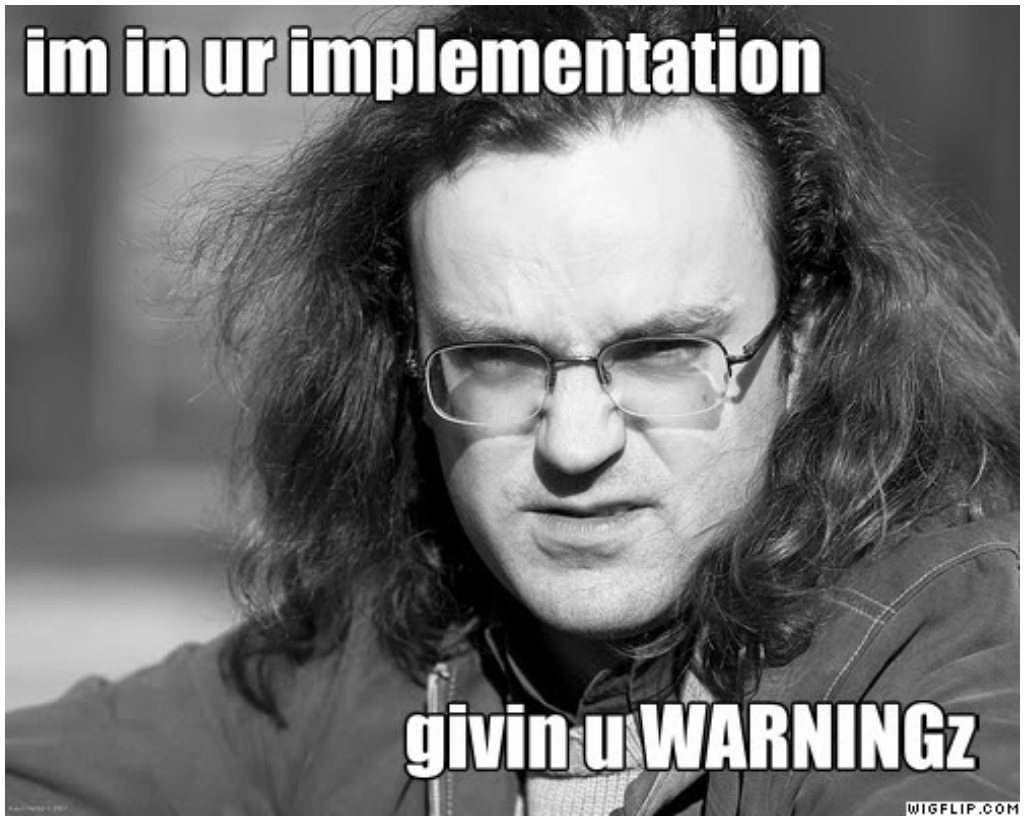
- ... and runtime
- ... and garbage collector
- ... and interpreter (or three)
- ... and developer tools (profiler, stepper, tracer)
- ... and language extensions
- ... and contributed modules

Steel Bank Common Lisp

A somewhat-optimising mostly-portable native code Common Lisp compiler

- ... and runtime
- ... and garbage collector
- ... and interpreter (or three)
- ... and developer tools (profiler, stepper, tracer)
- ... and language extensions
- ... and contributed modules
- ... and WARNINGz

Steel Bank Common Lisp



Prehistory: Spice Lisp

1980/1-1984/5
(I know very little about this period)

Prehistory: Carnegie-Mellon University Common Lisp

1984/5-

- [1987] David B. Macdonald, Scott E. Fahlman, Skef Wholey, *Internal Design of CMU Common Lisp on the IBM RT PC*
  - RT/Mach

Prehistory: Carnegie-Mellon University Common Lisp

1984/5-

- [1987] David B. Macdonald, Scott E. Fahlman, Skef Wholey, *Internal Design of CMU Common Lisp on the IBM RT PC*
  - RT/Mach
- [1992] Rob Maclachlan, *CMU Common Lisp user's manual*
  - MIPS/Mach
  - SPARC/Mach

Prehistory: Carnegie-Mellon University Common Lisp

1984/5-

- [1987] David B. Macdonald, Scott E. Fahlman, Skef Wholey, *Internal Design of CMU Common Lisp on the IBM RT PC*
  - RT/Mach
- [1992] Rob Maclachlan, *CMU Common Lisp user's manual*
  - MIPS/Mach
  - SPARC/Mach
  - (external use of CMU CL begins: volunteer port to SPARC/SunOS)

Prehistory: Carnegie-Mellon University Common Lisp

1984/5-

- [1987] David B. Macdonald, Scott E. Fahlman, Skef Wholey, *Internal Design of CMU Common Lisp on the IBM RT PC*
  - RT/Mach
- [1992] Rob Maclachlan, *CMU Common Lisp user's manual*
  - MIPS/Mach
  - SPARC/Mach
  - (external use of CMU CL begins: volunteer port to SPARC/SunOS)
- [1994] DARPA funding for CMU CL stops
- [1997] Generational mostly-copying conservative garbage collector implementation (Douglas Crosher; see also Scieneer Common Lisp)

Prehistory: Carnegie-Mellon University Common Lisp

1984/5-

- [1987] David B. Macdonald, Scott E. Fahlman, Skef Wholey, *Internal Design of CMU Common Lisp on the IBM RT PC*
  - RT/Mach
- [1992] Rob Maclachlan, *CMU Common Lisp user's manual*
  - MIPS/Mach
  - SPARC/Mach
  - (external use of CMU CL begins: volunteer port to SPARC/SunOS)
- [1994] DARPA funding for CMU CL stops
- [1997] Generational mostly-copying conservative garbage collector implementation (Douglas Crosher; see also Scieneer Common Lisp)
- [1997ish] Dan Barlow, CMUCL-on-Linux-HOWTO

Some hints on building the system which you might find useful

1. *if you change the compiler, you will have to recompile all the sources before the change is reflected in the system [...]*
2. *changing the miscops should be done with care [...] note that this requires building a new cold core file and a final core file before the change is reflected in the system [...]*
3. *changing sources in the code directory should be fairly straight forward [...] if you change something that a lot of files depend on [...] you will have to recompile everything and build a new cold core file and a core file*
4. *if you make a fairly major change, it is a good idea to go through the process of building a core file at least two or three times. If things are still working at the end of this, your change is probably correct and shouldn't cause any serious trouble.*

– Macdonald, Fahlman and Wholey (1987)

Example: byte

- `(byte size position)` ⟹ <implementation-defined>

Example: byte

- `(byte size position)` ⟹ <implementation-defined>
  - `(cons size position)` (SBCL)
  - `(ash (1- (ash 1 size)) position)` (CCL)
  - `(byte :size size :position position)` (CLISP?)

Example: byte

- `(byte size position)` ⟹ <implementation-defined>
  - `(cons size position)` (SBCL)
  - `(ash (1- (ash 1 size)) position)` (CCL)
  - `(byte :size size :position position)` (CLISP?)
  - `(cons position size)` (SBCL, briefly)

Example: byte

- `(byte size position)` ⟹ <implementation-defined>
  - `(cons size position)` (SBCL)
  - `(ash (1- (ash 1 size)) position)` (CCL)
  - `(byte :size size :position position)` (CLISP?)
  - `(cons position size)` (SBCL, briefly)
- `(ldb <byte> number)` ⟹ very much standard-defined

Example: byte

- `(byte size position)` ⟹ <implementation-defined>
  - `(cons size position)` (SBCL)
  - `(ash (1- (ash 1 size)) position)` (CCL)
  - `(byte :size size :position position)` (CLISP?)
  - `(cons position size)` (SBCL, briefly)
- `(ldb <byte> number)` ⟹ very much standard-defined

what is an optimizing cross-compiler to do?

- not optimize? Slow compiler, unhappy users
- try to optimize, get it wrong? No compiler, unhappy developers
- build a model of SBCL's `byte` in the cross-compiler...
  - and use that for cross-compiler optimizations

Example: byte

- `(byte size position)` ⟹ <implementation-defined>
  - `(cons size position)` (SBCL)
  - `(ash (1- (ash 1 size)) position)` (CCL)
  - `(byte :size size :position position)` (CLISP?)
  - `(cons position size)` (SBCL, briefly)
- `(ldb <byte> number)` ⟹ very much standard-defined

what is an optimizing cross-compiler to do?

- not optimize? Slow compiler, unhappy users
- try to optimize, get it wrong? No compiler, unhappy developers
- build a model of SBCL's `byte` in the cross-compiler...
  - and use that for cross-compiler optimizations

similar problem with `boole`, but luckily no-one uses that.

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining
  - ... and stores that information in a structure
  - a `defstruct-description` structure

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining
  - ... and stores that information in a structure
  - a `defstruct-description` structure
- what happens when we discover a bug in `defstruct`?

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining
  - ... and stores that information in a structure
  - a `defstruct-description` structure
- what happens when we discover a bug in `defstruct`?
  - we need to add a field to `defstruct-description`...
  - ... argh, we need to redefine the thing we're using

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining
  - ... and stores that information in a structure
  - a `defstruct-description` structure
- what happens when we discover a bug in `defstruct`?
  - we need to add a field to `defstruct-description`...
  - ... argh, we need to redefine the thing we're using

what is a sanely-bootstrappable compiler to do?

- same solution as before: build a model of SBCL's `defstruct-description` (represented as a host object), use it to cross-compile SBCL

Example: defstruct-description

- `defstruct` parses its body to understand what structure it is defining
  - ... and stores that information in a structure
  - a `defstruct-description` structure
- what happens when we discover a bug in `defstruct`?
  - we need to add a field to `defstruct-description`...
  - ... argh, we need to redefine the thing we're using

what is a sanely-bootstrappable compiler to do?

- same solution as before: build a model of SBCL's `defstruct-description` (represented as a host object), use it to cross-compile SBCL
- here's what happens if you don't do that:

Example: defstruct-description

```
;; For bootstrapping the conc-name inheritance fix, part 1.
;;
;; You need to do a build loading just this file but NOT boot6-b.lisp.
;; The next build MUST NOT use this file and load up boot6-b
;; instead.
;;
;; Use the CLOBBER-IT restart when prompted for a restart.
```

– CMUCL `bootfiles/18d/boot6-a.lisp`

Workarounds

don't implement your lisp in itself:

- write `eval` in support language (e.g. C, C++ or Java)
  - use that `eval` to bring up the rest of the system
- ECL, ABCL, CLASP, CLISP (and XCL, lisp500, Emacs Lisp)

Workarounds

don't implement your lisp in itself:

- write `eval` in support language (e.g. C, C++ or Java)
  - use that `eval` to bring up the rest of the system
- ECL, ABCL, CLASP, CLISP (and XCL, lisp500, Emacs Lisp)

bridge the gap with bootstrap files:

- mutate the state of the compiling instance
- examples:
  - mutate metadata about functions or structures
  - define a (maybe stub or no-op) function so that it can be called
- CCL, CMUCL, Lispworks*, Allegro*

Workarounds

Neither workaround is satisfactory:

- aesthetic / intellectual achievement of writing in the language you're implementing

Workarounds

Neither workaround is satisfactory:

- aesthetic / intellectual achievement of writing in the language you're implementing
- barrier to entry: sustainability of software through ease of ability to reason about it

SBCL's solution:

- Implement your lisp compiler in Common Lisp
- Use your lisp compiler to compile your lisp compiler

Workarounds

Neither workaround is satisfactory:

- aesthetic / intellectual achievement of writing in the language you're implementing
- barrier to entry: sustainability of software through ease of ability to reason about it

SBCL's solution:

- Implement your lisp compiler in Common Lisp
- Use your lisp compiler to compile your lisp compiler

(DANGER UNEXPLODED MINDS)

Announcement

```
Date: Tue, 14 Dec 1999 18:53:46 -0600
From: William Harold Newman <william.newman@airmail.net>
To: cmucl-imp@cons.org
Subject: It's alive! (SBCL, a CMU CL variant which bootstraps cleanly)

[...]

The differences are already significant enough that I believe that
calling the new system CMU CL would cause confusion, so I'm
tentatively calling it SBCL instead (for "Steel Bank Common Lisp",
after the industries where Carnegie and Mellon made the big bucks).
```

Announcement

```
[slothrop] it's tough to build apparently
[slothrop] cmucl, I mean
[wnewman] I did finally figure out how to do it, in my own way.
```

- `#lisp` IRC, 2004-08-30

Wild West

Break all the things

- no users: break backwards compatibility all the time
  - freedom to make mistakes
- experiment with things, learn

Events

- UKUUG 2003: Dan Barlow, *n things every programmer should know about signal handling*
- ELSW 2004: *Grouping Common Lisp Benchmarks*
- ECLM 2005: *Recent developments in SBCL*
- ILC 2007: *User-extensible sequences in Common Lisp*
- SSS 2008: *A Sanely-Bootstrappable Common Lisp*
- ELS 2008: Jim Newton, *Custom specializers in object-oriented Lisp*

ITA, QPX and RES

- QPX: airfare search
  - originally: CMUCL and Allegro
  - SBCL eventually (2005ish?)
  - funded work for Nikodemus Siivola, Juho Snellman
  - significant work from Martin Cracauer and team

ITA, QPX and RES

- QPX: airfare search
  - originally: CMUCL and Allegro
  - SBCL eventually (2005ish?)
  - funded work for Nikodemus Siivola, Juho Snellman
  - significant work from Martin Cracauer and team
- RES: airport operations middleware
  - 2006 venture capital, project announcement, Air Canada partnership
  - 2009 suspension of Air Canada involvement
  - 2010 Google acquisition

Oh no, now we have users

Google Summer of Code

- Four students over two years:
  - efficient interpreter (Matthias Benkard, 2013)
  - modernising register allocation (Alexandra Barchunova, 2013)
  - Improving Unicode support in SBCL (Krzysztof Drewniak, 2014)
  - Speeding up constant division in SBCL (Erik Krisztián Varga, 2014)
- (also, under LispNYC)
  - SBCL Windows Threading (Elliot Slaughter, 2008)

SBCL birthday parties

2004 (5th birthday): Bay Area Lispniks

> *Northern California party celebrating SBCL's 5th birthday will be at 8pm tonight at Jupiters in Berkeley, CA. I'll be there with my iBook.*

– Thomas Burdick

SBCL birthday parties

2009 (10th birthday): sbcl10, Goldsmiths, London

- talks from Nick Levine and Martin Cracauer

SBCL birthday parties

2009 (10th birthday): sbcl10, Goldsmiths, London

- talks from Nick Levine and Martin Cracauer
- as-yet-unfinished hackday projects include
  - llvm backend
  - calling functions with unboxed arguments
  - multi-word compare-and-swap
  - faster aref on non-simple non-displaced arrays
  - incremental allocation for dynamic space
  - subclassable structures

SBCL birthday parties

2019 (20th birthday): ?

1999-2019

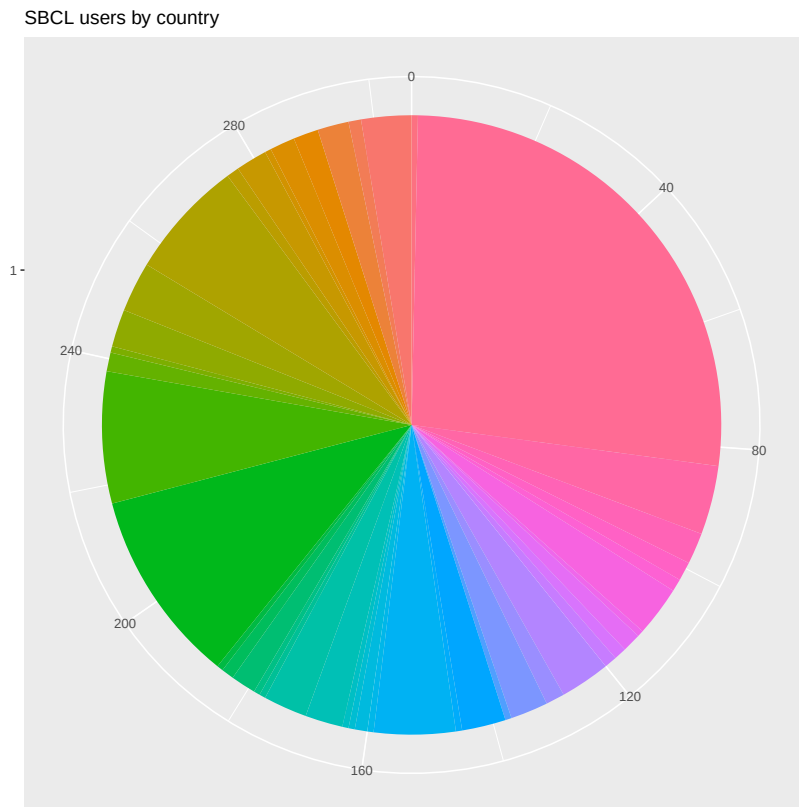
[video]

User survey

- last done in 2010
- similar questions:
  1. Personal information (age, country)
  2. Uses of SBCL and why
  3. SBCL platforms, versions
  4. Communications channels
  5. Development
- over 300 responses

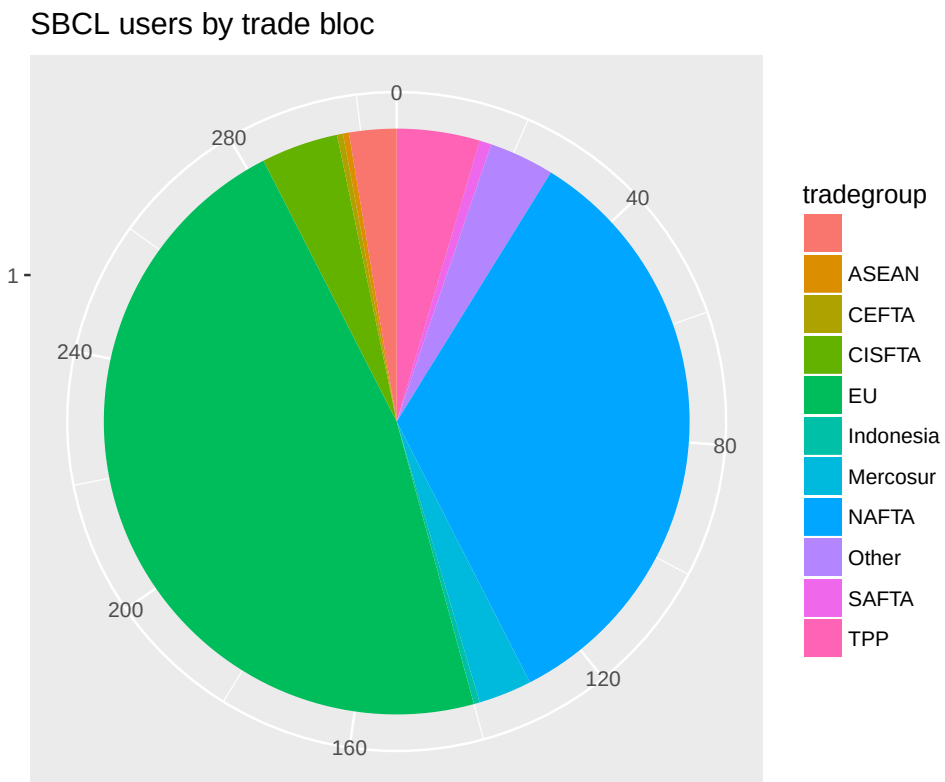
User survey: personal information

Country



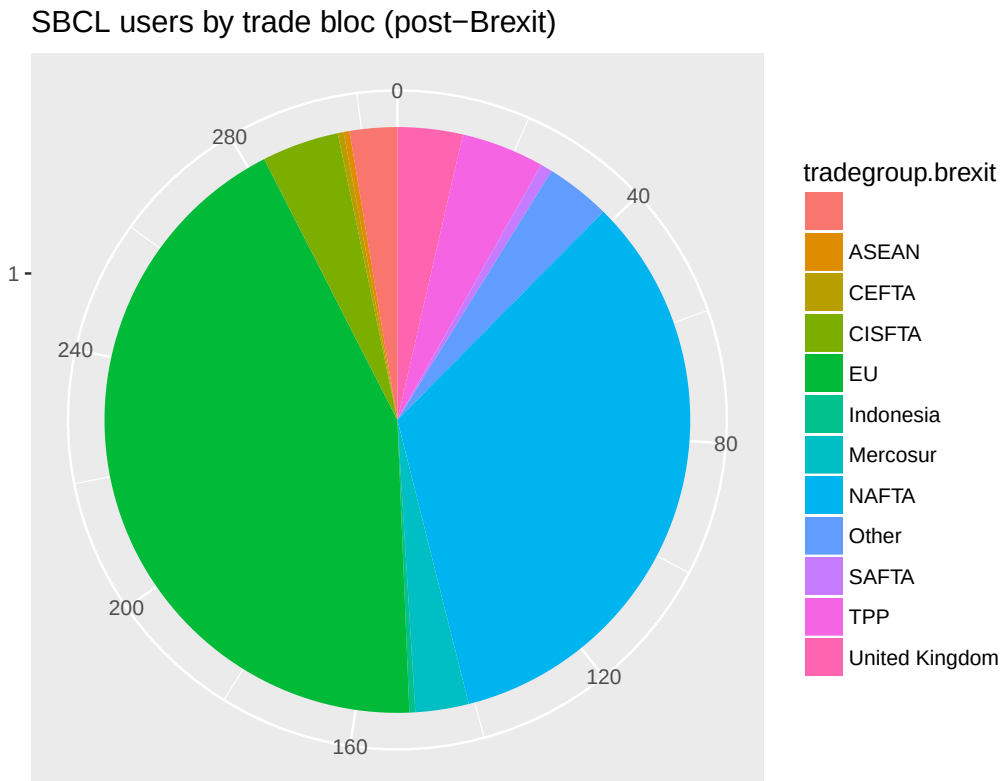
User survey: personal information

Trade bloc



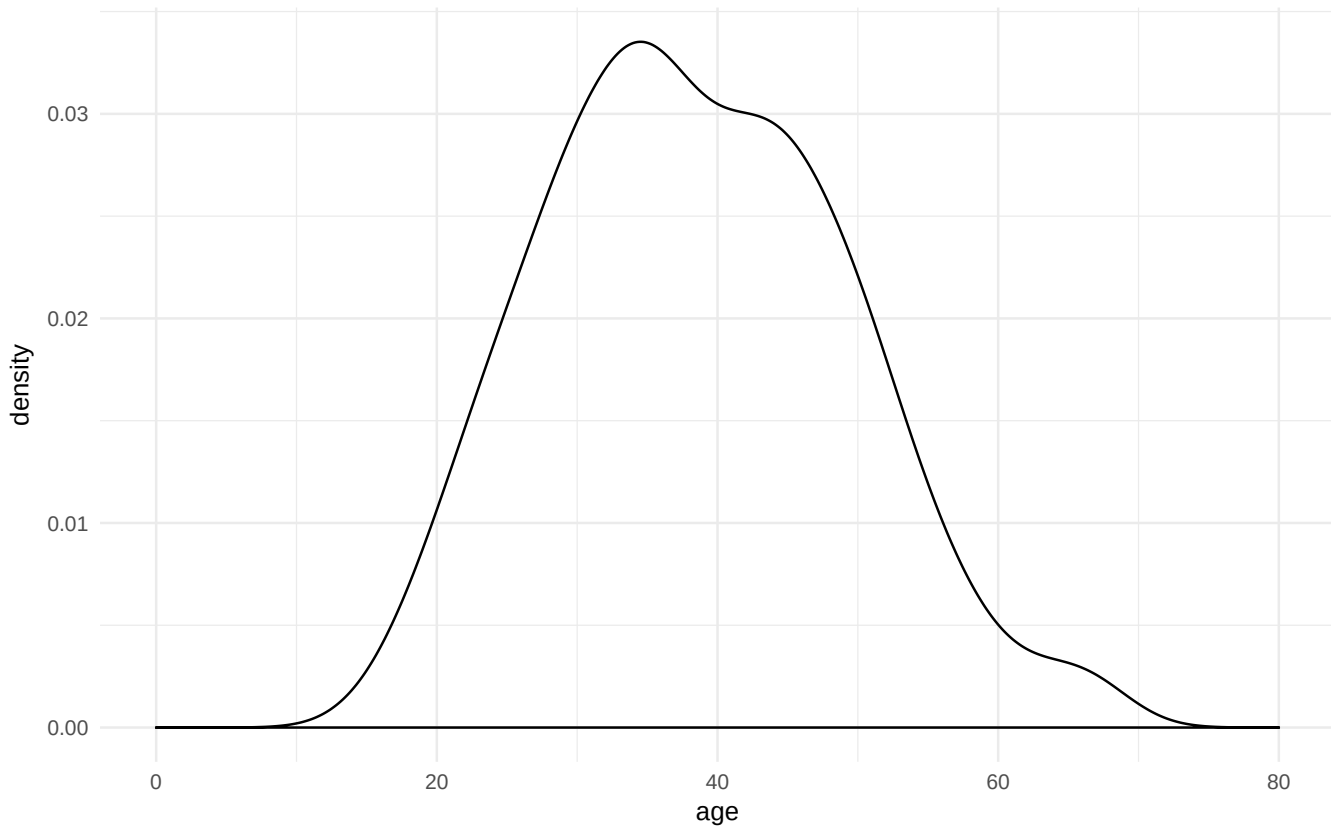
User survey: personal information

Trade bloc (post-Brexit)



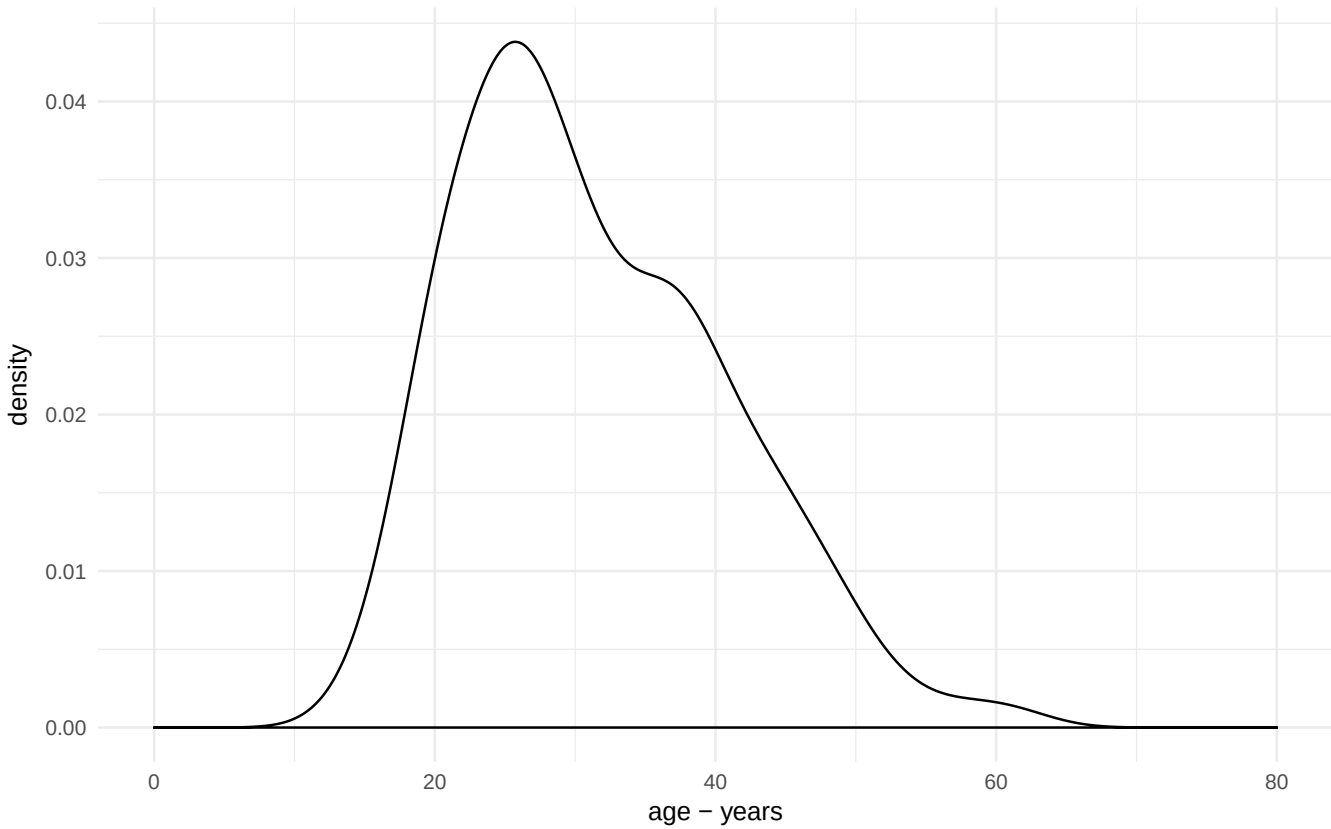
User survey: personal information

Age



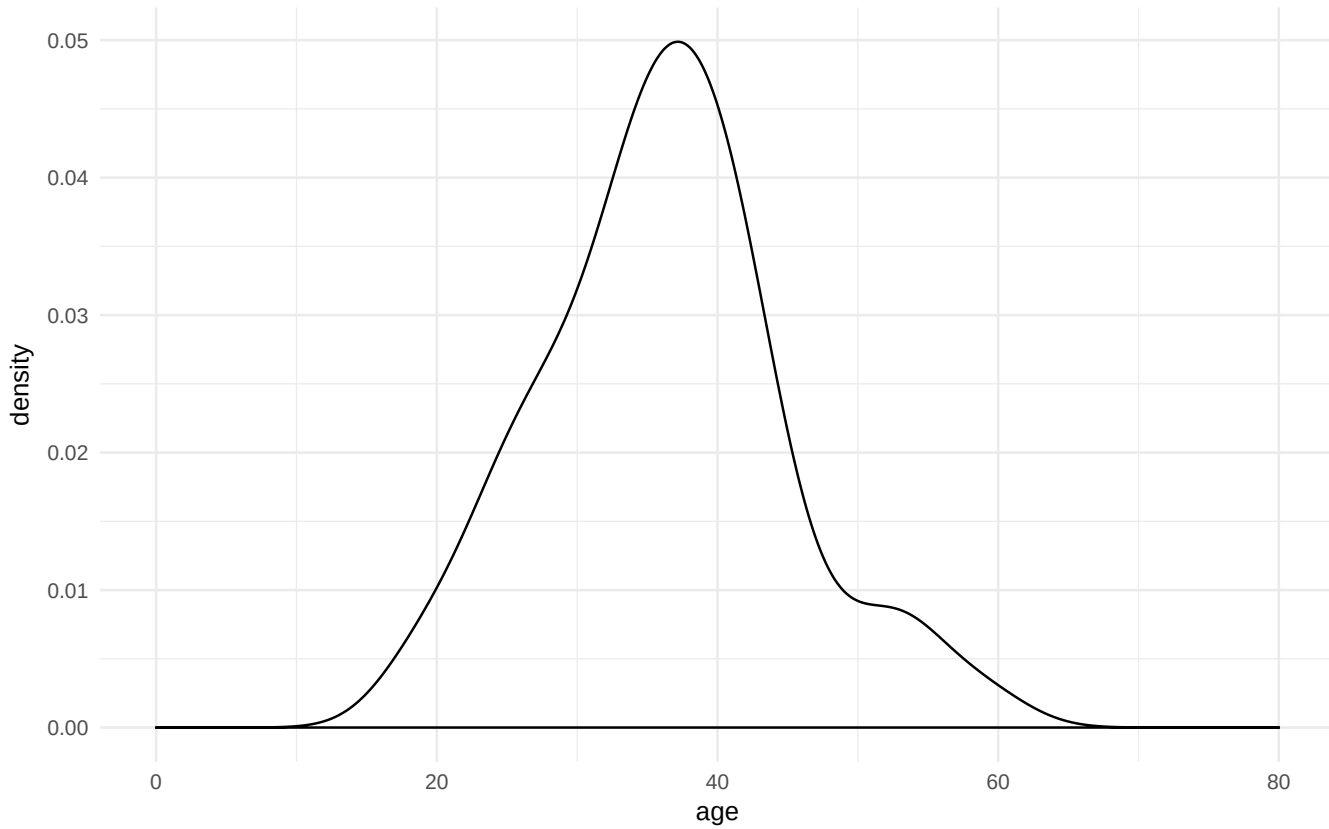
User survey: personal information

Age on starting to use SBCL



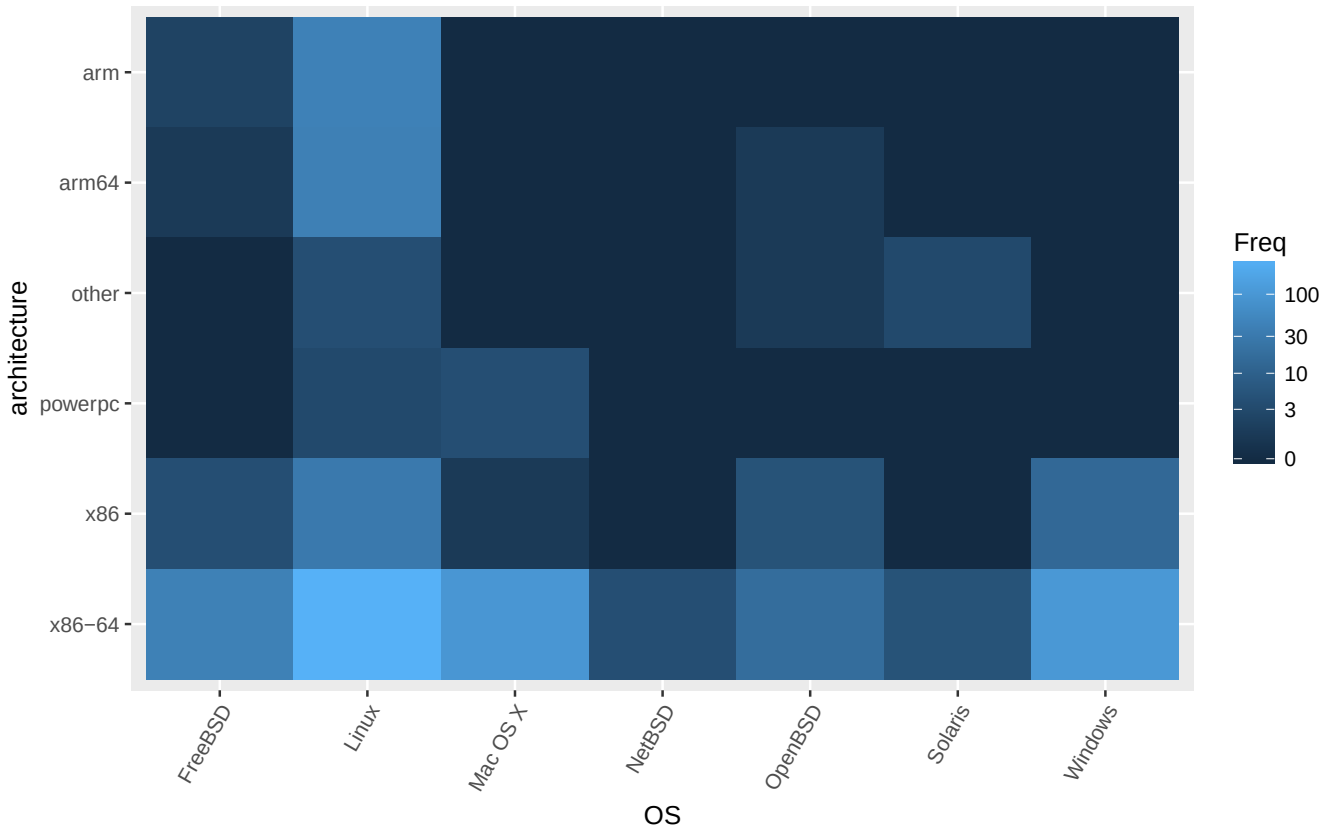
User survey: personal information

Age of IRC users



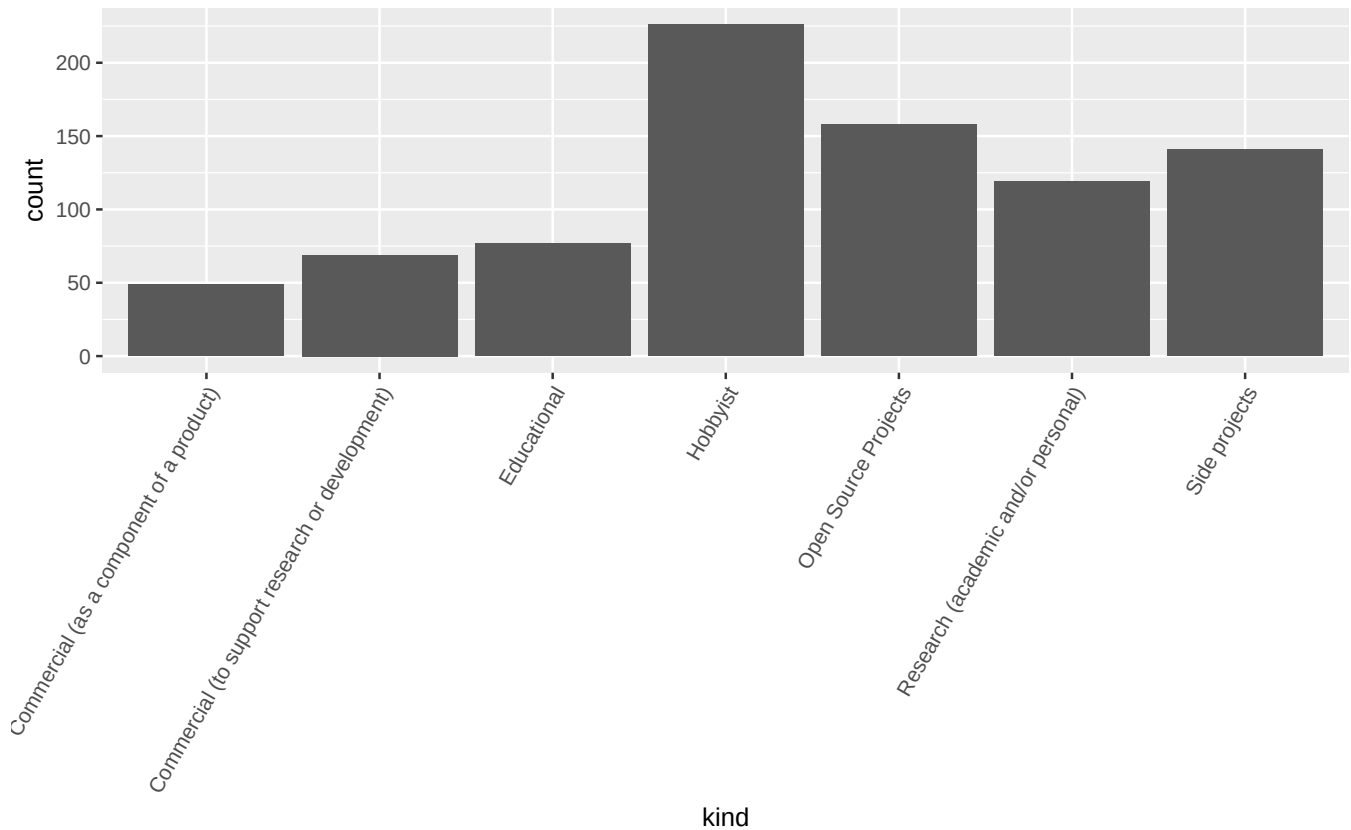
User survey: use of SBCL

Platforms

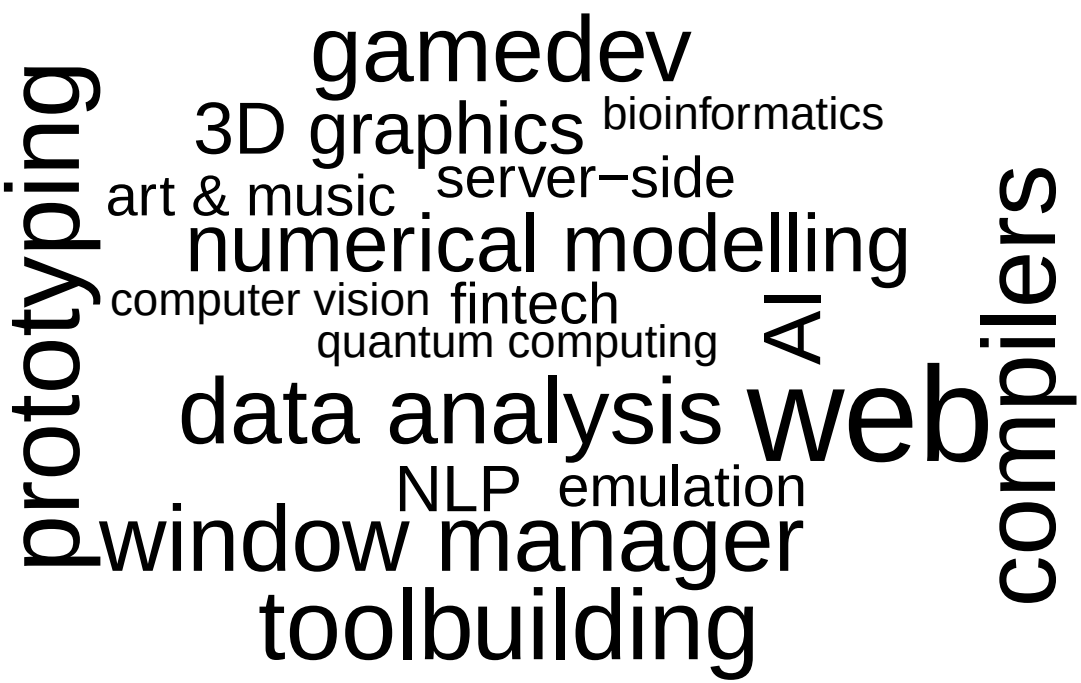


User survey: use of SBCL

Uses

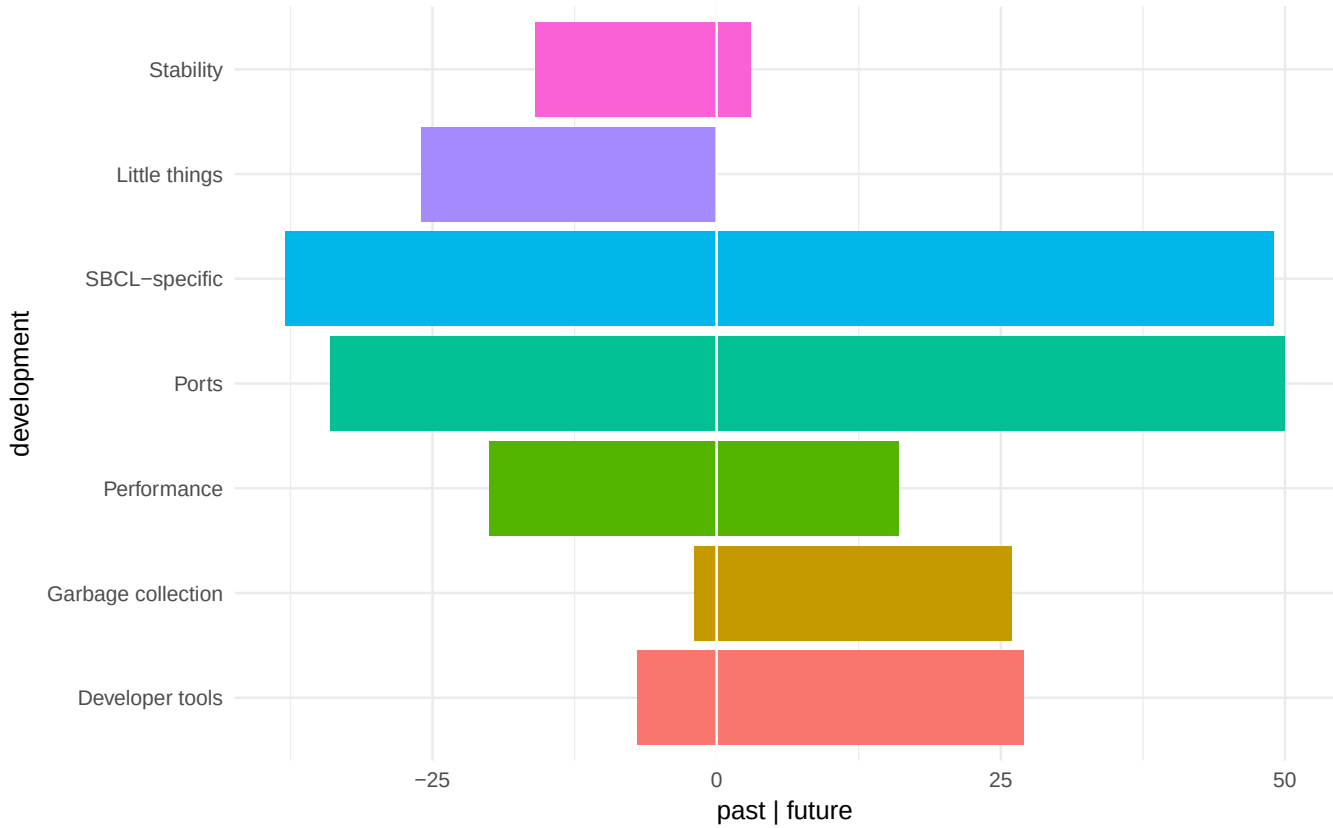


User survey: use of SBCL



User survey: development

Past and future development



Stability, little things

- regular releases
- quick bug fixes
- papercuts

SBCL-specific extensions

- SIMD support
- threading
- embeddability
- tree shaker

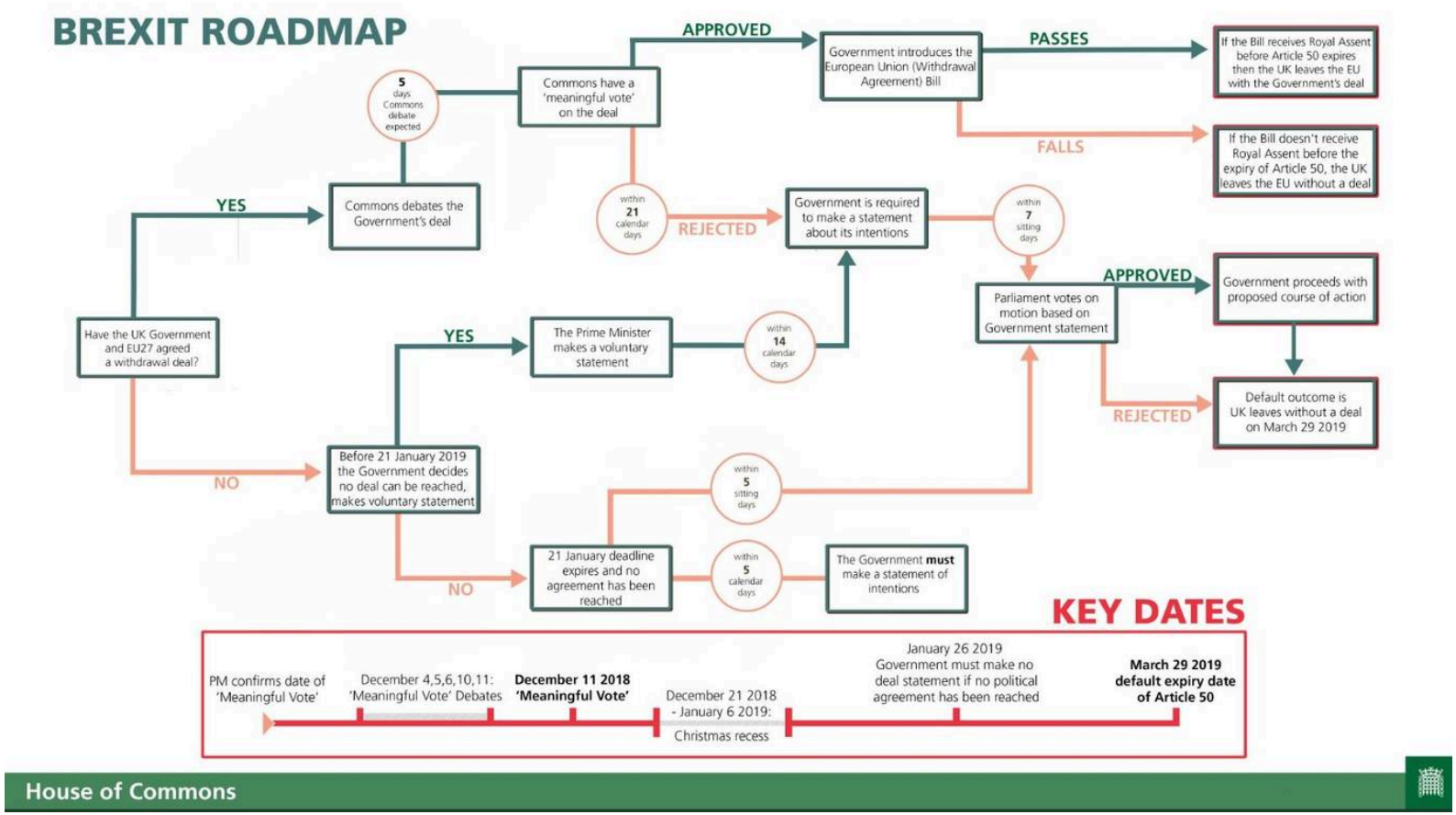
Developer tools

- compile-time diagnostics
- debugger, stepper, profiler
- friendlier repl

Garbage collection

- large, resizeable heaps
- multicore, parallel, incremental
- precise
- NUMA awareness

Ceci n'est pas un roadmap



New ports

- ppc64: partly complete
  - (Google Intern project, Brian Bokser)
- riscv: mostly complete
  - (spontaneous project, karlosz on `#sbcl`)
- s390x: requested

Existing ports

- arm: threads
- arm64: neon
- OS X: improve support for deploying applications
- windows:
  - improve support for deploying applications
  - remove the warning banner

Platform support

- lightweight threads
- shared library

Developer tools

- defadvice
- better backtraces when non-Lisp is involved
- LSI, Jupyter integration
- stepper improvements

Garbage collection

- resizeable heap

Governance

Currently no formal project structure

- low barrier to (development) entry
- high barrier to feeling ownership?
- high bus factor?
  - kmr incommunicado: can't add aliases to sbcl.org
  - admin access to sourceforge (project, mailing lists) vulnerable
- communication comes from individuals
  - lack of roadmap

Testimonials / Case studies

<Your case study goes here>

Onboarding

Growing new SBCL developers

- tangible project: new port

Onboarding

Growing new SBCL developers

- tangible project: new port
  - (oh no, run out of new platforms)

Onboarding

Growing new SBCL developers

- tangible project: new port
  - (oh no, run out of new platforms)
    - still some gaps in arch/OS combinations
    - finish ppc64, start s390x
  - possible to get quite stuck (buggy hardware, buggy kernel, buggy libc, buggy emulators, buggy debuggers)
  - only do this if you work (or plan to work) with that platform anyway

Onboarding

Growing new SBCL developers

- obviously impactful project: improve Windows support
  - combination of skills needed rare
    - SBCL has 25+ years of Unix heritage
    - Windows low-level programming quite different
  - userbase is large
  - much kudos available

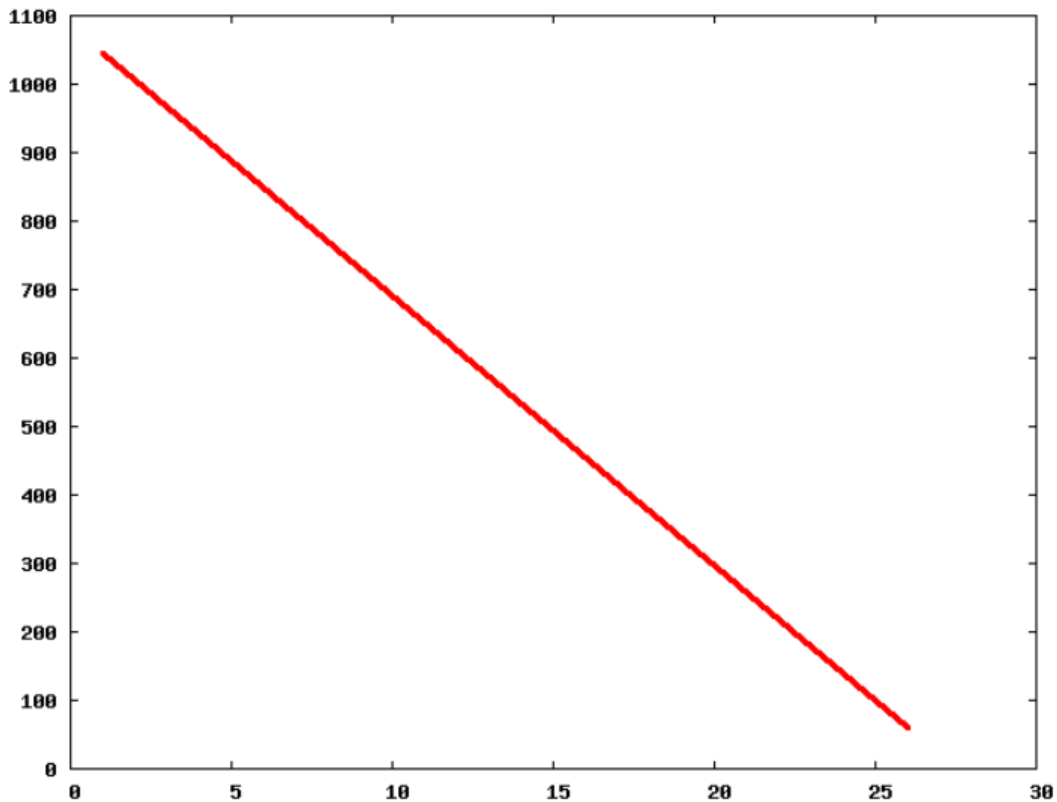
Onboarding

Growing new SBCL developers

- simple, bounded project: find a bug, fix it
  - not all that much low-hanging fruit
  - long-standing bugs might be gnarly
- we (I) need to be better at responding to bug fixes
  - quick merge (retaining patch author metadata)
  - lower-overhead commit access

Onboarding

0.8: fewer bugs



Thank you

Comments or Questions?