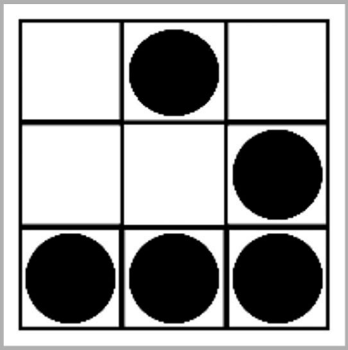


Lisp Hackers

Interviews with 100x More Productive Programmers

conducted by
Vsevolod Dyomkin
in 2012-2013

Lisp Hackers

Interviews with 100x More Productive Programmers

Vsevolod Dyomkin

This book is for sale at http://leanpub.com/lisphackers

This version was published on 2013-06-21

Tweet This Book!

Please help Vsevolod Dyomkin by spreading the word about this book on Twitter!

The suggested hashtag for this book is #lisphackers.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#lisphackers

Contents

A Short Intro

The idea of this series came to my mind after European Common Lisp Meeting 2011[1] where I had had a chance to meet in person some of those mythical 100x more productive programmers. And the points I wanted to make were:

- these guys do exist, and it's really interesting to know how they work and think
- it makes sense to show who are the real people using this obscure and mythical language Common Lisp, why do they use it and what they do with it

This book is a collection of short interviews with 14 prominent individuals from different parts of the world, from Australia to Canada, and of different occupations, from low-level programmers to physicists and musicians, asking them a more-or-less similar set of questions on the following topics:

- their general attitude to programming
- attitude towards and experience with Lisp
- stories of real-world Lisp projects

Overall, this should give an insight into why people use Lisp, as well as help the readers gain some new experience and improve as programmers.

How were these individuals chosen? I looked for people who possess at least 2 of the following 3 essential qualities:

- a renown Lisp open-source contribution
- prominence in the community
- experience working for successful Lisp companies
- … and a good perspective on other languages was considered as a pleasant bonus

Definitely, not all of the hackers I'd like to interview are here. For various reasons. There are lots more interesting Lisp people with fascinating experiences worth sharing, and, hopefully, this will happen in the future in one form or the other. Yet, here are these 14 unique lispers with their stories, world views and insights. Enjoy!

P.S. As a bonus you'll find some interesting bits of history of SBCL and Slime, the two most important open-source Common Lisp tools, told by their main developers.

[1] http://weitz.de/eclm2011/

A Few Quotes

I've been using CL exclusively for the last six or seven years. As I was working freelance, this was kind of easy — I either had projects where the customer didn't care about the programming language that I used as long as I got the job done, or I was hired specifically for my CL skills.

– Edi Weitz

I love hanging out with Lisp hackers: I find that we're an unusually diverse community. How often do you attend a small conference where attendees are building nuclear defense systems, running intensive care wards, designing aeroplane engines, analysing Lute tablature, developing cancer drugs, writing FIFA's legal contracts, and designing their own microchips?

– Luke Gorrie

When what I have to do is solve problems that no language has built-in libraries for, then I want to be working in a language where I can focus on the problem itself, rather than the detail of that language — and for me, Lisp permits that kind of separation of concerns, given its easy support for DSL and embedded language generation, protocols, and the ability to run partial programs.

– Christophe Rhodes

In one of our research topics we had to solve huge instances of 3-SAT problems using a very basic algorithm. The fact that Common Lisp comes with bignums reduced the development time to one afternoon and the program we produced was faster than the C++ prototypes we had.

– Juan José García Ripoll

In the end, Lisp won me over because it turns out that it is the mother of all languages. You can bend it and turn it into whatever language you want, from the most flexible and reflective interpreted scripting language to the most efficient and static compiled production system.

– Pascal Costanza

For typical projects I feel like I'm building more tools into my REPL workbench. I don't write scripts that I call from the command line, I write functions that I call from the REPL, and use the slime and Emacs environment to create and interact with data, whether it's data from a file, from a computation, from the web, a database, etc.

– Zach Beane

It's also very easy to turn Common Lisp REPL code into unit tests, which I tend to do a lot. That is something that's very hard to do with object-oriented code, which is why idiotic things like dependency injection and Test-Driven Development have to be invented.

– Vladimir Sedach

There's a clear difference between the Lisp workflow where you change the state of your image interactively to get the code into working shape very quickly (and then later try to remember what it was you did) and the more scripted approach of test-driven development in Ruby where you put everything (code, test setup, assertions) in files that you reload from disk on each run.

– Daniel Barlow

When I'm trying to understand someone else's code I tend to find the best way is to refactor or even rewrite it. I start by just formatting it to be the way I like. Then I start changing names that seem unclear or poorly chosen. And then I start mucking with the structure.

– Peter Seibel

The greatest thing about Common Lisp is that the standardization process drew from years and years of experience and organic growth. This makes the language feel singularly solid and practical. Most languages feel immature compared to CL.

– Marijn Haverbeke

We don't use Lisp, but much of our software is built on ideas borrowed from Lisp. We don't use it because we needed low level control — most of the code is written in C++, even with some bits of assembly. But we've borrowed an enormous number of ideas from Lisp. In fact, if we weren't Lispers, we would have built a very different (and I think significantly more inferior) product.

– Slava Akhmechet

On the upside, we certainly have been able to write quite advanced software that we might not have otherwise managed. A million lines of Lisp code, including its fair share of macros and DSLs, would be so many more million lines of code without the syntactic abstraction made possible by Lisp. However hard and expensive it was with Lisp, I can only imagine how many times worse it would have been with anything else.

– Faré Rideau

I think we would all be better off if we hadn't balkanised the different systems that we program for — and Lisp is one of the few programming languages with the flexibility to serve in all these roles.

– John Fremlin

The language itself is somewhat good enough and anyway Common Lisp makes it really easy to change most of itself to add the new and cool stuff or ideas of the day.

– Marc Battyani

Zach Beane (USA)

Zach Beane is the uniting link of the whole community. The creator of Quicklisp[2] and a number of useful open-source libraries, like Vecto[3] and ZS3[4], he is also always active on #lisp irc channel answering questions, as well as keeping a blog[5], where he shares interesting Lisp news, and supporting Planet Lisp[6] blog aggregator. And that's not all…

His twitter is @xach[7]

Tell us something interesting about yourself.

> I live in Maine and enjoy hacking Common Lisp projects for fun and profit.

What's your job? Tell us about your company.

> I work at a small telephone and Internet company.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

> I currently use Lisp to analyze some large data files we get from vendors and partners and produce reports. I've recently used Lisp to produce KML files for Google Earth visualization of some internal data. I use stuff like cl-ppcre[8], drakma[9], postmodern[10], cl-mssql[11], cxml[12], and more to gather data from various systems when preparing reports. The growing library ecosystem has made it really easy to get stuff done.
>
> Lisp is accepted because I can produce useful stuff pretty quickly with it. It's a small company with a small team so there isn't a bureaucracy to fight to use something non-standard. Does the tool work? Is the result useful? Then there's no problem.

[2] http://www.quicklisp.org/
[3] http://www.xach.com/lisp/vecto/
[4] http://www.xach.com/lisp/zs3/
[5] http://xach.livejournal.com/
[6] http://planet.lisp.org/
[7] http://twitter.com/xach
[8] http://weitz.de/cl-ppcre/
[9] http://weitz.de/drakma/
[10] http://marijnhaverbeke.nl/postmodern/
[11] http://code.google.com/p/cl-mssql/
[12] http://common-lisp.net/project/cxml/

In some cases, where it isn't a good fit for the final product, I use Lisp to prototype ideas before writing a final thing in some other language, e.g. C or Perl. But I even use CL to generate data and code in other languages, so it's still in the mix, still a part of my workflow.

What brought you to Lisp? What holds you?

Paul Graham's "Beating the Averages[13]" describes a really exciting way to use an uncommon tool to great advantage. That got me interested in how I could do the same thing. I started off with some stuff in Scheme because I thought CL was old and crufty and gross, but when I started using SBCL I found it was a great, practical tool, and I never stopped using CL after that.

I continue to use CL because it has a great mix of features (object system, first-class functions, fast compiled code, multiple implementations, a fantastic standard, etc, etc) and it works well in my favorite working environment, Emacs on Linux. I feel like I know it well and I can see quickly how to use it to explore and solve new problems.

What's the most exciting use of Lisp you had?

Making graphics with Lisp for wigflip.com[14]. It's always fun to have an idea about some kind of visual toy and then tinker with it until it's right.

It's also fun to do something fast. A few years ago I got a project to fill out some PDF forms automatically. The project started with a few days scheduled to research third-party solutions, but in those few days I had a working prototype that used CL-PDF[15] and CLSQL[16].

What you dislike the most about Lisp?

There are a lot of negative perceptions about Common Lisp that are reinforced by current and former Common Lisp users. I can accept that CL is not for everyone, but some of the criticism is just years (sometimes decades) of moaning and nitpicking about decisions made in the distant past that are not really up for review right now. I wish the people who are vocally, chronically dissatisfied with CL would go off and do their own thing and stop bothering people that are happy with Common Lisp.

There are some remarkable trolls that like to pick on CL, but they're not usually taken seriously by anyone, but "insiders" who complain about CL are perceived to be giving some sort of genuine insight. It's very annoying.

[13]http://paulgraham.com/avg.html
[14]http://wigflip.com/
[15]http://www.fractalconcept.com/asp/cl-pdf
[16]http://clsql.b9.com/

Among software projects you've participated in what's your favorite?

Quicklisp has been very rewarding. There's a lot of positive feedback from people who feel it really helped them use CL more easily. More generally, the whole CL ecosystem has been a lot of fun. I enjoy trying new libraries, sending feedback and bug reports, helping people get started, and all that stuff. Common Lisp has a lot of smart, helpful, friendly people who share a lot of knowledge, and I feel lucky to get to learn from them and to try to share what I know, too.

Describe your workflow, give some productivity tips to fellow programmers.

For typical projects I feel like I'm building more tools into my REPL workbench. I don't write scripts that I call from the command line, I write functions that I call from the REPL, and use the slime and Emacs environment to create and interact with data, whether it's data from a file, from a computation, from the web, a database, etc.

I find it really helps to have small, focused functions that are easy to follow and that do one thing and do it well. The fine granularity of testing, tracing, intermediate values, etc. can help wire together a very useful set of behaviors from a small core of simple functions.

Some productivity ideas…

Knock the rough edges off your working environment. Write code to automate stuff. Make it easy to repetitively do complicated but boring stuff. For example, I used to be afraid of the hassle of making new releases of my projects, but recently wrote a CL program that does everything for me, from PGP-signing the tarballs to uploading them along with the documentation to my website. Now I don't care if I make ten project releases in a day, it's just a few function calls.

Customize your environment to make it comfortable. Make it easy to look up info in the hyperspec or in other documentation sources. Make it easy to create new projects. I use quickproject[17] a lot for that, but I also have some Emacs templates that put some boilerplate into my files automatically. Make a program do your work for you.

How did quicklisp change your life? What are current plans for its development?

I can get up and running in a new environment very quickly now. Before Quicklisp, I could usually build up a set of libraries that were comfortable and useful, but it could

[17] http://www.xach.com/lisp/quickproject/

be a hassle to move them from computer to computer, or to make sure they were up-to-date. Now I just use Quicklisp and don't worry about it.

As Quicklisp maintainer, it's really helped me see where some people are coming from when they want to try Common Lisp. It's a totally different mindset than what I'm used to. I think there's room for some documentation or tutorial on the system-oriented, REPL-workbench style of Common Lisp development that I like to use.

I want to write more documentation for Quicklisp, particularly how users can publish their own software for it or customize it via its internal protocols.

I also want to gather and share documentation for all Quicklisp-loadable projects, so there can be a single place to learn what a given project does, and get access to all its documentation.

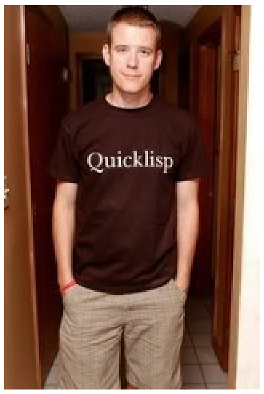
I'd also like to make it easy for people to share feedback about their experiences with a project, so you could decide if it's likely to fit your needs. Something like the feedback you see for products on Amazon, but for Lisp libraries.

If you had all the time in the world for a Lisp project, what would it be?

As a fantasy project, I'd love to make a system for interesting visualization of complex data, something where it's easy to splat something quick and dirty on the screen/page, but which can grow in capability as the need arises. Or maybe just some tool for making audiovisual toys, with cool pictures and noises coming out.

Anything else I forgot to ask?

Common Lisp is a great system that rewards in-depth study.



xach

2012-03-10

Edi Weitz (Germany)

Edi Weitz doesn't need to be introduced in the Lisp community. His vast contributions to open source Lisp, made during the last decade, collectively known as Ediware, include the indispensable regex library CL-PPCRE[18], written on a bet in Hamburg café, and a whole stack of web-related libraries with the most widely used Lisp application server Hunchentoot[19] and HTTP client Drakma[20]. Together with Arthur Lemmens[21] he co-organizes European Common Lisp Meeting[22].

Tell us something interesting about yourself.

> Well, I'll leave it to someone else to tell you what's interesting about me. I'll rather tell you what I find interesting in addition to Common Lisp: I collect photo books and I'm doing a bit of photography myself[23]. I like to listen to the music of Frank Zappa and to Jazz. I read a lot. I'm interested in mathematics, especially in set theory[24].

What's your job? Tell us about your company.

> I'm a professor for mathematics and computer science at the University of Applied Sciences[25] in Hamburg. I started this job in September 2011.
>
> Before that, I was a freelance hacker for about 13 years.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

> In my new job, I've been using CL in my math lectures a couple of times and will continue to do so.
>
> In my old job, I've been using CL exclusively for the last six or seven years. As I was working freelance, this was kind of easy — I either had projects where the customer didn't care about the programming language that I used as long as I got the job done, or I was hired specifically for my CL skills.

[18]http://weitz.de/cl-ppcre/
[19]http://weitz.de/hunchentoot/
[20]http://weitz.de/drakma/
[21]https://twitter.com/alemmens
[22]http://weitz.de/eclm2013/
[23]http://www.flickr.com/photos/hunchentoot/
[24]http://www.springer.com/birkhauser/mathematics/book/978-3-7643-6124-2
[25]http://www.haw-hamburg.de/

What brought you to Lisp? What holds you?

I came to Lisp via Emacs Lisp in 1999 or so. What got me hooked was the wonderful book "Writing GNU Emacs Extensions[26]" by Bob Glickstein. It opened my mind for the beauty of the Lisp language family — something I had missed the first time I had encountered Lisp (in university, a few years earlier). The two CL books by Paul Graham and Norvig's PAIP[27] then paved the way for Common Lisp.

What holds me is that I haven't found a better programming language so far — and I don't expect to find one very soon.

What's the most exciting use of Lisp you had?

I don't know if "exciting" is the right word, but it makes me happy that so many people use "The Regex Coach[28]" and like it. I stopped keeping track, but there must have been at least half a million downloads since 2003.

I'm also kind of proud that some of my open source libraries are used by various commercial and research projects around the world.

But probably the most awe-inspiring encounters I had with Lisp were the few occasions when I played around with Genera or watched someone else using it. I think this OS really was a work of art.

What you dislike the most about Lisp?

There's nothing I really dislike about Common Lisp. There are a few warts here and there, but so far I've found nothing that was serious enough to prevent me from being productive.

Among software projects you've participated in what's your favorite?

Working on the Windows port of Piano[29] — an extremely impressive application which has been around for almost 20 years and has been used by almost every aircraft manufacturer in the world. Dimitri Simos, Piano's main author, has been the most enjoyable client I've worked with so far.

[26] http://www.emacs.uniyar.ac.ru/doc/O'Reilly_Emacs/Writing%20GNU%20Emacs%20Extensions.PDF
[27] http://norvig.com/paip.html
[28] http://weitz.de/regex-coach/
[29] http://www.piano.aero/

Describe your workflow, give some productivity tips to fellow programmers.

I usually just start up the LispWorks IDE and hack away. The best productivity tip I can give is to stick with one implementation and IDE and to invest a lot of time to really learn how to use it — including all the implementation-specific goodies like debuggers, inspectors, steppers, browsers, and so on.

Ediware became hugely popular (by Lisp standards), and with this popularity came a lot of work and responsibility. You seem to have mostly handed over supporting it to Hans Hübner. What's up next for you in the land of programming and Lisp in particular?

I'm planning to give a lecture about the use of AI techniques in games in the next year and I might use some Lisp there. I might also — as a sideline — resume my CL consulting work sooner or later. I don't expect to publish new open source code in the near future, though.

If you had all the time in the world for a Lisp project, what would it be?

When I was still working as a hacker, I always dreamt of finding someone to pay me for working on an open-source CLOS object store — written in pure Common Lisp, OS-independent, portable, not relying on third-party software, fast, reliable, thread-safe, well-documented, etc.

edi

2012-03-14

Slava Akhmechet (USA)

Slava Akhmechet published several enlightening essays at his website defmacro.org[30], of which one I often recommend to people, interested in learning about Lisp: The Nature of Lisp[31]. He also created a continuation-based Lisp web-framework — Weblocks[32], backed by a delimited continuations library cl-cont[33]. Other then that he is a co-founder of a startup company RethinkDB, of which he tells a bit in the interview.

Tell us something interesting about yourself.

> For a long time I thought that human achievement is all about science and technology. In the past few years I realized how misled I was. Hamlet is as important an achievement as discovering penicillin. I wish I'd figured out earlier that science, for all its usefulness, is very limiting if one adopts it as an article of faith.

What's your job? Tell us about your company.

> I'm a founder at RethinkDB[34]. We spent three years building a distributed database system that we're about to open source and release in the next two weeks. The system allows people to easily create clusters of machines, partition data in a click of a button, and run advanced, massively parallelized, distributed queries using a very comfortable query language we've designed. The product is really delightful to use — we were just playing with it today to analyze census data for the upcoming presidential election in the U.S. and using it to play with the data is a real joy. I'm very proud of what we've done here — I hope it will make lots of people's jobs easier and let them do things they couldn't have done before.
>
> My job here is to do the most important thing at any given time. Sometimes it means fixing bugs, sometimes it means demoing the product to customers, and sometimes it means driving to buy supplies so our developers can get their jobs done.

[30]http://defmacro.org

[31]http://www.defmacro.org/ramblings/lisp.html

[32]http://common-lisp.net/project/cl-weblocks/

[33]http://common-lisp.net/project/cl-cont/

[34]http://www.rethinkdb.com/

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

We don't use Lisp, but much of our software is built on ideas borrowed from Lisp. We don't use it because we needed low level control — most of the code is written in C++, even with some bits of assembly. But we've borrowed an enormous number of ideas from Lisp. In fact, if we weren't Lispers, we would have built a very different (and I think significantly more inferior) product.

What brought you to Lisp? What holds you?

A guy named bishop_pass on gamedev.net forums about fifteen years ago. He was a really good advocate and I respected his opinions because of other subjects, so I decided to check Lisp out. I enjoyed it immensely, and spent years hacking in it. Today the only Lisp I still use is Emacs Lisp. I honestly don't know if I'll program in Lisp again (other than for fun, of course), but the ideas behind it will be with me forever.

What's the most exciting use of Lisp you had?

I built cl-cont — a macro that converts Lisp code to continuation passing style. I honestly think I learned more about programming from that experience than from anything else I've done before or after.

What you dislike the most about Lisp?

Probably the arrogance of the community that surrounds it. Knowing Lisp certainly doesn't make one a better person, nor even necessarily a better programmer.

Among the software projects you've participated in what's your favorite?

Definitely RethinkDB. We took a really complex subject (real-time distributed systems) and made them extremely accessible and super-easy to use. I love the product both because we made the user experience a joy, and because of the really advanced technology that goes inside to make that happen (from low-level assembly hacks, all the way up to abstract mathematics).

If you had all the time in the world for a Lisp project, what would it be?

I'd want to build my own Lisp dialect. I know, I know, it's been done to death, there is no need to do it, and it only hurts the community, but in the presence of infinite time, it's just too much fun not to do.

Describe your workflow, give some productivity tips to fellow programmers.

The most important thing I learned on productivity is this Alan Kay quote:

> Perspective is worth 80 IQ points.

You could be the most productive person in the world, but it won't make the slightest bit of difference if you're pointing your talents in a direction that isn't useful to other people. If you're talented, your gift is precious and your time is limited. Learn how to direct your talents, it will be the most important thing you do.

You're currently a co-founder of a startup company RethinkDB, which went through YCombinator. As an insider of the startup ecosystem, in your opinion, what are the areas for Lisp use in startups nowadays with the biggest potential upside and why?

This isn't a popular stance in the Lisp community, but I think that today Lisp is mostly valuable as an education tool, as a means of thinking, and as an engine of ideas. It's very important for that. But as far as practical use goes, there are better options today.

Slava

2012-10-25

Pascal Costanza (Germany - Belgium)

Pascal Costanza is a researcher, and an active Common Lisp programmer and community enthusiast:

- he's the maintainer of Closer to MOP library[35], that provides a common facade to the MOP implementation in different Lisps, and is the basis of some of his more advanced libraries like: ContextL[36] and Filtered Functions[37];
- the originator of Common Lisp Document Repository[38] (CDR) project, that collects proposals for improving the language (a la JCP for Java or PEP for Python);
- and the author of a Highly Opinionated Guide to Lisp[39], which can serve as introductory text for those who come to Lisp from other languages. (It was quite a useful text for me, when I started studying Lisp.)

In the interview Pascal shares a lot if insight into his main topic of interest — programming language design — grounded in his experience with Lisp, Java, C++ and other languages.

His twitter is @infoxpascal[40]

Tell us something interesting about yourself.

> I share a birthday with Sylvester Stallone and George W. Bush. I have been a DJ for goth and industrial music in the Bonn/Cologne area in Germany in the past. I once played a gay Roman emperor in comedic theatre play. I played a few live shows with a band called "Donner über Bonn" ("Thunder over Bonn"). My first rock concert I ever attended was Propaganda in Cologne in 1985. The first programming language I really liked was Oberon. I often try to hide pop culture references in my scientific work, and I wonder if anybody ever notices. My first 7" single was "Major Tom" by Peter Schilling, my first 12" single was "IOU" by Freeez, my first vinyl album was "Die Mensch-Maschine" by Kraftwerk, and my first CD album was "Slave to the Rhythm" by Grace Jones. I don't remember what my first CD single was.

[35]http://common-lisp.net/project/closer/closer-mop.html
[36]http://common-lisp.net/project/closer/contextl.html
[37]http://www.cliki.net/Filtered%20Functions
[38]http://cdr.eurolisp.org/
[39]http://www.p-cos.net/lisp/guide.html
[40]https://twitter.com/infoxpascal

What's your job? Tell us about your company.

I currently work for Intel, a company whose primary focus is on producing CPUs, but that also does business in a lot of other hardware and software areas. (Unfortunately, Intel's legal department requires me to mention that the views expressed in this interview are my own, and not those of my employer.)

I work in a project that focuses on exascale computing, that is, high-performance computers with millions of cores that will be on the market by the end of the decade, if everything goes well. I am particularly involved in developing a scheduler for parallel programs that can survive hardware failures, which due to the enormous scale of such machines cannot be solved by hardware alone anymore, but also need to be dealt with at the software level. The scheduler is based on Charlotte Herzeel[41]'s PhD thesis, and you can find more information about it in a paper about her work[42] and at http://www.exascience.com/cobra/.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

At Intel, I do all software prototyping in Lisp. The scheduler I mentioned above is completely developed and tested in Lisp, before we port it to C++, so that other people in the same project and outside can use it as well. It didn't require a major effort to convince anybody to do this in Lisp. It is actually quite common in the high-performance computing world that solutions are first prototyped in a more dynamic and flexible language, before they are ported to what is considered a "production" language. Other languages that are used in our project are, for example, MATLAB, Python and Lua. (Convincing people to use Lisp beyond prototyping would probably be much harder, though.)

The implementation we use for prototyping is LispWorks, which is really excellent. It provides a really complete, well-designed and efficient API for parallel programming, which turns LispWorks into one of the best systems for parallel programming of any language, not just in the Lisp world. The only other system that is more complete that I am aware of is Intel's Threading Building Blocks for C++.

What brought you to Lisp? What holds you?

I have participated in one of the first Feyerabend workshops, organized by Richard Gabriel[43], one of the main drivers behind the original Common Lisp effort. I have also

[41]http://soft.vub.ac.be/soft/members/charlotteherzeel
[42]http://dl.acm.org/citation.cfm?doid=1869459.1869491
[43]http://www.dreamsongs.com/

read his book Patterns of Software[44] around that time. Later we had a small discussion in the patterns discussion mailing list. He tried to promote Lisp as a language that has the "quality without a name", and I made some cursory remarks about Lisp's unnecessarily complicated syntax, just like anybody else who doesn't get it yet.

To me, the most important comment he made in that discussion was:

> True, only the creatively intelligent can prosper in the Lisp world.

The arrogance I perceived in that comment annoyed me so much that it made me want to learn Lisp seriously, just to prove him wrong and show him that Lisp is not as great as he thought it is. As they say, the rest is history.

I actually dabbled a little bit in Lisp much earlier, trying out a dialect called XLisp on an Atari XL computer at the end of the 80's. Unfortunately, it took too long to start up XLisp, and there was not enough RAM left to do anything interesting beyond toy examples, plus I was probably not smart enough yet to really get it. I was just generally curious about programming languages. For example, I also remember trying out some Prolog dialect on my Atari XL.

In the end, Lisp won me over because it turns out that it is the mother of all languages. You can bend it and turn it into whatever language you want, from the most flexible and reflective interpreted scripting language to the most efficient and static compiled production system. For example, the scheduler mentioned above easily gets in the range of equivalent C/C++-based schedulers (like Cilk+, TBB, or OpenMP, for example), typically only a factor of 1.5 away for typical benchmarks, sometimes even better. On the other hand, ContextL uses the reflective features of the CLOS Metaobject Protocol to bend the generic function dispatch in really extreme ways. I am not aware of any other programming language that covers such a broad spectrum of potential uses.

What's the most exciting use of Lisp you had?

When I decided to make a serious attempt at learning Common Lisp, I was looking for a project that would be large enough to prove to myself that it is actually possible to use it for serious projects, but that would also be manageable in a reasonable amount of time. At that time, I was intimately familiar with the Java Virtual Machine architecture, because I had developed compilers for Java language extensions as part of my Diploma and PhD theses. So I decided to implement a Java Virtual Machine in Common Lisp - under normal circumstances, I wouldn't have dared to do this, because this is quite a complex undertaking, but I had read in several places that Lisp would be suitable for projects that you would normally not dare to do otherwise, so I thought I would give it a try. Over the course of 8 weeks, with something like 2 hours per day, or so (because I was still doing other stuff during the day), I was able to get a first prototype that would

[44]http://dreamsongs.net/Files/PatternsOfSoftware.pdf

execute a simple `"Hello, World!"` program. On top of that, it was a portable (!) just-in-time compiler: It loaded the bytecode from a classfile, translated it into s-expressions that resemble the bytecodes, and then just called Common Lisp's compile function to compile those s-expressions, relying on macro and function definitions for realizing these "bytecodes as s-expressions." I was really impressed that this was all so easy to do.

The real moment of revelation was this: to make sure to reuse as many of the built-in Common Lisp features as possible, I actually translated Java classes into CLOS classes, and Java methods into CLOS methods. Java's super calls posed a problem, because it was not straightforward to handle super calls with plain `call-next-method` calls. Then I discovered user-defined method combinations, which seemed like the right way to solve this issue, but I was still stuck for a while. Until I discovered that moving a backquote and a corresponding unquote around actually finally fixed everything. That was a true Eureka moment: In every other programming language that I am aware of, all the problems I encountered until that stage would have required a dozen redesigns, and several attempts to start the implementation completely from scratch, until I would have found the right way to get everything in the right places. But Common Lisp is so flexible that at every stage in your development, you can tweak and twist things left and right, but in the end you still get a convincing, clean, and efficient design. As far as I can tell, this is not possible in any other language (not even Scheme).

What you dislike the most about Lisp?

There is not much to dislike about Lisp itself. There are some technical details here and there, some minor inconsistencies, but nothing that cannot be fixed in easy and straightforward ways. From a purely conceptual point of view, Common Lisp is one, if not the most complete and best integrated programming language that covers a lot of ground. Some rough edges are just to be expected, because nothing is ever perfect.

What concerns me a lot more is that there is too much unwarranted arrogance in the Lisp community. I don't know exactly where this comes from, but some Lispers seem to believe, just because they understand some Lisp concepts, that they are much smarter than anybody else in the universe. What they are forgetting is that computer science as a discipline is very, very young. In one or two hundred years from now, Lisp concepts will be common knowledge, just like elementary algebra. There is no reason to be arrogant just because you know that the earth is round, even if most other people still believe that it is flat.

Describe your workflow, give some productivity tips to fellow programmers.

I strongly believe that the one thing that made me most productive as a programmer is my interest in doing some form of art. I used to spend a lot of time making my own music, both by myself with synthesizers and computers, as well as in bands. I also was an actor in an amateur theater group. Art gives you a sense of making parts (notes, chords, melodies, rhythms, or acts, characters, plot lines) relate to each other and form a coherent whole. It also makes you aware that there is an inner view on a piece of music or a play, as seen by the artist, but also an outer view, as seen or heard by an audience, and if you want to make good art, you need to be able to build a bridge between those two parts.

Programming is exactly the same: you need to make parts (functions, data structures, algorithms) relate to each other, and you need to bridge the inner view (as seen by the designer and implementer) and the outer view (as seen by the user of a library or the end user of the final software).

The important aspect here is that you need to be able to change perspectives a lot, and shift between the local, detailed view, the global, architectural view, and the many different levels of a layered design. This is especially important when it comes to designs that incorporate meta-programming techniques and reflective approaches, but also already for simpler designs.

Like in art, the concrete workflow and the concrete tools that work best vary a lot for different people. It's also a good idea to just play around with ideas, expecting that most of them will turn out useless and need to be thrown away. Artists do this all the time. Artificial, seemingly nonsensical rules and restrictions can be especially enlightening (use only effect-free functions; use only functions that receive exactly one argument, not more, not less; make every function pass around an additional environment; use only classes with exactly two slots; etc., etc.), and then try to build larger programs strictly following such rules - this will make your mind a lot more flexible and train you to see new potential solutions that you wouldn't see otherwise.

(I actually believe that this is what makes fans of static typing so excited: Static type systems always impose some artificial restrictions on your programs, and enforce them to the extent that programs that violate these rules are rejected. If you then program in such a statically typed programming language, you will indeed have some interesting insights and see new solutions that you would otherwise miss. However, the category error that fans of static typing often seem to make is that they ascribe the results to the static type system, and therefore usually get stuck with one particular set or kinds of rules.)

Apart from that, I believe that LispWorks is a really good development environment.

Among software projects you've participated in what's your favorite?

I don't know how to answer that. At any point in time, I'm always most excited by the one I'm currently working on. So far, I am quite proud of ContextL the ClassFilters project for Java, because they are or were both used in one way or the other in "real" applications. Closer to MOP is a favorite project of mine, because it makes me feel like I can give something back to the Lisp community from which I otherwise benefit so much. But this doesn't mean I dislike anything else I have done in the past.

One of your papers, "Reflection for the Masses", researches the ideas behind 3-Lisp, "a procedurally reflective dialect of LISP which uses an infinite tower of interpreters". Can you summarize them here?

That paper was actually mostly the work of Charlotte Herzeel, and I was only her sounding board for detailing the ideas in that paper. Reflection is one of the essential concepts that was born out of Lisp. Every program is about something, for example a financial application is about bank accounts, money and interest rates, and a shopping application is about shopping items, shopping carts and payment methods. A program can also be about programs, which turns it into a meta-program. Lisp macros are meta-programs, because they transform pieces of code in the form of s-expressions into other pieces of code. C++ templates are also meta-programs in the same sense. Some meta-programs are about "themselves," and thus become reflective programs.

3-Lisp is "procedurally reflective" in two senses: On the one hand, it allows you to inspect and change the body of procedures (functions). Common Lisp, for example, also gives you that, in the form of function-lambda-expression and compile/eval, among others. On the other hand, 3-Lisp also allows you to inspect and alter the control flow. Scheme, for example, gives you that in the form of `call/cc`, but in 3-Lisp this is actually part of the `eval` interface. 3-Lisp goes further than Common Lisp and Scheme combined, in that it provides not only first-class access to function bodies and continuations, but also to lexical environments, always both with facilities to inspect and modify them, and provides a clean and integrated interface to all of these features. Unfortunately, because 3-Lisp goes that far, it cannot be fully compiled but always needs to be able to resort to interpretation, if necessary.

The reason for the requirement to always have an interpreter around is because the `eval` interface in 3-Lisp is so flexible that you can run programs inside an `eval` invocation that in turn can inspect and change (!) the environment in which `eval` runs, and can then invoke further `eval`s in those changed environments, to arbitrary levels of recursive invocations of `eval`. These recursive invocations of `eval` build what is called a reflective tower, where every level in the tower is conceptually an interpreter being executed by an interpreter one level up in the tower of interpreters. The amazing thing about 3-Lisp is that an implementation of 3-Lisp can actually collapse the tower into one level of interpretation, and arrange that one interpreter in such a way that several different levels of interpretations are only simulated, so the tower is actually just an "illusion" created by the 3-Lisp implementation.

This may all sound quite esoteric, but is actually practically relevant, because there is strong evidence that all meta-programming approaches eventually need such towers, and some ad-hoc way to collapse the towers. For example, you can find the tower in Racket's and R6RS's macro systems, where they are explicitly mentioned; in Common Lisp's macros, where `eval-when` is used to control which part of the tower sees which definitions; in the CLOS Metaobject Protocol, where the generic function dispatch can be influenced by other generic functions, which can in turn be modified by some meta-classes at a higher level; in the template metaprogramming system of C++, where "concepts" were devised for C++11 (and rejected) to introduce a type system for the template interpreter; and so on, and so on. If you understand the concept of a reflective tower better, you can also better understand what is behind these other meta-programming approaches, and how some of their sometimes confusing semantic difficulties can be resolved.

If you had all the time in the world for a Lisp project, what would it be?

I have some ideas how to design reflection differently for a Lisp dialect, which I believe has not been tried before. If I had all the time in the world, I would try to do that. I also have many other ideas, so I'm not sure if I would be able to stick to a single one.

Anything else I forgot to ask?

I think one of the most underrated and most underused Lisp dialects is ISLISP[45]. I think people should take a much closer look at it, and should consider to use it more often. Especially, it would be an excellent basis for a good teaching language.

Pascal

2012-04-16

[45]http://www.islisp.info/

Peter Seibel (USA)

Peter Seibel has helped more people (including me) discover and become user of Lisp as probably no one else in the last decade with his Practical Common Lisp[46]. Dan Weinreb[47], one of the founders of Symbolics and later Chief Architect at ITA Software, a succesfull Lisp startup sold to Google for around $1B in 2011, wrote, that their method of building the Lisp team was by hiring good developers and giving them PCL for two weeks after which they could successfully integrate under the mentorship or their senior Lisp people.

A few years after PCL Peter went on to write another fantastic programming book Coders at Work[48] — here's my summary[49] of it with the social network of Coders :)

Aside from being a writer he was and remains a polyglot programmer, interested in various aspects of our trade, about which he blogs[50] occasionally. His code, presented in PCL, laid the foundation for a wide-spread CL-FAD library[51], which deals with filenames and directories (as the name implies), and more recently he created a Lisp documentation browser Manifest[52]. Before Lisp Peter had worked a lot on Weblogic Java application server.

His twitter is @peterseibel[53]

Tell us something interesting about yourself.

> I'm a second generation Lisp programmer. My dad discovered Lisp when he was working at Merck in the 80s and ended up doing a big project to simulate a chemical plant in Lisp, taking over from some folks who had already been trying for quite a while using Fortran, and saving the day. Later he went to Bolt Beranek and Newman where he did more Lisp. So I grew up hearing about how great Lisp was and even getting to play around with some graphics programs on a Symbolics Lisp Machine.
>
> I was also a childhood shareholder in Symbolics—I had a little money from some savings account that we had to close when we moved so my parents decided I should try investing. I bought Symbolics because my parents just had. Never saw that money again. As a result, for most of my life I thought my parents were these naive, clueless investors. Later I discovered that around that time they had also invested in Microsoft which, needless to say, they did okay with.

[46]http://gigamonkeys.com/book
[47]http://lispm.dyndns.org/dlw
[48]http://www.codersatwork.com/
[49]http://lisp-univ-etc.blogspot.com/2010/03/whats-on-coders-minds.html
[50]http://gigamonkeys.wordpress.com/
[51]http://weitz.de/cl-fad/
[52]http://www.youtube.com/watch?v=COEgRaf6acU
[53]http://twitter.com/peterseibel

Oh, and something I learned recently: not only was Donald Knuth one of the subjects in my book Coders at Work, but he has read the whole thing himself and liked it. That makes me happy.

What's your job? Tell us about your organization.

A few months ago I started working half-time at Etsy. Etsy is a giant online marketplace for people selling handmade and vintage items and also craft supplies. I'm in the data group where we try to find clever ways to use data to improve the web site and the rest of the business.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

I always have a SLIME session going in Emacs for quick computations and sometimes I prototype things in Lisp or write code to experiment with different ideas. However, these days I'm as likely to do those things in Python because I can show my co-workers a sketch written in Python and expect them to understand it and I'm not sure I could do that with Lisp. But it makes me sad how slow CPython is compared to a native-compiling CL like SBCL. Usually that doesn't matter but it is annoying sometimes mostly because Python has no real excuse. The rest of my work is in some unholy mishmash of Scala, Ruby, Javascript, and PHP.

What brought you to Lisp? What holds you?

As I mentioned, I grew up hearing from my dad about this great language. I actually spent a lot of my early career trying to understand why Lisp wasn't used more and exploring other languages pretty deeply to see how they were like and unlike Lisp. I played around with Lisp off and on until finally in 2003 I quit the startup I had been at for three years, which wasn't going anywhere, with a plan to take a year off and really learn Common Lisp. Instead I ended up taking two years off and writing Practical Common Lisp.

At this point I use it for things when it makes sense to do so because I know it pretty well and most of my other language chops are kind of rusty. Though I'm sure my CL chops are rusty too, compared to when I had just finished PCL.

Did you ever develop a theory why Lisp isn't used more?

Not one that is useful in the sense of helping it to be used more today. Mostly it seems to me to be the result of a series of historical accidents. You could argue that Lisp was too powerful too early and then got disrupted, in the Innovator's Dilemma sense, by various Worse is Better languages, running on systems that eventually became dominant for perhaps unrelated reasons.

Every Lisper should read The UNIX-HATERS Handbook[54] to better understand the relation between the Lisp and Unix cultures—Lisp is the older culture and back when the UNIX-HATERS Handbook was written, Unix machines were flaky and underpowered and held in the same contempt by Lisp geeks as Windows NT machines would be held by Unix geeks a few decades later. But for a variety of reasons people kept working on Unix and it got better.

And then it was in a better position than the Lisp culture to influence the way personal computing developed once micro computers arrived—while it would be a while before PCs were powerful enough to run a Unix-like OS, early on C was around to be adopted by PC programmers (including at Microsoft) once micros got powerful enough to not have to program everything in assembly. And from there, making things more Unix-like seemed like a good goal. Of course it would have been entirely possible to write a Lisp for even the earliest PCs that probably would have been as performant as the earliest Lisps running on IBM 704s and PDP-1s. My dad, back from his Lisp course at Symbolics, wrote a Lisp in BASIC on our original IBM PC. But by that point Lispers' idea of Lisp was what ran on powerful Lisp machines, not something that could have run on a PDP-1.

The AI boom and bust played its role as well because after the bust Lisp's reputation was so tainted by its failure to deliver on the over-promises of the Lisp/AI companies that even many AI researchers disassociated themselves from it. And throughout the 90s various languages adopted some of Lisp's dynamic features, so folks who gravitated to that style of programming had somewhere else to go and then when the web sprang into prominence, those languages were well positioned to become the glue of the Internet.

That all said, I'm heartened that Lisp continues to not only be used but to attract new programmers. I don't know if there will ever be a big Lisp revival that brings Lisp back into the mainstream. But even if there were, I'm pretty sure that there would be plenty of old-school Lispers who'd still be dissatisfied with how the revival turned out.

What's the most exciting use of Lisp you had?

I'm pretty proud of the tool chain I've built over the years while writing my two books and editing the magazine I tried to start, Code Quarterly. When I first started working on Practical Common Lisp I had some Perl scripts that I used to convert an ad-hoc light-weight text markup language into HTML. But after a little while of that I realized both that Jamie Zawinski[55] was right about regexps and that of course I should be using Lisp if I was writing a book called Practical Common Lisp.

[54]http://web.mit.edu/~simsong/www/ugh.pdf

[55]http://www.jwz.org/

So I implemented a proper parser for a mostly-plain-text language that I uncreatively call Markup and backends that could generate HTML and PDF using cl-typesetting[56]. When I was done writing and Apress wanted me to turn in Word files, I wrote an RTF backend so I could generate RTF files with all the Apress styles applied correctly. An Apress project manager later exclaimed over how "clean" the Word files I had turned had been. For editing Code Quarterly I continued to use Markup and wrote a prose diff tool that is pretty smart about when chunks of text get moved and edited a little bit.

What you dislike the most about Lisp?

I don't know if "dislike" is the right term because the alternative has its own drawbacks. But I do sometimes miss the security of refactoring with more static checks. For instance, when I programmed in Java, there was nothing better than the feeling of knowing a method was private and therefore I didn't have to look anywhere but in the one file where the method lived to see everywhere it could possibly be used. And in Common Lisp the possibilities for action at a distance are even worse than in some other dynamic languages because of the loose relation between symbols and the things they name. In practice that's not actually a huge problem and some implementations provide package locks and so on, but it always makes me feel a bit uneasy to know that if I `:use` a package and then `DEFUN` a function with the name of an inherited symbol I've changed some code I really didn't mean to.

From time to time I imagine a language that lets you write constraints on your code in the language yourself—kind of like macros but instead of extending the syntax your compiler understands, they would allow you to extend the set of things you could say about your code that the compiler would then understand. So you could say things like, "this function can only be called from other functions in this file" but also anything else about the static structure of your code. I'm not sure exactly what the API for saying those things would look like but I can imagine it being pretty useful, especially in larger projects with lots of programmers: you could establish certain rules about the overall structure of the system and have the compiler enforce them for you. But then if you want to do a big refactoring you could comment out various rules and move code around just like in a fully dynamic language. That's just a crazy idea; anyone who's crazy in the same way should feel free to take it and run with it and see if they get anywhere.

Among software projects you've participated in what's your favorite?

Probably my favorite software I ever wrote was a genetic algorithm I wrote in the two weeks before I started at Weblogic in 1998, in order to build up my Java chops. It played Go and eventually got to the point where it could beat a random player on a 5x5

[56]https://github.com/mbattyani/cl-typesetting

board pretty much 100% of the time. One of these days I need to rewrite that system in Common Lisp and see if I can work up to a full-size board and tougher opponents than random. (During evolution the critters played against each other to get a Red Queen effect—I just played them against a random player to see how they were doing.)

Describe your workflow, give some productivity tips to fellow programmers

I'm not sure I'm so productive I should be giving anybody tips. When I'm writing new code I tend to work bottom up, building little bits that I can be confident in and then combining. This is obviously easy to do in a pretty informal way in Common Lisp. In other languages unit tests can be useful if you're writing a bigger system though I'm often working on things for myself that are small enough I can get away with testing less formally. (I'm hopeful that something like Light Table will allow the easy of informal testing with the assurances of more strict testing—I'd love to have a development environment that keeps track of what tests go with what production code and shows them together and runs the appropriate tests automatically when I change the code.)

When I'm trying to understand someone else's code I tend to find the best way is to refactor or even rewrite it. I start by just formatting it to be the way I like. Then I start changing names that seem unclear or poorly chosen. And then I start mucking with the structure. There's nothing I like better than discovering a big chunk of dead code I can delete and not have to worry about understanding. Usually when I'm done with that I not only have a piece of code that I think is much better but I also can understand the original. That actually happened recently when I took Edi Weitz's Hunchentoot[57] web server and started stripping it down to create Toot[58] (a basic web server) and Whistle[59] (a more user friendly server built on top of Toot). In that case I also discarded the need for backward compatibility which allowed me to throw out lots of code. In that case I wasn't going for a "better" piece of code so much as one that met my specific needs better.

If you had all the time in the world for a Lisp project, what would it be?

I should really get back to hacking on Toot and Whistle. I tried to structure things so that all the Hunchentoot functionality could be put back in a layer built on top of Toot—perhaps I should do that just to test whether my theory was right. On the other hand, I went down this path because the whole Hunchentoot API was too hard for me to

[57]http://weitz.de/hunchentoot/
[58]http://github.com/gigamonkey/toot
[59]http://github.com/gigamonkey/whistle

understand. So maybe I should be getting Toot and Whistle stable and well-documented enough that someone else can take on the task of providing a Hunchentoot compatibility layer.

I'd also like to play around with my Go playing critters, reimplementing them in Lisp where I could take advantage of having a to-machine-code compiler available at run time.

PCL was the book, that opened the world of Lisp to me. I've also greatly enjoyed Coders at Work. So I'm looking forward for the next book you'd like to write. What would it be? :)

My current theory is that I'm going to write a book about statistics for programmers. Whenever I've tried to learn about statistics (which I've had to do, in earnest, for my new job) I find an impedance mismatch between the way I think and the way statisticians like to explain stuff. But I think if I was writing for programmers, then there are ways I could explain statistics that would be very clear to them at least. And I think there are lots of programmers who'd like to understand statistics better and may have had difficulties similar to mine.

Peter

2012-07-01

Marijn Haverbeke (Netherlands - Germany)

Marijn Haverbeke is not only the author of several pretty useful Common Lisp libraries[60] (some of which he touches in the interview), but also a succesful JavaScript hacker, winning JS1K contest[61], and writing a profound book about the language — "Eloquent JavaScript[62]". Besides, he hacks on Mozilla Rust and his own language Hob, not to mention co-authoring a JavaScript-to-Common-Lisp transpiler, writing some games in Scheme, and experimenting with Haskell. In the interview he shares his perspective based in knowledge and experience with so many different languages.

His twitter is @marijnjh[63]

Tell us something interesting about yourself.

> I once thought I wanted to be a Sociologist. Before that, I also once thought that I wanted to become a treasure hunter. And a dinosaur researcher. I ended up a programmer, largely self-taught, and I'm very happy with how things turned out.

What's your job? Tell us about your company.

> I'm working free-lance. Currently, I'm doing a lot of work for Mozilla, hacking on their Rust[64] compiler. Rust is a programming language that's supposed to fit the C++ niche, without the insanity of actual C++. If the project works out, and it's going swimmingly so far, Mozilla will try to apply it to writing a next-generation browser, without the constant danger of buffer overrun exploits and threading bugs that comes with C++. True to Mozilla's spirit, all development on Rust is happening in the open, on github.
>
> Another things I'm busy with is my own CodeMirror[65] project, a code editor written in JavaScript. It's not a 'job', really, since it's open-source and no one is paying me to maintain it, but it's grown popular enough to generate a decent amount free-lancing gigs and a small income stream from support contracts.

[60]http://marijnhaverbeke.nl/
[61]http://marijnhaverbeke.nl/js1k/
[62]http://eloquentjavascript.net/
[63]https://twitter.com/marijnjh
[64]http://rust-lang.org/
[65]http://codemirror.net/

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

Not currently, beyond that I always have a REPL open in emacs, and switch to it to do quick computations and experiments. I've spent several years doing almost exclusively Lisp (mostly working on AllegroGraph[66], Franz' RDF database), and it has definitely been somewhat painful to find myself working with non-interactive, macro-less languages again, after being used to the luxuries of Common Lisp and Slime.

What brought you to Lisp? What holds you?

Ten years ago, during my sociology study, I randomly chose a Lisp course in university. I was spending so much time in emacs that I figured I might as well learn more about its scripting language. The course worked through Structure and Interpretation of Computer Programs[67], using the video lectures from Abelson and Sussman. Having been exposed mostly to Java, C++, and Python before that, the elegance and depth of this material was something of a revelation to me.

My professor for that course, Eduard Hoenkamp, was a real Lisp chauvinist with an MIT background, and properly indoctrinated me with regards to Lisp's glorious superiority. I wrote a text adventure game in Scheme for the end project of the course, my first non-trivial Lisp endeavor. I still have the code[68]. I ended up starting a PhD with this very professor, and though I never finished it, it did provide me with ample opportunity to learn Common Lisp.

What's the most exciting use of Lisp you had?

I think that'd be CL-JavaScript[69], a JavaScript-to-Common-Lisp transpiler that I wrote together with a friend. It actually handles the whole ECMAScript 3 language and produces programs that run reasonably fast—about on par with the 2009 generation of browser JavaScript engines. This is quite amazing, given that it's less than 4000 lines of code, at least an order of magnitude smaller than those engines. The Lisp compiler handles most of the optimization for us. (I should point out that the browser JS engines have a much higher speed of compilation, which is a necessity for handling web content.)

[66]http://www.franz.com/agraph/allegrograph/

[67]http://mitpress.mit.edu/sicp/

[68]http://marijnhaverbeke.nl/dunwich/

[69]http://marijnhaverbeke.nl/cl-javascript

What you dislike the most about Lisp?

I must say that the lack of static typing really gets in the way on larger projects. Being able to confidently change datatypes and function signatures, knowing that the compiler will point out most inconsistencies that you introduce, is something that I've really come to appreciate after working with Rust and Haskell. Test coverage helps, of course, but I'm not a very diligent test writer, and tend to feel that type signatures are easier to maintain than an exhaustive test suite.

Describe your workflow, give some productivity tips to fellow programmers.

My approach to writing software is embarrassingly unstructured. I'm not sure I can claim to have something that merits the term "workflow". One thing that is very important for me, and that, had I figured it out earlier, would probably have saved some past projects from dying, is to never have a system in a half-refactored, non-functional state for longer than an hour or so. A lot of momentum and motivation comes from having a working program in front of me. This sometimes means I have to split up big changes in a number of more indirect, trivial steps, and resist the temptation to start hammering out my big change in one go (which, even when seems straightforward in my head, will always hit some complications during implementation).

Also, really learning git in depth has had a big influence on how I code. Once you really understand that changes (if you bother to commit often and with care) can be reverted and picked at will, and that branches are cheap and easy to create and maintain, you start to see a codebase differently. I often take multiple shots at making a tricky change — if it feels like my first approach is not working out, I simply commit it, back up, and start again, knowing that if the new approach doesn't work out, I can simply flip back to the old one again. Knowing that this is theoretically possible is not enough—you need to have actually done it, often, to start applying it properly.

Among software projects you've participated in what's your favorite?

Postmodern[70], the PostgreSQL client, was my first non-trivial open-source project, and the first piece of actually useful software that I designed and executed on my own. I was extremely proud when I got it to work the way I wanted (modular, fast), and meticulously wrote out a long set of docs (copying Edi Weitz's style). Seeing people actually use it (and even write good things about it!) really drove home for me the fact that it is worthwhile and rewarding to package up the stuff I write, and share it.

[70] http://marijnhaverbeke.nl/postmodern

The community involvement in Postmodern, though always low-volume, has also been awesome. Most bug reports came with a patch, or at least an informed view of the problem and useful diagnostic information. This in sharp contrast with the JavaScript community, where of course great contributors also occur, but one has to deal with a lot of noise and lazy bug reporting.

If you had all the time in the world for a Lisp project, what would it be?

I am working on my own programming language, Hob[71], which tries to combine the virtues of Lisp and ML. This is a huge project, and unfortunately it's moving forward only slowly. If I didn't have financial constraints, I'd spend a year (or two) finishing it. Its "bootstrap compiler" (the compiler that will compile the first self-hosting compiler) is in the process of being written in Common Lisp. The writing that I currently have online about it is quite old and out of date. I just spent a week rewriting the compiler from scratch, moving the language to a completely regular syntax—think s-expressions, though Hob looks somewhat different. My work on Rust has convinced me that, even with all the good will in the world, you can't bolt macros onto a classical language with a non-regular syntax without making them very painful to write (and use!). They have to be part of the design from the start.

You've been actively using Common Lisp and JavaScript. When you use JavaScript what do you miss the most from CL? What do you miss from JavaScript when you're in Lisp land?

The greatest thing about Common Lisp is that the standardization process drew from years and years of experience and organic growth. This makes the language feel singularly solid and practical. Most languages feel immature compared to CL. I suspect that this is not just age, but also the fact that Lisp allows users to extend the language to an unprecedented degree, which allows for much more experimentation, and thus faster growth and invention.

The great thing about JavaScript is that it has so few concepts. It manages to strike a balance between being dumbed-down enough to be easy to grasp, which has allowed it to become as widespread as it currently is, and still being a more-or-less decent lambda language, which makes it scale up to complex programs relatively well. It is on its way to become a lingua franca of scripting, which Common Lisp will never be, because it is much more demanding of its programmers—they need to understand a lot more concepts.

2012-04-02

[71] http://marijnhaverbeke.nl/hob/

François-René (Faré) Rideau (France - USA)

François-René Rideau works at ITA Software, one of the largest employers of lispers, which was acquired by Google a year ago. While at ITA he stepped up to support and improve ASDF[72], the system definition facility, that is at the core of Lisp package distribution. He's also the co-author of the recently published Google Common Lisp Style Guide[73], which as well originated at ITA.

Faré is also an active writer: both of code and prose. He's thoughts and articles can be found on Facebook[74], Google+[75], Livejournal[76], and his site[77].

His twitter is @fare[78]

Tell us something interesting about yourself.

> I like introducing myself as a cybernetician: someone interested in the dynamic structure of human activities in general.
>
> Programming languages and their semantics, operating systems and reflection, persistence of data and evolution of code, the relation between how programmers are organized and what code they produce — these are my topics of immediate professional interest. For what that means, see for instance my slides (improved) from ILC'09: "Better Stories, Better Languages[79]" or my essay "From Creationism to Evolutionism in Computer Programming[80]".
>
> However I'm also interested in cybernetics as applies to Civilization in general, past, present and future. See for instance my essay "Identity, Immunity, Law and Aggression on the Rapacious Hardscrapple Frontier[81]" or my writings about Individual Liberty[82] and the basic principles of Economics
>
> Last but not least, I was recently married to my love Rebecca Kellogg, with whom I have since had a daughter Guinevere Lý "Véra" Kellogg Rideau (born last May). This gives me less free time, yet somehow made me more productive.

[72] http://common-lisp.net/project/asdf/

[73] https://google-styleguide.googlecode.com/svn/trunk/lispguide.xml

[74] http://www.facebook.com/fahree

[75] https://plus.google.com/108564127390615114635/posts

[76] http://fare.livejournal.com/

[77] http://fare.tunes.org/

[78] http://twitter.com/fare

[79] http://fare.tunes.org/computing/fare-ilc09-lightning.html

[80] http://fare.tunes.org/computing/evolutionism.html

[81] http://fare.tunes.org/liberty/hardscrapple.html

[82] http://fare.tunes.org/liberty/

What's your job? Tell us about your company.

For the last 7 years or so, I have been working at ITA Software, now part of Google Travel. I have been working on two servers written in Lisp, at first briefly on QPX[83] the low (air)fare search engine behind Orbitz and Google Flights then mostly on QRes, a reservation system now launched with Cape Air. These projects nowadays each count about half a million lines of Common Lisp code (though written in very different styles), and each keep growing with tens of active developers.

I suspect that my login "fare" (at itasoftware) was a pun that played in favor of recruiting me at ITA; however, it wasn't available after the Google acquisition, so now I'm "tunes" (at google), to remind myself of my TUNES project[84].

At ITA, I have been working mostly on infrastructure:

- how to use better compilers (moving from CMUCL to SBCL, CCL),
- how to build, run and test our software,
- how to maintain the free software libraries we use and sometimes write,
- how to connect QRes to QPX and follow the evolution of its service,
- how to persist objects to a robust database,
- how to migrate data from legacy systems,
- how to upgrade our software while it's running, etc.

And debugging all of the above and more, touching many parts of the application itself along the way.

I think of my job at ITA so far as that of a plumber: On good days, I design better piping systems. On bad days, I don gloves and put my hands down the pipes to scrub.

Since you're mentioning me as working at ITA and on ASDF, I suppose it is appropriate for me to tell that story in full.

In building our code at ITA, we had grown weary of ASDF as we had accumulated plenty of overrides and workarounds to its unsatisfactory behavior. Don't get me wrong: ASDF was a massive improvement over what existed before (i.e. `mk-defsystem`), making it possible to build and share Common Lisp software without massive headaches in configuring each and every library. We have to be grateful to Dan Barlow indeed for creating ASDF. But the Common Lisp ecosystem was dysfunctional in a way that prevented much needed further improvements to ASDF. And so I started working on a replacement, XCVB[85].

Now, at some point in late 2009, I wrote a rant explaining why ASDF could not be saved: "Software Irresponsibility[86]". The point was that even though newer versions of ASDF were written that slowly addressed some issues, every implementation stuck to

[83] http://matrix.itasoftware.com/
[84] http://tunes.org/
[85] http://common-lisp.net/project/xcvb/
[86] http://fare.livejournal.com/149264.html

its own version with its own compatibility fixes; no vendor was interested in upgrading until their users would demand upgrades, and users wouldn't rely on new features and bug fixes until all vendors upgraded, instead caring a lot about bug-compatibility, in a vicious circle of what I call "Software Irresponsibility", with no one in charge, consensus required for any change, no possible way to reach consensus, and everyone discouraged.

However, I found a small flaw in my condemnation of ASDF as unsalvageable: if, which was not the case then, it were possible to upgrade ASDF from whichever version a vendor had installed to whichever newer version you cared for, then ASDF could be saved. Users would be able to rely on new features and bug fixes even when vendors didn't upgrade, and vendors would have an incentive to upgrade, not to stay behind, even if their users didn't directly demand it. The incentive structure would be reversed. Shortly after I wrote this rant, the current ASDF maintainer stepped down. After what I wrote, I felt like the honest thing to do was to step forward. Thus, I started making ASDF self-upgradable, then massively improved it, notably making it more robust, portable, and easy to configure — yet fully backwards compatible. I published it as ASDF 2 in 2010, with the help of many hackers, most notably Robert Goldman[87], and it has quickly been adopted by all active Common Lisp vendors.

You can read about ASDF and ASDF 2 in the article I wrote with Robert Goldman for ILC 2010: "Evolving ASDF: More Cooperation, Less Coordination[88]". I'm also preparing a talk at ILC 2012 where I'll discuss recent enhancements. I have to admit I didn't actually understand the fine design of ASDF until I had to explain it in that paper, thanks to the systematic prodding of Robert Goldman. Clearly explaining what you're doing is something I heartily recommend to anyone who's writing software, possibly as a required step before you declare your software complete; it really forces you to get the concepts straight, the API clean, and the tests passing. That also did it for me with my more recent lisp-interface-library[89], on which I'm presenting a paper at ILC 2012: "LIL: CLOS reaches higher-order, sheds identity, and has a transformative experience[90]".

One double downside of ASDF 2 is that it both took a lot of resources I didn't put in XCVB, and made for a much better system for XCVB to try to disrupt. It isn't as easy anymore to be ten times better than ASDF. I still hope to complete XCVB some day and make it good enough to fully replace ASDF on all Common Lisp platforms; but the goal has been pushed back significantly.

Now one important point that I want to explicitly stress is that the problem with ASDF was not a strictly technical issue (though there were many technical issues to fix), nor was it strictly a social issue; it was an issue at the interface between the social and the technical spheres, one of how our infrastructures and our incentives shape each other, and what kind of change can bring improvement. That's the kind of issues that interest me. That's why I call myself a cybernetician.

[87]http://rpgoldman.goldman-tribe.org/index.html

[88]http://common-lisp.net/project/asdf/ilc2010draft.pdf

[89]http://www.cliki.net/lisp-interface-library

[90]http://common-lisp.net/~frideau/lil-ilc2012/lil-ilc2012.html

Do you use Lisp at work? If yes, how have you made it happen? If not, why?

I've made it happen by selection. I applied at ITA Software precisely because I knew (thanks to the Carl de Marcken article[91] published by Paul Graham), that the company was using Lisp to create real-world software. And that's what I wanted to do: create real-world software with a language I could use without wanting to kill myself every night because it is turning me into a pattern-expanding machine rather than a human involved in thinking and using macros as appropriate.

> I object to doing things that computers can do.
> — Olin Shivers

Yet, in my tasks as a plumber, I have still spent way too much time writing shell scripts or `Makefiles`; though these languages possess some reflection including `eval`, their glaring misdesign only lets you go so far and scale so much until programs become totally unmanageable. That's what pushed me over the years to develop various bits of infrastructure to do as much of these things as possible in Lisp instead: cl-launch[92], command-line-arguments[93], philip-jose[94], xcvb, asdf, inferior-shell[95].

Interestingly, the first and the last, cl-launch and inferior-shell, are kind of dual: cl-launch abstracts over the many Lisp and shell implementations so you can invoke Lisp code from the Unix shell; it is a polyglot lisp and shell program that can manipulate itself and combine parts of itself with user-specified Lisp code to produce an executable shell script or a dumped binary image; I sometimes think of it as an exercise in "useful quining". Inferior-shell abstracts over the many Lisp and shell implementations so you can invoke Unix shell utilities from any Lisp implementation, remotely if needs be (through ssh), and with much nicer string interpolation than any shell can ever provide; it is a classic Lisp library notably available through Quicklisp. With the two of them, I have enough Unix integration that I don't need to write shell scripts anymore. Instead, I interactively develop Lisp code at the SLIME REPL, and have a shell-runnable program in the end. That tremendously improved my quality of life in many situations involving system administration and server maintenance.

What brought you to Lisp? What holds you?

My first introduction to Lisp was in high school, in using the HP RPL[96] on my trusty old HP 28C (eventually upgraded to a HP28S, with 32KB of free RAM instead of 4KB!).

[91]http://www.paulgraham.com/carl.html
[92]http://cliki.net/cl-launch
[93]http://common-lisp.net/project/qitab/
[94]http://www.cliki.net/philip-jose
[95]http://www.cliki.net/inferior-shell
[96]http://en.wikipedia.org/wiki/RPL_(programming_language)

When I became student at Ecole Normale Supérieure, I was taught Caml-light[97] by xleroy[98] himself, I learned to use Emacs, and I met Juliusz Chroboczek[99] who introduced me to Scheme and Common Lisp, continuations and SICP[100]. Finally, during my vain efforts to gather a team to develop an operating system based on a higher-level language as part of the TUNES project, I have been introduced to Lisp machines and plenty of other interesting concepts.

I use Lisp because I couldn't bear to program without higher-order functions, syntactic abstraction and runtime reflection. Of all Lisp dialects, I use Common Lisp mainly because that's what we use at work; of course a large reason why we use it at work is because it's a good language for practical work. However, frankly, If I were to leave ITA (by Google), I'd probably stop using Common Lisp and instead use Racket or Maru, or maybe Factor or Slate, and try to bootstrap something to my taste from there.

What's the most exciting use of Lisp you had?

I remember being quite exhilarated when I first ran the philip-jose farmer: it was a server quickly thrown together by building green-threads on top of arnesi[101]'s (delimited) continuation library for CL. With it, I could farm out computations over a hundred servers, bringing our data migration process from "way too slow" (weeks) to "within spec" (a few hours). It's impressive how much you can do in Lisp and with how little code!

While I released the code in philip-jose, it was never well-documented or made user-friendly, and I suspect no one ever used it for real. This unhappily includes ITA, for my code never made it to production: I was moved to another team, our customer went bankrupt, and the new team used simpler tools in the end, as our actual launch customer was 1/50 the size of our first prospect.

What you dislike the most about Lisp?

For Lisp in general, I would say the lack of good ways to express restrictions on code and data. Racket has been doing great work with Typed Racket and Contracts; but I'm still hoping for some dialect with good resource management based on Linear Logic, and some user-extensible mechanism to define types and take advantage of them.

For Common Lisp in particular, though I do miss delimited continuations, I would say that its main issue is its lack of modularity. The package system is at the same time low-level and inexpressive; its `defsystem` facilities are also lacking, ASDF 2 notwithstanding;

[97]http://caml.inria.fr/
[98]http://pauillac.inria.fr/~xleroy/
[99]http://www.pps.univ-paris-diderot.fr/~jch/
[100]http://mitpress.mit.edu/sicp/
[101]http://common-lisp.net/project/bese/arnesi.html

up until the recent success of Zach Beane's Quicklisp[102], there wasn't a good story to find and distribute software, and even now it's still behind what other languages have. This is part of a vicious circle where the language attracts and keeps a community of developers who live happily in a context where sharing and reusing code is relatively expensive (compared to other languages). But things are getting better, and I have to congratulate Zach Beane once again for Quicklisp. I believe I'm doing my small part.

Among software projects you've participated in what's your favorite?

I unhappily do not have a great history of success in software projects that I have actively participated in.

However, I have been impressed by many vastly successful projects in which I had but a modest participation. In the Linux kernel, the Caml community, the Racket community, (QPX and QRes at work might also qualify but only to lesser degrees), there were bright people unified by a common language, by which I mean not merely the underlying computer programming language, but a vision of things to come and a common approach to concepts: not just architecture but architectonics. Another important point in these successful projects was Software Responsibility (as contrasted to the previously discussed Software Irresponsibility): there is always someone in charge of accepting or rejecting patches to any part of the system. Patches don't linger forever unapplied yet unrejected, so the software goes forward and the rewarded contributors come back with more and/or better patches. Finally, tests. Lots of them. Automatically run. All the time. Proofs can do, too, though they are usually more expensive (now if you are going to do testing at the impressive scale of sqlite[103], maybe you should do proofs instead (see CPDT[104]). I discovered, the hard way, that tests (or proofs) are the essential complement to programs, without which your programs WILL break as you modify them.

If you had all the time in the world for a Lisp project, what would it be?

I would resurrect TUNES based on a Linear Lisp, itself bootstrapped from Racket and/or Maru.

[102]http://quicklisp.org/
[103]http://www.sqlite.org/testing.html
[104]http://adam.chlipala.net/cpdt/

Describe your workflow, give some productivity tips to fellow programmers.

First, think hard and build an abstract model of what you're doing. Guided by this understanding of where you're going, code bottom up, write tests as you do, and run them interactively at the SLIME REPL; make sure what you write is working and passing all tests at all times. Update your abstract model as it gets pummeled into shape by experience. Once you've got the code manually written once or twice and detect a pattern, refactor it using macros to automate away the drudge so the third time is a piece of cake. Don't try to write the macro until you've written the code manually and fully debugged it. Never bother with low-level optimization until the very end; but bother about high-level optimization early enough, by making sure you choose proper data structures.

Unhappily, I have to admit I am a serial under-achiever. I enjoy thinking about the big picture, and I like to believe I often see it better and further than most people; but I have the greatest trouble staying on track to bring about solutions: I have so many projects, and only one life to maybe complete a few of them! The only way I can actually get a few things done, is to decompose solutions into small enough steps such that I can keep focused on the next one and get it done before the focus goes away.

A year ago Google bought ITA, which was, probably, the largest Lisp company recently. What were the biggest upsides and drawbacks of using Lisp on the scale of ITA? Does Lisp have a future inside Google?

On the upside, we certainly have been able to write quite advanced software that we might not have otherwise managed. A million lines of Lisp code, including its fair share of macros and DSLs, would be so many more million lines of code without the syntactic abstraction made possible by Lisp. However hard and expensive it was with Lisp, I can only imagine how many times worse it would have been with anything else. At the top of the tech bubble in 2008, we had over fifty Lisp programmers working just on QRes, gathered at an exponential rate over 3 years. That's a lot. We didn't yet have good common standards (Jeremy Brown started one, later edited by Dan Weinreb[105]; I recently took it over, expanded it, merged it into the existing beginning of a Google Common Lisp Style Guide and published it), and it was sometimes hard to follow what another hacker wrote, particularly if the author was a recently hired three-comma programmer[106]. But with or without standards, our real, major, problem was with lack of appropriate management.

[105]http://en.wikipedia.org/wiki/Daniel_Weinreb

[106]http://c2.com/cgi/wiki?ThreeStarProgrammer

We were too many hackers to run without management, and none of our main programmers were interested in becoming managers; instead managers were parachuted from above, and some of them were pretty bad: the worst amongst them immediately behaved like empire-building bullies. These bad managers were trying to control us with impossibly short, arbitrary deadlines; not only did it cause overall bad quality code and morale burnout, the renewing of such deadlines quarter after quarter was an impediment to any long-term architectural consideration for years. What is even worse, the organization as setup had a lot of inherent bad incentives and created a lot of conflicts, so that even passable managers would create damage, and otherwise good engineers were pitted against each other on two sides of absurd interfaces, each team developing a lot of scar tissue around these interfaces to isolate itself from the other teams. Finally, I could witness how disruptive a single bad apple can be when empowered by bad management rather than promptly fired.

I have had a lot of losing fights with QRes management at a time when, hanging on a H1B visa, I was too much of a coward to quit. Eventually, the bad people left, one by one, leaving behind a dysfunctional organization; and great as the people that manned it may have been, none was able or willing to fix the organization. Then finally, Google acquired us. There's a reason why, of two companies founded at about the same time, one buys the other and not the other way around: one grew faster because it got some essential things right that the other didn't. Google, imperfect as it necessarily is, gets those essential things right. It cares about the long term. It builds things to scale. It has a sensible organization. It has a bottom up culture. So far, things have only improved within QRes. Also, launching was also good in many ways. It makes us and keeps us real.

Lisp can be something of a magic tool to solve the hardest technical issues; unhappily it doesn't even start to address the social issues. We wasted a whole lot of talent due to these social issues, and I believe that in an indirect way, this is related to the lack of modularity in Common Lisp, as it fostered a culture of loners unprepared to take on these social issues.

So I'm not telling you this story just to vent my past frustration. There too I have a cybernetic message to pass on: incentives matter, and technical infrastructure as well as social institutions shape those incentives and are shaped by them.

As for the future of Lisp at Google, that million line of Common Lisp code ain't gonna rewrite itself into C++, Java, Python, Go, or even DART. I don't think the obvious suggestions that we should rewrite it were ever taken seriously. It probably wouldn't help with turnover either. But maybe, if it keeps growing large enough, that pile of code will eventually achieve sentience and rewrite itself indeed. Either that, or it will commit suicide upon realizing the horror.

Anything else I forgot to ask?

Ponies[107].

Faré

2012-10-19

[107] http://fare.livejournal.com/tag/pony

Daniel Barlow (UK)

Daniel Barlow was one of the most active contributors to the open source Lisp ecosystem, when its development took off in the early 2000s. Together with Christophe Rhodes he was the first to join SBCL hacking, after the project was started by William Newman[108]. He also had created a lot of early Lisp web tools, like Araneida[109] HTTP application server, and built on it the first version of cliki.net[110] which served the Lisp community for almost 10 years (the second version went live earlier in 2012). Studying Cliki source was a kind of zen experience for me, as it did so much, yet in a very simple way.

But his largest contribution is, probably, ASDF[111], regarding which, likewise Cliki, there are controversial opinions among Lisp programmers. And Dan explains his attitude in the interview.

In the mid 2000s his involvement with open-source Lisp gradually diminished, as he stopped working as a consultant and got a full-time job. Yet he remains fondly remembered in the community.

Tell us something interesting about yourself.

> I don't do interesting. Um, improvise. Hacker, skater, cyclist, husband, father to a seven-month-old son as demanding as he is adorable, computing retro-grouch who uses Linux on the desktop.
>
> I have a metal plate and some pins in my right forearm where I broke it a couple of months ago, inline skating in the Le Mans 24 hour relay event. I have now regained more or less complete range of motion in that hand and can advise anyone doing the event next year that the carpet in the pit boxes is unexpectedly treacherous when it's been waterlogged by a sudden thunderstorm.
>
> Still, we came fifth in category, which makes me very happy.

What's your job? Tell us about your company.

> I started a new job about three months ago, in fact. I'm now working at Simply Business in London, busily disrupting the business insurance market. Which is to say, writing web apps for the online sale of business insurance.
>
> It's not quite as buzzwordy as it sounds, actually. Business insurance is traditionally sold by brokers, who are humans and therefore although really good at dealing with

[108] http://www.advogato.org/person/wnewman/
[109] http://www.cliki.net/araneida
[110] http://cliki.net
[111] http://common-lisp.net/project/asdf/

complex cases and large contracts where the personal touch is required, tend to be a trifle expensive for straightforward policies which could be much more economically sold online. The industy is ripe for disintermediation.

What brought you to Lisp?

A combination of factors around the time I was at university: the UNIX-Haters Handbook[112], which I bought to sneer at and ended up agreeing with; Caml Light, which I used in my fourth year project; Perl - specifically my horror to learn that it flattens `(1,2,3,(4,5,6))` to `(1,2,3,4,5,6)` - yes, I know about references, but I think it's a bug not a feature that the sensible syntax is reserved for the silly behaviour - and meeting some people from Harlequin (as was) at a careers fair.

It took me another couple of years or so to find CMUCL - in fact, I think it was another couple of years or so before it was ported to the x86 architecture, so it wouldn't have done me much good if I had known about it earlier - but looking back I suppose that was where the rot set in.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

No, we're primarily a Ruby shop, with a sideline in large legacy Java app which we're working to replace. I think there've been a couple of uses of Clojure in hack days, but that's as far as it goes.

Why not? Well, apart from the point that I've only been there since July… It's The Ecosystem, I suppose. Ruby as a language has pretty good support for OO paradigms and a whole bunch of free software libraries for doing web-related things: Ruby as a community is big on Agile and TDD and maintainable design. And it's at least possible (if not exactly easy in the current bubble) to engage a regular recruitment agent and task him with finding competent programmers who know it. I'm not saying Lisp is exactly bad at any of that, but it's at least questionable whether it's as good along all of those axes, and it's certainly not better enough to make the switch sensible.

What's the most exciting use of Lisp you had?

SBCL was probably one of the most fun projects I've ever worked on. Working with people who were mostly smarter than me or had better taste than me or both, on a project whose goal was to make it possible for people less smart than me to hack on a Lisp system. And context switching between the CL with all its high level features and

[112]http://web.mit.edu/~simsong/www/ugh.pdf

(e.g.) Alpha assembly was a real kick - it's a bit like I imagine building a Lisp machine would be, except that the goal is achievable and the result is generally useful.

What you dislike the most about Lisp?

I don't really use it enough any more to react to that with the required levels of venom. I should probably say ASDF, everyone else does :-)

I guess if you force me to an answer, it'd have to be its disdain for the platform it lives on - take, for example, CL pathname case conversion rules. Whoever decided that Unix systems could reasonably be said to have a "customary case" had, in my view, not looked very hard at it.

As far as I can tell, you're currently mostly doing work in Ruby. What's the pros and cons of Ruby development compared to Lisp?

The `transpose-sexps` function in Emacs does nothing useful in Ruby mode. rails console is a poor substitute for a proper toplevel. Backtraces don't show the values of parameters and local variables. (Yes, pry helps a lot). And the garbage collector (in MRI, anyway) is sucky to the point that even 1990s Java GC could probably beat it in a fair fight.

On the other hand, libraries.

Here's an interesting thought experiment, though: there's a clear difference between the Lisp workflow where you change the state of your image interactively to get the code into working shape very quickly (and then later try to remember what it was you did) and the more scripted approach of test-driven development in Ruby where you put everything (code, test setup, assertions) in files that you reload from disk on each run. How would you meld the two to get repeatable and fast iterations? A lot of people are doing things like Spork (which forks your application for each test it runs, throwing the child state away after the test has run) but they never seem to me to be more than 80% solutions. My intuition is that you'd want to stick to a much more functional design and just make state a non-problem.

Among software projects you've participated in what's your favorite?

SBCL was a lot of fun, as I said earlier. ASDF is a candidate too, just because it must so obviously fill a need if people are still cursing it - as they seem to be - ten years later :-)

Describe your workflow, give some productivity tips to fellow programmers.

It’s taken me the best part of three months to get this interview back, I’m the last person anyone should be asking about workflow or productivity. :-)

Um. I’ve been doing a lot of TDD lately. Given that as recently as two years ago I was castigating it as a religion this might be seen as a capitulation or as a conversion, this might be perceived as a change of mind. What can I say? Actually, pretty much now what I said then: the value of TDD is in the forces it exerts on your design — towards modularity, functional purity, decoupling, all those good things — not so much in the actual test suite you end up with. Process not product. These days everyone thinks that’s obvious, but back then it was either less widely known or less explicitly stated or else I was just reading the wrong blogs.

(Of course, the tendency in Lisp to write code interactively that can be tested ad hoc at the repl probably has a very similar effect on coupling and functional style. My personal experience is that TDD doesn’t seem to be nearly as valuable in repl-oriented languages, but YMMV.)

More generally: go home, do some exercise, get some sleep. Sleep is way underrated.

Anything else I forgot to ask?

Some day I will write an apology for ASDF.

Pedants will note that the word “apology” not only means “an expression of remorse or regret” but also “a formal justification or defence”, and may infer from that and my general unwillingness to ever admit I was wrong that I’m not about to actually say I did a bad thing in writing it. Seriously, go find a copy of MK-DEFSYSTEM and try porting it to a Lisp implementation it doesn’t support.

In 2002 I presented a paper at the ILC (about CLiki, not ASDF) that said essentially “worse is better than nothing”, and — unless the “worse” has the effect of stifling a potential better solution from coming along later — I still stand by that.

danb

2012-10-08

Christophe Rhodes (UK)

Christophe Rhodes is the SBCL's principal maintainer. He was among the first two people to join the project, after it was solely developed for more than a year by William Newman[113]. And he is, probably, the most "stable" contributor, still remaining very active in the community, after more than a decade. What fascinates me personally about him, is the aura of a profoundly creative mind, fiddling with reality in interesting and unexpected ways.

His twitter is @ascii19[114]

Tell us something interesting about yourself.

> I'm a singer, specializing in performance (admittedly somewhat rare at the moment) of unaccompanied music of the European Renaissance. My next concert is Brumel's "Earthquake Mass" with the vocal ensemble De Profundis[115]. I did my doctoral research a few doors down from Stephen Hawking. And I was a cast member on "Una Stravaganza dei Medici", a reconstruction of the Florentine festivities for the marriage of Ferdinand de Medici and Christine of Lorraine - which means that there's at least some chance I have an Erdős-Bacon number.

What's your job? Tell us about your company.

> Oh, well, that's a bit complicated! :-) Most of my time at the moment is spent with Teclo Networks[116], a Lisp-flavoured startup which is bringing reliability and usability to mobile broadband. I'm on leave of absence to pursue this from Goldsmiths, University of London[117], where I am a lecturer in Computing, responsible for undergraduate teaching, postgraduate supervision and independent research.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

> I do. I should say that I'm no longer, if I ever was, a programmer by trade; most of my work at Teclo is not writing code, but when I do it is in C, R or Common Lisp. Working in a Lisp-flavoured startup (I'm not the most famous Lisp programmer in the company,

[113]http://www.advogato.org/person/wnewman/
[114]https://twitter.com/ascii19
[115]http://www.deprofundis.org.uk/
[116]http://teclo.net/
[117]http://www.doc.gold.ac.uk/

and quite some way from being the best…) has made Lisp a natural tool to use when it's appropriate; while it might be a bit of a shame that Lisp is not used exclusively, the balance of idealism and pragmatism is pretty good, and it's fairly satisfying to know that the high-performance network element that we ship has Lisp code controlling its operation.

I wouldn't make an argument that C is a Lisp by any stretch, but I've popped up at a couple of events in the last year to argue that R has some claims to being a Lisp; it has conditions, handlers, restarts, generic functions, macros operating on parse trees… and a SWANK backend[118].

Another part of what I do is write documents, both for Teclo and for Goldsmiths. I have in the last couple of years fallen in love with org-mode[119] and in particular with "reproducible research", or living documents; the ability to have executable sections of code for generating results, which can then themselves be further analysed by other bits of code — in other languages, if that's appropriate — and the whole exported as a document typeset to what I consider to be high-quality standards. The fact that there is also acceptable-quality web output from the exact same process is gravy.

What brought you to Lisp? What holds you?

Two seminal moments on my road to Lisp: the first was, as an undergraduate, getting a job in the summer of 1997 to work on simulating fluid dynamics in the context of galaxy formation, to assist in trying to understand how the spiral arm structures form. I'd being told "here is K&R, here is USENET, here is XEmacs, here is Tcl; in five weeks let's see what you've come up with" — I distinctly remember reading comp.lang.lisp and a "Lisp is slow" thread, with active participants combating the troll with disassembly.

The second, a few years later, was wandering into the #lisp IRC channel on openprojects, to find an almost-empty channel, about 10-20 participants; one of whom was there twice (dan_b and dan`b), using the channel to paste bits of alpha assembly between his laptop and his desktop, in his efforts to get SBCL on alpha working.

As for what keeps me, well, when what I have to do is solve problems that no language has built-in libraries for (scalable audio similarity detection, say, or musical structure inference) then I want to be working in a language where I can focus on the problem itself, rather than the detail of that language — and for me, Lisp permits that kind of separation of concerns, given its easy support for DSL and embedded language generation, protocols, and the ability to run partial programs.

[118]http://common-lisp.net/~crhodes/swankr/
[119]http://orgmode.org

Among software projects you've participated in what's your favorite?

I enjoyed most working on an editor for lute tablature[120], I think. Partly because it was a chance to learn about things (parsing, CLIM, user interfaces) that I hadn't really thought about before. Partly because of the feeling of being able to build up incrementally and rapidly from a very simple prototype to deliver something which was actually useful; and partly because it was also interesting to learn about the problem domain. Gsharp (G#), the score editor largely designed by Robert Strandh, is an editor for what you might call "common practice" notation: five lines in a staff; dots and lines to indicate pitch (by height) and duration (by glyph) of notes. In my research group at Goldsmiths, though, we have a lutenist (a player of the lute) who had a research project on cataloguing and transcribing a particular repertoire of 16th-century lute music, which was notated in "tablature": six lines, one per string of the lute; letter or number glyphs on each line to indicate finger position; and glyphs above the staff to indicate duration.

The point about this was that when I first came on the scene the project had a truly awful workflow, because they had a batch renderer from the textual format to encode this music to something visual. This made the process of encoding painful, in particular for the error correction phase: it was fairly easy to find an error in the visual rendering, but to go from there to the text corresponding to it was painful. So I wrote a little CLIM app with an editor buffer and a display; the display was a rendering of the "Tabcode" in the editor buffer, and each glyph in it was associated with a text cursor position, so that you could click on the display and have the editor cursor warp to the Tabcode corresponding to the glyph. This was a productivity win for the project.

What's the most exciting use of Lisp you had?

I don't get easily excited by technology. So for me, what was exciting was seeing a community of users and developers grow around SBCL — in particular, working with extremely focused and motivated individuals on improving the system, subject to all sorts of constraints, not least those imposed by real-world uses (including the QPX[121] poster-child of ITA Software).

[120]http://www.doc.gold.ac.uk/~mas01cr/papers/tabeditor.pdf

[121]http://www.itasoftware.com/products/shopping-pricing/qpx.html

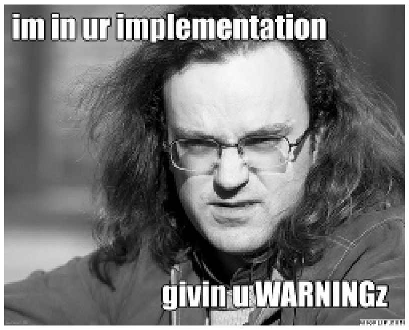
SBCL contributors seem to be a very active, closely-knit, and united community. How did that happen? Maybe you could share some insight into the history of the project. Also what were the factors, that determined its success?

I think part of SBCL's success as a community was the finding of like minds — which came about partly because it started off as a project when access to information channels was growing, but also partly because there were some explicit principles laid down: Bill Newman made it very clear that he was aiming for something maintainable, and also taught by example the meaning of maintainability. In the SBCL source distribution, there are over 100 lines of textual detail describing his ideas about what he was aiming for, and although it's nothing like an explicit charter which every SBCL developer (or user!) has to sign up to, it's nevertheless a useful indication of a direction to drive in. Both detailed and motivated enough that even a graduate student with no software development experience could understand that there were tradeoffs involved, and someone had thought hard about which side of the tradeoffs they wanted to be. I'd also point out some of Bill's advodiary entries of the time, including http://www.advogato.org/person/wnewman/diary/6.html and http://www.advogato.org/person/wnewman/diary/9.html, which perhaps illustrate that he was thinking about this kind of software maintenance all the time. ("To foil the maintenance programmer..." :-)

I think this is particularly important when the bulk of the work is done by volunteers; it helps to keep focus (I've written before about how software is only sustainable if it can stay alive in the minds of its developers).

If you had all the time in the world for a Lisp project, what would it be?

There's lots that I'd like to do, both hacking on Lisp itself and things I'd like to explore, for which I'm likely to reach for Lisp first. I'd also like to get more fluent with emacs lisp, enough to be able to quickly write the user interfaces I would like to use to various bits of software that I use every day.



Christophe

2012-03-26

Luke Gorrie (Australia - Sweeden - Switzerland)

Luke Gorrie is a proverbial hacker following his passion for programming to various places in the physical world, like Sweden or Nepal, or programming world, like Erlang, Forth, Lisp, Lua, or some other fringe language. And enjoying the process of exploration and meeting different people, while fiddling with computers, from OLPC to telecom equipment, working with world-famous technologists, including Joe Armstrong or Alan Kay, and, generally, doing whatever he likes in the programming world.

He was one of the main authors of SLIME[122] in its early days. And recently he founded a Lisp networking startup Teclo Networks, the story of which (as of fall of 2011) he told at ECLM 2011[123].

His twitter account is @lukego[124]

Tell us something interesting about yourself.

> I enjoy exploring the world. I was born and raised in Australia (Brisbane), I've lived for many years in Sweden and become a Swedish citizen, and these days I'm extremely happy to be settling myself into Switzerland. I've spent a couple of years traveling continuously with just my backpack and unicycle and no home anywhere to go back to. I've found this interesting. You can find links to my exploits on my homepage[125].
>
> I like to feel a bit out of my depth and to be "the new guy" who has a lot of catching up to do. This is why I so much enjoy learning new programming languages and visiting new programming communities so much: the feeling of having to think really hard about how to formulate even the most basic programs, just like when I was a kid, while other people do it so naturally.

What's your job? Tell us about your company.

> I'm currently starting a new project called Snabb[126]. It's too early to say very much about this yet, so I'll talk about the past.
>
> I've had a few major jobs that have shaped my thinking. I worked in each one for about 3-5 years.

[122] http://en.wikipedia.org/wiki/SLIME
[123] http://blip.tv/eclm/eclm2011-luke-gorrie-6263137
[124] http://twitter.com/lukego
[125] http://www.lukego.com/
[126] http://www.snabb.co/

The first was Bluetail AB, the first Erlang startup company. I was hired by Joe Armstrong[127] because I had enthusiasm pouring out of my ears and nostrils. I moved immediately to Stockholm in the middle of winter for this job — I was 21 years old and I'd never left Australia or seen snow before. I learned a lot of things at Bluetail[128], besides practical matters like how to walk on ice. The programmers there were all way above my level and I was routinely stunned at the stuff they had done, like when Tobbe Törnqvist mentioned in passing that he'd written his own TCP/IP stack from scratch as a hobby project and that his dial-up internet connection uses his own home-brew PPP implementation. I used to roam the corridors at night borrowing tech books from people's shelves, there must have been a thousand books in the office.

Bluetail was bought by Alteon[129], who were bought by Nortel at the same time, and become a productive little product unit in a big networking company.

Next was Synapse Mobile Networks[130]. This was an amazing experience for me: to switch from a big company to a small company and take care of everything from design to development to deployment to support by ourselves. Really getting to know customers and internalizing their needs and building the right solutions. The product is a device management system, which is a realtime database in a mobile phone network keeping track of which phone everybody is using and making sure all their services work. The whole thing was written in Erlang and BerkeleyDB, even with a from-scratch SS7 telecom networking stack in the end. I would routinely fly to a really interesting country — Kazakhstan, Jordan, Russia, Algeria, you name it — and spend a few weeks deploying the system and developing customizations on-site. Synapse was the first company in this market and they are still the market leader today. The system has by now configured approximately 1 billion actual mobile phones by sending them coded SMSes.

Synapse was also instructive in seeing how much a strong personality can shapee a company. I'm still in awe of our fearless leader Per Bergqvist. What a guy, as they say on Red Dwarf.

The most recent is Teclo Networks[131]. This is a company that I co-founded with friends from the SBCL community — mostly drinking buddies from ECLM — and friends from the telecom world. I served as CTO during the phase of finding the right problem to solve, building our series of prototypes and our real product, and finding our very first customers. I'm a Teclo-alumnus now, not actively working in the company, and: Wow, this was a really intense few years.

Teclo builds network appliances that optimize TCP traffic for cellular internet at about 20Gbps per server. The product speeds up the network, improves consistency, and

[127]https://twitter.com/joeerl

[128]https://groups.google.com/d/topic/comp.lang.functional/UYahf5fSNHI/discussion

[129]http://www.networkworld.com/news/2000/0829alteonbuy.html

[130]http://www.synap.se/

[131]http://www.teclo.net/

globally eliminates buffer-bloat. The company is currently moving forward with a lot of momentum: we have Lispers like Juho Snellman[132], Tobias Rittweiler[133], and Ties Stuij currently deploying systems in real live networks all over the world. Go Teclo :-)

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

I've mostly done Lisp hacking for my own pleasure on the side, but I've also used it quite a bit at work.

The first time was when I used Scheme at Bluetail. Specifically, I used Kawa to extend a Java application. I'm sure this raised some eyebrows amongst my colleagues but frankly I was having too much fun to really notice. Kawa is a great piece of software.

I wrote a bunch of Common Lisp networking code while reading books like TCP/IP Illustrated. This was a hobby project called Slitch and Netkit[134]. I used this slightly in my work at Nortel to replicate a DoS against the Linux kernel that was crashing some of our appliances.

Teclo is very much a Lisp-hacker shop. In the first year or so we used Lisp for absolutely everything. We've written and deployed a prototype TCP/IP implementation written in Common Lisp (SBCL), wrote network analysis tools for cross-referencing and visualizing traffic captures (often working on traces that fill up a whole disk i.e. hundreds of gigabytes), and developed all of the operation-and-maintenance infrastructure in CL. These days Teclo uses C/C++ for the main traffic handling, R for statistical analysis, and Common Lisp for the rest — mostly operation and maintenance and data visualization.

Among software projects you've participated in what's your favorite?

SLIME! This was wild fun. The project started when Eric Marsden, a CMUCL hacker, posted to #lisp his weekend hack to annotate individual S-expressions in an Emacs buffer with CMUCL compiler messages. His source files were `slim.el` and `skank.lisp`, after the Fatboy Slim song he was listening to at the time. I loved the idea so I started toying with it, renaming the files to `slime.el` and `swank.lisp` to make them easy to diff. I quickly posted my version to the cmucl-dev mailing list where Helmut Eller[135] jumped right into it too. We created a Sourceforge project and quickly snowballed from there to over 100 contributors.

[132] https://twitter.com/jesnellm

[133] http://trittweiler.blogspot.com/

[134] http://lukego.livejournal.com/4993.html

[135] https://github.com/ellerh

SLIME's mission was ultimately to replace ILISP as the standard Emacs mode for interacting with Common Lisp. ILISP worked by sending S-expressions between Emacs and Lisp via standard I/O and it had the unfortunate habit of getting stuck a lot. ILISP was also about 15 years old at that time (SLIME is catching up now at 8 years!) and really hard to work on due to the heavy use of reader conditionals in the source code, like `#+cmucl #-lucid #+sbcl #-lispworks #+acl #-acl4.2` and so on.

I really enjoyed the feeling of working on a growing and thriving open-source project. Seeing the steady stream of new names appearing on our mailing list, getting patches from dozens of people, feeling good about positive feedback we received, working hard on negative feedback right away, and so on. Twas a really productive flow.

I think that writing development tools is also really satisfying because your users are your peer group: people you really look up to and respect and sometimes drink beer with. It's a great feeling to build stuff that they like to use.

Helmut Eller and I also had a really great working style. Very often I'd do some late-night hacking and check in a bunch of new functionality, which Helmut would then read through and think about and then thoughtfully rewrite in a simpler way. Then I'd see what he'd done, think about it, and rewrite it to be simpler again. And so on. This was a really pleasant and productive way of working. He is also an absolute magician when it comes to suddenly checking in what seems like a month worth of work that he did over the weekend. (Juho Snellman has this uncanny ability too.)

I hacked on SLIME from the beginning and up to version 1.0. Then I was engulfed by an Erlang startup. Here're some posts that I fished out of the mailing list archives to give a sense of the early days:

- SLIME is born[136]
- Users![137]
- SLIME User Survey[138]
- SLIME 1.0 release party :-)[139]

What brought you to Lisp? What holds you?

My friend Darius Bacon brought me to Lisp (Scheme) by gently and persistently singing the praises of Abelson & Sussman back in the days when I was a teenager hacking Java. This book was a revelation for me: particularly the Digital Circuit Simulator. So I was a Scheme-lover for several years, but Darius also gently and persistently sang the praises of Norvig's Paradigms of AI Programming[140], which was another revelation to me, and made a Common Lisp convert of me.

[136]http://thread.gmane.org/gmane.lisp.cmucl.devel/3850
[137]http://lists.common-lisp.net/pipermail/slime-devel/2003-November/007946.html
[138]http://thread.gmane.org/gmane.lisp.slime.devel/1646
[139]http://lists.common-lisp.net/pipermail/slime-devel/2004-September/010048.html
[140]http://norvig.com/paip.html

What holds me to Lisp is the people. I love hanging out with Lisp hackers: I find that we're an unusually diverse community. How often do you attend a small conference where attendees are building nuclear defense systems, running intensive care wards, designing aeroplane engines, analysing Lute tablature, developing cancer drugs, writing FIFA's legal contracts, and designing their own microchips? Surely this describes few tech events other than Arthur & Edi's European Common Lisp Meeting :-).

What you dislike the most about Lisp?

I have a few stock rants that I'm tempted to rattle off — fragmentation, threads, GC-phobia — but honestly I doubt they are applicable these days. There have been so many improvements to SBCL and great developments like Quicklisp and so on.

I can say I'm personally disappointed about the missed opportunity for us to write Teclo's production traffic path in SBCL. Ultimately, the software has just a few microseconds' budget to process each packet, and can never spike latency by more than a millisecond. I don't know how to deliver that kind of performance in a high-level language like Lisp. So we fell back to C.

I'd also like to have embedded SBCL in the C program to take care of high-level work like slurping in configurations and spitting out statistics. But the SBCL runtime system is a bit heavyweight to make that practical. It gets in the way when you want to debug with `strace`, `gdb`, etc. So we wrote this stuff in C++ instead.

I'd have written a lot more Lisp code in recent years if I'd found good solutions to those problems. But at the end of the day it is C's niche to write lots of tiny state machines with extremely predictable performance characteristics, so I'm not especially shocked or disheartened. A language can't occupy every niche at once :-)

Describe your workflow, give some productivity tips to fellow programmers

I'm an incrementalist. I like to start from minimal running code, like `(defun program () (print "Program 0.1"))`, and move forward in a long series of very small steps. I tend to choose designs that I know are too simple or too slow, because I enjoy the feeling of hitting their limits and knowing that I didn't prematurely generalize or optimize them. Then I fix them. I'd say that I'm much influenced by watching the development of Extreme Programming on WardsWiki[141] in the 90s.

This isn't a hard and fast rule though. In Teclo I once spent a whole month writing a complex program without even trying to compile it once. This was a rewrite of the main traffic engine in C after having written a prototype in Lisp previously. This was a really

[141] http://c2.com/cgi/wiki?WardsWiki

fun way to work actually. I produced a tremendous amount of bugs in this style though and I wasn't smart enough to fix them. Christophe Rhodes did that part — don't ask me how :-).

I do have a tip for getting into "the flow". It's a simple one: make a little TODO list of some features you want to hack, and take the laptop into the park away from the internet for an hour or two until they're done. This works every time for me.

Oh, and I highly recommend printing out and reading your programs. This is the best way that I know for improving their quality. This is why I'm a bit picky about things like the 80-column rule and the layout of functions within a file. I want to be able to print programs out and read them on paper from top to bottom. I wrote a program called `pbook` to help with this — it's not a very good implementation though, with all those regexps, so I'd love if someone would make a much simpler one.

You have played around with so many languages, like Erlang, Lisp, Smalltalk & LuaJit. If you would design your own, how would it look like?

That's really hard to imagine. I'd have to find a reason that I needed a new programming language, and the details would probably follow from the problem I needed to solve.

I learn new languages mostly because I enjoy meeting new people and learning new ways of thinking. It's very seldom from any sense of dissatisfaction with previous languages I've used. My favourite languages are Common Lisp, Emacs Lisp, Forth, Erlang, Smalltalk, and C. So the best I can say is: those are the languages that I'd like to have designed.

Luke

2012-07-25

Juan José García Ripoll (Spain)

Juan José García Ripoll is a physicist and the principal developer of Embeddable Common Lisp[142] (ECL), an implementation that compiles to C instead of native code and provides a shared library, so that Lisp programs can be distributed without the Lisp runtime and be embedded in other programs. He also actively contributes to the improvement of ASDF, the foundational system definition facility (one of his posts on the topic[143]).

Juanjo is a representative of a large group of Lisp users — researchers and scientists, who are not professional programmers. Although Lisp is not generally considered alongside specialized research environments, like Matlab or R, it is quite useful in this field because of its solid mathematical foundation, coupled with truly interactive and robust environment.

Tell us something interesting about yourself.

> I do not consider myself to be an interesting character or have interesting hobbies, but my job is quite interesting and unconventional: I work on Quantum Computation, developing new ways of computing using, for instance, superconducting circuits[144], or arrays of atoms[145]. Actually I find that the fact that human technology has progressed to the point that we can trap and harness individual atoms[146] is pretty cool.

What's your job? Tell us about your organization.

> I am a physicist and work as a scientist for CSIC[147], the Spanish Research Council. It is the largest research institution in Spain and my group works on all kind of exotic things related to the implementation of quantum computers.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

> From my description above it might seem that it would be hard to use Lisp for anything, but quite the contrary: 50% of our time is spent doing simulations and programming. Quite interestingly, the programs that we develop are very short-lived. We have a

[142]http://ecls.sourceforge.net/
[143]http://tream.dreamhosters.com/tream/musings/49-lisp/76-analysis-of-existing-asdf-files
[144]http://www.bbc.co.uk/news/science-environment-12811199
[145]http://greiner.physics.harvard.edu/QGM.html
[146]http://greiner.physics.harvard.edu/Videos/hopping.mpeg
[147]http://www.csic.es/

question, write a simulation, get the data, move on. This is the reason why most people in our field use Matlab. I reached Lisp looking for an interactive programming language that was more powerful than Matlab in the fields of expressiveness, extensibility, optimization. That is how I started working on ECL, with the aim of empowering it for numerical simulations. Along the way I got distracted and more interested in the fields of language development.

Despite this I have found room for Lisp in my daily routine, from scripting all the way up to solving actual problems. For instance, in one of our research topics we had to solve huge instances of 3-SAT problems using a very basic algorithm. The fact that Common Lisp comes with bignums reduced the development time to one afternoon and the program we produced was faster than the C++ prototypes we had.

Among software projects you've participated in what's your favorite?

My favorite is ECL, but my second favorite was developing for an extinct operating system, OS/2. Back when I was at the university, OS/2 was still alive and I got some interesting projects finished, such as an X-Windows server and the port of Doom to it. It was pretty cool to learn multithreading and GUI development back then and OS/2 was pretty well thought out.

What brought you to Lisp? What holds you?

My job brought me to it, and what keeps me is the possibility of learning a lot about language implementations, compilers and making a tool that is useful for a larger community, including other projects such as Sage[148]. It is a weak link though, and sometimes personal interactions make me question whether this is useful at all.

What's the most exciting use of Lisp you had?

ECL. Really. After one year of development it was awesome to see ECL being able to build and compile itself from scratch for the first time. Also, every time I finish yet another feature, it feels really good. And finally, developing a Common Lisp implementation is a great excuse to learn a lot of things, from the subtleties of the POSIX specification, to the Unicode collation algorithms.

[148]http://www.sagemath.org/

What you dislike the most about Lisp?

The unspecified corners. Multithreading was not in the specification and implementations have evolved in somewhat random ways, with some common denominator and features (`process-kill`) which are not really well suited for existing operating systems and are nightmare to implement. I also miss a better specification of physical pathnames.

Describe your workflow, give some productivity tips to fellow programmers.

Sorry, I am a terribly disorganized developer and ECL users suffer from it. No advice here, hehe.

You are a physicist by main occupation, yet you manage to almost single-handedly support ECL. Why? and How?

Everybody needs some projects in life other than work and this one is a lifelong companion that I would like to see to the end. I had a lot of help from freelance programmers at the beginning and users cooperate with patches and bug reports, and a lot of patience for my mistakes and absences. That helps.

That does not mean I consider the current status an optimal one. I consider it a personal failure the fact that ECL does not have a larger community and a better support base. It still persists the wrong impression that ECL is a lesser implementation, covering a niche (embeddability) and inferior to other implementations which are more lispy in nature. That is not the case: we have sufficiently proven that the underlying core, C, does not prevent any feature from being implemented, performance is improving and stability, well, this is not always optimal, but it has to be blamed to the small size of the developer team, which has scarce resources for testing the code.

Juanjo

2012-06-25

John Fremlin (USA)

John Fremlin has created a couple of very performant Common Lisp programs beating on some microbenchmarks the fastest similar software written in any other language, including C: the teepeedee2[149] dynamic webserver, that managed to break the c10k record on a single core machine, and cl-irregsexp[150] regex library. Working at MSI[151] in Japan he also had written an object persistence DB for CL called manardb[152]. Besides, he writes interesting blogs[153] on topics of software optimization, programming languages and technology in general.

Tell us something interesting about yourself.

I've been to more than eighty countries; I want to go everywhere!

What's your job? Tell us about your company.

I work at Facebook on the growth team, on data-driven improvements to the sign-up flows.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

I used to at msi.co.jp. It is a Japanese consultancy based in Tokyo called originally Mathematical Systems Institute. Mr Kuroda leads the Lisp group there and I think it has hovered around five or six people over many years. He's done a great many very interesting projects for a range of companies over the years: for example a crash-test data inspection tool for a big Japanese car company, text mining, graph visualisation and so on. I worked primarily on building up a visualisation and mapping of a very large set of routers for the world's biggest telecoms, which led to the creation of manardb.

I really enjoyed working for Mr Kuroda and I'm sorry I had to leave for personal reasons. There were always many very fascinating problems around â€" with great people to discuss them and find solutions. It was a very stimulating workplace! At Facebook, I use PHP, Python, C++, Java and miscellaneous things. I think we would all be better off if we hadn't balkanised the different systems that we program for â€" and Lisp is one of the few programming languages with the flexibility to serve in all these roles.

[149] http://john.freml.in/teepeedee2-c10k
[150] http://john.freml.in/re2-benchmark
[151] http://cl-www.msi.co.jp/index.html
[152] http://john.freml.in/manardb-fast-objects-in-lisp
[153] http://john.freml.in/

What brought you to Lisp? What holds you?

My initial programming was following Michael Abrash's graphics books and building on his ideas, by doing things like runtime native code generation for drawing dynamically generated bitmaps efficiently. This is not so interesting for modern processors as they have good branch prediction but the idea of code generation stuck with me and Lisp is one of the few programing languages that makes this easy and efficient.

I appreciate the intellectual coherence of Lisp, and its sensible approach to numeric computations. In terms of using it today, I feel that Common Lisp has an advantage over many other programming languages in that it has multiple mature independent implementations. Running on multiple compilers tends to greatly increase the quality of a program in my opinion, as the code is exposed to different static analysis.

What's the most exciting use of Lisp you had?

I helped someone use Lisp for an automated trading project.

What you dislike the most about Lisp?

In trying to make efficient code one ends up fighting against the compiler and the runtime system and most of the time is spent in coming up with clever ways to circumvent and outwit both. This is not a good use of resources, and means that it usually makes more sense to start with C++.

Tell us about your approach(es) to optimizing Common Lisp code (and maybe code optimization in general)?

The most important thing is to try to hold in your head an understanding of where the program is going to spend time. Profilers can be misleading and inaccurate, and it is sometimes difficult to get representative workloads to profile. I think their main utility is in confirming that there is no sloppy mistake (in Lisp, typically, consing accidentally) that prevents you from achieving the natural performance of your approach.

Complexity analysis in terms of computation, network usage, disk accesses and memory accesses is a first step as obviously if you can improve the asymptotic usage of a bottlenecked resource, you will very likely do much better than trying to tweak some little detail. The second step is to try to characterize interactions with caches and, in Lisp, garbage collection, which is pretty tricky.

Among the software projects you've participated in what's your favorite?

I think the one I enjoyed most was an embedded H.264 decoder in 2005. This was for the VideoCore, a really wonderful CPU architecture that could deal with parallelizable problems incredibly efficiently if programmed correctly. It would have been awesome to use Lisp for it!

If you had all the time in the world for a Lisp project, what would it be?

I wish there were Lisp bridges to other runtime systems (Java, Android, Objective C, Perl, Python, C++, R, etc.) so that the libraries and tools for each could be leveraged efficiently in Lisp and vice versa. That would mean being able to call Java code and handle Java objects in Lisp, for example â€" perhaps initially by spinning up a Java implementation in a separate process running a `CL-SWANK` style interface.

I really don't think this would be that difficult and it would make a huge difference to the convenience of building programs in Common Lisp!

Describe your workflow, give some productivity tips to fellow programmers.

I use emacs and I have a bunch of elisp code that I keep meaning to publish!

John

2013-01-03

Vladimir Sedach (Russia - Canada - USA)

Vladimir Sedach is an active open-source Common Lisp developer and proponent, as well as a computing philosopher to some extent. At his carcaddar blog[154] he writes about decentralized social networks, forgotten bits of computer history and, surely, Lisp. He is the maintainer or originator of a few open-source libraries like parenscript[155] and Eager-Future2[156], and works on Vacietis[157] C-to-Lisp compiler, which he describes in more detail in the interview.

Together with Andrey Moskvitin[158] they were the driving force behind 2012 cliki[159] update effort aka cliki2[160].

His twitter is @vsedach[161]

Tell us something interesting about yourself.

> In 2012 my best friend and I rode our bicycles across the USA.

What's your job? Tell us about your company.

> Right now I work at ZestFinance in Hollywood. We're a relatively new company that's helping the underbanked receive access to credit on more reasonable terms than otherwise obtainable, by using machine learning. The only thing bad about my job is that I get so focused on programming at work that I don't have any desire to hack on Free Software projects when I come home, so a lot of my own projects are currently badly neglected.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

> I've used Lisp for new commercial projects where people trust me with the technology choices. I've also come in to work on existing Lisp projects. I have used Lisp for one-

[154] http://carcaddar.blogspot.com/
[155] http://common-lisp.net/project/parenscript/
[156] https://github.com/vsedach/Eager-Future2
[157] https://github.com/vsedach/Vacietis
[158] https://twitter.com/archimagdev
[159] http://cliki.net
[160] https://github.com/vsedach/cliki2
[161] https://twitter.com/vsedach

off tasks at "non-Lisp" companies, but I've never started a Lisp project at a company that wasn't using Lisp already. Most computer programmers tend to make technical decisions based on cliches they read about on the Internet, and unfortunately I don't know the cure for being dumb.

What brought you to Lisp? What holds you?

As a teenager I got interested in computer graphics, and came across this really cool 3d graphics program called Mirai from Nichimen Graphics. Mirai was written in Common Lisp, and I read more about Lisp in the Slashdot interview with Kent Pitman[162], and started reading SICP[163] based on Kent's recommendation. By the second chapter computer programming had finally made sense to me, and by the third I had decided to study mathematics in university.

As an aside, Mirai is directly descended from the S-Graphics software from Symbolics. Not many people know this, but Symbolics was actually the second producer of commercially available 3d computer animation software (the first was Wavefront). S-Graphics was a complete modeling, animation, and painting system, and it was all written in Lisp in the mid 1980s.

Common Lisp is still the best programming language I've come across. There is the feeling of freedom. No one is telling you what patterns or types to use. You can always just write code in a way that is most appropriate for what you're currently doing.

What's the most exciting use of Lisp you had?

Working with Daniel Gackle on Skysheet. We did some very awesome things with Parenscript.

What you dislike the most about Lisp?

No one has yet invented a good way to write "one-liners" in Lisp. I would love to replace my Unix shell with a Lisp REPL someday.

Among the software projects you've participated in what's your favorite?

Vacietis, for the sheer amount of hacks per line of code, and for the "I proved them wrong" factor. It was during the development of Vacietis that I came up with my current programming philosophy: "when stuck, do the stupidest thing possible"

[162]http://developers.slashdot.org/story/01/11/03/1726251/kent-m-pitman-answers-on-lisp-and-much-more
[163]http://mitpress.mit.edu/sicp/

Tell us about Vacietis: your vision for it, the project's progress and roadmap, what's lacking?..

Vacietis was originally intended to be a translator for a subset of C code to Common Lisp. I discussed the idea for Vacietis a few times online before starting to code, and people generally thought it wouldn't be doable. Scott Burson[164], who wrote the Zeta-C compiler for Lisp Machines in the 1980s, told me it would take at least a year of full-time of work.

As I worked on Vacietis, I realized that adding more and more of the "advanced" C functionality was actually easy. First came the C preprocessor (which is implemented as part of the Common Lisp readtable that also parses the C code), then large patches of the C standard library, and then one day Brit Butler[165] came along and sent me a one-page patch that actually used Vacietis as a stand-alone C compiler! I hadn't even realized the project had gotten to that phase. The compiler comes as the `vacietis.vcc` ASDF system as part of Vacietis.

Some big things that need to be done are struct call-by-value, pointer scaling, arguments to `main()`, some libc stuff, `setjmp`, and making VCC produce linkable Lisp "object" files (fasls) for the different implementations. I would also like to change the pointer representation to be a fixnum offset into a sparse array (right now pointers are represented by structures that look like `<offset, array>`).

Overall, I am amazed by how much progress Vacietis has made given how little time I spent on it. I am now convinced that Common Lisp is mostly a superset of C, and have a newfound appreciation for gotos and `PROG`.

If you had all the time in the world for a Lisp project, what would it be?

A single-address space Common Lisp operating system based on capabilities (Jonathan Rees wrote a dissertation[166] on doing this in Scheme in 1995) and a virtualized package system. I really like Azul Systems' hack of using the EPT/RVI virtualized page tables as a hardware read barrier for real-time garbage collection, and with a single-address space operating system I think you can do the same thing on hardware with a regular MMU.

If you follow the steps of the Viewpoints Research Institute's Fundamental New Computer Technologies project[167] and keep the system as simple as possible, it should be a surmountable amount of work to realize a somewhat working system.

[164] http://www.cliki.net/Scott%20L.%20Burson
[165] https://twitter.com/redline6561
[166] http://mumble.net/~jar/pubs/secureos/
[167] http://www.viewpointsresearch.org/html/work/ifnct.htm

Vacietis is actually the first step in the Common Lisp operating system project. I'd like to have a C runtime onto which I can port hardware drivers from OpenBSD with the minimal amount of hand coding. The way hardware drivers are written in OpenBSD (a set of patterns it borrows from NetBSD, where they originated and are colloquially known as "bus_dma") is very pleasant. Theo de Raadt was also the first well-known Free Software operating system developer to take a position against binary hardware drivers, and I respect him tremendously for that. Linus Torvalds doesn't care, and as a result it is now impossible to get drivers for any of your ARM devices, even though 99% of them run Linux. OpenBSD will continue to have great, stable Free Software device drivers that are easily portable for different hardware architectures and to the other BSD-derived operating systems. I hope OpenBSD's model of Free Software hardware drivers catches on more widely.

Describe your workflow, give some productivity tips to fellow programmers.

I tend to design software systems in advance only at the level of the domain. I find that programmers who try to come up with APIs or draw class diagrams ahead of starting to code tend to write very poor code.

I rely heavily on interactive programming for doing mundane tasks, and since most programming time is spent dealing with mundane things, I spend most of my time in the REPL. Fortunately the Common Lisp REPL is the best programming environment I have come across. Things like the Rails console don't even come close. The only comparable environment is the Unix shell.

It's also very easy to turn Common Lisp REPL code into unit tests, which I tend to do a lot. That is something that's very hard to do with object-oriented code, which is why idiotic things like dependency injection and Test-Driven Development have to be invented.

For difficult problems, it's always better to step back from the keyboard and just think about your code. Hacking around in the debugger is usually a waste of time (the only exception is when you're having data structure issues, but then technically you're in the object inspector). I mostly tend to debug with print statements.

The best productivity tip I've come across is the "Seinfeld technique" that I learned about from reading Hacker News. It involves doing something, no matter how small, on your project every single consecutive day, without any gaps or interruptions. Apparently Jerry Sienfeld uses a big calendar and tries to mark up large "runs" of consecutive days on it. That really keeps you focused.

One thing I try to focus on in my code is giving unambiguous, unique (i.e., easily `grep`-able) names to symbols. That makes it easy to go back to old code and quickly figure out what is going on. Common words like “status,” “mapping”, etc. make terrible names unless you qualify them.

Vladimir

2013-03-05

Marc Battyani (France - USA)

Marc Battyani is one of the people who put Lisp at the foundation of their business, and he doesn't seem to regret that decision. And the business itself is exploring very interesting aspects of high-performance and reconfigurable computing. Besides, he has a notable open-source contribution with cl-pdf[168]/cl-typesetting[169] libraries. He elaborates on all these and more in much detail in the interview.

Tell us something interesting about yourself.

> I think maybe the most unusual things I do is that I work on very different application domains which are even sometimes completely at opposite extremes both in electronics and software. For instance form ultra low power smart sensors based on $1 microprocessors which will run continuously for 5+ years only powered by a small coin battery up to the world's lowest latency supercomputers based on FPGA costing thousands of $ per chip. On the software side I use programming languages ranging from the lowest level languages like assembly or even below with VHDL up to really high level languages like Common Lisp.
>
> I've always enjoyed mixing electronics design and higher level computer science and all that diversity probably gives me a different and original view on say computing and programming in general.

What's your job? Tell us about your company.

> I'm the CTO of NovaSparks a startup I founded in 2008 to make ultra-low latency FPGA based supercomputers for the financial markets. BTW These things are really incredibly fast. For instance on 10Gb/s Ethernet market data packets coming from exchanges like the NASDAQ we process the IP/UDP/multicast network stack, extract the messages from the packets, parse/decode/filter/normalize those messages, maintain the indexed order book data structures, aggregate the price levels per stock, generate output messages and finally send them to a server through PCI-express or 10Gb/s Ethernet network stacks. The nice thing is that we do all that fully pipelined at a rate of one message every 12 nanoseconds! To have an idea of how fast it is, in 12 ns the light will only travel 3.6 meters (11.8 ft). Another way to view this performance is that the system can process 83 Millions of financial messages per second without any queuing.

[168] https://github.com/mbattyani/cl-pdf
[169] https://github.com/mbattyani/cl-typesetting

As an aside it is interesting to note that the Domain Specific Language Compilers and various other tools written in Common Lisp have been key enabling factors for the creation of NovaSparks.

I'm also the CEO and CTO of Fractal Concept which is the company where we were developing that technology before starting NovaSparks but as NovaSparks has been using more than 100% of my time for the last years Fractal Concept has been less active with only the development and maintenance of the smart sensors going on.

Do you use Lisp at work? If yes, how you've made it happen? If not, why?

I've always used Common Lisp for most of my work and even when I need to use other programming languages (like VHDL for NovaSpark's FPGAs or javascript for web stuff) I generally use Common Lisp at least for a lot of related tasks like prototyping, designing and testing algorithms, extracting statistics, performing simulations, generating test data, analysing test runs.

The next step is very often to generate some or all the code in other languages by designing various domain specific languages (DSL) which will take care of the tedious aspects of programming in less powerful languages. I really like it when from a few 100s of lines written in a easy to use high level DSL we generate 10000 to 60000+ lines of very low level VHDL code saving months of development.

About that point I think I would even say that I'm mostly doing Language Oriented Programming by trying to abstract the domain specific knowledge into various domain specific languages with very different syntaxes like s-expressions, C like or other languages or even sometimes adding to the mix some GUI or data-based inputs. Then those DSLs would generate most if not all the application code in whatever language is needed be it VHDL, asm, C, javascript or Common Lisp.

What brought you to Lisp? What holds you?

A friend of mine gave me a version of Le_Lisp[170] a version of Lisp used in the 80's by the French universities and engineering schools to teach high level programming concepts. At that time I was mostly programming in Z80 assembly language on a TRS80 and the straight jump from ASM to Lisp was quite a shock and an eye opener.

Since then I've used Common Lisp for countless projects ranging from a few hours of work to multi-years ones and I still find it awesome. Where else can you find a language providing such powerful and multi-paradigms features like CLOS, generic functions, the MOP, macros, closures, s-expressions, lambdas, an interactive REPL (Read Eval Print

[170]http://www.softwarepreservation.org/projects/LISP/le_lisp/

Loop), native compilation of applications, on the fly native compilation of generated code, the condition system, interactive live programming, real time live debugging of running software and more!

What's the most exciting use of Lisp you had?

I've got so many of them than picking only one would be difficult so here are a few of them:

- softscan[171]: real time driving and data acquisition of automated non-destructive testing installations with real time 3D display (a first in 1995)
- my web application framework: Automatic generation of really complex web applications (the whole stuff from the HTML/javascript front-end to the server back-end). It has been fully Ajax-based with dynamic modification of the displayed pages without reloading the page. This seems obvious nowadays but it was also a first in 2000 and IIRC the term "Ajax" was only invented 5 years later.
- hpcc: The awesome DSL to VHDL compiler. In many aspects VHDL is the complete opposite of Common Lisp. It's a hardware description language used to program FPGA and it deals with the lowest possible programming level with data types like signals, clocks, and bits. Programming at that level is really tedious, time consuming and verbose so it's really a relief to be able to generate tens of thousands of lines of highly optimized VHDL code from just a few hundred lines of some high level Domain Specific Language.
- cl-pdf/cl-typesetting: Being able to generate the first PDF files from scratch in only 107 lines of Common Lisp was an Haha moment.

What you dislike the most about Lisp?

The language itself is somewhat good enough and anyway Common Lisp makes it really easy to change most of itself to add the new and cool stuff or ideas of the day.

In fact what I dislike about Lisp is outside the language and more related to the (mis)perception that people have about it. Having almost everytime to justify its use and sometimes even to fight to be able to use it is somewhat annoying and tiresome.

Of course, Lisp is not for everybody but this is not a reason to have nobody using it. In fact I view Lisp as some kind of amplifier which will give awesome things when used by brilliant developers but will end up giving an incredible mess when used by people without any clues about what they are doing. That's one aspect that makes Lisp very different from other languages which are especially designed to try to normalize and average what people can do with them.

[171]http://www.fractalconcept.com/asp/softscan

Describe your workflow, give some productivity tips to fellow programmers.

I obviously use emacs and all the nice stuff that work with it like slime[172], org-mode[173], magit[174] and lots of other packages. I switched from subversion (svn) to git for all my projects and I now use git mostly form emacs with magit.

In general I have several instances of Lispworks running as standalone apps on my laptop but connected to Slime through `M-x slime-connect` rather than being started by slime. I do a lot of exploratory programming and interactively play with the code while I write it and if I have to optimize some code I very often use `#'disassemble`.

About my Lisp coding style I really do like generic functions, CLOS and the MOP. As mentioned earlier I try to generate as much of the code as possible by using macros in simple cases and full blown DSLs in more complex ones. Sometimes I do not dislike making some premature optimizations when I know that speed will be important for some project. BTW when speed matters I like to generate and compile code on the fly when it's useful (and possible) and try to generate really optimized code. It's always cool when you have Common Lisp applications running much faster than C++ ones.

I also refactor a lot. Generally when I start on some new stuff I try to make a first version that works well enough but then once it's done or generally after some time I have some cool ideas to make something much better. At that point I'm really happy that Common Lisp makes refactoring very easy thanks to optional and key args, macros and generic functions with multiple dispatch because all that enables me to make even very deep modifications very easily. I really find those Common Lisp features make software very resilient to code modifications.

What are the advantages of using Lisp in the HPC field? What are the drawbacks? Are you happy with your technology choice?

The HPC field is huge and I can only talk about the small corner of it I know which is the ultra-low latency processing of vast amounts of data we do in NovaSparks. For that the various DSL compilers generating VHDL code have really been a key enabling factor. Programming FPGAs is notoriously difficult and time consuming and this is the major factor limiting the use of reconfigurable computing in the HPC field. The use of those DSL compilers has made it possible to use those FPGA on a more practical basis for processing financial data.

BTW I'm looking for other HPC domains in which those DSL VHDL compilers could be used so feel free to contact me if you have some ideas.

[172] http://common-lisp.net/project/slime/
[173] http://orgmode.org
[174] http://magit.github.io/magit/

If you had all the time in the world for a Lisp project, what would it be?

Again that's a difficult choice. Speaking about pure Lisp projects I've been willing for a long time to clean up and modernize my web app framework before releasing it as open-source so maybe that could be a good project to start.

Otherwise as mentioned above, I'm looking for possible applications and opportunities of leveraging the DSL ⇒ hardware compilers and the FPGAs around some Big Data processing. I have a few ideas but nothing specific for now.

Anything else I forgot to ask?

A conclusion? (Everybody wants a conclusion.) So here is mine:

Common Lisp is Awesome! It is much easier nowadays to use it thanks to all the projects and libraries that Quicklisp makes available. If you do not know Common Lisp then learn it and this will make you a better programmer anyway even if you do not use it directly after.

Marc

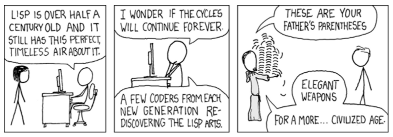
2013-06-12

Afterword

I hope you've enjoyed reading these interviews and have gained something of them. I surely did.

Some events occurred in the professional lives of many of the interviewees even in the short period since the interviews were conducted. FarÃ© had left ITA, yet he's produced ASDF 3, a lot of interesting things of which you can read in his presentation[175]. Zach Beane is now part of Clozure Associates[176] where he hacks on Lisp full-time and continues to improve Quicklisp. Marijn Haverbeke had a hugely successful Indiegogo project Tern[177] to improve JavaScript editing experience. Peter Seibel went to work for twitter. The next ECLM took place in Madrid[178].

The Lisp ecosystem continues its development little by little, yet consistently: new interesting implementations are arriving, new community resources are being developed, and more projects get started or mature. And, last but not least, hopefully, a new generation of Lisp hackers is growing…


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

(c) http://xkcd.com/297/

175 https://github.com/fare/asdf3-2013/blob/master/els-slides.org

176 http://www.clozure.com/index.html

177 http://www.indiegogo.com/projects/tern-intelligent-javascript-editing?browse_v=new

178 http://weitz.de/eclm2013/