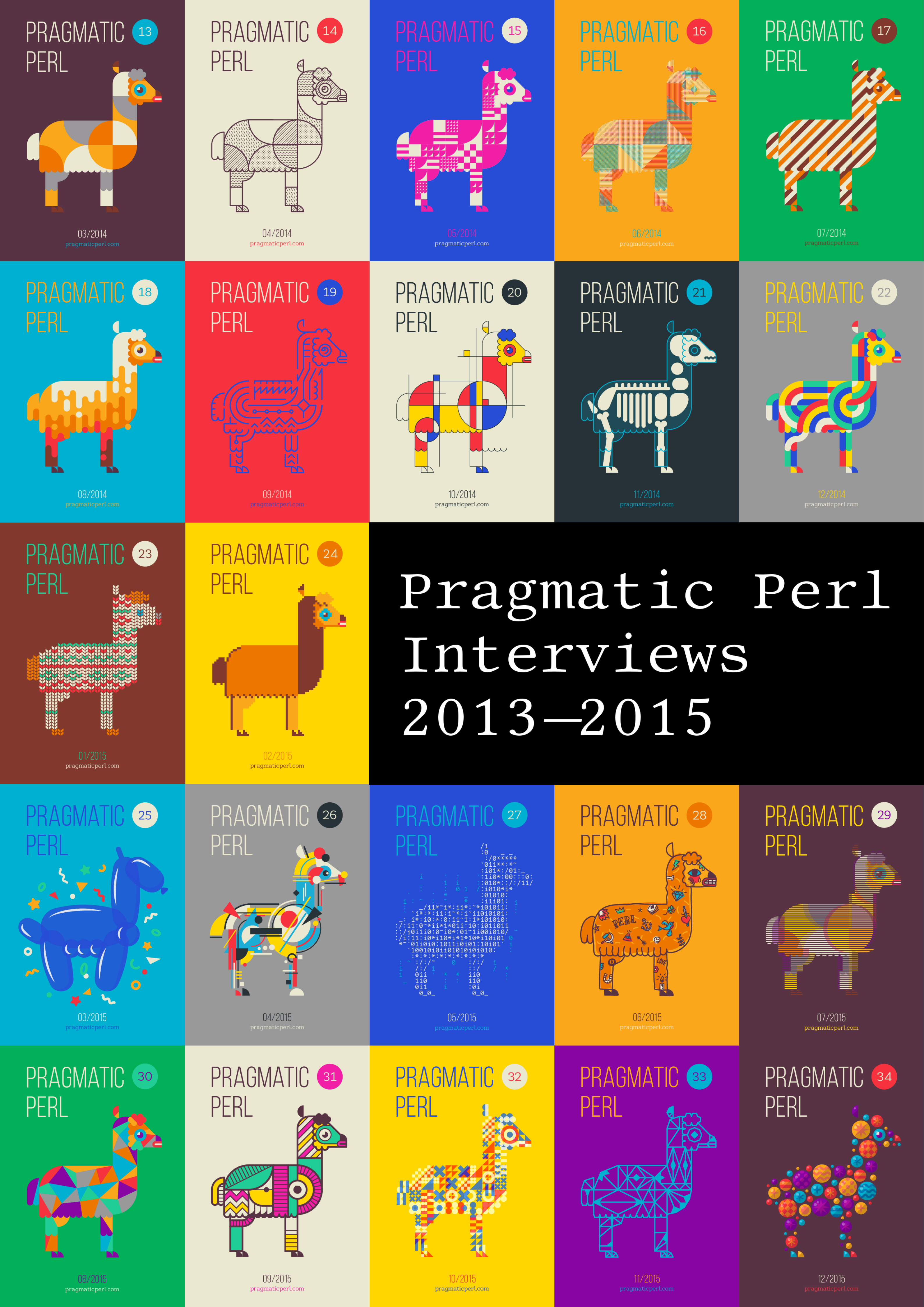
PRAGMATIC PERL 13 03/2014 pragmaticperl.com
PRAGMATIC PERL 14 04/2014 pragmaticperl.com
PRAGMATIC PERL 15 05/2014 pragmaticperl.com
PRAGMATIC PERL 16 06/2014 pragmaticperl.com
PRAGMATIC PERL 17 07/2014 pragmaticperl.com
PRAGMATIC PERL 18 08/2014 pragmaticperl.com
PRAGMATIC PERL 19 09/2014 pragmaticperl.com
PRAGMATIC PERL 20 10/2014 pragmaticperl.com
PRAGMATIC PERL 21 11/2014 pragmaticperl.com
PRAGMATIC PERL 22 12/2014 pragmaticperl.com
PRAGMATIC PERL 23 01/2015 pragmaticperl.com
PRAGMATIC PERL 24 02/2015 pragmaticperl.com
PRAGMATIC PERL 25 03/2015 pragmaticperl.com
PRAGMATIC PERL 26 04/2015 pragmaticperl.com
PRAGMATIC PERL 27 05/2015 pragmaticperl.com
PRAGMATIC PERL 28 06/2015 pragmaticperl.com
PRAGMATIC PERL 29 07/2015 pragmaticperl.com
PRAGMATIC PERL 30 08/2015 pragmaticperl.com
PRAGMATIC PERL 31 09/2015 pragmaticperl.com
PRAGMATIC PERL 32 10/2015 pragmaticperl.com
PRAGMATIC PERL 33 11/2015 pragmaticperl.com
PRAGMATIC PERL 34 12/2015 pragmaticperl.com

Pragmatic Perl Interviews 2013—2015

Pragmatic Perl Interviews

pragmaticperl.com

2013—2015

Editor and interviewer: Viacheslav Tykhanovskyi

Covers: Marko Ivanyk

Revision: 2018-03-02 11:22

Contents

1 Preface

Hello there! You have downloaded a compilation of interviews done with Perl programmers in Pragmatic Perl journal from 2013 to 2015. Since the journal itself is in Russian there was a lot of interest in publishing some of the content in English. Even though the interviews are already available at the website, they are hard to find for non-Russian speakers.

The Pragmatic Perl journal was started as a marketing campaign in order to attract more visitors to the YAPC Europe Conference in Kiev, Ukraine in 2013. Since the journal got some attention (in terms of Perl community) it made sense to keep going.

Unfortunately finding authors and editing articles takes a lot of time and finally in December of 2015 the journal stopped publishing new issues. It is hoped that this will change, but who knows when.

So enjoy the reading and if you have any questions, ideas or just want to say hi drop me a line at editor@pragmaticperl.com.

■ *Viacheslav Tykhanovskyi*

2 Alexis Sukrieh (April 2013)

How and when did you learn to program?

I started learning how to program when I was in high school, it was on my calculator, a Casio I think, there was a little BASIC language embedded in the system and I started playing with it. I think my first program was a little text-based lottery game. That's pretty much where I learned if-then-else structures and while loops.

What editor do you use?

Vim! I can't use anything else to write code. Every time I try another editor, I go back to Vim in less than 10 minutes, I suppose my brain is cabled with Vim.

When and how have you been introduced to Perl?

It was during my very first job (almost 13 years ago). I was asked to build a small statistics tool that should fetch a bunch of MySQL data and produce nice reports in PDF via LaTeX. They introduced me to the Perl language, giving me the O'Reilly "Introduction to Perl" book and of course, the "Camel Book". And that was it. After a couple of days playing with it, I was in love with the language. So easy to learn, so intuitive, I never stopped using Perl ever since.

What are other programming languages you enjoy working with?

I've been working with Ruby for a while, when I was working for Yoolink http://www.yoolinkpro.com (a company that I co-founded and where I was the CTO). The webapp we built is based on Ruby on Rails. That was very hype at that time (2008) and we wanted to experiment that new shiny tool. I've enjoyed a lot working with Rails at that time but never lost the pleasure of working with Perl.

I guess I've never found the freedom I have with Perl. It's like I know that I can code virtually anything with Perl, even extending its syntax with … a DSL. You see me coming, right?

What do you think is the most strongest Perl advantage?

Oh, I'm sorry, I've already answered that in the previous one! So yes, to me, the killer feature of Perl is the freedom it gives. You can really extend it as much as you like. It's also its drawback, because this freedom comes at the price of a very … "tolerant" syntax and lots of programmers prefer strictness (like Python offers for instance). That's understandable, I think both philosophies make their point, and it's good we have both styles out there. The Perl way really fits my needs.

Also, of course, I could not stop here if we speak about Perl's strongest advantages… CPAN, obviously. Even the worst language on Earth would be awesome if it had CPAN. This is why you come to Perl in the first place, I think. Because you know any problem you can face can be solved with a good use of appropriate CPAN modules.

And if we speak about CPAN, we speak about the Perl community. Many brilliant people are contributing to the language everyday, very smart ideas rise (`Moose` being one of the most impressive and Perl-world-changing one), and the tools we have to maintain, process and test our distributions are very powerful. The test smokers, the MetaCPAN, cpanminus, all that goodness contributes to the pleasure of working with Perl.

What do you think is the most important feature of the languages of the future?

Wow, that's a tricky question! Let me see… If I could snap my fingers and magically have the ultimate language of the future, what would it be? Hmm, I think it would have a syntax very close to a natural language. A syntax that would let the programmer "describe" (rather than program) the data he needs to work on and the behavior of that data (if we can say).

Maybe we could write the specification of our program in plain English (with of course a lot of conventions to respect) and then submit that description to that magical tool. It would then compile the document into a program and a test. We would be able to review the test and move from there. Something like that, I suppose.

I have the feeling that whenever we speak about the future of computer science we speak about a world where the machine gets better and better at understanding us, the human beings. So the languages of the future would definitely be something like that.

What drives you in being so active in Perl community and open source in general?

Well, firstly, I must be honest: I'm not as active in the Perl or open source community as I used to be. At my most "active" time, I was developing and maintaining

Backup Manager, contributing to Debian (I've been a Debian developer for three years) and - of course - contributing to CPAN. Sadly enough my personal and professional life don't allow me to contribute as much anymore. So I've focused all the "free software" time that I have on `Dancer`.

What drives me into that? Very good question. I suppose the pleasure of writing code for the only purpose of ... writing code. Free Software is the only place where you can find that: whenever money comes into play, the beauty of the code itself is altered, because of external priorities. Here, time does not matter. The only thing that matters is what you produce. From an intellectual point of view that is very satisfying, and very encouraging.

It's also one of the best way that I know to enhance your skills at programming: writing code in a free software environment will eventually attract a lot of experienced eyes on your lines of code. Eyes that are only motivated by the appropriate way of solving a problem, nothing else. That's why when I do job interviews I give a lot of credit to applicants who are open source contributors, for that very reason. It teaches you humility, a lot.

Writing, releasing and maintaining free software also makes you enhance a lot of skills that are different from purely programming. You need to market your product, to provide support to your users, to document the code and the product, to handle bugs... it really takes you to another level as a programmer, that's for sure and it is very valuable. It enlarges your vision as a software developer, you really see the big picture then.

Also, eventually it can make you answer a friendly interview, and that may be the real reason why you want to be involved in free software! Narcissism!

`Dancer` is no doubt one of the most popular Perl web frameworks. When and why did you start the `Dancer` project? Why do you think the project became so popular and attracted so many contributors?

It's a very hard question to answer. I can explain what I think contributed to that popularity but there is a lot of unknown there.

First of all, `Dancer` filled a gap. Back in summer 2009, there was nothing similar to Sinatra http://www.winatrarb.com on CPAN: a complete micro-framework that provides a feature-rich DSL for writing a webapp. That's a first point, `Dancer` introduced a new way of solving the "web development" problem with Perl.

Then there is its spirit: be as intuitive as possible. This is why you don't have $self in the route handlers, that's a deliberate choice. Everything that can be removed from the syntax should be. Less noise as possible. This is what brings at the end of the day that great feeling that developing a webapp with `Dancer` is easy, lightweight and fun. Most of the users say they love `Dancer` because it doesn't force them to do things as the framework wants, it doesn't get in their way. That's because of that very thin and intuitive layer that is the DSL.

And finally, the fact that I've always tried to promote community efforts contributed to attract more and more contributors, I think.

Very soon I granted commit bits to other developers, this rapidly led to the "core team idea and helped a lot the maintenance and created a lot of energy around the project.

Look at the changelog of `Dancer`, and count the names… That's probably one of the most satisfying feeling: seeing so many people joining the effort you've started. It must be that something was not too bad in the idea of `Dancer` in the first place!

Why do you think releasing and maintaining `Dancer` and `Dancer` 2 at the same time is a good idea?

Because it's actually two different ways of implementing the `Dancer` concept, and the design changes are so drastic it's not possible to provide a complete backward compatibility with all the existing `Dancer` 1 ecosystem. I've explained in details that decision on my blog: http://blog.sukria.net/2013/02/18/dancer-1-and-dancer-2-what-were-going-to-do/ The conclusion of that blog-entry summarizes the decision:

> There is out there a lot of real-life D1 applications. Most of them use engines that won't work transparently under D2. Also the app-scoping can break things with some applications, if the app is decoupled in many packages.
>
> For that reason, I'm going to release `Dancer` 2 in its own namespace, and that will bring a lot of energy in both projects:
>
> `Dancer` 1 is at the same time unfrozen, development can continue there (as long as the features added are not something that would lead to a new `Dancer` 2!) `Dancer` 2 is released on CPAN and its ecosystem can rise

> `Dancer` 1 users are happy `Dancer` 2 users are happy Migrating an app becomes something under control."

There have been several widespread attempts to bad-mouth the project. What helped you to face that? Why do you think this happens in open source world and how to deal with it?

Haha, those times are over now. But indeed the early days/months of `Dancer` were surrounded by passions and warfare! At the beginning, there was this misunderstanding about the origin of `Dancer` vs `Mojolicious::Lite`, as both projects were released almost at the same time, it created confusion. This confusion led to create "sides" and exposed `Dancer` as a competitor for Mojo (or the other way around), which was in fact completely wrong: `Mojolicious` is a great project with a tons of brilliant features. It's a complete suite for web development with a different philosophy (which is good). `Dancer` addresses another field, it has another vision: it's to web development what Perl is to programming: a minimal and extensible set of keywords to describe a solution to a problem. At the beginning, none of us (in both sides) realized that we were playing in different fields (myself included, I admit it). So we all were a bit childish and contributed to feed that trolls army!

In the end, there was even an individual who spent time polluting anonymously our CPAN ratings or who was flooding our Hacker News announcements, even posing as members of the Sinatra community to say how much they hated `Dancer`! The only result was this official statement on Sinatra's blog: Sinatra Loves `Dancer` http://www.sinatrarb.com/2011/07/21/sinatra-loves-dancer.html !

So in the end, you realize that the more harm you try to do online to a project, the more you give it strength, because it always comes back to the project as positive energy at the end of the day. Even badly intended, a spotlight is still light.

When all this happened I found the energy of continuing my work for a simple reason: the positive feedback of all the users we have, take a look at our testimonials page, it's really a booster: http://perldancer.org/testimonials I thought: if what I did is so wrong with `Dancer`, then why are there so many excited developers about it? Some of them being well-known and respected Perl hackers? I must have done something good somewhere, it can't be that bad!

Why does this kind of things happen in the open source world? Well, for sure because the way we communicate online is completely ... cold. There are no feelings in an email, no way to understand irony or being able to see that someone is hurt or amused. That's why getting upset by email is a very bad idea. Your words can be interpreted in hundred ways. This is why, I think, hackers can fight so hard when they disagree online!

But at the end of the day, you need to realize it's just about lines of code! It's not a big deal!

Where do you work right now? Does your company use Perl? Do you think it's important to support the Perl language and community on the business level?

I work at Weborama http://www.weborama.com one of the European leader (and we're in Russia since last summer by the way!) in the online advertising industry. I'm in charge of the R&D team and I work with very talented people, I'm very happy to be working with that team. We tried very hard to create our platform with the full power of Perl. Not only the language but also the "culture": we use CPAN-Mini to create our own private mirrors that host our in-house modules (in the Weborama namespace). Everything we code here is a CPAN distribution, written in Modern Perl (`Moose/Moo/Dancer`...) and is deployed in a perlbrew. We have mimicked the real-world CPAN to benefit of the whole tool chain: unit tests, smokers, cpanminus as our deployment tool... We use `Perl::Critic` plugged into Git hooks to make sure everyone respect our coding standards. Well you see the point, everything is based on the Perl way here. That's a really great pleasure to be able to work like that, and it helps us a lot to provide very high-quality software to the company.

Also we try to attract as much as we can talented Perl hackers, we recently posted a job offer on Linkedin and on http://jobs.perl.org and we got many interesting applications, from all over the world, people willing to relocate to Paris to join our team. That's really exciting to work in that environment! OK, I'm biased! I'm in charge of that team so I won't say it's a bad one, huh! But I think if you ask some of my colleagues they won't say the opposite (I hope!).

I do think it's important to support the Perl language on the business level, and we do at Weborama. We've been sponsoring Perl events for some time now, French Perl Workshops, YAPC::EU, or others. We also provide help to our employees who wish to attend such events like sponsoring their travel/hosting expenses or letting them go there without using a day off.

Promoting the Perl community is very valuable for a company that work with it, that's just good sense.

Should we encourage young people to learn Perl right now?

I think so. Perl is a language that is very easy to learn and with a huge potential thanks to its ecosystem and community. Working with Perl can be very efficient and for that reason I really think the more young Perl developers we have the better. Any effort that can attract new developers to the community is worth a try.

Do you use `Mojolicious`?

Of course! All my webapps are written with `Mojolicious`. I could not think of any other way of writing a webapp.

…

Did you remember when I said earlier it was hard to catch irony in an email? More seriously, I've not used `Mojolicious` but I've read part of the code when I started `Dancer` 2. I wanted to see how things were done here and there and that's when I really realized that `Dancer` is clearly not the same thing as `Mojolicious`::Lite. Lite is a little DSL over a huge MVC framework. `Dancer` is a complete feature-rich DSL. It's not exactly the same thing. Again, I really think both approaches are good.

So no, I don't use `Mojolicious` but I think it's a great project!

So you think you can dance?

I think my Moonwalk is not so bad! And no, I won't send you a video, you need to trust my words.

Do you know any working in production big projects written with `Dancer`?

First of all, at Weborama, our adserver tool (Weborama Campaign Manager) provides an API service that serves the web interface and third-party clients. This API webservice is written with `Dancer` (version 1) and is working like a charm. We also have released one of our last product recently with `Dancer` 2. Outside of Weborama, there are many "real-world" companies that use `Dancer`, for instance Moonfruit.com is proudly powered by `Dancer`. I've also heard of Novell who has released a deployment tool for servers called Baracus http://baracus-project.org/Site/Baracus.html, that tool has an embedded ReST API service that is built with `Dancer` as well. You can also take a look at this page on the website for more "real-world" users: http://perldancer.org/dancefloor

Will you come to Kiev for YAPC::EU?

I don't know yet, it depends on my personal agenda, who knows!

Did you leave `Dancer` 1 for the community or do you still participate?

`Dancer` 1 is now maintained by Yanick Champoux but it doesn't mean I'm out of the discussions. I'm not far from it even if all my energy is dedicated to `Dancer` 2.

Thanks for that interview, use Perl and dance!

3 Sawyer X (May 2013)

How and when did you learn to program?

When I was in junior high school there was a programming course offered off school hours. I joined it and learned some basic programming. I didn't really study with the class, I just had fun by myself. Then in the 10th grade I went on computer-oriented studies at school. We studied Assembly, C and Pascal. I actually didn't enjoy learning at school so I was behind the class. At some point I went and bought "A Book on C" and learned by myself instead of in class. When everyone else started the high school project I had already finished it. While everyone wrote a small snake game (we received a basic DOS graphics library to use), I wrote an encryption algorithm implementation cross-compiled for Windows and GNU/Linux with command line interfaces (I had used ncurses on GNU/Linux, specifically).

What editor do you use?

I currently use Vim without any plugins or special features. except its excellent syntax highlighting and a few comfortable little tweaks. I tried several plugins but eventually removed them. I find IDE configurations confusing and disruptive. When I need to hack some Perl on Windows, I use Padre, the Perl IDE. I don't really give a shit about editor/IDE wars. Use what you find comfortable, and hopefully try out new things once in a while.

When and how have you been introduced to Perl?

When I was in high school I met a hacker at hardcore punk shows who programmed in Perl. He was really cool and suggested I give it a shot, so I tried it and had a blast. Been using it ever since!

What are other programming languages you enjoy working with?

I used several languages throughout the years (notably Assembly, C, C++, Ruby and Python) but honestly, the only language I really enjoyed working with is Perl. C was fun too, but from a different direction, I guess. Perl definitely takes the cake... err.. or onion.

What do you think is the most strongest Perl advantage?

Technically-wise? Elasticity. Perl is not just flexible, it's elastic. You can move walls around, you can bend them, it's like the fucking matrix! Devel::Declare is just a reminder that literally anything can be accomplished in Perl. You are mostly limited by your own imagination.

I think what draws me and developers like me to Perl is the easiness of accomplishing things in Perl, since it just bends the world to your point of view, whatever it may be. So, if you can speak it, you can write it in Perl!

What do you think is the most important feature of the languages of the future?

Parallelism and community, hands down. At least to me.

Speed of execution is important, surely, but proper parallel execution seems to me like it's becoming more and more important. I reckon that's why Perl 6 has put so much weight on that.

Still, as much as parallel execution and speed are important, there is a limit to how involved you are when all you have is syntax. I mean, having a great language is... great, but having a community around it gives you a sense of belonging, which is one of the most basic instincts and desires we have as animals. We want to belong, to be part of something, to create for ourselves, for others, with others, and see it blossom and take shape. We want to exist and create existence. Community is exactly that. If we want a language that is sustainable beyond its technical capabilities, we need to have a good community, to make it more than just a logistical assortment of characters. It needs to be about people.

What drives you in being so active in Perl community and open source in general?

Several things do:

1. I like creating. The first time I picked up a guitar, I didn't learn a song, I tried writing my own. I wanted to create something new. Open source and free software promote that idea. They say "let's make something!" - Perl is centered around that. Perl always says "don't be afraid to try, don't be afraid to fail".
2. I like complaining about things and fixing them. I've become very good at pointing at things and saying "that sucks", and I'm improving at trying to fix

things that suck. Open and free code allows you to do just that. It pushes you to get involved. Perl is a tremendous example of getting people involved. If you look at how many Perl programmers are working on more than one or two or ten or more projects at the same time, it's quite stunning. You get people involved in over dozens of projects throughout a single year. It's absolutely crazy. Look at rafl! He's everywhere!

3. I like cooperating with people. If there's anything more motivating, it's the ability to work with others. I get to cooperate with the smartest people I've ever met. Perl has some of the greatest minds out there. The vast majority of them are pretty fucking awesome people and I'm thrilled to have met them and be considered a friend to at least some of them. There's generally something very exhilarating about working with others towards a common goal. It fills you with purpose and satisfaction.

You seem to write lots of async code. Why did you choose `AnyEvent` while everybody else bashes it?

I actually started with `POE`. When I tried to wrap my head around it, a long time ago, I went to the IRC channel and asked. People tried to get me to write an event loop to understand it better. This one person, after an hour of me doing random things, asked me "what don't you understand exactly?" He worked with me for a good half hour explaining `POE`, and helped me understand it. I later found out it was Rocco Caputo, the creator of `POE`.

I decided to try `AnyEvent` because of its slimmer interface and better speed. I still like `POE` (and Reflex looks impressive) but it's a bit too verbose for me. I will probably get to work on some Reflex stuff in the future though, and I'm excited about that.

I haven't noticed people bashing `AnyEvent` as much as expressing disdain for the way the author (Marc) has occasionally expressed himself and at least one decision to break a module that he considers misusing the `AnyEvent` API. I'd rather not get into it too much, except that I agree on some things and disagree on others. Besides, there's a lot I don't know about it.

The only thing that troubles me is that `AnyEvent` didn't pick up as a community. I hope that will change in the future.

You've joined the Dancer project quite early. Why do you think it attracted you?

Few things: how small it was, how thin the interface and DSL were and how warm the IRC channel was. Back when I started, there were three or four people in the channel, if I recall correctly.

When I wrote my first program with `Dancer`, I thought it couldn't handle `CGI` (I didn't even grep `PSGI` yet) and I wrote a blog post saying "too bad it can't, it looks kind of cool". That evening I went to eat hummus with my girlfriend and spent a few minutes checking online if someone had tried `Dancer` with `CGI`. I saw a post by someone saying "Sawyer wrote about `Dancer` and that it doesn't support `CGI`, but it does. This is how you should configure it!" — that was Alexis Sukrieh, who wrote `Dancer`. I was pretty amazed that he saw my post and had written an entire post back. It was so cool. That's when I joined the IRC channel.

The ease of helping out in `Dancer` and working on it with others was very compelling. You were a part of it as soon as you stepped in, and that just made me feel so wonderful working on it. People in the community are kind and thoughtful, and you always feel like investing time in `Dancer` is worthwhile for the `Dancer` users community and for the general Perl community.

Tell us about your role in TelAviv.pm and group in general. Do you think community meetings are important?

There had been a few organizers to Israel.pm, prominently Shlomi Fish and Gabor Szabo. After Shlomi decided to step down from organizing the meetings, I decided to try and organize it and boost the attendance. I called it TelAviv.pm because we met at the Tel Aviv university. Pretty quickly though we moved it to a place in Ramat Gan, which is 20 minutes from Tel Aviv but has its own municipality. I still kept the name anyway, since it's like a part of the larger Tel Aviv area (which is silly, since Tel Aviv is really small to begin with). I set up a crappy website (later revamped to a good website) and contacted people to give talks and gave quite a few myself as well. For the past few years I've organized the meetings, occasionally assisted by Shlomi or Gabor, who are both still active members in the community. The past few months we've decided to have each meeting organized by someone else. It was very successful. We have also successfully organized two fantastic Israeli Perl Workshops.

I think community meetings are extremely important! By going to meetings you can:

- Improve your knowledge of the language and its third-party modules
- Learn tricks of the trade
- Meet potential co-workers and employers
- Practice giving talks
- Making friends and possible cooperators
- Get free help with work or ideas you had
- Have a good time

It sometimes seems like it's not worth it, but once you start attending, let your guard down and meet new people, you begin to realize how incredible it is. It's like a free drug that makes you feel better, become smarter and better at your job without any side-effects except loss of time. :)

You tweet a lot about your git experience. What is so special about this vcs?

Git is one of the best tools for any creator, whether a writer, a programmer or even a graphics artist. Sure, there are other tools, but they all suck in comparison. Git is small, fast and powerful. In a way, it's like the Perl of version control systems.

You've visited Romania recently. What did you do there?

I had the honor of attending the Cluj.pm anniversary meeting. I gave a few talks and hung out with great people. I've written a detailed report available here: http://blogs.perl.org/users/sawyer_x/2013/03/clujpm-anniversary-meeting-report.html. In a nut-shell? Go there! Meet these people! Attending their conferences! We all have so much to learn from them!

Your talks at the tech events are very positive and energetic. What's your secret?

Thank you. :)

I guess I just think that what we have in the Perl community, both technical and social, is so fucking great, that it's hard to hold back.

There are so many exciting and fun things in Perl. When you look at other communities, at other languages, it doesn't seem like they have any glaring advantages (and many them don't have any advantages at all) but they know how to get excited about what they have. They show you some module that you know already exists in Perl for 7 years, but they get really worked up! In Perl you can write something amazing and people will just reply with a calm "cool" and that's it. We need to get worked up, we need to get excited, we need to realize how fantastic what we have is and to be ecstatic about it!

Another considerations in how I give talks is that if the talk isn't fun, you don't really learn much. It's hard to stay focused when it's just raw material. On a technical level it might be interesting, but you have to present it in an interesting and compelling manner. Look at Paul Fenwick and his talks. Can you imagine anyone not listening to him and learning from him? So of course none of us can be Paul (even Paul works very hard to be Paul), but we can do a hell of a lot more than just presenting a piece of code. We own our conferences and meetings, don't we? Let's have fun with it!

Where do you work right now? How much of your time do you spend writing in Perl? You've mentioned somewhere that you're an administrator too, is that still true?

I currently work in two companies: an e-commerce platform and a VoIP startup. I'm fortunate enough to have an amazing boss that lets me to work on a ton of Perl. I started with 50% of my time on Perl and 50% on systems administration. Once I stabilized the infrastructure, I was able to program more of it in Perl instead of chasing my own tail (which systems administrators do a lot). That way I got a stable ground and then started writing major components of the infrastructure in Perl. Nowadays about 80% of my time is spent writing Modern Perl, as part of the infrastructure and surrounding services.

At least until recently, because in June I will be moving to Europe and will be taking a different job. I will probably blog about it when the time is right.

Should we encourage young progammers to learn Perl?

Definitely! We just need to know how to be fucking excited about it. It takes time to realize why a certain solution is elegant, why a piece of code is nothing short of a work of art. Hell, it takes time for all of us to understand. Young programmers, who have even less experience, do not easily understand why Moose is spectacular. We need to be able to convey this. We need to be able to draw their interest in more than our experienced background, but in their inexperienced view. We need to speak their language and get them hooked. They will slowly learn to appreciate things from a more refined aspect, so we shouldn't worry about that now. Now, we should get them thrilled, get them interested.

Is Sawyer your real name?

Actually, No. It's a nickname my closest friends gave me a long time ago. It comes from the fictional character Tom Sawyer. There is a story behind it, but I'll keep that for another interview. :)

If you're ever unsure about calling me Sawyer, think of it this way: I always prefer to make a friend, so feel free to call me the way my friends do!

Why do you swear as 5 yo?

That was a slip of the tongue, I'll admit. I meant to say "13 year old". I guess swearing is one of the instruments I use to put emphasis when speaking. I mean, it's fucking noticeable! I also grew surrounded by trashy American action movies and profanities in Israel are generally very common, so I suppose my speaking patterns adjusted to all of that. Horrible, isn't it? :)

So, did you have a drink with Sebastian Riedel?

Not yet. I certainly hope to do so!

I got to meet Glen Hinkle who's a charming person. I hope next time I see him I'll be able to sit down with him some more and chat.

Are you going to visit YAPC::Europe in Kiev this year?

I was supposed to attend, then couldn't, and now there's a chance I might be able to attend anyway. I can't make promises because people get mad at me when I cancel, but it just might be possible for me to attend in the end. The question is how late could I submit talk proposals. :)

4 Stevan Little (September 2013)

When did you start programming?

I first started programming in BASIC around 1984 when my family bought an Apple][e. I mostly programmed simple graphics and made failed attempts at adventure games, I lost interest in about a year. I did not pick up programming again until late 1997 when I discovered HTML and Javascript and got a job working on the early web. Initially I just copied and hacked up simple Javascript widgets, but eventually I bought the big O'Reilly Javascript book and taught myself how to actually program. After that I spent the next several years reading computer science textbooks and learning other programming languages, including stuff like Ada 95, Erlang, LISP, Standard ML, Java and Python. It was not until 2002 — when I got hired at my current job at Infinity Interactive — that I learned Perl.

What text editor do you use?

I have always preferred GUI editors, up until recently I was a TextMate user, but about 6 months ago I switched to SublimeText 2. I really like it because it has some of the nice time-saving features you might find in a big IDE, but it doesn't force you into a particular workflow or anything like that.

When and how did you learn Perl?

As I said, I learned Perl when I got hired in 2002 at my current job. At the time I was still doing mostly HTML/CSS and Javascript for work and playing around a lot with Python in my spare time. At first, I was not very happy about having to learn Perl, until I found a copy of Damian Conway's Object Oriented Perl book and saw really what Perl could do. Having studied and learned a number of programming languages it was fairly easy for me to learn Perl initially, but it took several years before I really understood what it meant to write truly "Perlish" code.

What are other languages you like working with?

I am a big fan of programming languages in general, so I am always reading and learning about languages, both new and old. While I can read many different programming languages, I have only written substantial programs in about 7 or 8 or them.

Most recently I have been enjoying Scala, which I have been using for a couple experimental projects. I find it, on some level, to be philosophically a very similar language to Perl in that it seems to embrace the concept of TIMTOWTDI. However has very deep roots in academia and functional programming, which tends to eschew the more practical and “just get it done” philosophy of Perl. But it is still a very young language and it will be interesting to see the direction the community will ultimately take it.

At work we do a fair number of C# projects and I have always really liked working with that language. On some level it feels like “Java Done Right” in that the designers of the language clearly learned a lot from Java and where it got things wrong. I also have to say that the emphasis on tooling support in the core of languages like C# is really nice, this is something that many of the open source languages tend to not think about, which makes it very hard to add it in later.

I also recently played around with TypeScript, which is Microsoft’s attempt at making Javascript into a language more suitable for large scale programming. I found it quite nice to work with and I look forward to the 1.0 version.

In your opinion, what is the biggest advantage of Perl?

I think that Perl’s biggest strength is also it’s biggest weakness, and that is the philisophy of TIMTOWTDI.

Of all the languages I have learned and worked with, none of them comes anywhere near the level of flexibility of Perl. One good example is the new keyword API which literally lets you interrupt the parser in mid-stream and change the syntax of the language, I know of no comparable feature in any other language. And this is not just a new thing in Perl, we have long had access to the compiled op-tree via the B module, with which many evil things can be done. This level of access to the language runtime allows for so many interesting and useful things to be done with Perl. Along with this flexibility, Perl also has a robustness to match it, which allows for these features to be used safely, even in production systems.

But this same flexibility and robustness is also a problem for Perl.

Perl, because of TIMTOWTDI, is a language with many dialects; some good, some bad and some complete horrors to behold. I believe that this is truly where Perl gets it’s reputation as unmaintainable. Any sufficiently large codebase, in any language, will contain its own internal dialect created over time by the author(s) of the code, and if you are lucky, this will help a new programmer in reading and understanding the codebase (once they learn the dialect of course). For more restrictive languages like Python or Java, the variance in dialect capable of coexisting in a single codebase is fairly small. But for a flexible language like Perl, with it's extremely robust parser, is capable of supporting a very wide and varied assortment of dialects at the same time. This makes it much harder to a new programmer to learn, maintain and safely extend the codebase.

In your opinion, what features should the languages of the future have?

A good concurrency model I think it going to be the most important thing for any language of the future. Personally I really like the share-nothing Actor model because I find it to be predictable and much easier to reason about. It has been battle tested via the Erlang language for almost 20 years and is recently in the past couple of years been embraced by Java and Scala communities with great success. I am not saying that Actors are the best concurrency model, but they are certainly the most interesting right now.

Flexible and robust type inference I believe is going to be a key feature as well. In the past type inference has been found mostly in functional programming languages like Standard ML, OCaml and Haskell that have rigorous type systems, but recently people have been trying to apply it to more dynamic languages. A good example of this is the TypeScript compiler, given any ordinary Javascript (because TypeScript is a superset of Javascript) it will actually check all type usage even though no types are specified. While this does force the programmer to be more strict about their variable usage, meaning once you store an Integer you cannot later store a String, TypeScript also provides an "escape hatch" with an "Any" type, which allows the programmer to violate this at will. This kind of combination of flexibility and rigidity I think will be key for any future language.

And of course I think that any language of the future should have a robust object system with strong meta-programming capabilities.

Why did you write `Moose`? Why do you think it became so popular?

I wrote `Moose` after I had spent time working with the Perl 6 object system during my time working on the Pugs project. After spending a lot of time with the nice Perl 6 object system, coming back to plain old Perl 5 OO really highlighted how tedious and repetitive it can be. I had already prototyped the Perl 6 object system about 4 or 5 times for Pugs and I simply redirected those efforts to making something in that would work in Perl 5. I had about two or three failed attempts before I finally created Class::MOP and then I built `Moose` on top of that.

I think `Moose` became popular for the simple reason that people really wanted a nice Perl 6 style object system and had been waiting anxiously for it for years. `Moose` managed to fill that need for those people, and after that it just kind of went viral as those people introduced it to others.

What do you think about alternative implementations and so called "light" implementations of `Moose`?

I have no problem with them, I think they fill a need that `Moose` can never fill. Over the years there have been a lot of them, some good, some bad, right now I think Moo is the best one because it allows you to seamlessly "upgrade" to `Moose` if you need too.

What are `p5-mop` and `p5-mop-redux` projects?

The `p5-mop` project was my first attempt at building an object system, similar to `Moose`, that could fit into the core of Perl itself. Ultimately the project got crushed under the weight of its own complexity and some really tricky show-stopper bugs. I have to admit, I was quite disheartened by this and got very negative about the Perl core for a while. This actually lead me to start the Moe project, which was an attempt at writing a Perl 5 like language using Scala. Moe ultimately turned into a place for me to experiment with language concepts that I would like to see in Perl 5 but which were really hard to prototype in Perl 5 itself.

Then on the plane ride home from YAPC::NA this year I decided to give the `p5-mop` work a second chance, this time using a new approach based on things I had learned while writing Moe. This eventually became the `p5-mop-redux` project that I am currently working on. I have high hopes for this project and will be working over the next few months to try and get it into the core of Perl.

Can usage of OOP framework without understanding lead to even worse programming practices? Is it possible to write clean code without an OOP framework in Perl?

I think that not having a good understanding of your tools will often lead to bad usage of those tools, I don't think OOP frameworks are special in this sense. It is very possible to write clean code without an OOP framework, the Plack codebase is a perfect example of this.

Do you think that CPAN is now divided into two tribes: `Mo*` users and others?

No, I think the division is more along the lines of "Dependencies are a good thing" and "Dependencies are evil". `Mo*` users tend to be in the "dependencies are a good thing" camp, and non-`Mo*` users often tend to be in the "Dependencies are evil" camp, but I don't think `Mo*` is the dividing line here. It is my hope that Moo, having very few dependencies, will help to bridge the gap between these two camps.

This too is what I hope to fix with the `p5-mop-redux` projects, if we have an extensible object system in the core then people won't have to worry about the burden of dependencies just to have a nice object system.

Where do you work right now? How much time you spend programming?

I work for a small consultancy called Infinity Interactive, we have 24 full time employees and a handful of regular freelancers. We offer a full range of services such as; Perl programming, high end HTML/CSS development, sophisticated Javascript driven UIs, .NET and Java programming, system administration consulting and services and more. These days I tend to do more project management and architectural design then I do programming for clients, however this is not all bad because it means I have more time to work on Open Source projects like p5-mop.

Should we advice young programmers to learn Perl?

Yes, eventually. I say this because I do not think Perl is a good language with which to teach programming, I think either Java or Python are better for that. However, I believe that at some point in their careers they should try and really learn Perl because I think it can be a real eye opening experience for the programmer willing to dive deep enough into it. There really is no other language I have encountered quite like Perl and I think really learning it and understanding it can make you a better programmer in the end.

5 chromatic (October 2013)

How and when did you learn to program?

Which time?

Like many of my peers in the US (middle class white kids who grew up in the suburbs), the first time I saw a computer in person was elementary school. I was 5 or 6 years old. It was a Commodore 64. In those days, computers came with manuals and BASIC, and the manual showed you how to write simple BASIC programs on almost the first page. (Page 32 — I just checked. The first several pages tell you how to hook it up to a TV or monitor and how the keyboard works.)

I eventually talked my parents into buying a computer for the family.

When I was in high school, my interests changed and I spent more time playing music than programming, but in college I tried to pick up C++ (it didn't take) and Java (it took a little) and in my first real job in a career, I wrote a few programs for the printer group at HP and found that a lot more entertaining than anything else I was doing.

What editor do you use?

Vim.

When and how have you been introduced to Perl?

Java was new and interesting when I started at HP, but I was doing what work I could on HP-UX and Linux (Red Hat 4 or 5, I think). Java was way behind on Linux in those days, but at least it had basic Swing support.

The second serious programming project I worked on had to send email to a list of customers about updates to the printer networking support service. I wrote a proof of concept in whichever variant of the Bourne shell HP-UX 9 supported. It was about ten lines of code. Then I had to port it to Java because the group who wanted the code needed it to run on Windows NT. Back in those days, Java was immature. It didn't have many libraries. To send email, you had to write a lot of code to connect to an SMTP server and send the correct headers and everything. It took more code to set up the Java libraries than it did to do the whole task in shell.

I started looking around for other languages. My brother was studying and working in telecommunications at his university at the time, and he'd been automating a lot of things in Perl. I bought the second edition of the Camel book, read it cover to cover, and that was it.

What are other programming languages you enjoy working with?

Haskell, for the way it makes a good type system usable (and encourages you to write small pieces of code that join together sensibly).

Scheme, for the way it lets you build up a good system from a small set of usable pieces.

I keep wanting to like Go, but it's never quite been useful enough for me yet.

I keep thinking there's a good language in C++ struggling to get out, but then I feel like I'm not smart enough to find it out.

What do you think is the most strongest Perl advantage?

The lack of ceremony. You can give Perl to someone who has little experience programming and he or she can get something done more quickly than you would expect. He or she might make a mess — the discipline to program *well* takes time and experience to develop — but the pragmatism of Perl is a huge advantage.

What do you think is the most important feature of the languages of the future?

That depends on the language. Most people think speed of execution is the most important thing, and in a few contexts it is. If you want to get a lot of users, though, you have to let them do something that they want to do without frustrating them so much that they'll leave for something else. Again, pragmatism is a benefit here.

PHP didn't get a lot of users because it's a good language. PHP has a lot of users because it's just about the easiest way to make a dynamic web page.

You've invested a lot in Perl ecosystem and infrustructure. What is the reason behind that?

I want to make my life easier. Having good tests for CPAN modules means that I can trust CPAN modules more than I could before. Having simpler and better abstractions for things like email or OO development means that I can get my work done more correctly and with less effort than previously.

Having a good introduction like I hope the Modern Perl book is available freely means that more people can start writing better Perl than they would if they followed the awful, outdated tutorials that were already online.

During your work on Perl 6 you've done work on roles. Do you think they are essential for modern OOP? Are they a silver bullet or anther 'shoot yourself in the foot' pattern speaking of a real large and complex project?

If you read modern OOP tutorials, the bad ones will try to tell you that inheritance is essential to OOP. That's silly. Polymorphism is essential to OOP.

(Polymorphism is essential to programming. If you can't learn to treat things that are different in small details but similar in other details in the same way, you'll go crazy. People often struggle learning arrays because they want to name variables $recipe1, $recipe2, and $recipe3. Only when they realize that they can treat them all as members of @recipes do they start to understand the power of programming.)

In the development of roles, I kept arguing that we needed to solve two things:

1. how to identify behaviors and state that were conceptually similar enough that they could be polymorphic;
2. how to share code without enforcing a code sharing mechanism (delegation, inheritance, composition, cut and paste, role application).

The researchers behind the original Smalltalk Traits paper were trying to solve similar problems at the same time, so I think we were all on the right track.

Can they be misused? Yes. Can you shoot yourself in the foot? Sure. Are they essential to a good OOP project? That depends on the project.

In my experience, using roles and parametric roles effectively has improved the design of several projects. It takes time and experience to understand their effective use, but if you're a disciplined programmer who is willing to revise the design of your system as you discover what it needs to do, roles are another tool in OO design and analysis that you should consider.

Why do you think testing is important?

I want to improve my confidence that the code I'm using and deploying works. Testing by hand doesn't give me confidence. I'm okay if I don't have 100% confidence, but if I can look at the tests and they make sense, I have fewer worries about the code.

I don't care much about the distinction between unit tests, integration tests, behavior tests, spec tests and the like. I try to stay pragmatic. I want to know if the essential behavior I want to rely on actually works. I want to know that you've thought about edge cases. I want to know that if I make a change and break something, running the tests will catch that.

It's about quality and confidence and reliability. It's also a marker of discipline and quality of the programmer. You don't have to have automated tests to write great software, but if you're not writing automated tests, I want to know why I should trust the software you write.

After more than a decade is Test::Builder keeping up with the modern testing practices?

What it does, yes. What it looks like inside? No. Hopefully `Test::Builder` 1.5 or 2.0 will make it out and clean things up.

What inspired you to write the Modern Perl book?

Perl needed a short introduction that didn't pretend to cover the entire language. I thought that explaining the philosophy of the language, how to use perldoc, the necessity of the CPAN, and what to use and not to use would be useful.

Why did you decide to make it freely available?

Three reasons. First, people are going to get it for free through various copyright infringing means if they really want it. That's not worth my time fighting. Second, I hoped it would spread further (and promote the printed and paid electronic editions) if it were freely available. Third, I know there are people in countries in places like Eastern Europe and Southeast Asia and Africa and other regions who will never be able to buy a printed copy because it's too expensive to ship (or just too expensive). I can't write in all of their native languages, but I can give them a free edition in English that someone can localize.

Are you planning on updating it as Perl hopefully evolves?

I'd like to get a new edition out before the end of the year.

What's up with perl.com? It seems it has not been updated since January.

It's not easy to find authors and I don't want to be the only author.

It seems that you're involved in several businesses. What is your primary occupation right now?

Right now I'm consulting for a startup in the medical industry.

How do you see the future of Perl?

I'm hopeful for projects like Stevan Little's new MOP and Peter Martini's subroutine signatures in the core.

I'm doubtful about projects putting Perl on a new backend, at least without someone fixing the XS problem.

CPAN continues to grow.

I don't know what will happen with the Perl outreach problem. (Some people call this "marketing".)

Perl will continue helping my clients and projects get the right answers.

Is there a place for Perl 6?

If it finds something it does better than existing languages, maybe.

Should we encourage young people to learn Perl right now?

That depends what they want to build. I talked to a high school student yesterday who wants to study computer science and has experience with HTML and wants to learn JavaScript. Does Perl make sense for her? (Is there a simple Windows distribution that uses some Plack magic to make deploying Perl programs as easy as PHP programs, but doesn't require her to use CGI.pm? Perhaps.)

On PerlMonks you have a Perl 7 t-shirt user image. What are you thoughts on Perl versioning?

In retrospect, maybe calling it Perl 6 in 2000 didn't work out so well — but that's not going to change unless Larry changes it, so we're stuck with Perl 5.whatever for the foreseeable future.

What do you think about situation around smart match?

Perl 5 has long had the design idea that variables should be polymorphic while operators should be monomorphic. In other words, you can see `$a + $b` and know that's it's addition (no matter what `$a` and `$b` contain) and you can see `$a . $b` and know that it's concatenation (no matter what `$a` and `$b` contain).

Smart match was different, and any time you deviate from a core design principle of a language, you have to consider the pros and cons carefully. Smart match looks like a tool intended to solve several different problems all at the same time. In the absence of a strict set of monomorphic variable types, you can't look at the code and immediately understand what's going to happen.

It's similar to the problem of `each()` on references; you can't tell if each `$foo` is going to give you the keys and values of a hash or the indexes and values of an array.

Where do you get the energy to write your programming related blog posts regularly?

I've always been a writer, and like any habit you develop over time, you build up muscles (literal and figurative). By now it's a habit.

With that said, my day job and personal life are busy enough that my frequency of posting has dropped from several times a week to weekly. It would be nice to get a little more free time.

Was it hard to quit after several years working on Perl 6 and related stuff?

Yes and no.

It hadn't been enjoyable for a long time, for a variety of reasons. The biggest was probably my disappointment that, even after 9 years of work I'd put in doing un-fun things like fixing segfaults and finding optimizations, it still hadn't delivered anything I could use in work that mattered.

I couldn't justify working on it in my business hours, because I had enough paying work that needed deployable code, and I had enough other potential things to do in my spare time that I actually enjoyed.

On the other hand, if you work on something for 9 years you'll probably feel some sense of attachment to it. I'm proud of some of the work we did and I learned a lot, but if you told me in 2001 or 2002 what would have happened by 2013, I'd have spent my efforts elsewhere.

That lesson took me a lot of time and trouble to learn. I felt guilty for a while, thinking that Parrot would suffer if I weren't there fixing bugs and writing code. But then I realized that if Parrot weren't useful or usable for anyone else, maybe that was a good thing. If a project can't survive when it loses developers — if it's not being used for anything productive or if no one cares enough — then maybe the resources being used on it would be used better elsewhere. Certainly that was the case for me personally.

I spent almost a decade working on all of those projects, so I feel like I did what I could to make them succeed. If someone else wants to pick up the work I did and do something else with it, the code's out there. But none of those projects are worth my time or energy anymore, and I'm comfortable saying that. If you find yourself in a similar situation, anyone who tries to make you feel guilty for thinking there are more important things than working on software you're not getting any benefit from is being unfair. (Even if it's part of *you* trying to tell yourself that.) Donating your free time and talent to free software is a gift, and it's a gift you have the right to rescind.

Are you enyoing Perl conferences?

I took the year off from conferences for non-technology reasons and have enjoyed not travelling. With that said, it's always a pleasure to spend time around great people in the Perl community, so I've missed that this year.

6 Marc Lehmann (November 2013)

How and when did you learn to program?

I started programming when I was around nine years old or so, with some 8-bit board, the type where you enter instructions in hex. My parents “heard” that computers are “bad for your development” and were reluctant to buy a “real” computer, but that board passed their test. Luckily, a few years later they bought me a C-128, and shortly thereafter, an Amiga 500, which was the ideal computer for learning OS design, low-level coding and a lot more: The earlier computers mostly allowed me to understand hardware and very low-level programming, but the Amiga OS was as advanced as a modern OS kernel. When the Amiga became too old and too slow (despite the 68030 upgrade board), I switched to the much inferior PC+DOS platform, and in 1993, to HP/UX and shortly after that to GNU/Linux.

As for languages, I started with machine language, switched to BASIC and 6502, Modula-2 and m68k, then to Turbo Pascal + x86, and finally to Perl + C on HP/UX and GNU/Linux.

I never stopped learning how to program, of course — just this year I finally sat down to properly learn Javascript (as opposed to fool around around with it), and I still learn new ways of programming in Perl or C every year, even though these two languages are with me for a long time now. Perl is especially good at providing surprising new insights.

As to how — mostly I read the manuals that came with whatever language or computer I had, and worked from there. While studying informatics I learned a great deal of things, but I don’t think I learned any programming, so I learned most things though trying and failing on my own.

The most important way I learned (and still learn) new things is by reading other people’s code. I think I probably read 20 times more code than I write. Or maybe the ratio is even higher.

These days, I also often learn new languages by reading the relevant language specs to work around all the bad information you can find on the ’net — reading random blogs for basic info and then the spec to correct the misinformation works quite well for me.

What editor do you use?

VIM, exclusively so. I actually tried hard to learn Emacs, even getting the "Learning GNU/Emacs" book — I normally don't try to learn from books. To this date, I can still tell at what page in that book my brain suffered a fatal reality exception and rebooted. Put differently, even though I might have preferred Emacs for being the cooler editor, it turned out that my brain is incompatible with it in a very fundamental way, and I am stuck with VI for probably the rest of my life.

This is also why I think any editor wars miss the point: It's your brain that is VI-like or Emacs-like, and no matter how good the other editor is, it might not work for you at all.

As a sidenote, before I used VIM I used joe for a while — coming from Turbo Pascal/-Turbo C, joe was great because it used the same keybindings. The reason I mention joe, however, is that it still is occasionally useful because of a single killer feature: It can edit files that do not fit into memory. So when you need to interactively edit this 60GB text file — joe does it for you.

When and how have you been introduced to Perl?

That was in 1993 on HP/UX — I was looking for a nice language to start on Unix, and was overwhelmed by the Internet and the free availability of information, "standards" (RFCs), and the fact that almost everything (configuration files, protocols such as FTP) on Unix and the Internet was text-based.

Perl 4 came with good documentation (and for free, too :), and that's how I got accustomed to it. And when I switched to perl 5 in 1995, I simply read all the man pages that came with it in order, was amazed, and stayed with it ever after.

My first Perl program was a curses ftp client that could download files in the background — a very useful feature when the average download speed was 200 bytes/s. After it became popular it allowed me to do some security research (by gaining access to other people's accounts, strictly for educational purposes of course). When the university found out, they actually offered me a job, which, I guess, was good for my development after all.

What are other programming languages you enjoy working with?

He, "enjoy" is such a rubbery concept :)

I often pair Perl with C++ (if possible) or C (if not). I really love XS for its power and relative simplicity (in results, not in learning it), so I could describe my main language as "Perl+XS".

And while I dabble around in a lot of languages (for fun and profit), I couldn't say I really enjoy them. I regularly sustain posix shell, various assembler dialects and javascript, and avoid looking at php/python/ruby/... code at all costs, with mixed success.

As for truly enjoy, Perl is the only language I am afraid.

What do you think is the most strongest Perl advantage?

I am not sure Perl has clear advantages anymore, but for me, the thing that makes me prefer Perl over other languages in its category is the core language itself and the malleability of the interpreter.

One example is `Coro` — the fact that you can add such a thing as Coro as a mere extension to an unpatched perl binary is outright astonishing to me. Attempts to do this kind of thing to other languages such as Python usually end up with a rewrite or fork of the interpreter.

There are also many "micro-features" that don't look big, but really are. An example here would be `__END__` — it doesn't look like a big feature, but it can be used to elegantly bootstrap a perl process over, say, an ssh connection. `AnyEvent::Fork::Remote` does this for example, but I used this many times in the past in commercial projects. Other languages such as python have no equivalent to `__END__`, so implementing `AnyEvent::Fork::Remote` there would require ugly workarounds, probably involving temporary files and other hacks. In Perl the solution is simple and elegant, and Perl is full of these small but occasionally super useful features.

CPAN for example is great, and for many years definitely was a major selling point, but it's not something that other languages don't have these days, and the code quality of many modules is severely lacking.

Still, CPAN, the language, the malleability all come together in the one language that is Perl, and that is a pretty unique combination.

What do you think is the most important feature of the languages of the future?

I don't think I have a good answer to that. I hope languages of the future let me do what I want to do and don't get in my way.

As for the near future, I would hope that languages would allow easier data sharing between processes on the same and different nodes. Of course, I am working on that in Perl as well (my todo list is very long — `libev/EV` was on my todo list for more than a decade before it acquired the required priority), but I am not there yet.

As for the far future, I suspect future languages will be more graphical, mind-controlled (and possibly mind-controlling), and so annoying that I will probably still code in this antique Perl language while everybody else has nice and shiny quantum computers built into their head that listen to their subconscious thoughts and write software on their own. If the economy permits it, that is.

But, seriously, it definitely would be nice if I would just have to explain my problem to the computer (by having a silent conversation using abstract thoughts), and the machine would do the work of implementing and debugging it...

You have written several event loop related modules that became very popular. Why did you decide to make them open and why do you think they became so widespread?

That is a difficult question — it's rather easier to say what the reasons aren't: I don't go for publicity and I don't go for fame.

I think publishing what you write is the natural and default state of affairs. Altruism plays a major role — when I genuinely think something might be useful (as in, "I wish somebody else had done it already"), and I am allowed to, I feel I just NEED to publish it for the common good. After all, that's what I have seen other people do as well, and it works — if people do publish, other people will be encouraged to do so as well, which is good for everybody.

I try to document my modules mostly because otherwise they wouldn't be that useful to anybody else. It's the extra work that is needed to make something actually useful in practise, as opposed to just being theoretically available. Documentation is the only part of my software that I write for others — the code I write because I have use for it myself.

As for why some of my modules became so widespread — I can only guess, as I usually don't care, to the point of not really knowing which of my modules are popular or not.

With `AnyEvent`, I had a collaborator who thought `AnyEvent` is so useful, it should be better known, and this is why we tried to provide some basic “must have” modules, such as `AnyEvent::HTTP`. Whether that helped is hard to tell for me, because I didn’t try very hard, even the extra modules I wrote were chosen because they were useful to me.

Maybe they became popular because I studied event loops for almost two decades, silently complaining about various problems and lack of certain features, and as a result, tried to provide everything that I missed and considered essential, while not making the interface too complicated.

In fact, you can see this evolution in `AnyEvent` and `EV` — `AnyEvent` has an interface more like the `Event` module (using methods). Later, `EV` taught me that a much more minimal interface suffices (function calls with only a few fixed parameters), and thus the `AE` interface was retrofitted into `AnyEvent`.

In the end, I hope my popular modules became so because they helped somebody solve a problem, just as they generally help me solve a problem (or two, or three). I fancy that it is me striving for quality that makes them useful, but I only ever think that in secret :)

Is it possible to shortly describe why Coro is the only real threads in Perl?

Apparently not shortly, no.

The defining feature of threads in other languages (such as Python or C) is that address space (i.e. addressable code and data) is shared between threads. When you don’t share this, you end up with something called “processes”.

The “competing thread model” in Perl, ithreads, uses such threads in C to emulate Unix processes in Perl. On the Perl level, you end up with a (very buggy) process model, and sharing variables or code between ithreads is about as (in-)efficient and unnatural as sharing them between real processes, except real processes don’t have to emulate the MMU in software. Sharing of objects and code isn’t even implemented (array or hash-based objects come out empty after passing them to other threads). Sharing existing data structures isn’t possible at all, and so on.

Thus, the “only real threads in Perl” statement wasn’t meant to be controversial, but a rather obvious tag line, but in past years, there have been recurrent events where people define threads as something else and then contest the validity of the statement.

Basically all such cases are variations of "a thread is something that runs in parallel with other threads, so Coro aren't real theads, ithreads are", without realising that this would instantly a) rule out pthreads, python threads, ruby threads and most other thread systems, as they often cannot run in parallel either and b) would mean processes are threads as well, which makes this definition not very useful (we already have a word for process — "process").

While processes are certainly threads of execution, the common meaning when people refer to threads in imperative programming languages is to mean "multiple threads of execution share the same address space", and thus, Coro are the only real threads in Perl, sharing both code and data naturally.

What was the reason behind writing common::sense?

I always wanted to have some useful warnings, but a lot of warnings in Perl are just getting in your (my?) way, or are fatally flawed in some minor detail, making the whole warning useless.

Not having `common::sense` either meant duplicating its effects in basically all of my modules and programs, or not having any warnings at all.

So the reason behind writing `common::sense` is maintainability - unbundle code from various modules that need it and put it into a common library so it can be shared. If you look at the documentation for `common::sense`, you can see that the code isn't trivial and has changed multiple times. Without common sense, I would have had to make new releases of most of my modules each time something changes, with no benefit at all to the users.

What was the story with JSON::XS and Perl hash key ordering?

There are certainly multiple valid viewpoints on this topic, but here is mine:

Many years ago, there was a bug in CGI.pm that could trigger resource starvation, and for some reason, the perl 5 porters thought that making core perl a bit less useful was better than actually fixing that bug, or having resource limits in place to catch such exploits.

I thought that a bit strange at the time, after all, other languages with dictionary data types do not generally randomise them for security reasons, and thus C++ and other languages suffer from the same "security issue" (or at least fix the actual bug, namely resource starvation, not the symptoms).

Nevertheless, the documentation said that the ordering would be the same within one program run, and that was certainly true enough for more than a hundred modules to rely on it (there wasn't much breakage on CPAN at the time, as very few programs relied on hash ordering between multiple runs, probably because that never was a given anyways).

More recently, this randomisation apparently was found lacking (or broken), and a different system was implemented, where even the same hash would end up with a different order, within a single program run, something not actually needed to work around the original security issue.

There was no depreciation cycle — documentation and code was changed, and (fortunately) a lot of patches for modules were created, as otherwise, a lot of vitally important modules (i.e. not `JSON::XS`, but e.g. `LWP`) would no longer run with the next perl release without patches.

My involvement was in questioning the need for this drastic step — while a real security issue would probably justify that, I found the reasons were somewhat lacking (the few other languages that implement randomisation do it the "old" and dependable way, and there was no actual security exploit known), and to date, nobody I asked had an answer as to why the actual bug isn't being fixed (or checked for).

The only answer I got (from the perl pumpkin) amounted to: "That change fixes the issue, and we have no better patch at this time." This is reasonable, but is merely after-the-fact.

It's not the first "feature" that breaks existing code without a good reason in recent perl releases, and maybe it's just me, but I value bugfixes and stability much higher than cool new features.

In fact, Perl has "recently" acquired a much faster release cycle — roughly one major release per year. Unfortunately, backwards compatibility has effectively become a non-priority, so nowadays I have to deal with breakage regularly, which means I have to invest a lot more time in my modules, just to work around the latest incompatible change. It doesn't help that bug reports abut incompatible changes are often ridiculed or downplayed, and all that caused me to post some harsh criticism of the process.

When you follow perl since the 5.000 days, the difference between stability/backwards compatibility of Perl in the past, and recurrent breakage of Perl nowadays is pretty striking.

Can you give any examples when App::Staticperl might be useful?

I often send customers a `staticperl`-compiled executable in addition to the perl sources — often they don't know Perl, and requiring them to install a lot of modules would simply not compute. With a working `staticperl` setup, creating the binary is hardly more work than running the program in the first place, and even if the perosn who receives the program is good at installing dependencies, being able to instantly run the program without any extra work is just nice.

On systems that allow actual static linking (e.g. not glibc and not anything windows), you can even create binaries that "run everywhere without dependencies" — for example, I sometimes distribute GNU/Linux binaries that run on any x86 or amd64 system with a reasonably new linux kernel, regardless of libc or other details (alpine linux is a good starting point for building such binaries).

Sometimes customers don't want to know about Perl, and `staticperl` enables me to sneak in perl into a binary or shared library without the customer to freak out. It's amazing how fast and capable Perl is if it doesn't have to suffer from any stigmas because users think the program is written in C :)

It's also quite nice when you want to embed perl in an executable without having to have a lot of extra files. `staticperl` creates a .h and a .c file for me, and when I compile and link these and libperl into my program, I have a full-featured Perl interpreter plus Perl library and custom modules, without any external files and without any dependency on a writable filesystem. All in your own C program.

The practical relevance of this is that I can often embed perl in cases where I could normally only embed lua.

I also happen to sometimes test out new weird Perl build options or versions quickly using `staticperl`. That's just me, though: staticperl wasn't meant for that. I hear a lot of good things about perlbrew, so you should try that first if your goal is to have many perl binaries.

Now, this isn't advertising — all these features come with a hefty price tag. A few very broken modules (mainly `Module::Build`) make `staticperl` less than straightforward and only an option for experts, and having to understand how Perl builds and is configured, and what the difference between static linking and a static binary is can be more than most people want or need to know. And often I feel I am the only person left on this world who is embedding Perl on a regular basis (it feels natural to me, but apparently not so to others).

Once you are past that point, though, doing standalone binaries with staticperl are serious fun, and quite a bit faster and easier (for me) to create than e.g. with `PAR::Packer`. And they have a much higher chance of actually running, in my (biased) experience — if `PAR::Packer` would have worked reliably for me in the past there wouldn't be any `App::Staticperl` or `Urlader` nowadays.

(As for advertising, `Urlader + Perl::LibExtractor` are yet another option to bundle perl, which is what I actually use on Windows. It is still faster and more reliable than `PAR::Packer`, but works with any program, not just Perl, and — to me — is conceptually simpler).

You seem to heavily optimize your software. Is this usually a requirement or just a habit?

Actually I thought most of my software isn't very optimised, but I understand why it looks that way.

First, I do have considerable experience not just with programming, but also with what is fast and what isn't, and I indeed have trouble writing things inefficiently if, with little or no extra effort, I can do it better. Therefore, many of the optimisations you see are due to coding habit or because I wanted to avoid the pain of not writing good — according to my standards — code, when it doesn't result in some obvious gain.

Secondly, I heavily optimize basic and often-used functions (libraries, modules and so on) so when I write the actual program that uses these libraries later I can chose to write clear, small, and possibly not so efficient code.

Look at it that way: You can write your app in C, or you can write it in Perl. All else equal, in C it will be faster, but it's just such a pain to do it.

That's why you might want to write it in Perl — it's convenient, and since somebody else has heavily optimised Perl in C, it's probably fast enough.

The maxim at work here is "use the most convenient language that is still fast enough", and having heavily optimised C libs for Perl makes it possible to choose Perl more often for the convenient parts.

If you combine "use the right language for the job" with Perl+C, then you got everything covered usually, because Perl is good at things that C isn't, and vice versa, resulting in the best of both worlds.

So in reality I am actually too lazy to optimise my software, and this is why I apply optimisations where they matter most. This allows me to slack off and write less efficient but simple and straightforward high-level code.

However, optimisation always has lower priority than correctness.

Where do you work right now? Is Perl you primary language?

I work at nethype GmbH in Germany, a company I founded together with a friend while we were both studying together.

Luckily, Perl is our main language, which we use for anything ranging from millisecond, distributed domain trading or implementing high-performance DHCP or radius servers, to mundane things such as SMTP crawlers, web applications or even online games such as Deliantra — the full works, basically. We are also in the lucky position to often be able to publish modules that we developed for work as free software.

While C or C++ often aren't far away, Perl is the perfect language to write high level logic in it, not least because with XS you always have easy access to C to implement the few time critical parts.

Do you enjoy visiting Perl conferences?

Absolutely! Mostly for the ability to talk to other Perl freaks (and also normal attendees :).

However, making my actual body move to actually attend one is another story — I am really just lazy, which is why I attend even fewer conferences nowadays than in the past. All the planning effort, and then you have to fly or drive (and prepare a talk)… too much effort (I also do avoid the US these days, for complicated political reasons).

I have never regretted it when I forced myself, but it might require some coaxing.

What are your thoughts on Perl future?

By 2050 me and the other two people still using Perl make a lot of money maintaining the existing cobol^Wperl programs that the rest of humanity depends on to fight their (vain) fight against global warming, with me ending up rich and the rest of the world ending up dead and starving, until electricity runs out and my money becomes useless and we are all fucked.

Ok, sorry, I mixed my future in there.

Realistically, I think Perl 6 will to continue its slow and necessary death, giving Perl even more breathing space to survive and probably even grow.

I don't think any improvements (if they are improvements) to the Perl core will have much effect on the future of Perl — Perl is decidedly good enough, and CPAN is quite good as well.

I do think Perl isn't "fashionable", and I think that this and the fact that Perl is no longer alone in it's category will make its importance shrink, because fewer people will learn Perl, or see it as a "main language".

For me, that's not bad, and quite natural. I think the strength of Perl is not in it being innovative, but being dependable and stable. That isn't good for growth, but Perl might still be around and useful when other, more fashionable, languages are long forgotten.

However, I am not driving the Perl development process, so my predictions might be totally off and my wishes might not be shared by the people who shape the actual future of Perl.

Specifically, I would personally wish that backwards compatibility, and keeping CPAN in a working state by not continuously breaking modules would gain more priority.

This is far more important to me than iplementing new featurss in the perl interpreter — perl is so flexible that new features can often be implemented in a module without breaking stuff, and that's a better way to drive innovation.

So, I believe Perl is safe from extinction, will keep it's place among other popular languages, and can, in theory, even grow while keeping the enourmous body of existing code working.

Should we encourage young people to learn Perl right now?

Absolutely — it is a great and useful language. One might argue that Java is better for Money-Oriented-Programming (MOP), and C++ programmers might be more commonly sought after on the job market, but apart from a slightly harder job search due to it being more exotic, there is nothing wrong with Perl as a first or second programming language, and it's just so much more fun to code Perl than Java, which is good for your health.

So, when somebody would like to do something with ruby, python or a similar "scripting" language, suggesting Perl as another option will not do them any disservice: Giving "young" people the option of choosing between multiple languages enables them to choose the one they feel most comfortable with, and that is what counts.

Imagine I wouldn't have tried VI because I was told it was old fashioned or so — I'd be eternally unhappy with GNU Emacs by now. Shudder.

There is a minor problem with that though: Sometimes I get asked how to learn Perl, and I usually am at a loss there, having learnt Perl from reading its manual and source code, which probably isn't the most painless way to approach it. Looking at actual books, I find that many books are not written for beginners — "Learning Perl" from O'Reilly for example explicitly says it doesn't teach you programming, and most other books I have looked at kind of only teach Perl as a second language (or are simply crap). The one exception I found is "Beginning Perl" by Simon Cozens (and it's free to use, too!).

I think it's difficult to recommend Perl as a first language when most books treat it only as a second language, so make sure you have looked at a few books before recommending Perl, so you can recommend a suitable one.

But there is nothing wrong with the Perl language itself.

Why do you still use CVS?

Short answer: it works.

Long answer:

CVS still does everything I need quite well. Git for example requires more commands to do the same things that CVS does with checkout/update/commit and is a lot of trouble when working in a tight-knit group (while being far preferable to CVS when working in a more distributed fashion!). SVN really is just worse than CVS (same command structure, similar limitations, but longer URLs and this weird "copying" model that decidedly isn't how my brain works). If I had to, I would probably look into Mercurial, which looks like a saner alternative to git, should CVS not serve my needs anymore.

I am not religious about CVS (but it's astonishing how many people are religious about git and harass me to switch to it), it just happens to serve my needs. Why change something that works well for you: The time I can save by using CVS can be used more productively elsewhere.

Also, commit ids and tools such as cvsps bring CVS to about the same level as other systems these days, fixing most limitations, so the difference isn't as big as it once was.

You are famous for expressing your opinion in a harsh way. Is this intentional?

Yes and no — I usually have opinions about things or concepts, not about people, so I don't quite see why I should hold back — it's not as if I can hurt the feelings of a thing or concept. I can also always substantiate my opinions, so I don't feel that I need to hold back (even when wrong, I made a reasonable attempt of being right, as opposed to lightly making things up and having unsubstantiated opinions on everything). Both of these help to be able to have, and state, strong opinions, and be rarely wrong. It also helps to be able to shut your mouth on topics you don't know much about.

I also have the impression that I am not actually that famous there — the few people which bring this topic up usually were told about it by (some very few) other people without having seen any evidence themselves. In fact, the only people I see who publicly make this claim are disgruntled fans whose patches I didn't apply...

This impression is supported by my experiences with "non-Perl-communities", where I am not famous at all for this behaviour (I have a few popular non-Perl libraries and programs...).

Do you consider yourself a part of Perl community?

That critically depends on how you define "Perl community".

On the first Perl Workshop in Germany, I was astonished to see the enormous diversity of white-collar people, nerds, hobbyists and so on, who had little to no thing in common — except Perl. Did they form a community? I don't know, but I doubt it.

I contribute to CPAN, and consider me part of that "community". I don't consider me part of the "modern perl movement" community (if such a thing even exists).

So I really don't know what the "Perl community" is supposed to be, and I suspect there isn't a single one, but many different ones.

One thing I can say for certain though: most of my projects are not "community projects", or "community driven projects". I do develop software for my own needs, and that often means I will not provide a feature that I will not use myself (nor be able to maintain). This is the reason why I usually develop things to be extensible by other people — that allows me to decline feature requests.

Have things improved with rt.cpan.org?

Things have actually gotten worse. There are some cosmetic changes (such as displaying a notice on a subset of pages), but overall the effect is that rt.cpan.org makes it even harder to avoid it. The core problem is there just as much as in the beginning: It forces itself on other people's code and cannot be disabled (last checked: beginning 2013).

So, there has been an effort on the side of rt.cpan.org to hide the problem, but no effort to actually improve the situation. It feels like a game that I don't want to play. It's pretty annoying, really, just as being force-subscribed to some microsoft list or so, except you can't ignore it as spam because there are other people being victimised as well.

I just wished there was a simple setting to disable rt.cpan.org for my modules or change it into a mail forwarder, but the powers that are clearly are not interested in this, making it hard to configure it, and impossible to disable.

Is OpenCL Perl bindings are stable enough? What are advantages over using this library within Perl, rather than C/C++?

I haven't seen a need for a new release in the last 1.5 years, and it still works reliably enough for my usage. I suspect more work will come when NVidia finally supports OpenCL 1.2 (or, wow, maybe even 2.0), so I can improve support for that (I still have only a 1.1 driver to test against).

And the advantage of using OpenCL in Perl rather than C/C++ is that you can use it from Perl, silly :->

Why does AnyEvent treat `IO::Async` in a special way?

Short answer: Because the other backends don't work with `IO::Async`.

Long answer: `IO::Async` implements its own event loop (rather than being a framework that sits on top of one — it is a framework, but also comes with its own event loop). For every incompatible event loop there needs to be a custom backend — just as there is one for EV or Event, there needs to be one for `IO::Async`.

If `IO::Async` would only *use* other event loops (e.g. via AnyEvent, or directly using EV etc.), then no such backend would be needed. AnyEvent would then use the same backend as `IO::Async` and would mostly coexist peacefully.

(To the very technically inclined: normally, there would only need to be an interface to `IO::Async`::Loop, which implements the `IO::Async` event loop, but that module is the rare case of an event lib that doesn't support sharing between multiple independent users, so to be able to work with `IO::Async`, AnyEvent unfortunately needs to go via `IO::Async` (and still needs some extra manual setup, as `IO::Async` has the same design issue — multiple independent `IO::Async` users cannot share it, so you need to tell AnyEvent about the instance of `IO::Async` you wish it to use)).

Apart from that, `IO::Async` isn't treated specially, i.e. it's treated specially, but in the normal way for AnyEvent.

There is some controversial code that explicitly disables itself when used with another module though, named `IO::Async`::Loop::AnyEvent. That module is not needed to use `IO::Async`, nor is it part of `IO::Async`, but maybe this is what you wanted to refer to in your question.

The history of that module is that AnyEvent sits above `IO::Async` as an event loop, but this module tries to turn the whole thing on its head by putting `IO::Async` on top of AnyEvent, which then sits on top of `IO::Async`. This leads to endless recursion or other obvious or subtle problems, and did eventually lead to bug reports ending up in my inbox.

I mailed the `IO::Async` author about this problem and explained how `IO::Async` can properly use AnyEvent, but never received an answer (the author later publicly admitted that he ignored my mail because he didn't care).

This is why AnyEvent fails when that module is loaded, rather than causing subtle and hard to debug errors later — first, I don't want to deal witht he bogus error reports, and second, I want to protect my users from silent data corruption or endless debugging sessions.

The really ugly and most painful detail of this story, however, is the fact that the author of POE used this to start a public smear campaign against me, fooling people who can't be bothered to look at the evidence — a perl 5 maintainer even asked for CPAN to remove my modules (and later apologised for saying that without actually looking at the code themselves), and this is the only way I can conceive of how this question came to be: To say it clearly, AnyEvent *does not* and *did never* treat `IO::Async` differently than other backends, except where necessary due to differences in the API.

Summary: despite some ugliness, AnyEvent continues to support `IO::Async` as best as it can: two modules that use `IO::Async` cannot transparently coexist with each other without special care, which is up to the unfortunate module user to implement.

As long as you work around this design issue with `AnyEvent::Impl::IOAsync::set_loop`, `AnyEvent` users will work just as magically as with other modules, and `IO::Async` is not treated differently than other backends in any way.

What kind of a "Thank you" do you prefer? Litres or bottles?

I have trouble with any kind of "Thank you", because most likely, whatever it is, I didn't do it for you, so I have issues accepting "thank yous" (mild Asperger syndrome probably).

If you ever are in a position where you want to thank me for a module (or anything, really), don't tell me, tell others about it — if a module was genuinely helpful in some way, help others by teaching them about it. Conversely, if you don't like my code (as opposed to my person), feel free to criticise and "unrecommend" them.

Of course I do like to hear when people use my stuff (when I published `JSON::XS` 2.0, I got a storm of "why did you change the API of this widely used module" mails, when in fact I was under the impression that nobody used it because nobody bothered to tell me about it — maybe I should have added more bugs to receive more feedback?), so feel free to drop me a note on how it was useful (and thank me if you must, but keep in mind that it is easier for me to deal in objective things than in feelings, and don't be angry when I simply ignore your thank you part — I just don't know how to deal with it).

Rest assured that I will continue to contribute whether or not people thank me for it, but I might be more inclined to publish something when it is useful to more people.

7 Tokuhiro Matsuno (January 2014)

How and when did you learn to program?

My father gave me the first computer, when I was 10 years old. It was in 1994 and it was called the pocket computer (http://fr.wikipedia.org/wiki/Sharp_PC-1262). You could use up to 24 characters for representation. The computer could perform a BASIC language. It was the first programming experience for me.

When I became a junior high school student I went to the personal computer club. Back then I studied a language called HSP (known only in Japan). It was the evolution version of BASIC. It was the language designed specifically for game development. However, I did not make a game, I wrote a library.

When I entered the technical college my father gave me Visual Basic. And I studied programming further. Visual Basic could perform Windows programming. It was very easy to write a GUI application. However, I wrote a library for VB, without making any GUI applications.

There were many programming related books in a technical college, I learned how to program from them. I used Ruby and Python at that time and I thought that Perl was outdated. It was around 2000 - 2005. (Many books about Ruby were published by 2005 in Japan).

However, Perl hackers including miyagawa were playing an active part in Japan at that time. I came to think that I would also like to write in Perl.

Now, I am writing modules for Perl.

What editor do you use?

I am mainly using vim. The conditions for which I ask the editor used for programming are the following three.

1. It can perform in terminal.
2. It can be extended.
3. It understands Japanese.

When I tried to find a suitable editor, only vim and emacs existed.

When I am using Emacs my little finger hurts a lot. Therefore, I do not use Emacs.

When and how have you been introduced to Perl?

I was employed at the age of 20. The company was using Perl, and I studied it. I knew Python and Ruby back then. I thought “Perl is a very strange language.” However, while getting used to Perl, I came to thought that Perl was good.

What are other programming languages you enjoy working with?

I am using various languages. I use Objective-C for writing an iPhone application. When I need speed in a library I use C, or C++. In addition to this, I can code in Python, Ruby, PHP, etc. I think that each language has its own applicability. But mosly I write in Perl and C.

What do you think is the most strongest Perl advantage?

Perl is a wonderful language.

The culture which fights for compatibility is wonderful. The program written for Perl 5.8 operates almost without any changes in Perl 5.18. This is a wonderful thing. I do not want to rewrite an application for every upgrade.

I like reference counting in Perl 5. When I use a language with “mark and sweep” it is difficult to tune it. But it would great if Perl 5 supported optional “mark and sweep” like Python does.

I also like the “my” lexical scoping. Explicit variable declaration reduces bugs.

The culture of CPAN is wonderful. There is always a documentation and most of the modules have changes history.

What do you think is the most important feature of the languages of the future?

It is a difficult question. I do not have a good answer.

What do you think about Perl’s future?

I think that the development of Perl 5 will continue. Perl 6 will be put to the practical use. And both languages will enhance and complement each other.

What's the secret behind your energy in writing and maintaining so many modules on CPAN?

Developing software is my occupation. I have uploaded many modules to CPAN from my work. Others are written as a hobby. I like to write reusable modules.

Tell us about Amon2. How is it different from other Perl web frameworks?

Amon2 is very simple, robust, general purpose Web Application Framework.

The difference from Mojolicious

Mojolicious is great and I like the approach. Unfortunately it doesn't preserve backwards compatibility. Amon2 does. Because I think the breakage of compatibility should be performed with changing the namespace. When I decide to throw away back compatibility, I will release Amon3.

The difference from Catalyst

Catalyst depends on Moose, when Amon2 does not. It is because I want my applications to load quickly.

The difference from Dancer

There is almost no difference between Amon2 and Dancer, including Dancer2.

Are UnQLite bindings stable enough? Have you been using it in production? What do you think about the database itself?

UnQLite is very practical. UnQLite.pm is also usable (I myself do not use it in production).

I will explain why I wrote the bindings.

I wanted a practical file-based KVS. Perl core has GDBM_File, unfortunately it is not installed on several OSes. You can't install it from CPAN which makes it useless.

And UnQLite.pm fullfills the following conditions:

1. It is installable from CPAN.
2. It is portable.
3. It is high speed.

You seem to be a fan of as less dependencies as possible approach. Why do you thing this is important?

I dislike a low speed starting scripts. Generally, if there are few dependencies, the load speed of a script will improve. Therefore, I think that a direction towards few dependencies is very good.

There is also another answer.

Software development is multiplication. The whole thing will becomes garbage if one of the dependencies becomes garbage. If there are few dependencies the possibility of the application to become less maintainable will become lower.

Why did you write Minilla?

It is because the starting speed of dzil was very slow. I dislike software with a slow starting speed. Minilla would not be written if dzil were using Moo.

Another reason is "I just wanted to write it".

What is the biggest reason why people should choose `plenv` over `perlbrew`?

Miyagawa described the differences very well:

`plenv` is written entirely in bash (except perl-build, the part which downloads perl tarballs from PAUSE and apply patchperl), and provides "shims" in your `PATH` that locates the right executable of your perl and installed scripts, then calls exec on them.

That's it, and there's no magic shell functions that change `PATH`s before running commands, nor your need to switch perl from one to other, and later forgot that you're running a wrong version of the perl in one shell.

`plenv` allows you to switch perl in 3 ways: `PLENV_VERSION` environment variable, `.perl-version` file in the current directory, then `~/.plenv/version` global default settings.

For more details, look at: http://weblog.bulknews.net/post/58079418600/plenv-alternative-for-perlbrew.

Any future plans for `pvip` and `seis`? Can you explain what those things are?

PVIP is a parser library of Perl 6. It is written in C language and supports about 50% of Perl 6 specification. SEIS is a translator from Perl 6 to Perl 5. At a present stage, this is only a toy.

PVIP and SEIS are interesting projects. However, I needed time for another projects and the development is suspended.

Are there any of your not well-known modules from which many programmers can benefit?

`Test::Power` is very exciting.

When you write the following code in Perl:

```
use Test::Power;

sub foo { 4 }
expect { foo() == 3 };
expect { foo() == 4 };
```

The output will be like:

```
not ok 1 - L12 : expect { foo() == 3 };
#   Failed test 'L12 : ok { foo() == 3 };'
#   at foo.pl line 12.
# foo()
#    => 4
ok 2 - L13 : expect { foo() == 4 };
1..2
# Looks like you failed 1 test of 2.
```

`Test::Power` displays the progress of execution. Inserting manualy useless `diag()` functions it not needed any more.

Are you involved in YAPC::Asia organization?

No, I'm just a speaker.

Where do you work right now? Do your projects require fast processing?

I am an employee of the company which develops smart phone application. I am writing server side application in the Perl language. Our company is developing a lot of applications which 1 million or more users use. It depends on Amon2.

Should we encourage young people to learn Perl right now?

This is a difficult question. I do not have an answer.

Docopt is a nice implementation. But why did you put everything in one file?

It is because the Python version was implemented in one file too. `Docopt.pm` was translated from Python. When translating a code, it is important to do it line by line. Therefore, I did so. After the translation, code can be splitted. However, I do not have a motivation to do it. Patches welcome :)

Do you search CPAN before uploading a new module?

I am referring to http://search.cpan.org before beginning to write a module. It is foolish to already rewrite a certain thing. However, I have already uploaded several times modules that overlapped with a certain thing. That is because the existing module did not fill my demand.

Usually I upload an alternative implementation because of the following reasons:

1. Quality is very poor.
2. Too many dependencies.
3. I don't like the design.

Why did you start writing `tora`, and why did you stop?

I wanted to write a new modern programming language, very similar to Perl 5.

`tora` is a language with the following features:

1. Unlike Perl 5, there are function signatures.
2. Everything is an object.
3. It has scoping like Perl 5.

Development of TORA was interrupted because of the several reasons, like Amon2 and others. I had other interesting projects. Moreover, I needed more time to study Perl 6, and I developed SEIS and PVIP as by-products.

The development process of TORA was resumed the last month and I hope to work on it during the next spring.

8 Randal Schwartz (February 2014)

How and when did you learn to program?

I taught myself very early on, at least 45 years ago. I'm still learning. :)

What editor do you use?

GNU Emacs! I download the latest release from the git development archive each morning and compile it to make sure it still works on OSX.

When and how have you been introduced to Perl?

I downloaded Perl 1 from the Usenet postings as it came by, since I was already a fan of Larry Wall from patch and rn. Played with it a bit, went back to using awk and sed. Perl 2, now that made all the difference... started rewriting *everything* in that and encouraging others to use it. And, for Perl 3, I wrote the Camel book with Larry (although he retitled Perl 3.xx as Perl 4 just as the book was released).

What are other programming languages you enjoy working with?

I'm pretty fluent in Smalltalk, having used it since 1983. I'm getting fluent in Dart, which may be my next language to get famous in after Perl.

What do you think is the most strongest Perl advantage?

As Larry Wall puts it, "the right blend of Manipulexity and Whipituptitude". In simpler terms, it makes the simple things easy, and the hard things possible. Most languages fail at one end of that or the other.

What do you think is the most important feature of the languages of the future?

Ability to solve the Tower of Hanoi. :) Just kidding.

All practical languages are optimized for a particular problem space. I think Javascript is pretty poor for a command-line language, but it works well in the browser environment. (Dart is the first language that I think will live equally well for both.) So there no "most important feature", unless you'll allow the meta-feature of "making it easy to solve problems in the targeted problem space".

What do you think about Perl's future?

Perl's future is doing mighty well. It was stalled a bit in the post-dot-com-boom era, but it has really picked up since then. Perl now has regular releases, more modern features, and increasingly more CPAN uploads year-over-year, meaning more projects are choosing Perl. It's perhaps a smaller slice of a bigger pie, but the pie is getting huge.

What is your favourite JAPH?

```
$Old_MacDonald = q#print #; $had_a_farm = (q-q:Just another Perl
    hacker,:-);
s/^/q[Sing it, boys and girls...],$Old_MacDonald.$had_a_farm/
    eieio;
```

How did you get involved into FLOSS Weekly?

I met Leo Laporte on an InSightCruise (back then called "Geek Cruises"). We became friends and he gave me advice on how to start my "GeekCruisesNewses" podcast, which I ran for about 150 episodes. He was a guest on one of my early shows, and after we were done taping, we chatted about how he and Chris DiBona were starting this "open source" podcast. I suggested myself as a guest, and ended up on Episode 9 of FLOSS Weekly. After 17 episodes, Chris got busy (his wife had had a baby… it happens), and Leo put the show on hold. I asked Leo why the show had stopped, and he explained and said that he was looking for a new co-host. I volunteered, and took over as co-host for the next 100 shows or so. Leo started taking on more podcasts, and confident that I could be the show-runner and host, gave me the opportunity to go solo. I now have a rotating panel of co-hosts, and when I'm unable to host a show, two members of the panel take over the roles of host and co-host.

Does Perl consulting still have a broad market?

It's paying *my* bills. :)

Should we encourage young people to learn Perl right now?

Yes. There's a shortage of Perl programmers right now. I can recommend a couple of good books if they want to get started. :)

Do you write any Perl code right now?

I have Perl code open in the other window of this editor at this very moment. Yes, I write Perl code nearly every day.

What do you think about the current state of Perl?

I'm happy it came back from the post-boom-crash. I was starting to wonder what I would do next in the mid 2000's.

What's your opinion on Perl 6?

In some ways, I'm very happy it is happening. In others, I'm annoyed, because the early effort on Perl 6 could have been applied to Perl 5 to get us where we are now about five years earlier. (I'm probably wrong on that, but it's just a personal gripe.)

I *want* a language that has the features of Perl 6, especially the grammars, infinite lists, and all the other good stuff I'm seeing. But, I'm torn as a service provider about *when* I should start investing in developing talent (in myself and recruiting others) to provide consulting, training, and writing about Perl 6.

What are you thoughts on Modern Perl?

With no disrespect to Chromatic, I'm really just not following all that.

Leaving aside your famous transform, what do you like the most from your Perl discoveries?

I think it was when I discovered (and repeatedly re-proved) that any non-trivial Perl cannot be parsed statically, at least not without having a Perl execution engine available during the parsing. And thus, Perl is relatively unique in that space. (For more details, see my posting at http://www.perlmonks.org/?node_id=44722.)

9 Christian Walde (May 2014)

How and when did you learn to program?

That actually started very early, when i was given an Amiga in 1991, at the age of 8 years, and started messing around with its CLI and ARexx. Not much but playing around came from that, but i learned more later on, especially due to the games Clonk and Starsiege: Tribes, both of which implemented a lot their game logic in a C-like scripting language and were hilariously customizable. (Tribes had addons released that allowed the player to play Tetris while sitting on a hill waiting for sniping targets to come by.)

Later on in College i did PHP in private projects, while also doing C++ and Bash courses. In February 2005 i was then asked by random chance on IRC to help someone fix their forum (it was an affair sort of like 4chan) because they had managed to trash their SQL database. Luckily i had by that point mastered the fine art of reading error messages and googling them. In short order i was digging around in my first perl software and quickly got the site back up.

As a result of that i was handed the root password for that machine, which i still have, and had at my disposal a website with 20000 visitors daily, written entirely in Perl and MySQL. Using that and the help of nrr (twitter.com/nrr) on IRC i started learning Perl. About three years later i wrote the first Perl software i got paid for (a small tool to help in the translation job i was doing then), and another year later i was hired for my first serious Perl developer position (with a bioeng group looking to analyze and improve potatoes).

In short: I learned programming my entire life from the point i was given my first computer and learned the most of it in the least time when i made heavy use of IRC.

What editor do you use?

Mainly Komodo IDE. It is my main Perl editing tool because it combines in one single package a fully capable editor, a visual debugger for Perl and other languages (think Visual Studio), PerlTidy, PerlCritic, Perl syntax checking, a Regex debugger, a project manager, grep and most importantly, sane defaults. I can get many of these in other editors too, and have seen other people with these tools in their editor, but for me Komodo remains the only choice because it has all these things out of the box and requires no tweaking to be useful and only minimal tweaking to be comfortable.

In tandem i do also however use Notepad++, as it is very lean in memory and CPU, remembers everything opened in it even through crashes and simply provides the fastest low impact tool for simply munging a text file. Incidentally it also has the best syntax highlighting settings for Perl, which i've since then copied to all of my Komodo installations and adapted for every other language i edit often.

When and how have you been introduced to Perl?

Err, see above. :)

What are other programming languages you enjoy working with?

I do enjoy working with Javascript because it can do some useful things and there is no happier time in my day as when i am done with a JS file and can finally put that abomination of a language out of my mind again for a while.

Aside from that i do enjoy dabbling in graphics programming which means messing around with C and GLSL. I don't think i need to lose words about C, but GLSL merits special mention since it is a language that is superficially C-like, but requires special thinking since every program written in it will run on hundreds of microcores, not only in parallel, but in lockstep. That means things like if/else trees waste processing time instead of saving it. That pleases me.

What do you think is the most strongest Perl advantage?

As for the language, CPAN. It is the highest quality third-party library archive available for any language, due to the massive ecosystem of quality assurance tools that have sprung around it any which seek to prod the authors to make every library the best it can be by mercilessly pointing out and collecting their problems.

As for the perl executable itself, its biggest strength as far as i can tell, is that almost all of its features are implemented in a way that's convenient enough to do basic things when you have only the executable, but still low level enough that it is easy for CPAN modules to make the language amazingly easy to use. (See: Moo, IO::All)

What do you think is the most important feature of the languages of the future?

Humanity and parallelity.

Perl is already really good at the former. It is a language that is not a calculator. It is a language you read, write, speak and think in. Not only that, most of its uses come intuitively if you know enough English. Languages need more of that. Language designers need to make programming languages feel more natural to humans. Average humans.

Parallelity is the second bit. Technology is running into a wall when it comes to CPU performance. It's become hard to force single cores to be faster. That means we will need to learn how to distribute tasks of a single program to multiple cores, which means in turn that languages need to make this easy. I hear Go does it well, i know Java does it really well. However scripting languages like Python and Perl quite dropped the ball on that and need to catch up.

What is wrong with language popularity indexes? And how can we fix them?

At a very high-level layer: They do not answer a question that is useful to humans. One might ask "Which language is more popular?", but why does one ask that? Does the asker with to know which language will provide them with the biggest paycheck? The most secure paycheck? Which language will put them at the front of innovating technology? Which language will leave them happiest in their day-to-day work? These are the things people care about, but which popularity indexes do not answer.

Further, most people are not properly equipped to properly read and understand the output of popularity indexes. Simply looking at Google Trends requires enough knowledge of statistics to understand why a line going downwards can still mean that the number of users of that language is growing. Similarly, looking at TIOBE results requires understanding why languages with names that are similar to other things do better exactly because of the method TIOBE chose to prevent this similarity from skewing results. (To be clear: Their method simply creates a different kind of skew.)

How they can be fixed? Honestly, i don't think they can. The people who create them and use them to argue points are happy about the fact that they are inaccurate because it permits them to claim things that make them happy, or more perversely, cause people to click their links a lot and increase their advertisement income. To fix them requires fixing the entities they are using to generate their numbers and in practice this seems to be like tilting at windmills.

The best thing i can tell people who are concerned about this thing is what Getty said in his talk Moving the Needle (https://www.youtube.com/watch?v=VFJ672tpRG0): Use your time not for useless things, but to make things that you want to use and which others will want to use.

Why do you still use Win32 and how is Perl doing on the platform lately?

As mentioned with the editors: Sane defaults. But also: The Amiga.

I started as a child on a computing platform that already provided almost everything the modern Windows desktop has. (http://www.gregdonner.org/workbench/images/wb_13.gif) Unlike people who started on *nix or DOS i had as input devices both the keyboard and the mouse available to me from the very start and to this day prefer environments where for every task i wish to do, there is a way to it with both the keyboard and the mouse, allowing me to things in the manner most convenient to me at the time of doing them.

As for sane defaults: Windows has, since Windows 2000, had remarkably little change in the actual user experience. While the colors may have changed and machines became more powerful, little has changed about the way things are used. That means many app developers follow similar conventions. Additionally, due to a little bit of perversity in the human mentality, closed source development seems to result in software with slightly more sane defaults. On *nix developers often seem to take the tack of "Want this? Well you can implement it on your own.", thus absolving them from making the software fit the needs of others, while the authors of closed source software do not have this shortcut available.

As for how Perl does on Windows: Perfectly fine. There are many people who mainly use *nix-based platforms and have little practical experience with Windows or Perl on Windows, but will gladly spread their thus-gained prejudices as fact. In practice however almost all CPAN modules work fine on Windows, and the only occasional breakage comes from someone forgetting to mark their binary data streams. Nowadays all Perl distributions for Windows come with their own compiler, making it effortless to install XS modules, and will often also come with trickier modules pre-compiled. With ActivePerl, Strawberry Perl and DWIMPerl there is plenty of choice for those who need a Perl for their corporate environment, those who need a basic open source Perl and those who would like to have (almost) all the modules already installed out of the box.

Why did you start perl-tutorial.org? What is its main purpose, why was it needed?

In the summer of 2011, the first hit on google for "perl tutorial" was a Perl 4 tutorial. The rest of the results were of similar quality. This meant that people learning Perl had an extremely high chance of learning outdated Perl and missing out entirely on all the positive changes Perl as a language has had since 2000.

However the correct solution was not to write a tutorial. We had gotten into this situation because Perl has been around a long time and older tutorials have considerably more google juice that modern and recent tutorials. http://perl-tutorial.org fixes that by being a publically editable wiki that tracks tutorials and puts the most recent ones at the top, maximizing the chance that even in ten years someone clicking on that link won't find a tutorial aimed at Perl 5.12.

Are you playing any role in organizing German Perl Workshops?

Only incidentally, honestly. We have an active Perlmonger group here in Hannover (which was started by finding other Perl developers from Hannover at YAPC::EU 2012), and some of them were really gung-ho about the whole thing after being at the GPW 2013 in Berlin, so they decided on the spot to host the next one in Hannover. The other people in the PM group did the bulk of the work like finding the venue and with help of Frankfurt.pm setting up all the legal details and doing the talk time planning, while i mostly did odds and ends, like helping out at the venue, or sponsoring 102 plush camels for the goodie bags for attendees.

Where do you work now? How much of your time are you writing Perl code?

For the past 2.5 years i worked for Profihost.com here in Hannover, mainly writing the code for the backend of their website and guiding other developers at the company while they learn Perl. Recently i've switched to working freelance, still doing work for Profihost, but also looking for other work to spice things up.

That covers the for-pay part of my day. The rest of the time i spend on open source projects, either fixing things on CPAN that i see broken recently on my smoker, or implementing things on CPAN modules that i want to use, like the recent improvements to io layer handling in IO::All. I do however also spend a lot of my time on private projects, like this 3d viewer for Dwarf Fortress (https://github.com/wchristian/lifevis) (https://dl.dropboxusercontent.com/u/10190786/DF/perl%202012-06-12%2003-07-39-88.jpg https://dl.dropboxusercontent.com/u/10190786/DF/perl%202012-06-12%2003-08-30-52.jpg), this OpenGL shader toy (https://github.com/wchristian/perl_shader_toy) (which makes things like this: https://dl.dropboxusercontent.com/u/10190786/your_perl_on_drugs.avi) or more recently https://github.com/wchristian/Microidium a WIP multiplayer asteroid-like game, inspired by the excellent Luftrausers; which also allows me to work with SDL and build on a game framework which other people can use to make games.

In short, pretty much ALL of my time is spent on Perl.

Should we encourage young people to learn Perl right now?

Absolutely. Even if they won't end up using Perl in their work life later on, whether by choice, or by circumstance, they will end up being exposed to concepts that no other language possesses, especially if they learn Modern Perl.

Do you enjoy Perl workshops and conferences?

Immensely. Even if i don't learn anything new from the talks (i do), even if i don't meet any new people to connect with (i do), even if i don't meet newbies whom i can teach a lot of new things (i do), even if the evenings in the pubs aren't invariably amazingly fun (they are), i still go home from one of these with a page full of notes about things to do and look into, and most importantly: Extremely motivated to do things. There is nothing that motivates me more to get some code done, than to spend a weekend at a conf or workshop.

What is the best beer for connecting Perl developers?

Without a doubt belgian beer! I come from Germany, which is said to be the country of beers, but to be honest, all people make here is boring bitter water. I hated beer until i learned about belgian beers and found subsequently that they are the optimum for social gatherings like this, since they cater to every possible taste. Even if there's someone in your group who normally doesn't drink, they might be glad to experience the almost non-alcoholic fruit beers; for the finns or other heavy drinkers, there are the triple-brewed 10%+ Gulden Draak 9000. In between there's a variety of beers of different tastes, including the Gueuze style beers, which are sour instead of bitter.

That said, if you don't have those available, the british palette with plenty of Ciders is almost as good.

10 Florian Ragwitz (rafl) (June 2014)

How and when did you learn to program?

I still am and I don't think I'll be done any time soon.

They say you never forget your first, but apparently that's a lie. I've been dabbling around in programming starting from around the age of 10, but I can't quite remember what initially sparked my interest.

I do however have fond memories of a certain "children's learning computer". It had the form factor of a laptop, but was really just a children's toy. It had a monochrome 4x20 character display and came with a couple of educational programs such as various word and maths puzzles installed. A lengthy internet search seems to suggest it was called "V-Tech Genius Leader 4004 Quadro L" (http://www.pinterest.com/pin/146930006562843060/) and apparently only sold in Germany. In addition to the various learning games it also had a BASIC interpreter and the manual included a brief introduction to the language. I have no recollection of what I programmed with that device, but I do remember being fascinated by BASIC and spending a lot of time with it.

A less fond memory is of the i486 my family used in their business for some time. I enjoyed playing games and learning about Windows 3.1, DOS, and batch file commands, but I got into trouble a few too many times for breaking the whole system and my family having to call support to be able to do their work again.

The next programming related memory I have is from some time later during secondary school. We were all using Casio graphing calculators of the CFX-9850G series (http://en.wikipedia.org/wiki/Casio_9850_series). It was programmable through a dialect of BASIC and a great distraction during the more boring classes. I recall writing a couple of programs I'd use in maths or physics class to solve certain problems, but most of the time I'd be writing games for it. One of my proudest programming achievements of that time was a graphical 2 player chess game. Sadly, both players had to play on just a single calculator. To this day I blame a bug in the implementation of the `Send()` and `Receive()` commands of my calculator model for the network multiplayer version never fully working. My attempts at writing a chess AI on that device were never successful either.

I've also had some exposure to languages like Logo and Pascal in school and I tried to learn about Visual C++, Visual Basic, and Java once I had my own personal computer at my disposal. None of those things really stuck, though. It wasn't until I stopped using Windows that I've started programming on a regular basis to solve actual problems I was having. Most of my programs from that time were quite simple and written in shell. Some of the more complicated ones were written with awk, PHP, and Perl.

What editor do you use?

Both!

Quite some time ago, probably around the year 2002, I've switched from Windows to Linux as my primary operating system. The first text editor I've been introduced to as part of that switch was Vim. A little while later, after I had figured out how to exit the editor and had gotten a chance to read some of its documentation, I had grown quite fond of its modal editing and expressiveness.

Vim is a great editor and even today I still use it on a regular basis. However, my primary editor these days is Emacs.

I had been wanting to switch from Vim to Emacs for a fairly long time. I found some of it's features and extensions quite intriguing. I spend a lot of my time in org-mode (http://orgmode.org) and probably wouldn't wanna switch to another editor that didn't provide similar functionality. Being able to use different fonts within the same buffer is very helpful in editing structured text like LaTeX documents and Perl POD - having `=head1` headings displayed larger than `=head2` headings, for example, makes the document's structure a lot more clear to me.

In addition to that, I also have a tendency to spend too much time configuring and fine-tuning my most commonly used tools, text editors included. I'd much rather spend that time writing Emacs Lisp than Vim Script.

Switching from Vim to Emacs wasn't easy and it took me several tries. It went surprisingly smooth, though, after I had finally stopped using Vim to edit my Emacs configuration and forced myself to use Emacs for everything with something along the lines of `alias vi=emacs` in my shell configuration.

I'm occasionally trying other editors and IDEs and am excited about how easy they make certain things that'd take me an hour or longer to configure in Emacs. I'm looking forward to switching editors again in the future, but I have yet to find a new editor with enough distinguishing features to make the pain of switching worthwhile.

When and how have you been introduced to Perl?

I vaguely recall me getting interested in Perl when I was doing web programming with PHP a lot, still during secondary school, probably around 2003. I might've come across it in a UNIX book I was reading, or perhaps I've stumbled upon it searching for an alternative to the PHP language that I didn't feel particularly happy with at the time.

Version 5.8 was recent at the time I started using Perl. It was a marvelous language compared to what I was used to. Lexical scope alone was worth switching to me. PHP, at the time, didn't have namespaces, closures, or even anonymous functions. Perl had all that, and then some.

I would still sometimes write the occasional program in PHP due to how easy it was to deploy the result, at least when compared with Perl and other languages at the time. Perl excels in many things, but ease of deployment still doesn't seem to be its strong suit, albeit the many improvements it has seen since when I started using it. That makes me a little sad.

What are other programming languages you enjoy working with?

Most of them.

Almost every language I've encountered so far has something unique and interesting about it. I feel that with every new language I learn about, I become a slightly better programmer. Many languages seem to impose a certain world view onto their users, but rarely, if ever, is that world view the only valid one.

Exposure to different ideas as expressed in different programming languages often makes me reconsider my preconceptions about those ideas. Many dynamically typed languages being popular currently and some older static type systems, such as C's and Pascal's, having lots of practical problems doesn't make static typing a bad idea. Threads aren't bad because Perl's threads are bad. Functional programming paradigms can express many problems very naturally, but yet there's problems that are much more elegantly solved with object-orientation.

There's many of these somewhat contradictory concepts in the field of programming. Most of the time, though, I feel they aren't actually mutually exclusive. Typically each option has its own merits that have to be considered in the context of the problem you're trying to solve. You're not going to be able to do that if you only know one of the options because that's the only one supported by your programming language.

Therefore, I try to learn and use a lot of languages whenever I get the chance. The ones I'm currently most fond of include Haskell, F#, OCaml, Go, and Scala.

What do you think is the strongest Perl advantage?

Historically Perl had a lot of advantages over many of the other language at the time: it's "manipulexity" and "whipuptitude", the support for many different programming paradigms, it's portability, the great support for text processing, it's culture of testing, and so on. Since then, however, it appears that other programming languages have caught up.

At first I was gonna answer this question saying CPAN would be Perl's strongest advantage. CPAN most definitely is one of the main reasons I continue writing Perl code, but even in that area other languages seem to have made huge leaps forward and CPAN probably isn't the leader of the pack anymore, at least as far as the comprehensiveness of the module archive is concerned.

One thing Perl still has going for itself is its, to my knowledge, unparalleled malleability. Many things usually considered to be impossible in other languages are quite doable in Perl, even though doing so might be a bit involved. I would like Perl a whole lot less if it weren't possible to write modules like Moose, `List::Gather`, or `Scope::Escape::Sugar` for it. On the flipside, modules like that tend to just add language features that are already available in one form or another in various languages. I don't know if that's a good or a bad thing.

What do you think is the most important feature of the languages of the future?

I can't think of a single most important feature I'd like future languages to have, but I'm hoping those languages are going to liberally steal from existing languages such as Lisp, Haskell, and Scala. I feel there's many language features in those that haven't quite made it into the mainstream programming world just yet.

I hope I won't have to program in languages without pattern matching and destructuring binds for much longer. I also find the prospect of automatic parallelisation of operations for which it can safely be done quite exciting. I'm also looking forward to see static type systems evolve further so I won't have to debug quite as many of my dumb mistakes because the compiler will tell me about them upfront and reject the program.

Another area I'm hoping to see improvements in is interoperability between different languages. Many languages now sharing a common run-time environment like the JVM or CLI is a great step forward for code reuse across language barriers, but I think there's still a lot of room for improvement. I don't believe there can be a single language that's good at solving every kind of problem, and I'd like to have more freedom to mix languages as I see fit.

Do you write CPAN modules because you lack functionality or as a proof that it can be done in Perl?

Most of the modules I wrote and the contributions I made to other people's modules were motivated by actual problems I've been having. Almost all of the time I'd contribute to existing modules trying to fix bugs I encountered or to make them more general in order to solve my specific problems. Only rarely, for example in the case of List::Gather, have I had to write new modules because the existing ones just weren't good enough for my purposes.

Most of the new modules I wrote were provide functionality I wanted to use that was not previously available on CPAN at all. A lot of those seem to just be bindings to existing C libraries I wanted to use.

The only module I recall creating as a proof something could be done in Perl was my signatures.pm experiment.

How did you end up releasing perl distributions?

I don't quite recall how exactly that happened, but I did my first release of Perl in 2010, during Jesse Vincent's reign as a pumpkin. I imagine he either asked me to do a release or I volunteered to do so, probably on IRC. Back then the new release process establish by Jesse was already pretty good and shipping a new version of perl turned out to be surprisingly easy.

My favourite releases remain 5.15.4, which I got to release on Larry Wall's sofa, 5.17.5, which I was able to release live during a talk about Perl release management at YAPC::Brasil, and 5.14.2, the only non-development version of perl I've ever shipped.

How does one start contributing into perl core?

I don't have a good way to recommend, but what I did was scratching my own itches.

It appears that my first core commit was in 2008. Back then given/when and smartmatch were still new and exciting to me, and apparently B::Deparse, which I use a lot when working with Perl, didn't seem to handle that new construct very well yet. That bugged me enough to go ahead and fix it, and it turned out to be super simple - it was a one line code change in the B::Deparse module.

Many of my early commits followed the same pattern. At some point I discovered that I couldn't easily combine perl's `-E` and `-p`/`-n` options when using the `perl -n '} END { ...'` pattern in oneliners. A tiny 3-line patch later it worked. Whenever I'd find a typo or inaccuracy in the documentation I'd submit a patch. It felt very rewarding.

In general, there's nothing magic about the perl core (well, maybe with the exception of `sv_magicext()` and related functions). A lot of the perl core is written in perl itself - the same kind of perl code you deal with every day. There's absolutely no point in being intimidated by it. I feel that almost every Perl programmer would be able to contribute to the core of their language in one way or another if they set their mind to it.

Does perl have enough volunteers to keep its code evolving? What are the current most important needs?

I'm not currently worried about Perl not having enough volunteers to maintain and improve the language, though there's certainly room for more.

I don't know what Perl's most important need right now. For anyone wanting to start contributing, the perlhack documentation and the Porting/todo.pod that are part of the perl code distribution should contain plenty of starting points. I'm sure the perl5 porters community would also gladly discuss ways for you to contribute if you approached them on the p5p mailing list, the #p5p IRC channel, or otherwise.

Personally I'm hoping to see alternative implementations on Perl that are able to run in different environments such as on top of the JVM, the CLI, or perhaps LLVM.

Projects like `Compiler::Lexer`, `Compiler::Parser`, and `gperl` seem like a step in the right direction, but it seems to me that there's still a lot of room for improvements in that area.

What do you do for Debian?

Not much anymore. I used to maintain a relatively large number of packages. Most of them were Perl-related, but I also packaged a couple of non-perl things such as the XMMS2 music player and various clients of it. These days, most of my former packages have found new maintainers that are doing a great job at keeping them up to date and in working order.

Where do you work now? How much of your time are you writing Perl code?

I work at a small IT consultancy called Infinity Interactive (http://iinteractive.com). There I get to work with whatever tools are appropriate to fix our partner's problems. Sometimes that's Perl, and sometimes it's something else. I quite enjoy the technical diversity of my work.

As of late I've been working mostly on systems automation tasks. Application programming in Perl is currently about 10% to 20% of my work time.

Should we encourage young people to learn Perl right now?

I don't think Perl is one of the better languages for teaching and learning programming. I'd probably recommend different languages for a newcomer's first foray into the field.

Nonetheless, I believe there's many good lessons to be learned from studying Perl which are gonna be useful in a programmer's life, no matter if they're gonna use Perl to solve their problems or not. In that sense, I feel that encouraging developers, young and old alike, to learn about Perl is generally a good thing.

How do you notice that your talk is too hardcore for an average developer?

Sadly, I usually don't.

How did `Acme::rafl::Everywhere` started?

That happened during YAPC::NA 2012 in Madison, Wisconsin. On my first day there I joined a group of Perl folks organised by Dave Rolsky for an (anti-) arrival-dinner. Also amoung us were Robert Blackwell and Sawyer X.

I don't quite recall how the conversation got there, but at some point Robert pointed out how I appeared to be present in a lot of different communities and/or projects. I believe he alluded to the relatively large number of very diverse CPAN modules I maintained at the time, me being a contributor to many non-Perl projects such as Git, Linux, XMMS2, and Arduino, and possibly also the fact that I had visited all of the 5 YAPCs as well as many international Perl workshops in that year.

At some point, the phrase "you are everywhere!" was uttered, and Sawyer just ran with it. A lot of the jokes in `Acme::rafl::Everywhere` originate from that dinner. Later on, once Sawyer had published the module shortly after the conference, a few other silly people sent pull requests to add additional facts.

Apparently there's even a "!rafl" bang command on DuckDuckGo.com these days.

I feel ambivalent about all of this.

How often do you go outside?

If the slight sunburn on my forehead and neck right now is any indication - too much.

There's many exciting things going on in Brooklyn and New York City in general, and sadly most of them don't happen in my home. In order to be able to see those, I'm sometimes forced to go outside. That's usually a sacrifice I'm willing to make. Also the selection of beers on tap in my home is quite limited, so I often have to resort to working from various bars and beer gardens in the area.

11 Curtis “Ovid” Poe (September 2014)

How and when did you learn to program?

I first learned to program back in 1982, when a geometry teacher let me play with the computers in the lab, even though they were for older students. I taught myself BASIC and by 1984, I had written a relatively full-featured (for the time) graphics editor for the TRS-80, but the graphics were so low-res that any image took several screens (and files) to build and you could dump them out to dot matrix printers. Students in the computer lab lost a lot of time playing with it. A couple of years later, I was writing a text adventure (for fun, I still didn’t have a programming job) and I remember that my test sentence took eight seconds to parse in BASIC. That sentence was “hit the skeleton with the sword, take his ring and then walk north and touch the alter.” I taught myself assembler to speed that up and later learned C. Sadly, because I lived in a small town and had no degree, I stopped programming entirely by 1988 and only programmed sporadically until 1998, when I got my first real programming job. I’ve been doing it full time ever since.

What editor do you use?

I use vim, but it’s due to muscle memory, not because I have any particular feeling that vim is necessarily better or worse than other editors. I was forced to learn to use vim at my first Perl job in 2000. I hated the damned thing and kept working through the tutorial. Today, I have a heavily customized vim setup which is fantastic, but makes it very hard to pair program because I can navigate a code base so fast in vim that other developers can’t follow what I’m doing. Or I change a line of code and suddenly, tests are running in the console and I’ve never left vim. Or I hit `<leader> rb` and suddenly my entire project is being rebuilt. Or I select a range of lines and bam, extract method! I’m fast at refactoring and cleaning up code, but it’s mainly because of all of the tools I have built into vim.

Curiously, the fact that my vim setup is so powerful might be a drawback in the “text editor versus IDE” debate. I don’t consider myself particularly talented in vim, but nonetheless it’s sometimes painful to watch other developers use the editor. In my years of experience, I’ve found that most developers never learn to use their editor properly. Thus, a standard, generic IDE means that while there may be some bloat (I’m looking at you, Eclipse) and lots of features that are never used, developers can easily share tips and tricks by simply pointing out shortcuts or menu commands. My environment is heavily setup for how I program and it would be hard to share or explain. It optimizes me, but it doesn’t optimize the masses. I suspect an IDE could be better for that. I actually tried to make a pared-down version of my vim setup because other devs kept asking for it, but it was a mess.

When and how have you been introduced to Perl?

In 1999, I was working as a COBOL programmer for an insurance company and befriended one of the Unix admins. He kept telling me to check out this thing called Perl. I was trying to fix a problem with a COBOL program that was converting a CSV file from an NT system to the fixed-width format that COBOL prefers. COBOL has many weaknesses and working with text is one of them. The code was 150 lines long, but that’s because the author didn’t understand how the COBOL’s unstring function worked. I got it down to 80 lines of COBOL. Out of curiosity, I tried it in Perl and got it down to 10 lines of clean code, and that included error checking. When that Unix admin left to start his own company, I happily jumped ship from COBOL to Perl.

Ironically, many Perl devs write Perl like C, but I started writing it like COBOL (that’s too long of a story to explain). It was the Perl community which happily embraced me and taught me how to really program. I’m very grateful for that.

What are other programming languages you enjoy working with?

My favorite would be Javascript. Writing things like my 3-D rotating star map brings me silly amounts of pleasure in areas that Perl just isn’t well-suited for. I’ve also had fun writing Python and Ruby, but sadly, I’ve had Perl on my CV for so long that companies willing to offer me contracts in other languages simply won’t pay as much as I can earn with Perl.

What do you think is the strongest Perl advantage?

Perl’s strongest advantage is the community, hands down. I’ve watched other language communities struggle with issues that the Perl community dealt with a long time ago. It’s a bit older than other communities and I feel there’s more of a sense of maturity in the community than I find in other languages.

Other than that, I would say the Unicode support is phenomenal and while the CPAN does have a lot of cruft, once you know your way around it, there are many excellent modules out there that turn hard programming problems into trivial ones.

What do you think is the most important feature of the languages of the future?

Concurrency. We're not really facing it yet, but Moore's law is coming to an end in a decade or less. If we want faster systems, a robust concurrency system is imperative. Forking is often too expensive for many smaller programs and threads are a nightmare (how many experienced developers can tell you the four conditions necessary for threaded code to lock?). I recall Jonathan Worthington describing threads as the assembly language of parallel programming and as a former assembler programmer, I have to agree. All it takes is one tiny, tiny overlooked bug and you have a nightmare of a hard-to-reproduce problem on your hands. There's a huge amount of work being done in better concurrency models and we're going to need them in the future. If chips can't get faster, we have to take advantage of concurrency if we want to see those performance increases.

(Though there's a small part of me which feels that CPUs will become so cheap and widespread that many devs will still have plenty of work just cranking out standard code to get stuff done, no concurrency required)

How did you like the experience writing Beginning Perl? What was your biggest motivation?

When Wiley first contacted me about writing the book, I was going to turn it down. My wife, Leïla, had just given birth, we had just moved to Amsterdam, and life was chaotic. However, Leïla was insistent that I write it, telling me that she'd take care of our daughter. It was hell for both of us. I spent eight months writing what I think may be the largest Perl book written by a single author. My technical editor, chromatic, did a heroic job catching all sorts of issues in the work, but sadly, typos crept in. I'm still happy with the book and the reviews have been fantastic, but I wish I had more time to fix the typos.

Here's a little inside story that I don't think I've shared before. On one of my early drafts, I made a joke about terrorism (in a programming book!) and my editor asked if I thought it was appropriate given today's political climate. I simply replied "yes." Generally speaking, I defer to the editor's judgement, but in this case, I put my foot down. Shortly thereafter he was assigned to work on another book and while I was told that it was merely an internal reorganization, I always wondered if he was so offended by my sense of humor that he didn't want to work with me. That leads me into giving a shout out to Mary James, Maureen Spears, and San Dee, three incredibly awesome people at Wiley who worked with me on the book. If it weren't for them, I doubt what little was left of my sanity would have remained intact. I can’t speak for other’s experience, but given mine, I’d happily work with Wiley again.

Are Perl books still popular if judging from the purchased amounts?

That depends on how you define “popular.” I’m happy with the sales of mine and given what I know, my book is doing better than some others, but that being said, there does appear to be a decline in the Perl book market in general. Because I do a lot of freelance consulting and training through http://www.allaroundtheworld.fr/, I see that Perl is still a popular language and new projects and companies are springing up and using it all the time, but there’s been a public perception that Perl isn’t as strong as it was. I think that’s hurting the book market.

What was the reason to write yet another testing module for Perl? How is it different from others?

I’ve written many of them, but I’m guessing that you’re referring to `Test::Class::Moose`. I originally wrote it to be an xUnit replacement for Adrian Howard’s venerable `Test::Class` module. Unfortunately, the latter is showing its age, while `Test::Class::Moose` is designed to be a modern xUnit framework. What surprised me is that more and more companies are starting to adopt it because it’s so powerful. Once you get used to it, it’s easy to use, but it’s truly an “enterprise ready” test framework.

Out of the box it gives you faster test execution, easy-to-use parallel testing (including being very easy to write parallel schedules for), reporting, slicing-and-dicing of test suites (only run tests that are relevant to what you’re working on), tools for migrating from `Test::Class`, running tests in different contexts (thanks Dave Rolsky!), and complete `Moose` integration for powerful OO testing capabilities. I still remember demonstrating it for a company in the US and they immediately agreed to adopt it when they saw they could trivially run subsets of tests based on differing customer requirements.

That being said, I’ve toyed with the idea of writing Testament. This would be a successor to TCM which would solve two of the biggest problems: it would stop people from thinking that it was only useful for `Moose`-based code, and it would allow the company to choose what testing modules they wanted to bundle, rather than simply accepting `Test::Most` as the default.

Why do you think people should have experience living and working in different

countries? What are the benefits of being an expat?

It’s easier to hate someone you don’t know. I currently live in France and I must say that most of the stereotypes of the French are dead wrong. They’re not arrogant. They’re not lazy. They’re not dirty. They’re a warm, rich people with a fantastic culture and a deep appreciation of great food and wine (and, well, bureaucracy). In fact, I’ve lived in five countries and every one is different from what outsiders think. The main thing you learn, if you get to know the locals, is that people are, well, people. We’re not “Americans”, “French”, “Japanese”, “Iraqis”, or anything like that. We’re people. When you’ve lived in as many countries as I have, you learn that most people are inherently good and it’s the unfortunate tendency of the news to report on the bad stuff that really warps people’s perceptions.

Being an expat is one of the most rewarding things any person can experience, particularly if you find yourself long term in a country whose common language is different from your own. Even though we’re all fundamentally the same, the way it’s expressed is different from culture to culture and when you experience that first-hand, you can have a beautiful understanding of why people are the way they are. I wholeheartedly recommend it.

Among many things you’re doing the recruiting job in a special way. Can you tell us how is your way of doing it differs from the common one?

There is a very low barrier to entry for recruiters. All you need to do is say “I’m a recruiter” and bam!, you’re a recruiter. This pushes quality down and prices down. Companies tend to spend very little on recruiters and then complain bitterly when they get what they pay for. Recruiters, in turn, often just scan for keywords, have no understanding of the position they’re recruiting for, and then push the CVs to the companies. Wash, rinse, repeat.

We don’t like to do that and we’ve turned down recruiting contracts asking for us to just throw CVs at them. We like to work with companies who want a recruiter to send them quality candidates, particularly for hard-to-fill positions, and are willing to let us do the job right. That entails understanding the hard skills the company needs (the measurable technical skills), but also the soft skills (work with deadlines, convince colleagues, handle rapidly changing requirements, and so on) necessary to fill a role properly. We actually interview the company to start this process.

When we’re ready, we start interviewing candidates. We first filter CVs. If they get past that, we generally administer a technical test — the vast majority of candidates wash out at this point — and then we assess their soft skills using what is called a “structured interview”. Unfortunately, they’re not well known to hiring companies, but unlike your typical interview (which isn’t much better than random chance in assessing good candidates), structured interviews are excellent predictors of whether or not a candidate will be successful in a given role. They take more time to learn, to conduct, and to assess, but it means that when we hand a candidate to a company, their final interview can be little more than “would I want to hang out at a pub with this person?” and there’s much less risk.

I might add that this works well with our specialty in international recruiting. If your company is in a small town, such as Parthenay in France, you might find it very hard to attract French workers to it. In fact, even though France is part of the EU, EU citizens moving to France typically want to move to cities such as Paris, Bordeaux, or Nice. Who wants to move to the middle of nowhere? Well, a company in Parthenay whose willing to open themselves up to the global labor market is now flooded with candidates who would be delighted to live in France, even if it’s a small town. That’s where we come in: we tell the company how immigration works, how they can sponsor employees, how the European Blue Card works, how relocation is handled, and we handle the recruiting process to ensure that candidates moving to France are already prepared to be integrated, plus are excellent choices for the company to hire. It’s a huge amount of work, but I love being an expat and it makes me very happy to see others realizing their dreams of living abroad.

What are you thoughts on Perl’s future?

Perl is still doing better than many people think. It was incredibly popular in a space that was ripe for competition, and now that competition has arrived, its market share has declined. Unfortunately, many people who don’t understand economics have misunderstood. When there’s a low barrier to entry in a hot field, competitors will arise. This means, the leaders often have a reduced market share, but those who don’t understand economics see the leaders as failing, rather than competing.

Currently Perl is still one of the fastest dynamic languages, has magnificent Unicode support, and a thriving community. Part of what’s hurting has historically been two things: lack of method/sub signature and lack of a decent core OO system. With Perl 5.20’s new signatures, we now have fantastic signatures. And Ricardo Signes has said that if he gets a decent MOP (metaobject protocol), he’ll put it in core. With that, two of Perl’s biggest limitations will be gone and our OO system won’t be on par with other dynamic languages: it will leap-frog them entirely. At that point, I feel some of the biggest technical issues will be behind us, though marketing is still something the Perl community hasn’t done well.

That being said, our marketing push over the last few years has seemed to revive the morale of many in the community. Now is the time to push it beyond the community.

Should we encourage young people to learn Perl right now?

We should teach them to program. While I’d be happy to see more young people learning Perl, in reality, once you’re a competent programmer, you should be able to pick up most languages. Don’t tie yourself to a single technology.

Will you write another book on Perl? Or maybe other language or technology?

Right now, I don’t know, but I have no plans in the works. I’ve thought it would be fun to write “Perl for Crimelords”, showing a novice programmer using Perl to improve the efficiency of a criminal syndicate. It would certainly get noticed, but perhaps not in the way we would like. That being said, I’m so busy with my company, being a husband and father, and books pay so poorly that I don’t anticipate getting around to it.

How to find a job overseas?

This would take far too long to cover here, I’ll punt and and direct people to the “Start Here” page on my expat blog.

12 Leon Timmermans (October 2014)

How and when did you learn to program?

The first language I really learned was Javascript actually, in a time when hardly anyone was using it for anything serious (around 1999 I think). Ironically everyone is using it nowadays, and I haven't touched it in years. After that I picked up C, Java and Perl, though I don't remember the exact order anymore (they were in close succession).

What editor do you use?

Vim. I picked it up when I was just starting with Linux and never looked back at anything else (I even used it on Windows).

When and how have you been introduced to Perl?

I think around 2000-2001 I picked up a Perl book and started playing with it on my Linux machine (that came with 5.005004). It wasn't until 2008 that I started uploading some of my stuff to CPAN, and a few months later I attended my first perl workshop. I've been hooked on the Perl community ever since.

What are other programming languages you enjoy working with?

I rather enjoy C++, specially now that C++11 (and since this summer C++14) is out. Much like perl, it has a reputation of being old and outdated, while when you look at it it's vibrant and in active development. And much like Perl it's a language that treats you like an adult: it doesn't tell you how to get something done, but provides you with plenty of tools to do what you need or want to do. Don't get me wrong, C++ is a fickle lover, and any competent C++ programmer has some level of frustration towards it (more than other programming languages), but to me it's still worth it.

My other main language is C. Nowadays I pretty much only use it when hacking on perl core or XS modules. I still like it as much as I used to do, but C++ is allowing me to solve the same problems in a much more effective way.

What do you think is the strongest Perl advantage?

Its expressiveness. It gives me power to say things in words that would take paragraphs in some other languages. It's willingness to give me all the power it could possibly give me, instead of having opinions on how I should do that.

What do you think is the most important feature of the languages of the future?

In the long term: extensibility. Doing things outside of the core, things that its original authors didn't forsee.

On a shorter term: good concurrency support. We live in a multicore world, but very few programs actually make use of it because most programming languages don't enable them to do that. There is a vacuum to be filled there.

How did you come up with experimental pragma?

I noticed experimental features were too awkward to use. While I think turning them off and warning is a sensible default, needing two lines of code just to convince perl you know what you're doing seemed a bit too much (and rather unperlish). So I came up with a module that emphasizes that you're doing something naughty, without actually trying to stop you from doing it.

How do you combine biology and programming?

I'm actually trained as a biologist, not as a computer programmer. At some point in my study, I decided to mix my hobby with my study.

Roughly said, bioinformatics can be split up in a more computational field (like 3D protein modeling) and a big data field (like genetic analysis), though in practice almost any problem is a bit of both. What I was doing was the latter. Various innovations over the last two decades cause the amount of data that we have available to grow faster than both computational power and storage availability, which is causing all sorts of interesting challenges.

What is the problem with File::Slurp?

There are three issues.

The first is the interface. In modern IO the encoding of a file is about as important as the filename. This was an innovation of perl 5.8, but File::Slurp predates it. Quite frankly, its whole paradigm is outdated. `read_file($filename, binmode`

`=> ":encoding(utf-8)")` is too long, which leads to people ignoring encoding. In File::Slurper that would be `read_text($filename)`, which is a much better huffmanization in my opinion.

Secondly, the implementation is buggy. Because it predates IO layers, it tries to reimplement them, badly. In particular, it will decode/encode incorrectly for non-utf-8 characters sets (such as `UTF-16` and `KOI-*`). It also has portability issues in doing crlf transformations correctly. Much of this is due to a minor performance optimization for a very specific use-case (slurping a binary file at once) that also greatly complicates the codebase. I'm planning to add a similar but correct optimization to core in perl 5.22 to make that point moot anyway.

The third problem is its maintenance. The distribution hasn't seen a new release in more than 3 years despite the previously mentioned bug being known for more than a year and a half. It has previously known such periods of inactivity. I accept that all software sometimes has bugs, but I wouldn't want to depend on authors that are so careless about fixing them.

Is it really ok to use threads in Perl?

Usually not.

It's not necessarily evil, but in practice hardly anyone knows how to use them effectively. They're not a good model for nearly anything, they won't scale if you share a lot of data.

I've been working on an actor model library, which is a much better fit for perl's internals, but making them usable for end-users is rather difficult. The current uncertainty around smart-matching may mean I'll be forced to reconsider the entire interface.

Is libperl++ a ready project? What is its current state?

Ready? Sadly not. I have to admit it's a somewhat neglected project, though I have plans to revive it.

libperl++ was an extremely ambitious project, that has taught me a great deal about both C++ and the perl API. In fact I'd say it's the most complex programming project I ever wrote. So complex that I painted myself into a corner. Combining templates, multiple inheritance and implicit conversions makes for an explosive mix.

I probably should make a version 2.0 which much leaner and much less magical.

Has Perl's documentation improved since 2010?

Somewhat. Not nearly as much as I would like. Some stuff got rewritten, like the new perlopentut, a lot of stuff hasn't changed. This could really use an influx of motivated contributors,

Should we encourage young people to learn Perl right now?

Yes, of course. Then again, I encourage aspiring programmers to learn lots of languages, preferably very different ones. Perl has a combination of practicality and inventiveness that makes it educational and useful at the same time.

Are you going to start blogging again?

Probably. Not sure when yet. I usually have a hard time finishing my blogposts, despite having plenty of ideas. Generally, coding gets a higher priority, so it tends to not happen.

13 Olaf Alders (December 2014)

How and when did you learn to program?

For some reason when I was in elementary school, I took a very introductory computer course at the public library. That was my first hands-on programming experience.

I first properly learned to program while in high school. Our class worked in Waterloo BASIC on a network of Commodore 64s. My family didn't have a computer at home, so I didn't apply my skills outside of school. It was fun, but I don't think I ever suspected I'd write code for a living.

In university I started out in science program. My lone computer course was in Fortran 77. I hated it. I eventually switched from Science to the Humanities. That's where I got back into computers and picked up on Perl.

What editor do you use?

For a few years now, I've been using Vim. I'm not a power user, but I get by. https://github.com/oalders/dot-files/blob/master/vim/vimrc. I've happily used various GUI editors over the years, but after a junior programmer at a previous job showed me a few vim tricks, I was hooked.

When and how have you been introduced to Perl?

I was introduced to Perl while in University. I was studying Greek and Latin and created a web site for the Classics Club. I ended up writing a hit counter cgi script for the web site. That was fun, so then I built a service based on that counter script and emailed a bunch of friends to tell them about it. Word got out and eventually I had thousands of users. It got to the point where I had to start charging money to cover my hosting costs. Soon I had a proper business on my hands and the whole thing was written in Perl.

What are other programming languages you enjoy working with?

I did a fair bit of Objective-C when writing iCPAN. I dabble a bit in shell scripts and JavaScript when I need to. I've just started reading a book on Go.

What do you think is the strongest Perl advantage?

I don't know if I have a strong argument about what makes Perl better or worse than other languages, but for me it's a comfortable fit. I like the way I can express myself with it.

I'd also have to say that I really like the people of Perl. I've had really great experiences getting help when I've needed it. I've made some good friends in the "community" and it's a comfortable place for me to be. YAPC::NA is always a lot of fun. I've also had the chance to be at the QA Hackathons in Paris and Lyon and I'm planning to attend the 2015 hackathon in Berlin. When I'm able to attend these events, they're real highlights for me.

What do you think is the most important feature of the languages of the future?

I'm no language designer, but as there are more and more portable computing devices, I'd say the languages which make it easy to run embedded software will have a real advantage.

How was MetaCPAN started, what was your role?

It's kind of a roundabout story. At some point I wanted to teach myself how write an iPhone app. I thought of having a copy of CPAN docs on your phone because I like browsing random modules. I was commuting to work on the subway and there was no cellular signal underground, so I couldn't explore CPAN while on the move. I convinced a friend to work with me on this app.

My focus was on extracting the Pod from a minicpan and inserting it into an SQLite db. As I dug deeper and deeper into the problem, I realized that it wasn't easy and that there was no existing solution in place. It seemed like a problem that many people before me had already dealt with. And, since the modules on CPAN are constantly being updated, it felt like a problem that was best solved by a web service.

We talked it over at PerlMongers in Toronto one evening. A fellow PerlMonger mentioned using Elasticsearch, which I had never heard of. A bunch of people also passed me $20 bills. Over the next 6 weeks, I spent my evenings writing code to extract data from the iCPAN database and put it into Elasticsearch. I hosted it in the Rackspace cloud with the money I had collected. Since Elasticsearch gives you a RESTish API out of the box for free, I had a working web service at the end of this time.

Mark Jubenville (who had been working with me on iCPAN) wrote http://search.metacpan.org as a proof of concept in pure JavaScript. We used this to test the API.

Around this time Moritz Onken came along. As part of his Google Summer of Code he created metacpan.org Since then MetaCPAN has become much more popular than I would have imagined. We've gone from a tiny cloud-based account to 6 servers in 2 data centres. We have a group of core developers and many committers. It's a busy project and it has a lot of moving parts, but it's fun.

Did you expect that MetaCPAN would practically replace the SCO?

No, and it hasn't really done that on a global scale. In some circles MetaCPAN is the more popular choice, but we have maybe 20% of the traffic that search.cpan.org has. Gabor Szabo could confirm or deny that. This is partly because the top results for more Google searches will lead you directly to search.cpan.org. There are still many people who swear that search.cpan.org is just better. Gabor even started work on an SCO clone which uses MetaCPAN as the back end: https://github.com/szabgab/MetaCPAN-SCO.

We're working on improving search this winter, so I have high hopes that we'll have an even better user experience quite soon.

I should note that MetaCPAN is not meant to be an SCO replacement. It's good to have choices.

What was the biggest obstacle in MetaCPAN development?

The biggest obstacle continues to be finding developers who know or are willing to learn Elasticsearch. It was an amazing tool to get up and running with. It can do very complex and amazing things, but it does require a fair amount of skill to really get to shine in some cases. Also, you have to install and configure a lot of different things to mimic our production environment. We've mostly solved the environment now, though. Leo Lapworth spent a lot of time moving our deployment to Puppet and creating a developer VM using Vagrant. So, now you can spin up your own MetaCPAN front end + back end in a very short time https://github.com/CPAN-API/metacpan-developer.

How to start contributing to MetaCPAN?

First off, I should mention that you can get lots of help on `#metacpan` on irc.perl.org.

It's a very friendly channel with very few experts, so you can feel comfortable asking any questions at all. We're all trying to figure things out together.

If you just want to work on the web site, it's quite simple:

```
git clone https://github.com/CPAN-API/metacpan-web.git

cd metacpan-web

carton install
./bin/prove t
carton exec plackup -p 5001 -r
```

If you want to work on the api, then starting with the VM is the best option. https://github.com/CPAN-API/metacpan-developer.

How is iCPAN doing these days? Can you explain what it is?

iCPAN is a universal iOS app, meaning that it runs on both iPhones and iPads but with slightly different interfaces. It has almost all of the Pod from the recent CPAN uploads in it. It's meant to be something you can use on an airplane, in the subway or in line at the bank. No connection necessary. It also allows you to bookmark your favourite modules (on the iPhone, at least). However, iCPAN hasn't been in the app store for many months now. If you don't pay your developer fees to Apple, they pull your apps from the store and that's what happened to me. Lately I've been thinking about reviving it. If enough people upvote this ticket, I'll get the app back online https://github.com/oalders/iCPAN/issues/19.

Where do you work now? How much of your time are you writing Perl code?

I work at MaxMind.com. Most days, apart from our standup meetings, I actually spend almost all of my time writing Perl code and I'm able to contribute to open source projects as part of my work. I feel quite fortunate in that regard.

Should we encourage young people to learn Perl right now?

I think so. I don't think we should push young people into learning just one language, but Perl is a good language for young people to have in their toolbox. I've been involved with the Google Summer of Code and GNOME's Outreach Program for Women for a couple of years and I've seen some very bright young people doing interesting things with Perl.

I'm not sure if the number of working Perl developers is dwindling but our demographic does seem to be getting older. We could solve that by encouraging young people to learn some Perl. "Girls Who Code" seems to have the right idea. http://girlswhocode.com/.

My girls are 3 and 5 years old now, so they're not ready to start with Perl. If they do express some interest in programming, I'll start them with something like Scratch, but I'd certainly introduce them to Perl at the right time.

How do you combine working as developer, doing open source and playing in a band?! Does making music help in programming?

I combine all of those things very badly. I don't spend nearly as much time playing music as I'd like to. Basically I try to spend most of my free time with my kids, but I'm fortunate to be able to contribute to open source projects as part of my day to day work. Everything else has to get squeezed in whenever I have a little bit of free time.

I don't know that music helps at all with my programming. I actually studied Ancient Greek and Latin in University and that helped me immensely. If you can read Ancient Greek, you're well equipped to decipher someone else's code.

Why does http://wundersolutions.com return default nginx page?

Oops! I just fixed that. :)

Dancer or Mojolicious?

Mojolicious. We had a long discussion on frameworks at $work recently. I spent a lot of time researching the various options and I presented what I found. As a group we then decided to move to Mojolicious for our next project. I'm also using it for a personal project and I'm very happy with how easy it is to get up and running with.

Aside from the painless start, the quality of the documentation is what really sold me. I don't have to look hard to find what I need and that's very valuable to me.

Will you start blogging again?

I actually did just start blogging again. I've got a backlog of things I need to write about, so there are lots of articles on the way.

14 Ricardo Signes (January 2015)

How and when did you learn to program?

When I was very young, my family had a TI-99/4A, which was a contemporary of the Commodore 64. I was probably four or five, and I taught myself enough BASIC to write guess the number games or little templates that would ask for your name and birthday and then tell you how old you were. I was just learning to write anyway, and what I got out of this was that a computer was a special kind of writing device, where whatever you wrote could just happen. I still sort of think of them that way.

I was an on-again off-again programmer until I left college. It was just something I did for fun or to solve little problems. I hacked on BBS servers in Pascal and C, and wrote little tools to scrape USENET and the web. It all worked, but it was all very jury rigged. I didn't learn how to do professional-quality work — if you're willing to accept that's what I do these days — until I left college and had to get a full time job.

What editor do you use?

The first text editor I remember using was KEDIT, which was an XEDIT clone. My father worked with 3270s, so he was used to using XEDIT, and so we had KEDIT at home. It was weird, but it was clearly superior to EDIT.COM, so I used it. Later, I moved on to elvis, because I used Slackware, and then JED which I remember fondly. I remember that all of the S-Lang software looked gorgeous in 1995. All of my terminal application color schemes, now, take inspiration from JED and slrn.

Nowadays, I use Vim. I'm not a Vim zealot or even much of an expert, but I know my way around it and have written enough Vim script to know how little I want to write any more. I like looking at new editors as they come out, but I have a hard time believing I'll change editors again.

When and how have you been introduced to Perl?

When I first started using Slackware, around 1995, I wanted to write programs to do a lot of the little boring things that I was doing by hand, like stitching together USENET posts and uudecoding them. I'd written (horrible) C, and I could've done it in C, but I knew it would be painful. Slackware had a bunch of Perl code holding it together, so I decided to learn Perl.

Now, that was Perl 4. Slackware, as I recall, was very slow to adopt Perl 5. Eventually, it was available, but you had to install it as its own package, and it was `/usr/bin/perl5`. I never bothered. Perl 4 could do everything! Also, I had no clue why I'd want modules or references.

I didn't really learn Perl 5 until around 2001. I had gotten my first full time job, and was writing software from scratch, alone, in PHP 4. It all worked, but it was very clear to me that the language wasn't helping me build software that I'd be able to maintain for long. I needed something that could enforce more discipline, and I had inherited a copy of the Camel. I read it, ported all of my PHP to Perl 5, and have never regretted it.

I'm probably the only person who decided to use Perl because it would encourage discipline.

What are other programming languages you enjoy working with?

Most of the serious work I do is in Perl 5, so in most other languages, I dabble. Occasionally I work on a larger project, though, and I'm always looking for reasons to use the other languages I like even more.

First, there are the languages that are a lot like Perl: PHP, Python, Ruby, JavaScript. Of those, I like Perl best. The rest, mostly, I can take or leave. People overstate the differences.

Last year, I was really into Forth. In the past, I'd tried to learn assembly languages, and I always loathed it. Forth felt like the assembly that I'd always wanted. It was simple, straightforward, close to the metal, and had a model that I could really wrap my head around. More than one Perl conference attendee had to listen to me rant about how they needed to write more Forth.

I really like Smalltalk (the language, if not always its bizarro operating environment), and I like trying to solve little problems in Prolog now and then, to keep my brain a bit limber. I wish I had a reason to write more Io.

I look back happily on every language I've had to use, for different reasons. Except for Visual Basic 5.

What do you think is the strongest Perl advantage?

There are object oriented languages, functional languages, logic languages, and other paradigms. Perl's paradigm is results-oriented. Perl is a language for getting the job done. If the job only has to run once, you can throw together a monstrosity that will do the job, and then you can never speak of it again. If the job has to run reliably every day for ten years, you can write pretty robust code and rely on the program to keep working even as Perl itself changes out from under it. This isn't any one feature of the language, but it's a guiding principle of how the language's evolution is guided over time. We keep trying to make sure that we're helping you get the job done without affecting the jobs you've already got in flight.

Incidentally, I think you can learn a lot just by watching the goings-on of the Perl community with this idea in mind.

What do you think is the most important feature of the languages of the future?

Concurrency. Really, that's one of the most important feature of languages of the present. It's clear that a lot of software that we write could be made more flexible and performant if it was designed with a greater emphasis on concurrency. Perl 5's not terrible at concurrency. It has a ton of frameworks for different kinds of concurrency. That's a pretty common situation for current languages.

I think the really important feature for languages of the future is a simple and ubiquitous abstraction for concurrency throughout the whole language. Go and Erlang are definitely languages to watch on this front.

Why so many modules? When is it going to stop?

Hasn't it already stopped? I'm almost afraid to look at a chart of my activity!

When I solve a problem in a way that I might use in more than one project, I try to modularize it. Sometimes, modularization presents its own problems, so then I have a second order of solutions to modularize. After a while, it all adds up, and I ended up with modules broken out of modules written to help maintain the modules I modularized out of my modules. Yow!

Fow a while now, I've felt like I had most of the tools I've needed for a while. Until I hit some weird new problems, I think my output will remain slow and steady. Then again, I do have a backlog of ideas I'd like to get out the door: a URI.pm replacement, a Mason-like template system, a zasm compiler…

What is your current role in perl development?

I'm the "pumpking," which is something like "benevolent dictator for now." That's a job that's been done differently from person to person. For my part, I try to keep discussion on the perl5-porters list from stalling, to make decisions where there is no consensus forthcoming, and to apply pressure as needed to keep potentially-stalled work moving. I've also tried to firm up expectations of civil behavior on perl5-porters. I don't want to work in a place full of rudeness, and I work on the assumption that most other people don't, either.

What to expect in 5.22?

So far, 5.22 is looking pretty different from 5.20. In 5.20, we added a bunch of exciting new features that you could point at and say, "Wow, that's a feature I want to use!" So far, 5.22's hit list has been *mostly* been performance improvements — and a lot of them! Method calls are faster, deep dereferencing is faster, and there are a number of other optimizations in there. Of course there quite a few bug fixes and nice little features, too.

If I had to pick one big *feature* that's most exciting, I might pick the new experimental reference aliasing. With this feature, you can say:

```perl
\$x = \$y
```

…and both will be the same variable, aliased one to the other. You can use it in loops, too:

```perl
my @input = ( [ 1, 2, 3 ], [ 8, 9, 10 ] );
for \my $x (\ (@input) ) {
  $x++;
}
```

…and end up with ([2,3],[9,10]). In other words, you can get the aliasing behavior you'd have with `$_`, without having to use the topic variable.

What is the current main focus for perl language development?

I don't think it would be accurate to claim that there is one. There are quite a few people working on perl, each according to their own interests. I try to encourage the work that seems beneficial and discourage those rare projects that I think can't ever get merged.

We're definitely seeing a number of people working on speed issues, right now, but there are a lot of other threads of development going on: deparsing improvements, float handling improvements, memory optimizations, and even restored EBCDIC compatibility. My view is that as long as people are working and sharing their work and plans, things are doing well. Trying to channel people onto topics they're not interested in wouldn't be a win for perl, the way things stand now.

There are people who say that backwards compatibility has become a non-priority. What is your answer to that?

It's clearly not the case. When changes to the language are discussed on perl5-porters, the question of backward compatibility invariably comes up, and it's a big factor in deciding what's good for Perl. After all, as I said above, we don't want to break your existing code that you've got getting the job done.

Different people want different rates of change, and they want change to different parts of the language, so there's often a lot of contention around these issues. Still, I think it will be clear from any observation of the process on perl5-porters that backcompat is a definite priority. Earlier this year, we decided to make clearer statements of the consideration given to those rare breaks in backcompat that are made, and that will be going into the "perl policy" document before 5.22.

Are there any things about internal perl development you would like language users to know?

It's really interesting! Long before I took any active part in the development of perl itself, I read p5p for fun. I deleted some of it unread, because it wasn't interesting enough, but tons of it was just fascinating. I learned all kinds of weird quirks of the language, some intentional and some not. I got a great sense for the different priorities assigned to different parts of the language by different people, and why that might be. I learned about operating systems, data interchange, algorithms, data structure, and group social dynamics. It's also just great to watch smart people work together.

Even when I wasn't interested in working on perl, it was fascinating to watch all this, and it turned out that it made me feel a lot more prepared later when I wanted to make a few modest suggestions or changes, and be more heavily involved.

What are your thoughts on Perl 6?

I'm cautiously optimistic. I've tried to keep an eye on Perl 6 through its whole development, which has also been interesting. Every year or so I write a little program in Perl 6, and it's fun. Perl 6 is one of the only languages I know of that seems to be trying to provide universal abstractions for concurrency while also being a very dynamic language. I'm definitely going to keep playing around with it. There's a lot of language there, though, and so far I've been unable to hold the whole thing in my head at once. That may very well change as I use it more often than once a year, though!

Where do you work now? How much of your time are you writing Perl code?

I work at Pobox.com, and I've been there for around ten years. We provide services for using your one email address for life, and we've been doing it for about twenty years now, which still impresses me, given the lifetime of most of the other internet services I've tried using.

Our internal code is almost entirely Perl. There's a little bit of Ruby, Python, C, and PHP, and of course JavaScript, but I'd say that 95% of my time, I'm writing Perl. We try to get most of the generically useful hunks of that Perl onto GitHub or the CPAN, too, so a fair bit of what you find in my CPAN listing is code built for work.

Should we encourage young people to learn Perl right now?

We should encourage young people to keep in mind that computers are general purpose tools for solving all kinds of problems, and that might mean encouraging them to program, depending on the problems they want to solve. I don't think the choice of language is particularly important, except insofar as it's able to help them solve the problem and feel continually more ready to go solve the next one.

For me, Perl was that great at that. I think it probably would be for many other people. On the other hand, if I saw a kid solving problems with what they learned from "JavaScript for Kids," the last thing I'd say is, "You should've written that in Perl."

For kids who want to be serious programmers, though, I might recommend Forth! Let me tell you a little bit about it…

Are you going to FOSDEM?

I am! I've heard lots of good things about FOSDEM, and kept thinking I should go someday. This year, I got just the right amount of nudging about it, and so I'm booked and I'll be there. I'll be in the Perl room, but also everywhere else, I hope. I look forward to sampling all the good stuff Brussels has to offer, and to meeting a lot of the people who never make it to the US for conferences.

What are you addicted to?

Lately, I'm trying to prove to myself that I'm not addicted to sweets. I've cut out soda, pastries, chocolate, and lots of other such things, with only a few exceptions. It's not making me miserable, but it sure isn't easy.

Beyond that, I try to maintain a redundant array of inexpensive hobbies. I could quit any *one* of them any time I wanted, but after that, I'd start needing to find something new. I'm not good at just killing time. Maybe I'm addicted to keeping busy.

Did you finally restore all your backups?

Ugh! This still smarts! Presumably this is about my big music migration disaster in June. The big problem, basically, was that I permanently deleted some backups that I *thought* were restored, but weren't. So, I was left missing tons of files. I restored quite a lot of it, but now I've got missing tracks on all my favorite albums. I'm not missing any whole albums, so it's even harder to motivate myself to go down in the basement, unpack CDs, and rerip missing tracks. Still, the current count is 1598 missing files. I do need to do something about it. Mostly, though, "something" is turning out to be "listen to more Spotify."

15 Neil Bowers (February 2015)

How and when did you learn to program?

In 1980 my Physics teacher brought a ZX-80 into class (a kit-built computer only available in the UK). Sometime after that the school got a Commodore Pet and I started playing round on it. The same physics teacher took a few of us on a trip to London, where I bought my first book about writing programs (mainly games like Hunt the Wumpus) in BASIC.

Some time after that I got my first computer, a Vic-20. That was the start of my addiction.

There's no one time I can point to and say *that* is when I learned to program. We're always learning, but there are definitely milestones and key experiences I can point to. For example: I spent one undergrad summer working for my CS department, porting a VLSI design program from Unix to VMS, to run on the graphics terminals we had. I had a lot of guidance from a postdoc (thanks Neal!). I learned a lot about C, make, software in the large, graphics terminals, perseverance and when to ask for help.

What editor do you use?

I use vim.

My first experience with a true text editor was SOS, used on a DEC-10 in my first year at University. At the end of the year I got an account on a Vax running BSD Unix, and had my first exposure to vi. After SOS it was fab. I also used Eve and EDT at Uni, but whenever possible I wanted to use vi.

I periodically try new text editors, and I had an extended affair with emacs in the 90's, but whenever I need do some serious *editing*, I always want vi.

I laughed when I read in your RJBS interview that he said "I'm not a Vim zealot or even much of an expert". At the QA hackathon last year I sat next to him working on a PAUSE testsuite. I came away thinking "man, I need to improve my vim knowledge!", and subsequently bought a book. I better read it before this year's hackathon! :-)

When and how have you been introduced to Perl?

In the late 80's / early 90's I was doing a lot of text processing, writing scripts with a lot of awk and sed, as was a friend in the department. He came across Perl and said we should take a look. Wow, pretty wild.

I moved to New Mexico and was working on a visualisation and image/data processing system written in C, which had to work on as many variants of Unix as possible. We had a wide range of machines on our desks, and would get loaners as well. When I got there, they had a build system for compiling across all the machines, which was a collection of shell scripts. I rewrote it in Perl 4 and had great fun. From that point on I steadily wrote more and more Perl.

What are other programming languages you enjoy working with?

I spend a lot of time writing Javascript as well as Perl at the moment. Sometimes I enjoy it. I've enjoyed writing C, Postscript, Inform and assembly at various points in the past, not always because of the language, but what I was doing with it.

Perl has always seemed like the language which best matches how my brain is wired. Scala is the first other language where I've felt something similar, but I haven't done enough to know if it's just a crush.

A bunch of Perl people did an online course on programming languages, and seeing their tweets made me wish I'd signed up for it. I fancy playing with Haskell, Clojure, Rust and Go. Someone should do a language challenge: a language per month with randomly assigned tasks!

What do you think is the strongest Perl advantage?

CPAN and the community around it, but then you might have expected me to say that.

Personally, I find myself able to express myself most efficiently in Perl, though some of that is obviously just down to experience.

But what really makes me efficient in Perl these days is CPAN. That can be at very different levels of abstraction, from a full-blown framework to a single sub in a utility library. Writing a script for the Pull Request challenge at the weekend, I needed to randomise an array. I could easily have done that myself, but I thought "probably something in List::Util", and sure enough, there was shuffle().

Among many ratings/graphs on your website which one do you think more people should know about?

It's not quite a rating or graph, but I'd say the adoption list.

Open source is largely built, and particularly maintained, in spare time. People's interest, motivation and time for it naturally ebb and flow. They scratch an itch, release some code to CPAN, then move on. Later on other people have that itch, but the code may need some work before it provides the right scratch.

If you want to learn about CPAN, the toolchain and the community, but you don't have an idea for a module of your own, then find a CPAN distribution in the need of some love and which appeals to you, and get to work.

Once you've done that, I suspect you'll be more likely to release something of your own as well.

Do you think CPAN needs curation?

Ha! Yes, it does. Actually, I think that curation needs to take a number of different forms:

- Removing distributions from CPAN entirely. You need to be careful doing this, because you've no way of knowing whether a module is being used, other than when a module is used by another dist on CPAN. But there are very old dists on CPAN that haven't been usable for years. And with a long enough deprecation cycle, there are plenty of more recent dists that could safely be removed from CPAN, I think.

- Deprecating modules, particularly when there are one or more better dists for doing the same thing on CPAN. Deprecating a dist is the right thing when a dist is no longer going to be maintained, but you can't remove it (yet), because there are other dists on CPAN using it (and don't forget DarkPAN). Any::Moose is a good example; it's now deprecated, but there are plenty of dists still using it at the moment. Hopefully they'll gradually be moved to Moo, or Moose.

- Improving the documentation for modules. Obviously third parties can only improve the doc by sending patches or pull requests. AnnoCPAN was kind of an attempt to address that, but unless it's fully integrated with services like MetaCPAN, that's never going to take off. Github has made me much more likely to submit a documentation improvement.

- Writing a good SEE ALSO section in the doc for your modules is one of the best ways you can help people who are looking for modules, because it aids discoverability on MetaCPAN, for example.
- Community generated metadata / annotations. Favorites are one mechanism we have for this now, but tagging and other types of metadata could be provided externally from distributions themselves. CPANratings is a good idea, but one problem is that if you improve a once-badly-rated dist, it can be hard to to improve its ratings (eg by getting negative reviews that are referring to a previous release). This sort of annotation should let us identify the "best of CPAN", without needing a separate CPAN for it.
- Convergence. Where there are a lot of modules for doing the same thing, there is often no obvious "best" module: each is prioritising different things, for example. Sometimes, but not always, it would benefit CPAN to take the best parts of various modules and work towards a single "best of" module (hopefully based on one of the existing modules, rather than adding a new one). It's not clear what the motivation is here for the authors, since the individual modules were probably created by people scratching their own itch. Case in point: I've tried doing this with Olaf Alders, related to one of my reviews, but we both keeping getting distracted with other projects.
- The toolchain and documentation need curating as well. Right now it's clear that it's not as easy as it should be for people to start contributing dists to CPAN. It's easy to write the code, but getting from there to a good dist on CPAN is non-trivial.
- Bringing in ideas from other languages and tools. We're terrible at this, for the most part.

What is CPAN Pull Request Challenge? How can one join?

It's a challenge for 2015 to encourage people to contribute to CPAN and get involved with "the community" a bit more. If you sign up, then on the 1st of each month I'll send you an email with a randomly selected CPAN distribution (which also has a github repository).

You have one month to do at least one pull request. You can choose to do more if you want. The idea is that you should do something useful for the dist, which might be fixing a bug, improving the documentation, updating it to follow modern CPAN conventions, or improving the coverage of the test suite, for example.

It's based on the following thoughts, which I had after participating in the 24pull-requests.com challenge:

> Every dist on CPAN can be improved in some way that is useful to CPAN. Following on from a thought above, that way could be deprecating it. Picking the repos for the 24pullrequests challenge was hard. So I'll just randomly assign one to you. One a day was too much like hard work. One a month sounded a lot more feasible. I score CPAN dists according to some measure of "worth someone looking at", and select from the highest ranking dists.

The hope was that people might have fun while learning a bit more about the toolchain, and improving the overall quality of CPAN at the same time.

You join by emailing your github username to me: neil at bowers dot com. If you've got a PAUSE id, let me know that as well please, so I can make sure I don't assign any of your dists to you.

Did you expect CPAN PRC be so popular? What are the next steps?

Not at all. I've come up with a couple of CPAN related challenges in the past, and based on those I expected a dozen or so people might sign up. There are 372 people signed up to the PRC at the moment! There were three waves of signups: first when I posted it on my blog in late November, resulting in 20 signups. Then I posted about it to blogs.perl.org on Christmas Eve, and got another 50. Then someone put it on Hacker News on New Year's Eve, and the email started pouring in!

There's an IRC channel and emailing list, where participants are helping each other, for example with ideas on what to do, but also in usage of github, Travis, etc. It's a very positive supportive channel, because everyone there is trying to do the same thing. Plus there's a self-selection in the population as well. We're all the same kind of nutters, maybe?

I expect a hundred or so of those to drop out over the next month or two, but at the moment the energy is really high, and lots is getting done. It's very satisfying and slightly scary.

What's next? I'm working on extending the scheme I use to rank distributions. When people get their assignments, the email they get lists any known ideas, to help people get started. The more we can give people things like this, the more likely they are to pick at least one of them and submit a PR.

I had originally planned to start a second challenge, but this one is keeping me plenty busy at the moment. A secondary challenge I will be starting soon is to encourage more people to put their CPAN distributions on github. If you do (and the metadata lists the repo), then it will become a candidate for handing out in a later month.

Where do you work right now, how much time do you spend writing Perl code?

I work for a company called Cogendo, which I started with a friend when we were both made redundant.

I do a lot of Perl (and Javascript) coding, but as with any small company I do a lot of other things too. Pretty much every day I'm coding, but I'll also be doing ops stuff, product design, customer support, etc.

Should we encourage young people to learn Perl right now?

It depends on how young you're thinking?

My son is 7 and has so far slightly engaged in Logo. I wouldn't try him on Perl yet, but will probably try some other language soon. I think children are better served with a less complex language, which is simpler and thus easier to learn.

He mainly thinks computers are for Minecraft and flash games right now.

When CPAN Report 2014 is coming?

Ha! Hopefully soon. I planned to do it around New Year, but then the PR challenge took off. Now everyone's got their February assignments, I'm hoping I'll have time to finish the CPAN Report.

When I first got these questions from you, I thought "Oh I'll have that done before I've replied to your questions!". Hmm. My yak shaving tendencies don't help.

There will be a different slant to the report this year. I think you need to be careful what figures you present in a leaderboard, as it can easily end up driving a different behaviour than you intended, or hoped for.

A different theme each year is no bad thing anyway, to keep things fresh.

How long is your cpan review backlog and when to expect new reviews?

Before I give you a number, I should explain: whenever I realise there are a number of modules for doing roughly the same thing, then I think "that's a potential review", and put it on my trello board for reviews.

I have a number of reviews in progress, and some that are just a list of modules, and others that are just a card on trello.

Altogether my backlog has 22 reviews. The next review I'll publish is on Exporter modules. I've written up most of the 40+ modules, and even did a talk on it at the London Perl Workshop last year. I keep getting distracted from finishing it. See answer to previous question :-)

What do you think is the best way to give back to the Perl community?

Sign up for the Pull Request challenge and do at least 11 PRs in the rest of the year! :-)

I think the best way to "give back" is to help other people in concrete specific ways, particularly beginners. Helping others often requires you to learn things. I'm learning a lot doing the PRC: people ask me questions and I'll think "oh, I've no idea. That might be useful to know though, so let's find out".

And if you want something more to do, adopt a distribution on CPAN. Polish it up and possibly be ready to hand it on.

If you don't want to adopt a distribution, contribute to some of the less "fun" parts of CPAN development & maintenance: improve the documentation, fix a nasty bug, or improve the test coverage. It's a lovely surprise for an author to get a PR for one of those.

And blog about it. Share your experiences and encourage others to as well.

One of the bits that's making me most happy about the PRC is how friendly and helpful everyone is being towards each other.

16 Renée Bäcker (June 2015)

How and when did you learn to program?

My first program was a simple calculator written in QBasic in the mid-1990's. A command reference for QBasic was printed in the manual of the PC my father had at the time. So I sat down and tried what I can do with that thing called QBasic. It was very easy to get some basic stuff done. But I wasn't very excited then.

A few years later I learned Pascal in school. Very basic stuff, and still I wasn't very fascinated by programming. After school - in 2001 - I started a training as a qualified IT specialist where we learned Java in the company and C++ in training school. I did the training in a small bioinformatics company and I did a lot of stuff in Java. Half a year later I changed the company and came to a bioinformatics department of a big pharmaceuticals company. The people there where mathematics, chemists, biologists but not programmers. And they used Perl...

That was the time when I learned to program. I always learned by trial and error and with the help of Google ;-) I'm not very good in reading IT books, because I want to do something and read less. When I want to do something, I think about how to do it and try it. When anything doesn't work, I read manuals, forum threads and blogposts about this specific thing and try to understand what (and why it) happens.

This way I learn things better and get a deeper knowledge compared to just reading books.

What editor do you use?

I'm not addicted to a specific editor. I use what's available... When I'm in Windows land, I often use Kephra or Padre, but at my current project I use Notepad++ and Komodo. When I'm in *NIX land I use vi/vim.

So I'm the wrong person for a editor-flamewar.

When and how have you been introduced to Perl?

I already mentioned it in the answer to the first question that my first contact with Perl was in a bioinformatics department. That was back in 2002. The people there did some programming without being real programmers. They only wrote (very) small Perl scripts.

They couldn't teach me Perl, but they wanted me to do the programming in Perl so that they can - at least - read the stuff I do. So I had to teach Perl myself. But that wasn't very hard as Perl's documentation is very good and there were Perl forums like http://perl.de (the community behind that forum started http://perl-community.de a few years later) and Perlmonks.

So it was JDIWP (Just do it - with Perl) without any guidance. Maybe this was one of the reason that I still use Perl as I saw how fast one can get things done even without any Perl knowledge. Sure the scripts had lots of global variables, the first script had no "strict" and no "warnings" (but I learned about that a week later) and I didn't knew all the idioms, but the scripts worked.

I still learn new things every day…

What are other programming languages you enjoy working with?

I rarely do some programming in C/C++ and PHP and I do not really like it. I have to do a lot of JavaScript and recently I started to look at Rust. With frameworks like jQuery JavaScript is ok. You can do a lot of stuff really fast. And Rust looks nice…

What do you think is the strongest Perl advantage?

The community, CPAN and backwards compatibility.

The community might not be an advantage for everyone. I know a lot of programmers who do not participate in a community (no matter what programming language). They use the programming language as a tool and nothing more. In my opinion the community is important especially for tools that do not have a big company that develops the tool.

Without CPAN I couldn't have done all the stuff that I've done. I want to say "Thank you" to all the authors of the great distributions on CPAN.

And backwards compatibility is a thing that I hear very often from my customers. They are happy with Perl and use Perl since many years and upgrades were a no-op.

What do you think is the most important feature of the languages of the future?

That's a hard question - I don't know what will be important in 10 years, but at the moment applications hosted in the cloud is an important topic. And the amount of data that have to be handled gets bigger and bigger. I don't believe that one language must fit both areas, but a language should be good in one of those areas.

What are you thoughts on Perl 6?

I didn't follow the Perl 6 development in detail until recently and I use Perl 6 only for some small scripts, I think it has some great features. In my dayjob Perl 6 won't replace Perl 5 for the next years, but for my own stuff I use Perl 6 more and more.

We should stop worrying about the impact of Perl 6 on Perl 5 (e.g. was the decision to name the thing Perl 6 when the language is a sister language of Perl 5; Did the start of the Perl 6 project slow down the development of Perl 5; ...). Now we should start enjoying Perl 6 and its features.

How and when did you start `$foo` magazine? Why did you decide to stop?

In 2005 I talked with a friend of mine at the German Perl Workshop about a German speaking Perl magazine. We agreed that such a magazine is missing and that it would be nice if one can do it. But nothing happened.

In 2006 we talked about the very same topic at the German Perl Workshop, but nothing happened.

Around November 2006 I thought if no one starts a magazine, I have to do it. I created a list of possible topics and started to write the articles. I downloaded Scribus to layout the first issue... I knew nothing about publishing and even less about "designing" a magazine. Nevertheless I did it. I didn't want to spend a lot of money for an experiment like this. So I did everything myself.

And in February 2007 the first issue of `$foo` magazine was printed. I sent it to some Perl programmers I knew and one of them worked for a company that does a lot of design/layout stuff and they told me all the things I've done wrong. Creating a magazine layout is hard work! In the end they created a layout and I paid by doing some programming for them.

I created a small website and sold the magazine over this website.

I've published the magazine for eight years and I stopped as it was too time consuming. Publishing a magazine is not that easy. You need authors and as a very small magazine you can't pay royalties, so you need volunteers. You have to contact the authors, ask for articles. Sometimes it happened that authors said a week before we sent the magazine to the printer that they couldn't make it. It's ok, because they do it for fun. But that means that I have to get an article within a week.

I have to say "Thanks" to Herbert Breunung who was a regular contributor and wrote a lot of articles.

After you get the articles, you have to proofread them. Then you have to compose the magazine, send it to the printer, get it from the printer and send it to the readers.

Since 2007 when the first issue came out, a lot in my life changed. I married and I have two sons. In the beginning it was very easy to write articles as my projects covered a broad range of topics. But the last years it became more difficult. I had to think about new topics, read about that topics, test the code and write the article. And I had my day-to-day job. That didn't fit.

I love my family life and I want to spend a lot of my time with my wife and my kids. So I decided to stop the magazine. It was a hard decision, because `$foo` magazine was my baby too.

Why did you build perlybook.org?

Since many years I use Pod as my main markup language. For the articles in the `$foo` magazine, for documentation, for training handouts etc. It's very simple and you can convert it to many other formats. And on business trips I want to use the time to prepare my trainings, read documentation and other stuff, but I do not want to read the texts on my laptop. Ebook readers are way better for that. So I needed to convert my texts and the documentation of CPAN modules to ebooks.

So I wrote a small commandline tool that did the conversion for me. When I showed that tool to some friends they asked me if I can put that tool online so they can create Ebooks from the documentation on CPAN.

Fortunately there is MetaCPAN with a great API, so I moved the commandline tool to a proper module that can have plugins to support different sources like local files, MetaCPAN, Github etc. and different target formats like text, PDF and EPub. Thats what the module EPublisher is for. Then a small webapplication around that module and perlybook was born.

So this website started as a small tool that helped me to scratch an itch…

What is perl-academy.de?

In the past many customers decided that I should not only do some programming for them but also teach their employees - covering a wide range of Perl topics (starting with basic stuff like the right usage of DBI and references to object oriented programming using Moose). And I gave several talks at Perl and non-Perl events. I really enjoyed both giving talks and teaching people.

I also held some courses in the university where I studied and gave some Perl courses for training companies. All these experiences led to the decision to start my own training business. At http://perl-academy.de you can find various courses about Perl that you can't find at other training companies (in Germany).

At the beginning, I offered the trainings on a subdomain of Perl-Services.de, but I recognized that the trainings were not as visible as I wished. So I registered the domain Perl-Academy.de. This helped a lot to make the trainings more visible.

Giving trainings is a nice change from the everyday programming life.

You have sponsored or helped organizing Perl booths during various computer events in Germany. What is a motivation behind that?

I like Perl and I like to talk about Perl. The thing with the booths started in 2010 when we organized a Perl booth at CeBIT, a big IT fair. I can't remember who had the idea to apply for a sponsored booth in the opensource project lounge, in the end we got the booth and we had a good time there talking to about 400 people about Perl.

We learned that lots of companies use Perl but they knew Perl as it was in 2000 or 1990. They didn't know how Perl and its ecosystem has evolved. So we told them about best practices and new modules and many other things.

Since then I have been at various events running a Perl booth and talking to many people.

In my opinion it's very important that the Perl community is open to talk to people outside its echo chamber. It helps to understand what other people think about Perl and what they need.

But to be honest there was not only this altruistic motivation. Running Perl booths were also marketing for the `$foo` magazine and my own business.

Sponsoring is a different topic. I sponsor the events to give something back to the community. In 2004 I attended my first German Perl Workshop and it helped to become part of the community. Those events really help to get to know the people behind the modules you use every day. So events like the German Perl Workshop should still be there in a few years. Sponsoring helps the organizers and keeps the prices low.

In my opinion, every company that uses OpenSource should give something back. And sponsoring is a very simple way for this. And again, it's marketing ;-)

What are you doing for Frankfurt.pm?

Currently too little... I'm a member of the board of directors. We established the club in 2011 when we started organizing YAPC::EU 2012. And in 2013 we took over the responsibility to help local groups to organize the German Perl Workshop.

As I can't make it to Frankfurt.pm's social events I only do the organizing stuff.

Where do you work right now, how much time do you spend writing Perl code?

I still run my own business doing only Perl (plus some HTML, SQL and JavaScript). I do Perl for 8 to 10 hours a day and I love it. Currently I have a long running project at Deutsche Bahn (German Railway), but I still do some OTRS customizations. OTRS is a request tracking system (a competitor to RT that most CPAN authors should know) written in Perl that is widely used especially in Germany.

Should we encourage young people to learn Perl nowadays?

Yes! We shouldn't force anyone to learn Perl, but we should encourage young people to look at Perl. They should decide what they want to use then.

Perl might be an old language, but it's not outdated. I gave some lessons in a school and I sponsored prizes for a competition called "Jugend forsicht" (A youth research program). Some of the teenagers showed a talent for Perl and they liked it.

And young people have new ideas, they don't have biased opinions. That can lead to great new developments. Having those new fresh ideas in the Perl community is great. And maybe once we get the next big application everyone wants to use and that application is written in Perl.

Will you publish `$foo` issues?

All `$foo` issues are available in PDF format at http://abo.perl-magazin.de. We plan to publish the articles in other formats, too. But that might take some time.

Why Mojolicious?

Short answer: It's a great webframework.

Long answer: I started web programming with plain old CGI scripts. Then I switched to CGI::Application and CGI::Application::Dispatch. Some of my programs still run using those modules. I have used Catalyst, too. But for one project - a very small webapplication - I wanted to use something more modern than just CGI or CGI::Application and Catalyst was too heavy.

I had three alternatives in mind: Dancer, Mojolicious and plain Plack. The customer isn't a programmer at all and I didn't want to leave him in the dependency hell. That's what I like with Mojolicious - no dependencies beside core Perl. And @vti it's your fault ;-) You wrote about Mojolicious in your blog and that looked interesting.

I really liked all the features that Mojolicious has in core like support for Websockets. I know that it can be a disadvantage to bundle everything in core. Only one person or a small group is responsible to fix bugs.

An other interesting thing was the Mojolicious and Mojolicious::Lite thing. I can stay with one framework for both "big" webapplications and small applications that fit on one screen. And I do not have to create a directory structure for a small app. Just do it and deliver that one file…

In the meantime the numbers of plugins increased steadily and it's very easy to develop webapplications of any size.

17 David Golden (July 2015)

How and when did you learn to program?

I think I was nine years old when my elementary school class got a TRS-80 and I started doing some BASIC. Soon after, I had a TI-99/4A of my own at home and continued programming in BASIC. I'm sure it seemed quite serious at the time, but now I can't imagine what I was doing with it.

By high-school, I was programming in Pascal and starting with some C. In college I worked a lot with C, and then started with a little C++ and Java around graduation time.

What editor do you use?

In college, I learned emacs because most of the people around me used it, but it was never more than an editor; I never did any customization or programming.

When I started playing with Linux in the late 90's, I wound up needing vi for configuration. I got tired of re-learning it every time, so decided to make vim my primary editor on Linux. Eventually, I bit the bullet and installed gvim on my Windows laptop and never looked back.

Vim has been particularly good for me, as I struggle with RSI occasionally and can keep my hands the home row; I've disabled the arrow keys entirely in vim, and do `CTRL-[` at least as often as I whack the escape key.

When and how have you been introduced to Perl?

I got into Perl in the late 90's when I was running a micro-ISP for friends and family on a Linux box on one the first DSL lines in the US. I was trying to decipher the "logwatch" program and got sucked in from there.

I spent a lot of time on Perlmonks in the early days. Reading and later answering questions there is probably how I got past my novice stage.

Using Perl on Windows led me to get involved in the CPAN Testers project (to more easily report the many broken modules on Windows). From there, I got further into the world of the Perl toolchain and eventually the core.

What are other programming languages you enjoy working with?

Most of my programming work is in Perl, C and shell. I have a love-hate relationship with C. It's often painful, but the ability it gives to be really specific about how data is represented and handled is really refreshing sometimes.

I'm currently infatuated with Go, though I've only had chance to dabble. It fits my brain in much the same way Perl does, and the speed (relative to Perl), concurrency model, and static compilation address several of the most glaring weaknesses that annoy me about Perl.

What do you think is the strongest Perl advantage?

I think Perl's whipuptitude is still its dominant feature. It scales immensely well between one-liners, small programs, and small systems, which means you get to take advantage of the same expertise for tasks big and small.

In some ways (but not all) I think Perl's TIMTOWDI philosophy is an under-appreciated strength that leads to constant evolution in ideas.

For example, I've started to explain to people outside the Perl community that Perl does OO with frameworks the way most languages do web programming. Need a full featured OO framework? Choose Moose. Want something lighter? Choose Moo? Do you have specialized needs? Fine, there are dozens of OO frameworks to choose from.

Ten years ago, no one knew much about traits or roles. Now, it's clearly a dominant feature of Perl's leading OO frameworks. You don't get that in a "batteries-included" language. You get it through lots of people trying to reinvent wheels that the language doesn't provide.

I should add that I don't think CPAN is one of Perl's strongest advantages anymore, at least not on a relative basis. It's still immensely useful, but many other languages have libraries just as deep. Startups are shipping APIs for "popular" languages before they ship them for Perl (if at all), so if anything Perl is usually late in providing libraries for suddenly popular services.

What do you think is the most important feature of the languages of the future?

It might be an over-used term, but I think "scalability" is the most important thing, but across numerous dimensions.

I think people focus on single-system scalability a lot, particularly single-system concurrency to take advantage of multi-core, multi-CPU systems.

But I think multi-node scalability will be equally important. The "cloud" already has people taking about servers as "cattle, not pets" and the popularity of containers will only push the elastic, utility computing trend forward faster.

I think that favors languages with a well-developed, message-passing model over ones that rely on shared-memory. While they are theoretically equivalent, in practice, I think programmers will have an easier time applying experience with the message-passing model to multi-node systems development.

I think it also favors languages with deployments that fit the "cattle" model. Static builds or easily-repeatable deployments will be crucial.

Another interesting scalability dimension is people. Some languages have a reputation (true or not) of being good (or not) for large teams. The best languages should be as good for a single developer as for a large team.

What are your thoughts on Perl 6?

My opinion hasn't really changed from my Feb. 2015 blog post:

> It's still not clear to me what the "killer feature" is. If Perl 6 can be a much faster Perl 5 (by absolute performance or better concurrency), then I think it will be familiar enough that we'll see adoption by Perl 5 developers willing to trade the learning-curve for higher performance.
>
> But I don't see anything that would drive adoption by other communities or new developers. The sheer size and complexity will be daunting. By contrast, Go – a small, highly-opinionated language – has become wildly popular in certain niches. It solves several serious developer pain points that appeal to both static and dynamic language enthusiasts.
>
> I think the Perl 6 team needs to find a few key examples of features of the language that really shine compared to other languages. Not "clever features" – features that get back to the mantra of making hard things possible in areas that programmers encounter every day.

You maintain so many modules. How does it work?

First, I write a lot of modules! I've developed the habit of writing things I need as re-usable, independent libraries. That has pushed me to streamline every aspect of the module development process.

For example, I have a single shell function that creates a new distribution skeleton with Dist::Zilla and creates public and private repositories for it. That lets me go from an idea to coding a module in just a few seconds.

Second, I try to automate everything I can, both to minimize the amount of time I spend with routine stuff and to prevent repeating mistakes. Using Dist::Zilla for this has radically transformed how I ship code. The only thing I need to do before I ship is make sure the Changes files is up to date. Everything is on autopilot, including version numbering, pre-release tests, boilerplate Pod generation, git tagging, etc.

I also use Ingy's git-hub tool, which lets me manage Github comments and pull requests entirely from the command line. I can fetch a pull request, rebase it, test it, close the pull-request with a comment and ship it with Dist::Zilla in a matter of minutes without ever leaving the terminal.

Who wrote the first version of HTTP::Tiny, why was it needed?

Back in 2010, several people on p5p were discussing how Perl 5 couldn't bootstrap CPAN over HTTP without relying on command line tools like curl or wget. We considered putting HTTP::Lite in core, but it had a number of deficiencies and seemed under-maintained.

In response, Christian Hansen whipped up the first draft of HTTP::Tiny as a proof of concept. The goal was to have an HTTP client that was just enough to get to CPAN.pm working via HTTP and download LWP.

After the first draft came out, I started collaborating on the public API and refactoring the back-end code. Most of the networking code is still Christian's, even if git blame credits me because of the refactoring.

We've had a really good partnership. We try to talk through any major changes before implementing them and I think that's kept a good balance between features and minimalism. I've been really surprised (but pleased) that it has become a useful alternative to LWP for some people.

Do you think more ::Tiny modules should go into core?

Not just because they are ::Tiny. The original idea of ::Tiny modules was that they would do 90% of the job with 10% of the size/memory of some more popular module, while depending only on the Perl core. They are intended to be trivial to install or bundle, so they don't really need to be in the core itself.

Why are QA hackathons important?

A lot of the challenges in the Perl QA/toolchain world are hard to address in the scattered bits of time people have for them in their day-to-day lives. Either they need concentrated hacking time or they depend on getting the owners of different parts of the toolchain to all agree on how things should get done. The QA hackathons give QA/toolchain developers both the time and face-to-face contact to move things forward.

I elaborate a lot more about how and why the QA hackathons work in my annual writeups.

(If you want to support next year's QA Hackathon, you can donate to the 2015 hackathon and the funds will carry over to next year.)

As a p5p member, what do you think about stableperl?

Speaking only for myself, not p5p, I'm disappointed that the concerns that prompted it couldn't have been addressed in the normal development cycle.

That said, I think it's a wonderful experiment. Just like rperl, any experimentation around Perl 5 indicates some sort of demand in the user community for something they aren't getting from Perl 5 itself. If forks gain traction, that's a strong signal about what users find valuable.

Where do you work right now, how much time do you spend writing Perl code?

I work for MongoDB, where I lead development of the MongoDB Perl driver. That means that almost 100% of my coding time is spent on Perl (or XS).

Over the last year, I've been working on the "next generation" Perl driver, which is an almost total re-write of the original driver to improve consistency, compatibility, portability and robustness. According to git blame, I'm now responsible for over

90% of the code base!

I'll be shipping a release candidate in the next few weeks, so if you use MongoDB, keep your eyes open for it.

Should we encourage young people to learn Perl nowadays?

I think we should encourage young people to learn to program and the language is not as important as whether the experience is engaging.

When I was learning to program, green text on a black screen was pretty cool, so terminal-oriented programming was plenty engaging. These days, though, I think young kids' expectations are set higher and they'll want a more graphical experience for a first-language.

For older kids, I have mixed feelings about whether a dynamic language of any sort is the right thing to study as it obscures some fundamentals that are important to learn. But that might just be my own bias from having starting with Pascal and C.

Do you still have to work with Perl on Windows?

When I worked as a consultant, I had a work-issued Windows laptop and used it as my everyday Perl platform. Those struggles led me, ultimately, into CPAN Testers and helping Adam Kennedy create Strawberry Perl.

But once I could get free VM software, I shifted to a Linux VM for my day-to-day Perl programming. Much of my software was still "Windows-conscious", but I stopped wrestling the Perl rock up the Windows hill anymore.

Now that I'm at MongoDB, I have to care about Windows portability, so I'm starting to do more work on Windows again. I've been particularly happy with David Farrell's berrybrew, as it has made testing on various Perl versions much easier.

What kind of games do you play?

I mostly play board and card games these days (i.e. offline games). I recently got to try Seven Wonders and enjoyed it a lot. At home, my game playing is a bit biased towards things the kids can play, too. Ticket to Ride is in heavy rotation and my kids are pretty fond of Uno, as well.

18 Philippe Bruhat (Book) (August 2015)

How and when did you learn to program?

I got my first computer in my teens (I was probably 13 or so). It was a TO7/70 (a French microcomputer). The only available language was a Microsoft BASIC. I remember typing in pages and pages of programs read from the pages of Hebdogiciel, a French weekly paper with printouts for every computer of the time (Oric, Sinclair, Commodore, etc.).

All back issues are online! http://www.abandonware-magazines.org/affiche_mag.php?mag=7&page=1

I also remember that in the end, most programs where actually binary dumps (in hexadecimal), with a loader. It took me a while to understand why the only letters were A to F, and why some sequences showed up more often than others.

In the end, I bough the "Assembler" cartridge, but the examples in the accompanying book were wrong, so I never went very far.

What editor do you use?

I'm a vim user, although my `.vimrc` is really sparse. When I started using Unix, I remember the choice was between Emacs and Vi, and someone had told me that, no matter how few utilies were available on a system, there would at least be a version of `vi`, so it would always be useful to learn. This is how I picked it.

For years I used vim without knowing that many commands. In 1999, I bought a vi mug on Internet, and learnt a few more tricks from it. The real breakthrough, though, was when I was told by a friend who paid attention to the mug that any `:` command can be preceded by an "address" and will then apply to the corresponding section of the file.

The mug seems to be still available, via cafepress: http://www.cafepress.com/geekcheat.11507711

The cheat sheet printed on the mug looks like this: http://f2.org/image/archive/vi-mug-detail.gif

I'm a terminal user. I remember putting the finishing touches on my entries for the 5th Obfuscated Perl contest in a `screen` session on an actual VT320 hooked to my PC via a serial cable. I still have that tty on a shelf at home, catching dust.

My first real Unix email client was Elm (my first email account was actually on a VMS machine, but I only exchanged a dozen emails before leaving for Unix and have never looked back). Then I switched to Pine, and had a short period using Eudora (while stuck on a Windows 95 box for a few months). I switched to Mutt in November 2002, and I'm still using it as my main MUA.

When and how have you been introduced to Perl?

My first Perl program ever was probably a stat counter. And I didn't write it, I just copy-pasted it. Later on (with my website getting more hits), I vaguely remember adding something to deal with file locking, so that the counter wouldn't get corrupted.

Another program I "wrote" early on was a guestbook (those were the days). I very clearly remember a very cryptic line of Perl code. Today, I'm about 99.9% sure that the code that was given to me came from the infamous Matt's Script Archive, and that the cryptic line was this one:

```
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
```

The guestbook is still visible on the Internet Archive, http://web.archive.org/web/19970705232331/www2.ec-lille.fr/~book/goldenbook.htm and among the silly messages is the first contact I ever had with the woman who would later become my wife. Who needs online dating services when there's Perl and CGI?

What are other programming languages you enjoy working with?

Even if I agree with the popular wisdom that says we should know several programming languages and use the best tool for the job, I'm a little ashamed to say I use Perl for about every programming task.

I have written quick prototypes in Bash, but then usually turn to Perl as soon as the program gets more complex.

I have written a few simple PostScript programs, when I need to control exactly how something will look on paper. And I'm very proud to say that I have once written a whole presentation in PostScript, and that some of the slides used the `rand` function to draw some graphs. I also had a `Makefile` to recompute the slides (basically replacing a few constants at the top) so that they would always look good, no matter the aspect ratio of the projector (4:3 or 16:9).

I also wrote at least one non-trivial program in Befunge.

Obviously, to work with today's web, I had to learn some JavaScript (jQuery is cool). I have also written a few lines of Python, since it's the main language used in my team (we have this weird application where most of the work is done in Python, and the crazy bits are done in Perl).

I have a several books about programming languages I want to learn (Haskell, Erlang, Lisp), but because Perl fills all my needs, I never have enough time and motivation to actually learn a new programming languages.

What I really enjoy is the history of computing. I own many books about the history of Unix and the Internet, and I'm always looking for the "classics". One of the last computer books I read from cover to cover was "Elements of Programming Style", 2nd Edition, by Kernigham and Plauger (1978). Which means I learnt bits of PL/I and Fortran while reading it. And before that, I enjoyed "Life with Unix" by Don Libes (1989). And of course the "Unix-Haters Handbook", by Garfinkel, Weise and Strassmann (1994).

What do you think is the strongest Perl advantage?

The easy answer is CPAN, with CPAN Testers as a close second, but those are not my answer.

Perl is a wonderful programming language, but Larry's great success has been to grow a wonderful community around it.

In my view, Perl's strongest advantage is its people. The community.

I didn't consider myself a Perl programmer until I joined the Paris Perl Mongers in April 1999. I've been one ever since.

My favorite Perl interview was given by INGY in August 2004. I still get goosebumps where I read the last paragraph, the last sentence of the interview: "But in the final analysis, it's the Perl community that keeps me around. These people are my weird looseknit family. Perl attracts a certain type of people; these are my people."

INGY's interview: http://osdir.com/Article1534.phtml

What do you think is the most important feature of the languages of the future?

The ability to use the multiple cores of a CPU without the need for forking and IPC. And then, good networking on top of that ability, to grow your computing power further. "Big data" has pretty much become a buzzword nowadays; I know a few people who are deeply involved with that at work, and clearly, on the technical side, any sufficiently successful project will end up with one single issue: scale.

Success, when you're dealing with the Internet, means that whatever you're dealing with can grow by a few orders of magnitude in very little time. If you don't want to drown in your own success, you'll need tools that can keep up with the growth. In some areas at work (a very very successful Internet company), we've reached the limits of Perl 5.

What are you thoughts on Perl 6?

I'm old enough in the community to have been around when the Perl 6 project was announced. I remember the excitement, the RFC process. I remember hearing about mugs being thrown at a wall (but not the Wall!). I remember Dan Sugalski leading the Parrot project.

I've tried to learn Perl 6 when Pugs was hot, but never went farther than a few examples.

I've followed Perl 6 progress from afar, knowing that one day I'd have to actually learn it. I very much like how Larry presents Perl 5 and 6 as siblings in the Perl family of languages. (Also, have a look at his "State of the Onion" talk from 2006 about families: http://www.perl.com/pub/2006/09/21/onion.html)

I love the Perl 6 logo, the butterfly, and its name, Camelia. (And how this name is tied to the name of the Perl 5 camel, Amelia.) I also love the fact that this logo pisses off "manly" programmers (should I say "brogrammers"?). I don't think Larry should present it as "appealing to seven year old girls", though, because in my opinion stating this is reinforcing the stereotypes he wants to go against.

I wanted to go to FOSDEM for years, but never managed to fit that in my schedule.

I finaly went this year (although I spent less than 24 hours in Brussels including the time spent travelling there), just to hear Larry's announcement about Perl 6 coming out in 2015.

A few weeks ago, I started to try and program in Perl 6 again, this time with a clear goal: to port one of my Perl 5 modules to Perl 6, and blog my journey, the journey of a fairly experienced Perl 5 programmer trying to write a real module in Perl 6.

What are you doing for YAPC::EU Foundation and why?

I'm one of the co-founders of the YAPC Europe Foundation. In 2003, right after the YAPC Europe in Paris, the organisers from the past four YAPC Europe (London, Amsterdam, Munich and Paris) gathered together in a bar to discuss the future.

YAPC Europe had taken a life of its own, and didn't need the people from YAS (Yet Another Society, the non-profit that later became TPF), who were an ocean and several time zones away, to help the European community organise its conferences.

The Paris team had set up an online payment system connected to our web site (which was rewritten the year after for the first French Perl Workshop, and named "Act"), and we wanted to make this available to other Perl conferences, so that the work and money that went into that payment system would not be lost.

Because the team was international, the organisation was registered in the Netherlans as a Dutch Stichting, with a bank account in France (to keep the same bank and online payment system we used in Paris). And our first chairman was German.

The YAPC Europe Foundation (http://yapc.eu/) has two main roles: selecting the organising team for the next YAPC Europe conference, and managing the online payment system. This online payment system is actually offered for free to the Perl community, since YEF is paying all the bank fees. The entire YEF budget comes from donations, and is spent on the online payment costs and the kickstart donations to various Perl conferences. http://www.yapceurope.org/finance/kickstart.html

In 2008, I took over the treasurer role, because our previous treasurer didn't have enough time for the Foundation any more. So I'm the one who's sending the money paid by the attendees to the organisers. Since 2005, YEF has helped close to €350,000 to flow from the attendees and sponsors to the organisers of 45 Perl conferences.

Where do you work right now, how much time do you spend writing Perl code?

I work at Booking.com mostly remotely from Lyon since March 2007. The company has grown by a factor of at least ten (and sometimes much much more than that) on absolutely every metric since then.

I write Perl most of the time (I'm still a developer), in a team where most of my colleagues swear by Python. My current project uses Moo (for a while I used both Moose and Moo, and then factored Moose out), Dancer and DBIx::Class. Because the project has a web interface, I also write some JavaScript.

Should we encourage young people to learn Perl nowadays?

I can't find the exact quote from Larry, but he explained once that Perl is more likely to be the last programming language that someone will learn, rather than the first.

People should definitely learn some Perl, in addition to all the other languages in their curriculum. Everyone says, people should know more than one language, and the best time to do that is when learning to code. Teachers should teach as many languages as possible, in as many paradigms as possible (imperative, declarative, structured, functional, OO, etc,) so that their students can understand the differences and similarities between them.

And then the students can discover Perl and mix everything together.

How did you decide to collect in one place all the perl secret operators?

I've been hanging around on the Fun with Perl mailing for a long time, and this is where José Castro started to ask for the names of the most famous secret operators for his OGSOP (Obfuscation, Golfing and Secret Operators in Perl) book project. I came up with the name "baby cart" for @{[]}, and I loved that name so much that I wanted to make sure it would be the one that everyone would use. The only way to do that was to write the reference manual myself...

I also did a few presentations about the Perl secret operators in 2008-2009 at various Perl conferences. And I naturally started to collect them.

So I had this outline for a "perlsecret" manual page (modeled after the "perlop" manual page) sitting somewhere in my projects directory (the oldest version I could find was from 2010) and someday I decided to finally work on it. My other goal was to get it included in the Perl core documentation, but it turns out that making a reference to "Goatse" is not a desirable feature for a core manual page.

Git::Repository uses git command, would there be any advantage in using libgit2?

The fact that there is no libgit is a big issue with Git. That means that the only way to keep track of git and all of it options and subcommands is to run git itself. (A little bit like only perl can parse Perl.)

Speed is not really the issue: git is extremely fast, and the cost of forking would still exist in a shell script running the same commands. A program using the hypothetical libgit would probably be faster than a shell script chaining git commands, but I doubt another library (like libgit2) can be faster than git itself for doing git operations.

Git::Repository is wrapping repositories in simple objects that provide all the context needed to run git commands. For me, the benefit of using Git::Repository is the fact that I can work with multiple repositories without having to worry about the current working directory. It was also explicitely designed to be able to run any of the 128+ git commands, and deal with a stream (of any size) produced by Git.

There are some bindings for libgit2 on CPAN (Git::Raw), but I never used those. One of the things I read about libgit2 is that the objects can be stored in more than just the git loose objects and the git packs. This can be useful when dealing with git objects at a large scale (like GitHub does).

Why pink?

This is a long story.

I've been wearing pink at Perl events for a long time. I've had some Perl conferences (and not just the French ones) make a pink version of the conference shirt just for me. I own one of the only two pink Perl Dancer T-shirts. At YAPC in Pisa, Nick Clarck and BinGOs impersonated me by wearing a pink shirt. At work, some of my colleagues demand that I wear pink, or at least ask in a demanding voice why I'm not wearing pink (when sometimes I'm not).

Here's the story of why I wear pink shirts. Back in 2001, during the YAPC Europe conference in Amsterdam, a small group of French Perl mongers thought: "why not organise a YAPC in Paris?". Then we went to a meeting with Kevin Lenzo (from YAS) and the Amsterdam organisers, to talk about the next YAPC. To our surprise, there was another team willing to organise YAPC Europe in 2002. In the end, it was decided that the first team to be ready would get the next YAPC. It turns out the

Munich team was ready first. And therefore we had two years to prepare for our YAPC.

Back in 2002, auctions were still pretty popular at YAPC conferences. During the auction at the end of the Munich conference, many silly items (a Perl 6 opcode, a Perl 5 optimization, Schwern's pants, a topless arm-wrestling contest between Schwern and Damian Conway, ...) were sold. Remembering the fight between factions of London.pm (the regulars and the heretics) over the date of their monthly social meeting during the Amsterdam auction, I hoped to re-create such an epic fight by offering the colour of the organisers shirt for the YAPC in Paris (everybody knew it would be us, so there was no competition this time).

My idea was that Léon Brocard would bid for his favourite colour (orange) and that some faction of London.pm would gang up against him for some other colour, and that lots of money would be spent. Alas, it turned out that the British like to tease the French more than themselves… Greg McCarroll bid for pink, and it went quickly for €100. Some of the Paris team started to get really angry at me very quickly…

I decided to turn the joke made at our expense into a marketing trick, and we made pink the official conference colour. The web site was pink (see http://conferences.mongueurs.net/ye2003/coming_soon.html and http://conferences.mongueurs.net/ye2003/, the flyers were pink, the organisers shirts (all 12 of them) where pink.

The year after, at the auction in Belfast (a story to be told some other time), I realized that "pink is us" (the French YAPC), and later on started to use pink as my distinctive colour (since none of the other French guys wore their pink shirt ever again). And I have done so for the past ten years or so. This is usually how people recognize me at Perl events.

The sad moral to this story is that one does not need to be a brilliant coder to be recognized at a Perl conference (I don't claim to be either brilliant or famous), one just need to do something stupid and stick with it.

I actually like wearing pink, because it's also, in a small way, a way to stand up against gender stereotypes. I'm quite happy that my two daughters think that pink is dad's favourite colour (it's not really), rather than a girl's colour (Colours should not be gendered! Everybody should be free to wear whatever they want!).

19 Author

Viacheslav Tykhanovskyi is a russian programmer who is occasionally busy with consultancy work and maintainership of his open source projects. Right now he is running a side SaaS project devoted to software quality kritika.io in the hope of it becoming his full time activity.

In his free time he rides his road bicycle and participates in long-distance audax events (so-called randonneuring).

GitHub Twitter CPAN