Perl 6

The story so far

Kirrily Robert
e-smith, inc

Why Perl 6

At TPC 4.0 ("The Perl Conference", run by O'Reilly and Associates in July 2000) a meeting of Perl 5 developers decided it was time to start planning for the next major version of Perl: Perl 6.

The reasons for this decision included:

- Perl 5 internals are becoming increasingly difficult to maintain
- Perl 6 offered a chance to improve the process by which Perl itself is developed
- Increasing "competition" from other languages (particularly Java, PHP, and Python) prompts us to reassess Perl's usefulness for real-world development
- Perl 5 developers have been blessed with 20/20 hindsight on some decisions (e.g. pseudo-hashes) and would welcome the chance to make those decisions differently a second time

Constraints on Perl 5 (Larry's list)

Larry Wall (the founding father of Perl) offered the following list of constraints he perceived on Perl 5, at a speech at the Atlanta Linux Showcase in October 2000. Emphasised phrases are Larry's; the plain text parts are my own annotations of Larry's points.

- *The need for backward compatibility:* Perl 5 attempted to be backward compatible with Perl 4, which attempted to be backward compatible with Perl 3. This leads to a situation where legacy or deprecated language constructs are still supported, even if they really should be thrown out.
- *My leisurely mental growth:* Although most people would be stunned to hear Larry describe his mental capacities as less than godlike, it is a fact that Perl 5 was Larry's rewrite, and needed to fit within the scope of his comprehension.
- *Cultural model with too many bottlenecks:* Perl 5 development centred around the perl5-porters mailing list ("p5p"), and was presided over by a single "pumpking" (project leader) at any one time.

- *Lack of knowledge of the future:* When Perl 5 was first envisaged, the World Wide Web was only just gaining in popularity, XML had never been heard of, and numerous other technologies that we now take for granted were only a dim light on the horizon.
- *Lack of round tuition:* There were a lot of things that might have been done with Perl 5 if Larry and/or the other developers had got a "round tuit".
- *Complexity of the core:* Perl 5 was a significant departure from Perl 4, and introduced a lot of new concepts. The implementation of many of these turned out to be quite complex, to the point where a potential Perl 5 developer may not be able to understand it.
- *Complacency:* At the outset of Perl 5, Perl was acknowledged as one of the best and most widely used tools for toolsmithing on Unix systems, and for CGI scripting for the World Wide Web. For some years, Perl rested on its laurels and failed to address new fields of programming as they rose in prominence.

Tinkering with the development process

Having decided to start work on Perl 6, the first issue addressed was how to improve the development process. An examination of the existing process will underline why this was considered necessary.

How it used to be (and still is, for Perl 5)

Perl 5 development centred around one mailing list, perl5-porters, aka "p5p". This mailing list was home to a core group of perl hackers and an ever-changing procession of hopeful contributors, with Larry presiding over the whole thing as a benevolent, but mostly uninvolved, dictator.

The p5p list was very high volume, and many subscribers dropped off it when they were unable to keep up with the discussions. For those without the courage to brave the deluge, a weekly summary was posted on the web and available by email. This provided links to threads or specific posts of significant interest, all of which were available via a web archive.

As well as being high volume, p5p had a reputation for unfriendliness and for being the home to some particularly heated flame wars. Certain personalities on the list were renowned for the frequency and vehemence of their flames, and newcomers were often scared away by the argumentative atmosphere or the cold welcome they received.

More recently, a system of "refereeing" has been introduced, in which a small group of trusted regulars are charged with keeping the peace. The means at their disposal include out-of-band warnings to offenders and moderation of posters who repeatedly ignore warnings.

Why change it?

A number of reasons were given for why the Perl development process needed changing. These included:

- Reverse p5p's bad reputation
- Give everyone an opportunity to contribute -- not just the old-timers and Unix users, but also the Windows contingent, the WWW scripting crowd, and those who use Perl for purposes ranging from astronomical data-crunching to mainframe applications development
- Record discussions and decisions for future reference, so that late joiners could read a clear archive of what had gone before
- Apply a bit of real project management
- Foster the kind of working environment that doesn't drive valuable people away

How did we make decisions about this?

Straight after the TPC 4.0 meeting and announcement, a mailing list called perl6-bootstrap was set up, with the goal of determining what shape the subsequent process and community should take.

Unfortuneately, the bootstrap phase was given only a couple of weeks to make decisions, and the first week of that was largely wasted while travelling TPC attendees returned home, recovered from their jetlag, and caught up on their email backlogs.

The result was rushed decisions that upset some people. However, most of the people were upset because the decisions had been rushed, not because they were necessarily bad. Few suggestions were offered that differed significantly from what we ended up with -- most differences were in the naming of mailing lists and other such (relative) trivia.

The shape of Perl 6 development

Here's what the bootstrap list came up with:

- Project manager: Nathan Torkington
- Working groups, lead by WG chairs
  - Language (Kirrily Robert)
  - Internals (Dan Sugalski)
  - Standard Library (Graeme Barr)

  - Build (Jarkko Hietaniemi)
  - QA (Michael Schwern)
  - Licensing (Bradley Kuhn)

  Each working group has a mailing list, and may spawn sub-lists (with chairs for each one) as required.

- Working groups to develop RFCs describing possible changes to make in Perl 6. Note that these are actually "Requests For Comment", not fixed standards as IETF RFCs tend to be.
- Larry would create an overall design for Perl 6 based on the best suggestions to come out of the RFC process.

Project timeline

- July - September: Working groups develop RFCs
- October: Larry to work on design based on RFCs
- November - December: more detailed design discussion; sort out teams and process for implementation
- January: start coding
- July (TPC 5.0): present Perl 6 alpha

Where are we now?

- 350+ RFCs were submitted by September 30th (mostly Language related)
- Larry is still working on design
- Larry gave a talk at ALS outlining some of the design, and discussing how he's using the RFCs
- Waiting, waiting, waiting...
- Meanwhile, we are working on specifying how the detailed design and implementation process should be managed. In particular, we are setting out a format for Perl Design Documents (PDDs) which will

describe specific aspects of the design for the information of new developers, or for reference purposes for old developers.

- An apprenticeship program has been discussed to assist new developers in becoming familiar with the internals and the development process.

What will Perl 6 be like?

Again from Larry's talk at ALS, here is his list of constraints on the development of Perl 6:

Constraints on Perl 6 (Larry's list)

- Can Larry understand it?
- Do people really need the new feature?
- Can we implement it efficiently and robustly?
- Does it preclude translation from Perl 5?
- Does the utility grow faster than the complexity?
- Does the size grow slower than Moore's Law
- Must take time to maintain Perl 5
- Must take time to design Perl 6 right

Skud's psychic powers predict the future (with help from Larry's talk)

- Perl will still be Perl
  - TMTOWTDI ("There's more than one way to do it") -- just because one Perl language construct duplicates the functionality of another is not sufficient reason to avoid it.
  - Make easy things easy, hard things possible -- if a feature achieves this, it is in keeping with the Perl spirit and is likely to be considered for inclusion.
  - Perl is not Python, Java, C, BASIC, COBOL, Lisp... and will not become any of these, despite the loud arguments put forward by some proponents of these languages during the RFC process

  - ... but it should be easy for people from those languages to pick up
  - May also be easier to connect directly to those languages (e.g. Inline.pm, cleaner XS replacement, etc.)

- Language changes
  - Death to global variables! (especially punctuation vars such as `$/`); these are likely to be converted to some kind of OO interface.
  - Deprecated features removed. For instance, the use of `'` as a synonym for `::` will die a peaceful death of old age
  - Non-critical builtins move out to modules, including formats, shared memory functions, `gethostbyname` and friends, etc
  - Easier and safer OO programming
  - Easier functional programming
  - No more typeglobs (but none of their functionality lost)
  - Filehandles become objects
  - Easier interpolation of complex expressions

    ```
    print "The answer is $obj->answer()";
    ```

  - Optional typing of scalars (int, float, etc)

    ```
    my int $answer = 42;
    my int @array = (1, 2, 3);
    ```

  - Better subroutine parameter specification mechanism
  - A switch statement, based on Damian Conway's excellent (and very Perlish) suggested implementation
  - wantarray() becomes a more generic want()

- Standard library changes
  - Greater consistency
  - More applications development support: web, XML, CORBA, ???
  - Some stuff moved from the core to the library: date/time, shm*, formats, etc

- Internals changes
  - More modular
  - Easier to maintain
  - Much easier to extend
  - Better garbage collection
  - We'll know more when we start on detailed design and implementation

- Licensing changes
  - Possible rewrite of the Artistic License

Where next?

perl6-meta is in limbo right now; we need impetus, which should come when Larry releases his design

The detailed design and implementation will take a lot of careful management to happen smoothly. I believe that Nat Torkington and Dan Sugalski are up to the job, and I hope to be able to help out as well.

Will an alpha be ready for TPC 5.0? Who knows. I hope we'll have something to show off, but I also hope the world understands what "alpha" means, and doesn't expect a fully fledged Perl, because it won't be. I'm guessing we'll present a design, but no working code beyond individual modules of no use in the real world.

Long term...

Perl 5 will continue to exist and be maintained. Perl 5.7 is under active development now, 5.8 is ahead of us, and there may be more 5.X versions beyond that. ActiveState have committed to supporting Perl 5 after Perl 6 is released, and a few individual developers have said they will continue to work on Perl 5 for as long as necessary.

It will be at least 2 years until a stable Perl 6, and even longer until it's widespread. For comparison, consider that there are *still* people using Perl 4 more than five years after the release of the much more featureful Perl 5. Of course, popular uptake of Perl 6 depends on it being as much of an improvement over Perl 5 as Perl 5 was over Perl 4.

Nevertheless, this is not worrying us. We hope to do it right, rather than fast.

More information

- http://www.perl.org/perl6/
- http://dev.perl.org/