How (not) to create a language specification for Perl 6

Lessons learned

Patrick R. Michaud
pmichaud@pobox.com

FOSDEM 2015
Brussels, Belgium
January 31, 2015
Overview

What are language specifications?

Perl 6's design process, some mistakes

Important features of language specifications

Where to go from here
Reflections on Perl 6 language specification

Imprecision in our discussions

Time to apply some rigor

Release of Perl 6.0.0 will require it

Opinions expressed here are solely those of the presenter

May be unwise, untested, incorrect, etc.
Language specification basics
What is a “language specification”?

Many languages have them (Ada, Pascal, C)

Many languages don't have them
  Perl 5
  PHP prior to 2014

Different forms of specification
  Explicit definition using syntax and formal semantics
  Natural language description
  Model implementation
What is a “specification”?

From Wikipedia (italics added):

Specification (often abbreviated as “spec”) may refer to an *explicit set of requirements to be satisfied* by a material, design, product, or service.

Specification generally contains requirements, not conjectures.
Example language specifications: C

*The C Programming Language*
Kernighan and Ritchie, 1978


C99

C11
Example language specifications: Python

Python doesn't have an official language specification

Python Language Reference describes the language, but leaves some details ambiguous
Example language specification: HTML / RFCs

HTML is in fact a language
It has a formal specification
Maintained by W3C
Early versions were RFCs

Many RFCs are specification documents
HTTP, SMTP, URLs, etc.
How is the Perl 5 language specified?

The camel book?

The interpreter?

The test suite?

Larry?
Perl 5 doesn't have a separate written language specification.

The Perl 5 interpreter and its functional tests serve as the *de facto* specification.

Whatever behavior the Perl 5 interpreter has, that's the “standard” behavior.

If Perl 5 misbehaves, see the previous rule. Even if Perl 5 changes its mind.
The Perl 6 language spec history
Brief Perl 6 history

Perl 6 announced July 2000

RFCs commissioned
361 submitted

Larry refined these into Apocalypses and Synopses

Unlike Perl 5:
Perl 6 would first become a specification
Then realized by one or more implementations
Oops.
In retrospect, targeting a “language specification” before implementation is a mistake.
“Specification” is too loaded with meaning

Implies a level of rigidity and permanence

“design plan”? Yes
“Synopses”? Sure, that works

Careless use of “Perl 6 specification” has led to much confusion about development of Perl 6
Confusion in the community

“When will Perl 6 be ready?”

“Is the Perl 6 specification finished yet?”

“Well, no wonder it's taking so long, if you can't even decide on a specification first.”
I've always disliked this one...

“Perl 6 needs to freeze a specification immediately, implement that, and release it.”

No.
Misunderstanding language design

Many assume a specification precedes language implementation

It's a common misconception

Descriptions of Perl 6 development reinforced this incorrect notion … and still do!

Reality: Successful languages and systems are striking counter-examples:

Perl 5, PHP, C, Ruby, HTML, HTTP
Premature specification examples

HTML+ (1993)
   Effectively delayed HTML and browser development

C99 (1999)
   After C99, the C standards committee adopted guidelines to limit adoption of new features untested by implementations.
“Writing a specification before an implementation has largely been avoided since ALGOL 68 (1968), due to unexpected difficulties in implementation when implementation is deferred.”

– Wikipedia
Lesson learned

Specification freezes aren't like code freezes.

Specification releases aren't like code releases.

Specs should be (very?) retrospective.
Proposals start as Internet Drafts

Become Requests for Comments (RFCs)

May be considered on the Standard Track as a “Proposed Standard”

Promotion to “Internet Standard” requires:
   Two independent operating implementations
   No errata against specification
   No unused features that increase complexity
   Two independent uses of any licensing restrictions
Key features of (Perl 6) specification
Lesson: Evolution is a constant

A programming language is never “frozen”
(until it's dead)

Perl 6 design explicitly recognizes evolution:
  Lexically scoped language modifications
  Versioned specification
  Versioned modules
  Macros
  Custom operators / parsing / DSLs
  Slangs
  Augmented classes / MONKEY_TYPING
Classes are never “final”
Sharp distinction between “specification” and “implementation”

“Perl 6” and “Perl 6.0.0” refer to the language

No “official” implementation of Perl 6

Multiple implementations are key to long-term adaptation, evolution, success
Lesson: Synopses should not be spec

Synopses were the original “Perl 6 specification”

These change frequently with language evolution and discovery

Difficult to version synopses as spec

Changed circa 2008 to:

“Perl 6 is anything that passes the official test suite.”

– Synopsis 1
Where we go next
Establish / release “Perl 6.0.0”

Better understanding of “Perl 6 spec”

Specification follows implementations

Specification is a set of tests...

...not the design documents
Some documents still refer to Synopses as “official Perl 6 specification”

Various histories of Perl 6
Wikipedia and similar articles

Fix these!
Attention!

Our GitHub repository for Synopses is (mis)named “specs”

This bothers me

I will change this

Soon

Without further warning

Forgiveness > Permission
Perl 6 specification will be defined by test suite

Current test suite has things that are not yet “spec”

It will always have such “extra” tests

Mechanism to identify / extract the test suite for a given Perl 6 version

Git tags useful but likely not sufficient
The “roast” test suite already exists in subsets

Per-implementation “todo” and “skip” markers pre-processed into test files

```perl
#?rakudo.moar todo "Not yet implemented"
```

May be able to use similar markers for Perl 6 versions

```perl
#?v6.0.0 omit 5 "Conjectural"
```
Consider feature lifetimes

Language features may have lifetimes:

Conjecture
Work in progress
Adopted
Discouraged
Deprecated
Retired

Perhaps specification should explicitly recognize this somehow
Version guidelines

Criteria for declaring new versions of Perl 6

Time-based language specification?
Tag Synopsis documents?

Develop way to tag Synopsis documents with language version information

Perhaps at section / paragraph level

Doesn't have to be static or snapshot, can evolve over time
Recap

Widespread misconceptions about the role of “specifications” in language development

Specifications work best as historical markers

Languages evolve

Perl 6 has robust features for evolution

Separate specification and implementation

Test-based specification

Need to design versioning standards
Questions?