NAME

Module::Install::Philosophy − The concepts behind Module::Install

SYNOPSIS

This document describes the personal philosophy behind the creation of **CPAN::MakeMaker** (the predecessor of **Module::Install**). The views expressed here belong to Brian Ingerson; if they are not of interest to you, you can safely ignore this document.

I HAVE A DREAM

I say to you today, my friends, that in spite of the difficulties and frustrations of the moment, I still have a dream. It is a dream deeply rooted in the Perl Module dream.

I have a dream that one day this community will rise up and live out the true meaning of its creed: ‘‘We hold these truths to be self-evident: that all Perl authors are created equal.’’

I have a dream that one day even the state of the `CGI::` namespace, a desert state, sweltering with the heat of injustice and oppression, will be transformed into an oasis of freedom and justice.

I have a dream that my four modules will one day live in an archive where they will not be judged by the number of their prerequisites but by the content of their source code.

I have a dream today.

DESCRIPTION

The above is obviously a mutation of the monumental speech by great Martin Luther King (<http://web66.coled.umn.edu/new/MLK/MLK.html>). While the contexts are vastly different, I feel that there are some serious parallelisms.

The CPAN has become a place that is not free of injustice. This situation has arisen not out of directed oppression, but from a failure of our community to keep its tools sharp. It is the culmination of many small decisions made in the name of practicality. This is a sad state for an institution that was created to allow all interested people to contribute equally to the best of their ability.

This assertion is rooted in my personal experience as an author. When I created my first Perl module, Inline.pm, I knew that I had done something important. But how was I to make a dent in vast Perl community?

As a complete unknown in the Perl community, my voice did not travel far. I repeatedly tried to get even an acknowledgment from the gurus familiar with XS. No success. I resorted to sending messages with ridiculous subjects to `modules AT perl DOT org`. (<http://www.xray.mpe.mpg.de/mailing−lists/modules/2000−08/msg00078.html>) No response. Through sheer determination and shameless self-promotion I eventually got the word out, and I hope the world is a slightly better place for it.

Since then, Inline has won awards and I have had the privilege to meet almost all of Perl’s finest. But I still remember the pain of starting out, and want to help invite more people into this wonderful world.

One thing I have learned from experience is that the Perl community (and throw in the Python and Ruby people as well) is a small drop in the vast ocean of programming. It’s a giant pot of Java out there; and a sea of C. Perl may not be the biggest fish, but with some care and cunning we could become a much bigger school.

These are the current problems that I see with CPAN and the core modules:

- New Modules don’t help Older Perls

  If I were to guess what percent of all Perl5 installations were at the current release level (5.8.0 in October 2002) I would say 3−5%. That may even be generous. I’d say that over 40% of installations might still be at 5.005 or earlier.

  The biggest problem with adding a module to the core is that it only helps a small subset of Perl users for a long long time. Worse yet, a good module author will still probably avoid using the core additions as prerequisites, because they want their new module to work as well on 5.005 as on 5.8.

  CPAN::MakeMaker should be able to help in this regard. For example, instead of putting Inline.pm into the core for 5.9, I can now effectively get it into the core for every version of Perl that Inline supports.

- Author Exclusiveness

  Not just anybody can get a module into the core. It seems you have to know people in high places. If I were a brilliant new talent with a great new module, it would have a harder time getting the ear of the pumpking, then if I were, say, Damian Conway. In fact, I probably wouldn't even know where to start.

- Reduced Competition

  One comment I've heard from some very good Perl programmers is ''Everything important has already been done''. Their feeling is that even though a module is suboptimal, it would be a waste of time to write a competing module. Who would use it instead of the one already in the core?

  When I write a competing module, I know that I have to make it at least twice as good as the existing one to even get noticed. That's not a bad thing, but should everybody be forced into that situation?

  For example, let's say that you have created a really useful CGI script. Let's also say that it makes use of your own **CGI::Special** module, because **CGI.pm** doesn't meet your needs. Even though your script might be generally useful and worth sharing, the fact that it requires a non-standard module can only negatively affect its acceptance. Trying to get general acceptance for the superior **CGI::Special** module will be harder still.

  Core modules are assumed by the general public to be ''Best of Breed''. While this may be true for some modules at some point in time, it keeps talented people from attempting to ''breed'' something better.

- Core Bloat

  Every time we add a module to the core it gets bigger and bigger. And we can't ever remove modules from the core, once they've been added.

  If I had my druthers, we'd remove all modules from the core that weren't necessary for either running Perl or installing modules. Of course, we'd need to set things up so that installing modules was so easy, that it could be done on the fly if necessary. Is this easily accomplishable? Nope. Is it impossible? Nope. We have the best language in the world to help us do it!

- Maintenance Bitrot

  Believe it or not, Perl authors can sometimes acquire a ''Life Beyond Perl''. They get families or new hobbies or even hit by a bus. (This would be a ''Death Beyond Perl''.) The fact is, that once somebody writes a piece of code and shares it with the world, they are expected to maintain it for all time.

  That is being generous. There are others that think that once their module has become popular or made it into the core, they don't need to keep fixing and improving it. I have personally been guilty of this sin.

  And then there's the Damian Conway Effect. This plagues the exceptional authors who are so innovative and prolific they simply don't have time to maintain everything they have written.

I initially formalized these opinions at the YAPC (Yet Another Perl Conference) in June 2001. Since then I have been trying to think of technological solutions to fix these social problems.

One idea was dubbed NAPC. NAPC is CPAN backwards. It is a large system of precompiled modules that can be installed on the fly, with the goal of reducing the number of modules in the core. NAPC hasn't got started yet. I'd still like to do it someday, but it's a big problem with a lot of issues.

CPAN::MakeMaker (and now **Module::Install**) on the other hand, is simple and ultimately flexible. It should work with all of the existing CPAN processes without requiring any changes from them. And new features can be continuously added. Even though it doesn't scratch all of my philosophical CPAN itches, it's a good start.

CONCLUSION

This is all just food for thought. Take it with a pinch of salt.

AUTHOR

Brian Ingerson <INGY AT cpan DOT org>

COPYRIGHT