In Search of Silver Bullets for Polyglots

Arto Bendiken

Overview

1. The Challenge
2. A Solution?
3. A Paradigm?
4. Something Crazy
5. An Opportunity
6. Silver Bullets?
7. Questions & Answers

Polyglot Programming

The Challenge

polyglot | ˈpälēˌglät |

adjective

- knowing or using several languages

noun

- someone who knows and is able to use several languages
- a mixture or confusion of languages

WHICH PROGRAMMING LANGUAGE

SHOULD I LEARN FIRST?

WHAT IS PROGRAMMING?

Writing very specific instructions to a very dumb, yet obedient machine.

LANGUAGES

PYTHON · JAVA · C · PHP · C++

JAVASCRIPT · C# · RUBY · OBJECTIVE-C

START

WHY DO YOU WANT TO LEARN PROGRAMMING?

- I don't know, just pick one for me
- Just for fun
- I'm interested
- Improve myself
- For my kids → Start with Scratch, then move on to...
- Make money → I have a startup idea!
- Get a job → Which platform/field?

Which platform/field?

- I want to work for big tech companies
- Front-end (web interface)
- Doesn't matter, I just want $$$
- Web → Back-end ("brain" behind a website)
- 3D/Gaming
- Mobile → Which OS? → iOS / Android
- Enterprise
- Corporate → What do you think about Microsoft? → I'm a fan! / Not Bad / Suck
- I want to work for... → Startup

Have a brilliant idea/platform in mind?

- YES → Which platform/field?
- Nope. Just want to get started → I prefer to learn things...
  - The easy way
  - The best way
  - The slightly harder way → Auto or Manual car? → Auto / Manual
  - The really hard way (but easier to pick up other languages in the future)

Do you want to try something new with huge potential, but less mature?

- NOT SURE
- NO → Which one is your favourite toy? → Lego / Play-Doh / I've an old & ugly toy, but i love it so much!
- Web → Does your web app provides info in real-time, like twitter? → YES / NO

“**We are entering a new era** of software development. For most of our (short) history, we’ve primarily written code in a single language. […] Now, increasingly, we’re expanding our horizons […] Applications of the future will take advantage of the polyglot nature of the language world.”

— Neal Ford, Polyglot Programming (2006)

“**We’re entering a polyglot era** in software development, driven by cloud and multicore systems architectures, as new languages emerge to challenge, and coexist with, the long hegemony of Java and .NET. [...] IT isn’t getting any easier, and scale demands are increasing exponentially. Therefore – it’s time to start seeing other languages.”

— James Governor, The Polyglot Revolution Continues Apace (2011)

Why Polyglot Programming?

- Best tool for the job at hand?
  - And there are just many more programmers and many more tools now…
- Best-of-class frameworks and tools have driven novel language adoption
  - **Ruby on Rails**!
- Pursuit of productivity on critical platforms with subpar base languages
  - **Client side**: JavaScript
    - Stagnation of JS language development until ES6 (2015)
    - Two polarized responses:
      - Double down on JS (cf. Node.js), for both client and server
      - Don't develop in JS, just generate it (e.g., GWT, RJS/SJR, CoffeeScript, TypeScript, Elm, ocaml_of_js, and hundreds more)
  - **Server side**: the Java Virtual Machine (JVM) ecosystem
    - Stagnation of Java language development until Java 8 (2014)
    - Emergence of alternative language ecosystem (Groovy, Scala, Clojure, Kotlin, etc)

“There was no way this polyglot reality could persist. Not given its cost. I’m not referring as much to the cost of the enterprise—which is very real—but rather, the cost to developers in terms of time and attention.
Make no mistake: the cost is enormous.”

— Matt Asay, Developers Are Calling it Quits on Polyglot Programming (2014)

“There is a real cost to this continuous widening of the base of knowledge a developer has to have [...] One of today’s buzzwords is “full-stack developer”. Which sounds good, but **there’s a little guy in the back of my mind screaming**: you mean I have to know Gradle internals *and* ListView failure modes *and* NSManagedObject quirks *and* Ember containers *and* the Actor model *and* what `interface{}` means in Go *and* Docker support variation in Cloud providers? Color me suspicious.”

— Tim Bray, Discouraged Developer (2014)

Some Polyglot Problems

- The maintenance cost
  - Initial implementation is often a small part of the total effort over an application's lifetime
  - People tasked with maintenance need be at least comfortable with languages used
- The paradox of choice
  - The set of technologies isn't static, new entrants appear frequently (cf. Dart, because Flutter)
  - The continuing trade-off analysis is exhausting
- The Red Queen effect
  - *"Now, here, you see, it takes all the running you can do, to keep in the same place."*
  - Technology never stays still, and it takes continuing effort just to keep up with existing tech
- The cognitive load
  - We aren't CPUs: we multitask rather badly, with large context-switch costs
  - Cognitive costs are proportional to quantity (# of languages) and quality (differing paradigms)

Cognitive Load: The Shallow End

Language	Type System	Main Paradigm	Class Naming	Method Naming
JS	dynamic, weak	object-oriented*	CamelCase	mixedCase
Ruby	dynamic, strong	object-oriented	CamelCase	snake_case
Elixir	dynamic, strong	functional	CamelCase	snake_case
Go	static, inferred	procedural	CamelCase	{M,m}ixedCase
Java	static, manifest*	object-oriented	CamelCase	mixedCase
Kotlin	static, inferred	object-oriented	CamelCase	mixedCase
Swift	static, inferred	object-oriented	CamelCase	mixedCase
Dart	optional	object-oriented	CamelCase	mixedCase

Cognitive Load: The Deeper End

Language	Type System	Main Paradigm	Class Naming	Method Naming
Julia	dynamic, strong	multi-dispatch	CamelCase	snake_case
Erlang	dynamic, strong	functional	snake_case	snake_case
Common Lisp	dynamic, strong	multi-dispatch	lisp-case	lisp-case
C/C++	static, weak	multi-paradigm	*various…*	*various…*
D	static, inferred	multi-paradigm	CamelCase	mixedCase
Rust	static, inferred	functional*	CamelCase	snake_case
OCaml	static, inferred	functional*/OO	CamelCase	snake_case
Haskell	static, inferred	functional, lazy	CamelCase	mixedCase

Cognitive Load: Simple Tasks

```java
// Java 8+
import java.nio.file.*;
new String(Files.readAllBytes(Paths.get("input.txt")));
```

```kotlin
// Kotlin
File("input.txt").readText()
```

```javascript
// Node.js
require("fs").readFileSync("input.txt")
```

```java
// Java 6: 20+ lines omitted
```

```ruby
// Ruby
File.read("input.txt")
```

```go
// Go
data, err := ioutil.ReadFile("input.txt")
```

Cognitive Load: Ecosystems

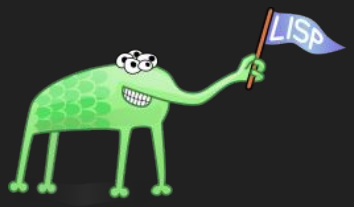
Language	Package Mgr	Test Framework	Code Coverage	Doc Generation
JS	NPM	Mocha	Istanbul	JSDoc
Ruby	RubyGems	RSpec	SimpleCov	YARD
Elixir	Hex.pm	ESpec	ExCoveralls	ExDoc
Go	Go/Git	Ginkgo	Go	GoDoc
Java	Maven/Gradle	JUnit5+AssertJ	JaCoCo	Javadoc
Kotlin	Maven/Gradle	JUnit5+AssertJ	JaCoCo	KDoc
Swift	Git	XCTest	Xcode	Jazzy
Dart	Pub	pkg:test	pkg:coverage	pkg:dartdoc

Code Generation

A Solution?

“Will write code that writes code that writes code that writes code for **money**.”

— seen on comp.lang.lisp



“I **object** to doing things that computers can do.”

— Olin Shivers

Code Generation FTW

- Code generation is a **force multiplier** for productivity: it gives you **leverage**
  - One line of high-level input code can be worth ten or twenty lines in the target language
- For some problems in computing, already the de-facto solution:
  - **Lexing & parsing**: writing parsers by hand is tedious and rarely needed (*exceptions: the C++ grammar*); parser generators (ideally) take a declarative EBNF grammar spec and churn out the code for a complicated automaton to parse it
  - **On-the-wire serialization**: the serialization & deserialization code for binary RPC protocols is tedious and prone to error: commonly, specs written in interface description languages (IDLs) are used to generate the actual code (Avro, Protocol Buffers, Thrift, etc)
  - **Foreign-function interfaces**: interfacing higher-level languages (such as Python and Ruby) to large low-level native APIs in C (for example, Qt) is tedious and prone to error: hence SWIG to churn out thousands of lines of glue code

Model-Oriented Programming

A Paradigm?

“MOP works as a layer on top of everything you know today [...] MOP works for every kind of area you write code for. Whether you write games, Linux drivers, servers, apps, plugins, whether you use Java, C, Perl, Ruby, Python, Gnome or KDE... once you start to see the world as models you’ll find yourself writing more code, faster, than you ever thought possible.”

— Pieter Hintjens, Model-Oriented Programming (MOP)

Model-Oriented Programming (MOP)

- You already know model-oriented programming, kind of…
- MOP is writing behavioral specs with RSpec instead of tests with xUnit
- MOP is writing HTML and CSS instead of PostScript
- MOP is writing Makefiles instead of Bash scripts
- MOP is part and parcel with metaprogramming, declarative programming (the *what* instead of the *how*), and domain-specific languages (DSLs)
- No single do-it-all modeling language can cover every possible abstraction or solve every problem; instead, need the right models and abstractions
- Need tech to quickly and easily build arbitrarily modeling languages
- MOP is immune to tech changes: it is abstract from specific programming languages, operating systems, and trends; good models will work for decades

“GSL is a code construction tool. It will generate code in all languages and for all purposes. If this sounds too good to be true, welcome to 1996, when we invented these techniques. Magic is simply technology that is twenty years ahead of its time. In addition to code construction, GSL has been used to generate database schema definitions, user interfaces, reports, system administration tools and much more.”

— Pieter Hintjens, imatix/gsl on GitHub

Case Study: OpenAMQ

- An AMQP middleware server by Pieter Hintjens et al
- The reference implementation for the original AMQP (pre-1.0) protocol
- Designed as high-level models fed into a code-generation process
  - Classes to encapsulating functions, finite state machines for protocol handlers, grammar definitions for parsers and code generators, project definitions for building and packaging sources, a test scripting language, etc
- Used C as the target language for maximum portability and performance
- Generated almost 100% of the middleware server—more than 500 KLOC of C code—from about 60 KLOC of modeling code

“**We can produce extremely high-quality code**. This is an effect of doing code generation: the generated code we produce has no errors, and is as good as a human programmer can write, consistently. [...]

“On many projects where we’ve used MOP, I’m able to deliver hundreds of thousands of lines of code, and say, with confidence: **there is not a single bug in this code**.”

— Pieter Hintjens, Model-Oriented Programming (MOP)

What If…

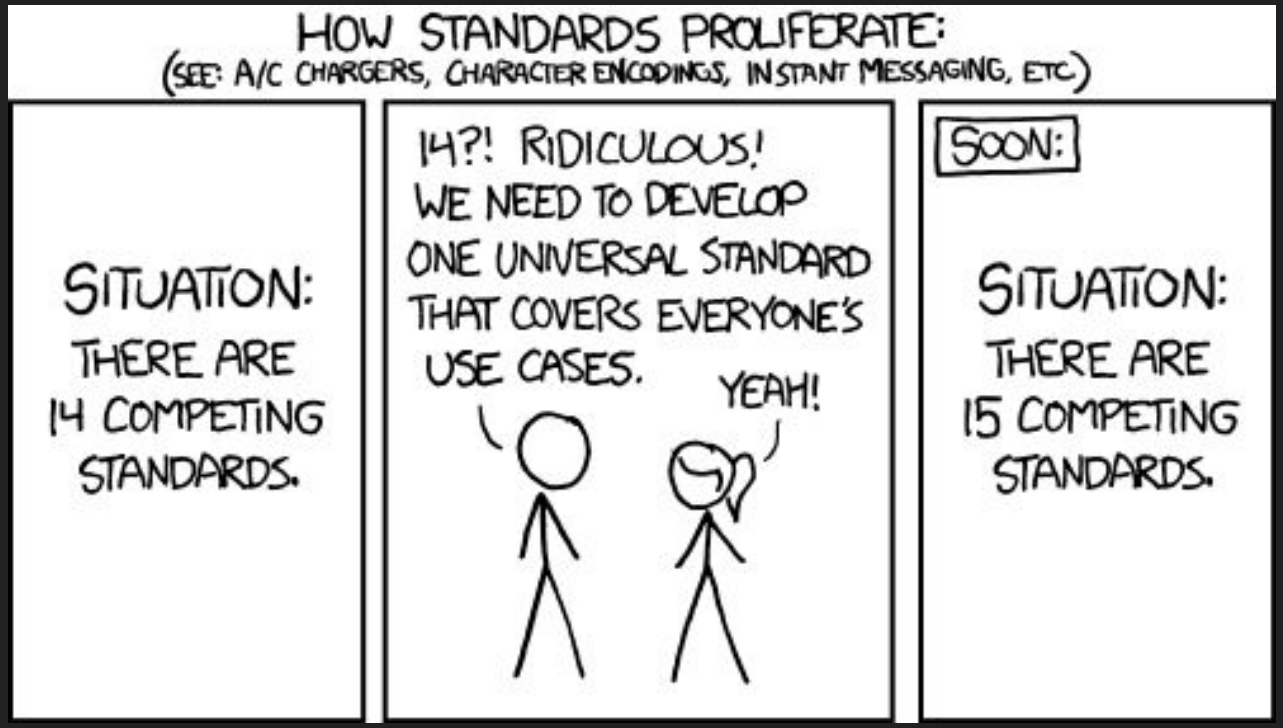
Something Crazy

What If…

- What if there existed a uniform surface layer on top of all these languages…
  - Clearly, there would be some variation across languages in terms of naming conventions
  - However, the overall package/module/class/term taxonomy would be a close match between languages, reducing cognitive load when switching between languages
- What if this **universal standard library** shim simply wrapped those parts of each target language's standard library that are adequate
  - And provided **polyfills** for what the language was missing or didn't adequately implement natively
  - For example, plugged the UTF-8 string situation in Java and JVM languages…
- What if didn't carry with it all the legacy baggage of standard libraries we're used to?
  - A good place to start: **null safety, immutability by default,** and **safe arithmetic**.

What If…

- What if this library accommodated concepts that actually matter in the world?
- Why do so few standard libraries provide models for real stuff that matters?
  - **Contacts**: email addresses, phone numbers, street addresses
  - **Identifiers and locators**: UUIDs, URIs, URNs, URLs, ISBNs, etc.
  - **Locations**: WGS84 latitudes & longitudes, altitudes, angles, cities, countries, etc.
  - **Countries** (ISO 3166 codes) and **languages** (ISO 639 codes)
  - **Quantities**: lengths, durations, masses, the SI units, and the combinations thereof
  - **Tensors**: scalars, vectors, matrices, and beyond
- The notable exception is Wolfram Language, used in Mathematica (demo)



You know what they say about standards…

What Then?

An Opportunity?

“**I call it my billion-dollar mistake**. It was the invention of the null reference in 1965. [...] I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

— Tony Hoare, QCon London (2009)

“**Greenspun’s Tenth Rule of Programming**: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.”

— Philip Greenspun, 1993

The Numerical Tower

Ad-Hoc Numerical Absurdity

“If PHP encounters a number beyond the bounds of the integer type, it will be interpreted as a float instead. Also, an operation which results in a number beyond the bounds of the integer type will return a float instead.”
http://php.net/manual/en/language.types.integer.php

"If you compare a number with a string or the comparison involves numerical strings, then each string is converted to a number and the comparison performed numerically."
http://php.net/manual/en/language.operators.comparison.php

```php
<?php
var_dump("1e3" == "1000"); // bool(true)
```

How to round amounts correct? #Update 1

Ask Question

3

3

I have a little problem with round() in php. I don't know, if I really make it correct. (it is an order system)

$amount can be decimal 14,8 $price can be decimal 10,5 (in the database)

I am using the following rounding at this moment The $price is for one $amount

```
function round_amount($amount) {
  return (float) round($amount, 8);
}

function round_price($amount) {
  return (float) round($amount, 5);
}

//if some one have more decimal chars like 10.000000009
round_amount(10.000000009); //will be 10.00000001

//if some one have more decimal chars like 10.000009
round_price(10.000009); //will be 10.00001

//also this is possible
round_price(round_amount(10.000000009)*round_price(10.000009))
```

Is this way correct to use round? Some user are using more than 16 decimals. I am deducting / adding the results in the user database.

But I see, that some user have about 5-10 cents too much!

Is the only way to resolve this, to allow ONLY 8 and 5 decimals? And warn the user, if he tries to use more? But than I will get an problem with the round_cur(round_amount(10.000000009)*round_cur(10.000009))

I hope some one understand what I am meaning, or can tell me, if my way to round is correct.

asked 5 years, 1 month ago

viewed 7,386 times

active 4 years, 2 months ago

Related

2782 How can I prevent SQL injection in PHP?

972 How to round a number to n decimal places in Java

3 Rounding floats with non-exact representation

129 Rounding float in Ruby

2641 How do I check if a string contains a specific word?

How to round amounts correct? #Update 1

Ask Question

3

I have a little problem with round() in php. I don't know, if I really make it correct. (it is an order system)

$amount can be decimal 14,8 $price can be decimal 10,5 (in the database)

I am using the following rounding at this moment The $price is for one $amount

```
function round_amount($amount) {
  return (float) round($amount, 8);
}
```

3

asked 5 years, 1 month ago

viewed 7,386 times

active 4 years, 2 months ago

FEATURED ON META

Should we burninate the [gaming] tag?

But I see, that some user have about 5-10 cents too much!

Is the only way to resolve this, to allow ONLY 8 and 5 decimals? And warn the user, if he tries to use more? But than I will get an problem with the round_cur(round_amount(10.000000009)*round_cur(10.000009))

I hope some one understand what I am meaning, or can tell me, if my way to round is correct.

in Java

3 Rounding floats with non-exact representation

129 Rounding float in Ruby

2641 How do I check if a string contains a specific word?

“One of the biggest causes of crypto losses is **bad code**, and it’s not usually the fault of the coin’s developers. Instead, third parties, including shoddy smart contract developers and shady exchanges, are to blame for losses that have reached **half a billion dollars in the last seven months**.”

— Bad Code Has Lost $500M of Cryptocurrency in Under a Year (Feb 2018)

“The [Ariane 5] launch [in 1996] ended in failure due to [**integer overflow**]. This resulted in the rocket veering off its flight path 37 seconds after launch, beginning to disintegrate under high aerodynamic forces, and finally self-destructing by its automated flight termination system. The failure has become known as one of the most infamous and expensive software bugs in history. The failure resulted in **a loss of more than $370M**.”

— Cluster (spacecraft), Wikipedia

“The Mars Climate Orbiter [was a] space probe launched by NASA [in 1998] to [Mars]. However, [comms] with the spacecraft [were] lost as the spacecraft went into orbital insertion, due to ground-based computer software which produced **output in non-SI units of pound-force seconds (lbf·s) instead of the SI units of newton-seconds (N·s)**. The spacecraft [came] too close to the planet, causing it to pass through the upper atmosphere and **disintegrate**.”

— Mars Climate Orbiter, Wikipedia

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, **not smart enough to debug it**.”

— Brian Kernighan, paraphrased in Kernighan’s Lever (2012)

Silver Bullets?

“I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems. If this is true, building software will always be hard. **There is inherently no silver bullet**.”

— Fred Brooks, No Silver Bullet: Essence and Accident in Software Engineering (1986)

Questions?

Find me at http://ar.to