“the prevalent theme is programming languages, while this kind of work can sometimes lead somewhere, projects mostly fail while promising grandiose general results”

-- anonymous reviewer
JavaScript Programmers Hate You
Program have types
Types increase productivity
Static is better
Correctness is the goal
<table>
<thead>
<tr>
<th>Years of Dynamism</th>
<th>Language</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1991</td>
<td>VB</td>
<td>dyn</td>
</tr>
<tr>
<td>1991</td>
<td>Python</td>
<td>dyn</td>
</tr>
<tr>
<td>1993</td>
<td>Lua</td>
<td>dyn</td>
</tr>
<tr>
<td>1993</td>
<td>R</td>
<td>dyn</td>
</tr>
<tr>
<td>1995</td>
<td>Java</td>
<td>stat+dyn</td>
</tr>
<tr>
<td>1995</td>
<td>JavaScript</td>
<td>dyn</td>
</tr>
<tr>
<td>1995</td>
<td>PHP</td>
<td>dyn</td>
</tr>
<tr>
<td>1995</td>
<td>Ruby</td>
<td>dyn</td>
</tr>
<tr>
<td>2007</td>
<td>Clojure</td>
<td>dyn</td>
</tr>
</tbody>
</table>
ObjC 12%
C 9%
C++ 16%
Lua 63%
Premiepensionsmyndigheten’s Pluto

“This system is behind the Swedish Premium Pension. It automatically invests and manages 220 billion SEK across 5 million accounts.”

Pluto = 320 000 lines of Perl

Lundborg, Lemonnier. PPM or how a system written in Perl can juggle with billions. Freenix 2006
contract('do_sell_current_holdings')
  -> in(&is_person, &is_date)
  -> out(&is_state)
  -> enable;

sub do_sell_current_holdings {
  my ($person, $date)   ...
  if ($operation eq "BUD") {...
    return $state;
  }
}
Objective C

“Obj-C used in 9.3% of world's software, while C++ stands at 9.1%”
-- TIOBE Index
Objective C

dynamic object types on top of weak static types
JavaScript
Programmers
Hate You
because you design
languages no one uses!
A Clash of World Views

**Computer Science**

Fixed programs, transient data

\textit{there will always be another input}

**Data Science**

Fixed data, transient programs

\textit{there will always be another query}
Programming for the masses

ML, Haskell, Scala, C++ are all domain specific languages

The “domain” is programming in the large by experts
Programming for the masses

Programming languages should be gateway drugs to computational thinking

Instead enforce a rigid programming discipline
Dynamic Typing

If static typing has benefits:

• preventing some errors ahead of time
• simplifying generation of efficient code
• providing machine-checked documentation

Why is it a bad idea?
Dynamic Typing

Static typing only catches trivial errors
  most systems can't even catch NPEs, or off-by-one errors
Static typing ossifies code and hinders evolution
  make the type checker *globally* happy before testing a *local* change
Static typing slows down the rate of development
  pessimistic typing, in case of doubt just say *no*
Dynamic Typing

Hypothesis:

No difference in time solving semantic bugs with a dynamically or statically typed language

Steinberg. *What is the impact of static type systems on maintenance tasks?* MSc Thesis U.Duisburg-Essen
Programming for the masses

R learned in one lecture

Most PHP programmers never read a language manual

Design languages for all
JavaScript Programmers Hate You because you ignore the real world!
Programs as Data

Programming language design should be informed by empirical studies of actual use.

Design languages with the same attention to detail Apple pays the iPhone?
This suggests that eval, while primarily preoccupied with arrays, is overrepresented in benchmarks. It seems likely therefore that a JavaScript implementation that favored other object types would be poorly suited to the kinds of allocations found in real-world sites. (Enhancements tend to heavily stress a few types of objects which may not be representative of the full range of object dynamism seen in real-world sites.)

There are several popular benchmark suites used to determine the quality and speed of JavaScript implementations. However, using these benchmarks as metrics assumes that they are representative of a particular library's feature set. The benchmarks associated with Google's Chrome browser include deserialization and another raytracer.

The behavior of most sites at construction time is modulated by polymorphism—coercion of argument types to fit the requirements of some function call. However, the lifetime of objects in benchmarks is not dissimilar to the real-world sites, with most objects fairly long-lived. Polymorphism was rarer in the benchmarks than in real-world sites. Variadicity in the benchmarks was not dissimilar to real-world sites; however, the benchmarks also showed a steep dropoff in living objects in Figure 17. This peculiarity was not seen in any real-world sites.

The dashed line indicates the end of object construction. The extensive use of constructed objects in benchmarks is now more representative, as seen in Figure 17. The behavior of most sites at construction time is modulated by polymorphism—coercion of argument types to fit the requirements of some function call. In general, the presence of most libraries precludes a major change in the profile of a benchmark. However, the benchmarks can provide simplified and well-tested coding patterns for problems in JavaScript.
Firefox Speedup SunSpider vs JSBench

Richards, Gal, Eich, Vitek. **JSBench: Automating the Construction of JavaScript Benchmarks.** OOPSLA’11
The Ostrich posture does not make Reflection go away

Accept that reflection is here to stay and deal with it
Eval Usage

100% of top 100 sites use JavaScript

82% use eval!

Richards, Hammer, Burg, Vitek. The Eval that Men Do. ECOOP 2011
The Shape of Eval

Identified common patterns:

- JSON
- JSONP
- Read
- Assign
- Typeof
- Try
- Call
- Empty

<table>
<thead>
<tr>
<th>Patterns</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callsites</td>
<td>27553</td>
<td>303</td>
<td>92</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
window.width = 10;
window.height = 20;

function getDimension(x){
  d = eval("window." + x);
}

getDimension("width");
getDimension("height");
JavaScript Programmers Hate You because you solve irrelevant problems!
Avoid non-Problems

Decades of research on Alias analysis for Java, Ownership types, and Information flow…

Programmer Productivity First

The metric that matters is time-to-solution

Late answers are wrong answers
Failure Obliviousness

Dynamic languages keep the program running...
... by execution of incomplete programs
... by converting data types automatically
... by swallowing errors

“Best effort”, optimistic, execution
Failure Obliviousness

Getting an error in JavaScript is difficult

```javascript
x = {}; // object
x.b = 42; // field add
y = x["f"]; // undefined
z = y.f; // error
```
Failure Obliviousness

- New JS VM that aborts untrusted JavaScript code on policy violation
- From program’s point of view these are random failures
- Most programs are resilient and keep working

<table>
<thead>
<tr>
<th>Policy</th>
<th>Functional</th>
<th>AdBlock</th>
<th>Partial</th>
<th>Broken</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddOnly</td>
<td>36</td>
<td>8</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>SendAfterRead</td>
<td>42</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Programmer Productivity First

Don’t prove theorems because you know how but because you need to

What are the properties that really improve time to solution
Introduce a novel type construct that mediates between static and dynamic.

- fast
- static
- catch errors
- catch some errors
- still flexible
- ?
- flexible
- dynamic

Wrigstad, Nardelli, Lebresne, Ostlund, Vitek. *Integrating Typed and Untyped Code in a Scripting Language* POPL10
Gradual Typing

```python
def id(x) = x;

x = id(["toobad"])

x[0] + "!"
```
Gradual Typing

def id(x : [Int] ) = x;

... id( [42,24] ) ...
Gradual Typing

```python
def id(x :like[Int]) = x;

x = id(["toobad"])

x[0] + "!"
```
Gradual Typing

• In Thorn, a simple contract:
  • adding type annotations will never slow down the program
  • adding type annotations will not break a running program
  • type system is “always on”

Bloom, Field, Nystrom, Ostlund, Richards, Strnisa, Vitek, Wrigstad.

Programmer Productivity First

Programming languages matter much less than we like to admit

Millions of lines of PHP at Facebook, Sweden’s pluto is in Perl
Variable lookup in R

\[
\begin{align*}
\text{getfun}(H, \Gamma, x) &= \delta & \text{[FORCEF]} \\
C[\overline{\text{x(a)}}] \Gamma * S; H & \rightarrow \overline{\delta} \Gamma * C[\overline{\text{x(a)}}] \Gamma * S; H \\
\text{getfun}(H, \Gamma, x) &= \nu & \text{[GETF]} \\
C[\overline{\nu(a)}] \Gamma * S; H & \rightarrow C[\nu(a)] \Gamma * S; H \\
\end{align*}
\]

\[
\begin{align*}
H(\nu) &= \lambda \overline{\text{f}}.e, \Gamma' & \text{args}(\overline{\text{f}}, \overline{\text{a}}, \Gamma, \Gamma', H) &= F, \Gamma'', H' \\
C[\nu(\overline{a})] \Gamma * S; H & \rightarrow e \Gamma'' * C[\nu(\overline{a})] \Gamma * S; H' \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &= \iota * \Gamma' & \iota(H, x) &= \nu & H(\nu) &= \lambda \overline{\text{f}}.e, \Gamma'' & \text{[GETF1]} \\
\text{getfun}(H, \Gamma, x) &= \nu \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &= \iota * \Gamma' & \iota(H, x) &= \nu & H(\nu) &= e, \Gamma'' & \text{[GETF2]} \\
\text{getfun}(H, \Gamma, x) &= \text{getfun}(H, \Gamma', x) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &= \iota * \Gamma' & \iota(H, x) &= \delta & H(\delta) &= \nu & H(\nu) &= \lambda \overline{\text{f}}.e, \Gamma'' & \text{[GETF3]} \\
\text{getfun}(H, \Gamma, x) &= \delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &= \iota * \Gamma' & \iota(H, x) &= \delta & H(\delta) &= \nu & H(\nu) &= e, \Gamma'' & \text{[GETF4]} \\
\text{getfun}(H, \Gamma, x) &= \text{getfun}(H, \Gamma', x) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &= \iota * \Gamma' & \iota(H, x) &= \delta & H(\delta) &= \nu & H(\nu) &= \lambda \overline{\text{f}}.e, \Gamma'' & \text{[GETF5]} \\
\text{getfun}(H, \Gamma, x) &= \text{getfun}(H, \Gamma', x) \\
\end{align*}
\]
Variable lookup in R

\[ c() \neq d \leftarrow c; d() \]

< 0.05% context sensitive lookups

symbols that affected are \texttt{c} and \texttt{file}

Morandat, Hill, Osvald, Vitek. \textbf{Evaluating the Design of the R Language}. ECOOP’12
I HATE YOU THIS MUCH!
• Programming language techniques are needed in practice

• But we must change our value system to reward and measure benefits to users

• Language research must be informed and motivated by real-world usage

• We must embrace dynamism and push static techniques into the runtime