What makes or breaks a programming language?

Tero Hasu

BLDL and University of Bergen

Bergen, 22 February 2017

"syntax...inconcise" "syntax is great" "yukky syntax" "nice syntax" "real syntax, i.e., C-like" "weird syntax" "syntax is pretty odd" "much more standard syntax" "eccentric syntax" "syntax is verbose and ugly" "less-cluttered syntax" "syntax...it's weird as hell" "unneeded idiosyncratic syntax" "beautiful syntax" "parallel indecipherable syntax" ...

if we understood...

- as programmers, could better evaluate language choices
- as creators, could better design languages for adoption

overview

1. stories of some programming languages
   - apparent reasons for success/survival/failure
2. factors influencing programming language adoption
   - based on theories, data mining, and surveys

§1: stories

language selection

language	projects	status
C	273067	popular
C++	366941	popular
Lua	32046	moderately popular
Haskell	35037	moderately popular
Racket	3452	surviving
BETA	N/A	retired

(number of GitHub projects, February 2017)

programming language popularity

"Popularity follows a power law, which means that most usage is concentrated in a small number of languages, but many unpopular languages will still find a user base."
– Meyerovich & Rabkin (2013)

case: C

creators

- at Bell Labs
- Ken Thompson created B
- Dennis Ritchie extended B to C

C: motivation

- systems language, for implementing Unix
- desire for a high-level language
  - none available for hardware (DEC PDP-7, etc.)
  - had used PL/I and BCPL in the Multics project

C: the language

- procedural language
- typed (statically, weakly)
  - B was untyped
    - "cell" as the sole data type
  - wanted to reduce pointer use overhead
  - wanted direct support for values of different sizes

C: development

- essentials of C created 1969–1973
- focus on portability in 1977–1979
  - real growth only after portability
    - DEC VAX 11/780 (1977), etc.

C: reasons for success

- success of Unix itself
- efficiency
- small and simple to compile

(Ritchie, 1993)

case: C++

creator

- Bjarne Stroustrup, at Bell Labs

motivation

- frustrated after having had to use BCPL for a project
  - for efficiency
  - instead of Simula

goal

- Simula's code organization facilities
- still retaining the efficiency of C

C++: reasons for success

- zero or low overhead abstraction
  $\rightarrow$ performance
- essentially a superset of C
  - easy to migrate from C to C++, gradually

case: Lua

creators

- Roberto Ierusalimschy (computer scientist)
- Luiz Henrique de Figueiredo (mathematician)
- Waldemar Celes (engineer)

- Computer Graphics Technology Group of PUC-Rio in Brazil
  - industrial partner: Petrobras (an oil company)

Lua: motivation

- products for Petrobras used little languages SOL and DEL
- DEL users began to ask for control flow, etc.
- SOL in turn was going to require procedural programming
- $\rightarrow$ replace them by a single, more powerful language

Lua: distinguishing features

- tables (associative arrays)
  - the sole data structuring mechanism
- extensible semantics
  - "metatables" and "metamethods"

Lua: C/C++ embedding

- a library with a C API
- highly portable C (or C++)
  - "compiled for platforms we had never dreamed of supporting"
    - "many games run on non-conventional platforms"

Lua: reasons for success

- goals: simple, small, portable, fast, and easily embedded
  - turned out to be appealing especially to game devs

case: Haskell

motivation

- multiple independent efforts on pure, lazy languages
  - Miranda, Lazy ML, Orwell, Alfl, Id, Clean, Ponder, Daisy, ...
- consolidation of effort on a new, *common* functional language

Haskell: creation

- initiated in 1987
- by researchers, in working groups, on mailing lists
- design by committee until Haskell 98
- Haskell 98 brought stability
  - further developments by community encouraged

Haskell: the language

- lazy and "remorselessly" pure
- novel feature: type classes
  - violated the goal of incorporating only "well-tried ideas"

Haskell: non-mainstream success

- popular CS research language
  - esp. a "laboratory" for type-system extensions
- influential in language design
- used widely in teaching
- also some industry use
  - "executable mathematics" is appealing in some domains

Haskell: success despite committee design

- committee members' "strong shared, if somewhat fuzzy, vision"
- "mathematical elegance" was important
  - helped avoid ad hoc features creeping in
- "Syntax Czar" to help focus on semantics

case: Racket

"Some programming languages become widely popular while others fail to grow beyond their niche or disappear altogether."
– Meyerovich & Rabkin (2013)

Racket: creators

- originates from the PLT group at Rice U
  - still maintained by PLT, now with multiple sites
- lead visionary: Matthias Felleisen
- lead developer: Matthew Flatt

Racket: origins

- motivation: programming environment for teaching programming
- initial version by Flatt in 1995, based on `libscheme`
  - with MrEd GUI subsystem
- later with DrScheme IDE (later DrRacket)
  - with "language levels"

Racket: keeps going

- programming education niche
- language development niche
- ≈Scheme

user community

- academia
- schools
  - e.g., Koodiaapinen (FI)

case: BETA

"Some programming languages become widely popular while others fail to grow beyond their niche or disappear altogether."
– Meyerovich & Rabkin (2013)

BETA: creators

- Kristen Nygaard, Bent Bruun Kristensen, Ole Lehrmann Madsen, Birger Møller-Pedersen, etc.
  - informal collaboration among team members
  - travel by ferry between Århus and Oslo to meetings
- work started in 1976
  - lasted some 25 years

BETA: quite an early plan

- create a continuation for SIMULA
  - i.e., a language for modeling as well as for programming
- explore the "one abstraction mechanism" idea
  - the "pattern"
    - subsumes: classes, types, procedures, functions, etc.

BETA: realized

- the first complete compiler in 1987
- programming environment

BETA: limited success, for a time

- research vehicle, with several papers
- some use in teaching
- Mjølner Informatics Limited
  - sold Mjølner BETA System as a product

environment

language	projects	research institution
C	273067	industrial
C++	366941	industrial
Lua	32046	academic, with industry partner
Haskell	35037	various, mostly academic
Racket	3452	academic
BETA	N/A	academic, non-profit

C Unix kernel written in C in 1973

C++ delivered for real projects, within ½ year of idea

Lua from the start had real users

- who care only about how to use it productively

funding

language	funding enablers
C	business needs
C++	liberty to "do something interesting"
Lua	business needs of industrial partner
Haskell	multiple parties
Racket	research and teaching vehicle
BETA	some reported troubles

Haskell
- lacked funding for further research for Yale Haskell

BETA
- no attempt to implement a compiler for some time
  - mainly due to lack of resources
- "difficulties getting funding in Norway"
- a large Mjølner project helped BETA get realized

§2 adoption

adoption of new innovations

$$\text{Change Function} = F(\frac{\text{Perceived Crisis}}{\text{Perceived Pain of Adoption}}) \qquad (1)$$

Coburn (2006), Meijer (2007), Meyerovich & Rabkin (2012)

§2.1 reducing pain of adoption

bring in innovations into existing language

- Meijer (2007) has brought in FP features into Visual Basic
- C $\rightarrow$ C++ transition is similar

easier-to-learn programming language?

"...a sense of aesthetic as embodied by the Algorithmic Language Scheme: universality through minimalism, adequation through self-improvement, flexibility through rigorous design, and composability through orthogonal features."
– Scheme'17 CFP

learning a language: perceived ease

- underlying simplicity of formal semantics matters little
  - learn PHP "well" in 1–3 months, C in 6–12 months
- size does matter
  - learn C++ "well" in 1–2 years

(Meyerovich & Rabkin, 2013)

"familiar" syntactic style

- ~~BCPL~~

  $\rightsquigarrow$ {B, C, C++, C#, D, ...}

  $\rightsquigarrow$ Go $\rightsquigarrow$ Magnolia $\rightsquigarrow$ Objective-C $\rightsquigarrow$ Rust

use of C-style syntax

language	rank	%	C-style
Java	1	16.676	✓
C	2	8.445	✓
C++	3	5.429	✓
C#	4	4.902	✓
Python	5	4.043	
PHP	6	3.072	✓
JavaScript	7	2.872	✓
Visual Basic	8	2.824	
Object Pascal	9	2.479	
Perl	10	2.171	✓

TIOBE Index, February 2017

- not used by Lua, Haskell, Racket, or BETA

interoperability

language	projects	interoperability
C	273067	close to the machine
C++	366941	supports C calling convention
Lua	32046	designed for C/C++ embedding
Haskell	35037	(FFI)
Racket	3452	(FFI)
BETA	N/A	(FFI)

language variants

language	projects	interoperability
Clojure	29336	targets Java
ClojureScript		targets JavaScript

§2.2 benefits from adoption

factors in picking a language

factor	important %
open source libraries	64
existing use	63
familiarity	61
performance	54
safety/correctness	40
particular language feature	33
simplicity	27

(Meyerovich & Rabkin 2013)

open source libraries

- C libraries "always remained in touch with a real environment" (Ritchie, 1993)
- C++'s "worst mistake": 1.0 should have had a larger library (Stroustrup, 1993)
- Lua "should have provided ... policies for modules and packages earlier" (Ierusalimschy, de Figueiredo, Celes, 2007)
- Racket is a ≈Scheme "with batteries"
  - with great reference documentation

factor	important %
open source libraries	64

performance

- C's types and operations well-grounded in those of real machines
- C++ was taken seriously due to C-level performance
  - other OO languages of the time, not so
- Lua is "one of the fastest scripting languages"
  - response times are important in games
- Haskell: "laziness has its costs"
  - extra bookkeeping to delay evaluation
  - hard to predict space consumption

factor	important %
performance	54

type and memory safety

language	typing	safety
C	static	unsafe
C++	static	unsafe
Lua	dynamic	safe
Haskell	static	safe
Racket	dynamic	safe
BETA	static	safe

- C++ retains unsafe features of C
  - but eliminates the need to use them except where essential

factor	important %
safety/correctness	40

language features

feature	important (%)
object inheritance	72
classes w/interfaces	68
exceptions	66
higher-order functions	47
functional purity	45
generics	36
eval	30
templates	23
macros	18
continuations	17

perceived value of features (Meyerovich & Rabkin 2013)

unproven features

language design vs. feature design

The language designer should not "include untried ideas of his own. His task is consolidation, not innovation." – C. A. R. Hoare (1973)

Haskell type classes

BETA "patterns"

invasive features

"Pure lazy functional languages such as Haskell will remain a niche" for a long time. – Meijer (2007)

§2.3 application domains

special-purpose → general-purpose

language	rank	domains
JavaScript	7	web browser → general-purpose

Atwood's Law

Any application that can be written in JavaScript, will eventually be written in JavaScript.

Jeff Atwood

an adoption strategy

Design for niches and grow.

Leo Meyerovich (2014)

killer application

language	application
AutoLISP	AutoCAD
C	Unix
Emacs Lisp	Emacs
PHP	CGI
Ruby	Rails
Vala	GNOME

- e.g., 21000+ GitHub repos for Emacs Lisp

language adoption strategy

Code a brilliant application, and integrate a half-decent language.

typical application domains

language	rank	domains
COBOL	24	business systems
Dart	25	web applications
Fortran	27	numeric and scientific computing
Lua	28	application scripting & configuration, games
Rust	40	systems programming
Erlang	41	scalable and fault-tolerant systems
Elm	>50	client-side web
Julia	>50	technical computing
Tcl	>50	GUI programming
Racket	>100	language development, education

programming education domain

language	rank
Scratch	20
Logo	36
Haskell	38
Alice	47
NXT-G	75
Racket	$>$100

TIOBE Index, February 2017

competition within domains

- familiarity is a limited resource:
  - devs maintain a working set of ≈4 languages known "well" (Meyerovich & Rabkin, 2013)
- e.g., Perl, Python, *or* Ruby for build scripts

synopsis

- histories of some programming languages
  - C, C++, Lua, Haskell, Racket, BETA
- programming language adoption
  - pain, gain, domains

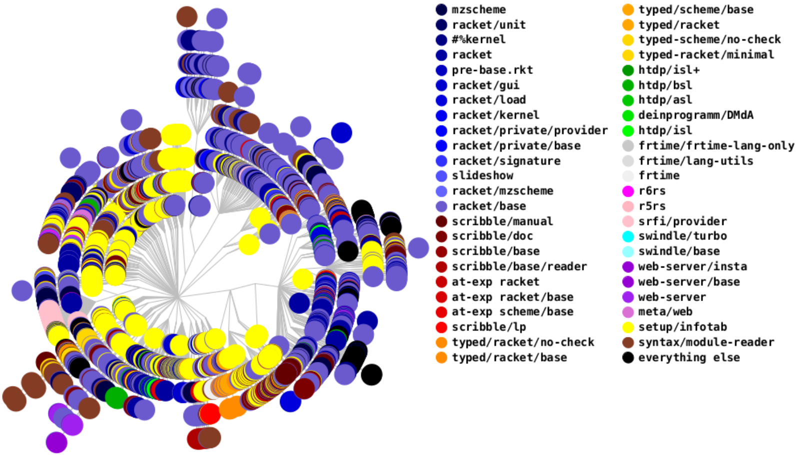
references

- HOPL II & III papers
- Meyerovich and Rabkin: Empirical Analysis of Programming Language Adoption (2013)

contact

tero@ii.uib.no

language implementation domain



`https://github.com/samth/lang-slide`

recent language attempts

language	GitHub ★	idea
Magnolisp	7	inadequate for real world; targets C++
Heresy	27	BASIC-inspired, functional
RacketScript	51	Racket subset; targets JavaScript
Hackett	214	Haskell-like, with macros

static types and unit testing

	see the value %	enjoy using %
static types	36	18
unit tests	62	33

(Meyerovich & Rabkin 2013)