The Programming Experience: Beyond Language

Gilad Bracha

Google
What are Programming Languages for?
What are Programming Languages for?

For building programs
How do you Build a Program?
How do you Build a Program?

With a Programming Language?
How do you Build a Program?

No, with an Editor
How do you Build a Program?

No, with an Editor & a Programming Language
How do you Build a Program?

No, with an Editor & a Programming Language & a Compiler/Interpreter
How do you Build a Program?

No, with an Editor & a Programming Language & a Compiler/Interpreter & a Debugger, a Profiler …
How do you Build a Program?

With Tools
Tools are a Legitimate Research Area

- Tools are no more “just engineering” than languages and compilers
  - Tools enhance productivity
  - Tools interact with language design
- There are real issues of performance and design
A Literate Tradition

“Il n'y a rien en dehors du text”

Foucault, referencing Derida
Is There Nothing Outside the Text?

Actual programming is more than just text.

Languages do not exist in a vacuum. Languages are part of a broader **experience**, dependent on quality of implementation, libraries, tools and support.

Once upon a time, we knew this.
An Empirical Tradition

(loop (print (eval (read)))))
BASIC Interactivity
Beyond the REPL

REPLs still focus on text, but they are a first step moving from text to experience
Smalltalk: The Alto
Lisp Machines
Integrating the Traditions

Experience requires more than mathematics; more than electrical engineering. It requires psychology, and even art.
Neither is science
Neither is pure art
Neither is pure engineering
Each combines art & engineering.
Novelty & Utility

- New knowledge is the name of the game in science.
- In engineering, utility is more important than novelty.
- In art, novelty can be the incarnation of new knowledge, or mere fashion.
- In PLs, as in architecture, both matter.
What are Programs for?
What are Programs for?

For modeling real or imaginary worlds
What are Programs for?

For modeling real or imaginary worlds
Beyond Language: Snapshots and the Sea of Objects

- What matters is the live process, not just the text
- One must be able to save the process and restore it
  - Quickly and reliably
- The saved process should be portable and shareable
Explore & Modify the Heap

- Things you can’t do without the right primitives
- become:
  - allInstances
Language Still Matters

SequenceableCollection subclass: #OrderedCollection
    instanceVariableNames: 'array firstIndex lastIndex'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Collections-Sequencable'

I represent a collection of objects ordered by the collector.
Language Still Matters

- The sea of objects is landlocked
  - It is really hard to move an object from one sea to another
- Objects are hard to serialize - we need values
  - If we want to share programs, language text is valuable
Language & Snapshots can Cohabit

- Smalltalk can be separated from the image model
- We did it in Strongtalk 20 years ago
- Even more so in Newspeak
- Images are good in as much as they provide a great experience

Language Still Matters

- The sea of objects is landlocked
- There is no such thing as an “interpreted language”.
Language Still Matters

- Phrases like “interpreted language” make no sense. Ditto a “compiled language”.

Language Still Matters

- Phrases like “interpreted language” make no sense. Ditto a “compiled language”. Or even “dynamic language” or “static language”. 
A Static Language
Experience Dominates

- Interpreted language - a language whose main implementations are interpreted
- Compiled language - a language whose main implementations are compiled
- Dynamic language - a language whose program texts may be dynamic
- Static language - a language that isn’t dynamic
Language Still Matters

- The sea of objects is landlocked
- There is no such thing as an “interpreted language”.
- Types can be useful
There is Work to be Done
Ampleforth : Live Literate Programming

- Issues: namespaces and versions
- Better text processing tools
- See Eve
Code and Data

- Editors, class browsers divorce code from data
- REPLs, workspaces, object inspectors, debuggers show data & code but code is de-emphasized.
- Idea: bring code & data together at all times
Debug Mode is the Only Mode

Live 2013 demo
http://gbracha.blogspot.com/2013/04/making-methods-live.html
Where to get Data?

- Tests?
Where to get Data?

- Tests
- Type annotations?
Where to get Data?

- Tests
- Type annotations
- Type inference?
- Machine Learning?
Where to get Data?

- Tests
- Type annotations
- Type inference
- Machine Learning
- All of the above; they can re-enforce each other
Time Traveling Debugging

What do we want?
Time travel
When do we want it?
It's irrelevant!
A time traveling debugger is worth a thousand language features

Query the history of the computation for events of interest and go back to the time of the event
Is FP going live?

- Liveness may be the killer feature for functional programming
- Reactive UIs
- Time traveling debugging becomes a lot easier
- See Elm, Clojure
Conclusions

- Programming Languages are more like Architecture than Civil Engineering
- Programmers are experimenters, empiricists engaged in modeling
- They need tools beyond text
- We must attend to their overall experience
This file is licensed under the Creative Commons Attribution ShareAlike 3.0 License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license identical to this one. Official license.