Design Principles Behind Smalltalk

Daniel H. H. Ingalls, 1980

Jeff Carpenter - @jcarp

- I’m Jeff Carpenter (<- not a {Smalltalk,} expert)
- I work on payments at Braintree using object-oriented languages
- Thank you for all your work, Papers We Love

- Two reasons why I love this paper:
  - It’s about something that’s not super hot right now – OOP – but contains lessons we can apply to any kind of programming
  - It represents the origins of paradigms we are deeply familiar with today
- 17 design principles, I will cover 11

- 1972, Alan Kay, Ted Kaeher, Dan Ingalls talking in PARC hallway
- Ted & Dan: language must be large to have “great power”
- Alan Kay: no you can define the “most powerful language in the world” in “a page of code”
- To prove it, Alan Kay came in at 4am and worked till 8am for 2 weeks to develop the language
- That language was Smalltalk

Matt Savona, *Smalltalk’s Influence on Modern Programming*, 2008

Smalltalk-80

- Fast forward to 1980
- PARC released Smalltalk-80 to the public
- This paper was included along with a number of other related articles in the initial public release of the Smalltalk language in 1980


BYTE®
the small systems
AUGUST 1981 Vol. 6, No. 8
$2.50 in USA/$2.95 in Canada
A McGraw-Hill Publication
SMALLTALK

Is the Smalltalk-80 System for Children?

Adele Goldberg and Joan Ross
Learning Research Group
Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto CA 94304

For many years our work on the Smalltalk project has carried with it the purpose of creating new technologies that can be used effectively for instruction, both to teach programming and to support the implementation of educational activities. While the Smalltalk-80 system is not specifically designed for school-age children, most of the applications that we developed as tests of the earlier Smalltalk systems were.

This article will present a brief history of the development of the Smalltalk-80 system that focuses on the instructional uses of its various predecessors. A significant part of this history is the redesign of the language syntax. Programming in Smalltalk involves creating a language for communicating among objects; this language is created within the syntactic restrictions of the Smalltalk-80 system. Often the programmer adds an additional level of syntax in which the language for communicating among objects is presented in terms of graphic images. An example of an instructional activity, the *Dance Kit*, illustrates the idea of such a language. Its design was motivated by the rich support for generalization and interactive graphics available in the Smalltalk-80 system.

Our original intention in writing this article was to *disabuse* readers of the idea that the Smalltalk-80 system, like LOGO, is a language for children. We concluded, however, that the other articles in this issue and the two books on the system (see references) will easily accomplish that task. It remains, then, for us to comment on the style of use of the system that our instructional work has taught us. Although there are a few places where knowledge of the Smalltalk-80 system is helpful, this article does not, in general, require such knowledge.

Learning to Program in Smalltalk

Initially when we ventured out into the schools to teach programming classes, we used a version of Smalltalk known as Smalltalk-72 (see reference 3). Our purpose in teaching these classes was threefold. First, we wanted to know if the language was teachable. In particular, we wanted to devise an appropriate pedagogical approach that could provide feedback on the design of the user interface as well as a basis for language redesign. And we wanted to begin to find out if software based on the concepts of objects and message-passing offered something special in the way of problem-solving tools for children and adults alike. The outcome of these investigations reinforced the value of the semantics of Smalltalk: that is, from the point of view of supporting computer-based problem solving, we found that the ability to organize information into objects that can be independently explored and linked together to create new kinds of behavior is a powerful computational tool.

Smalltalk-72 took the approach that the syntax was defined by the receiver of the message: the receiver read as much of the message as the receiver's method determined and then passed control to the next remaining token, which was seen as the receiver of the remainder of the message. This design came out of our assumption that the system user should have total flexibility in making the system be, or appear to be, *anything* that the user might choose. However, this meant that the only way that a reader could understand an expression was to execute the methods in his head. Furthermore, if a human could not parse an expression without executing the methods, the system itself would not be able to parse it. Thus Smalltalk-72 was a purely interpretive system, and its performance suffered accordingly.

The syntax design (or lack of it) was an example of taking the "flexibility" position to an extreme. Our experience in teaching Smalltalk-72 convinced us that overly flexible syntax was not only unnecessary, but a problem. In general, communication in classroom interaction breaks down when the students type expressions not easily readable by other students or teachers. By this we mean that if the participants in a classroom cannot read each other's code, then they cannot easily talk about it. Our intention was that the Smalltalk system serve as a communication mediator, but the lack of communication due to the runtime parsing of expressions was hindering this goal.

The Smalltalk-76 system took a stricter approach to

TL;DR: yes

Design Principles Behind Smalltalk

Daniel H H Ingalls
Learning Research Group
Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto CA 94304

The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone. Our work flows from a vision that includes a creative individual and the best computing hardware available

The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone. Our work flows from a vision that includes a creative individual and the best computing hardware available.

course of our work. While the presentation frequently touches on Smalltalk "motherhood," the principles themselves are more general and should prove useful in evaluating other systems and in guiding future work.

Just to get warmed up, I'll start with a principle that is more social than technical and that is largely responsible for the particular bias of the Smalltalk project:

Personal Mastery: *If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual.*

The point here is that the human potential manifests itself in individuals. To realize this potential, we must provide a medium that can be mastered by a single individual. Any barrier that exists between the user and some part of the system will eventually be a barrier to creative expression. Any part of the system that cannot be changed or that is not sufficiently general is a likely source of impediment. If one part of the system works differently from all the rest, that part will require additional effort to control. Such an added burden may detract from the final result and will inhibit future endeavors in that area. We can thus infer a general principle of design:

Good Design: *A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible; and all parts of the system should be held in a uniform framework.*

Language

In designing a language for use with computers, we do not have to look far to find helpful hints.

Personal Mastery:

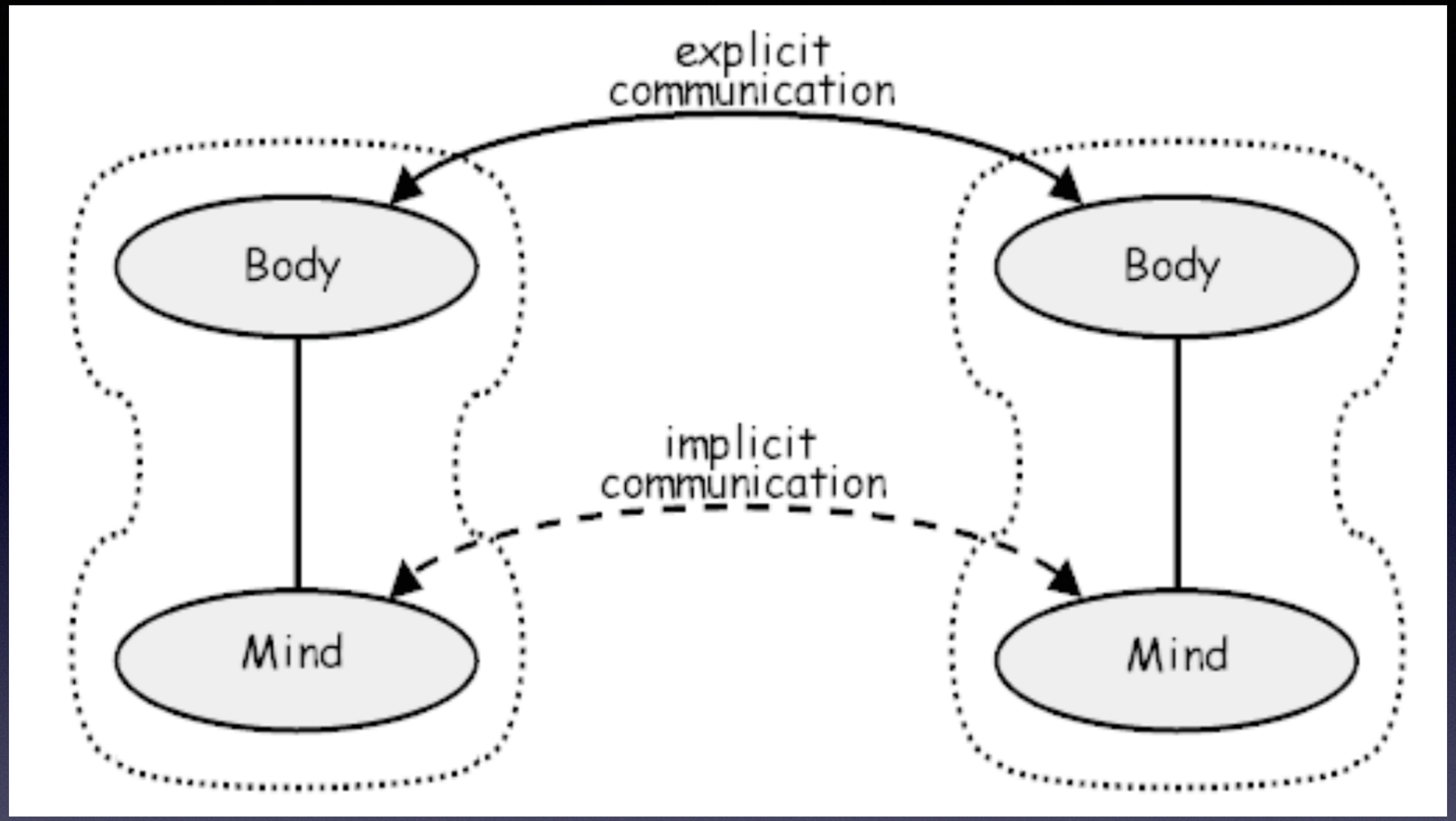
If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual.

Good Design:

A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible; and all parts of the system should be held in a uniform framework.

Purpose of Language:

To provide a framework for communication.

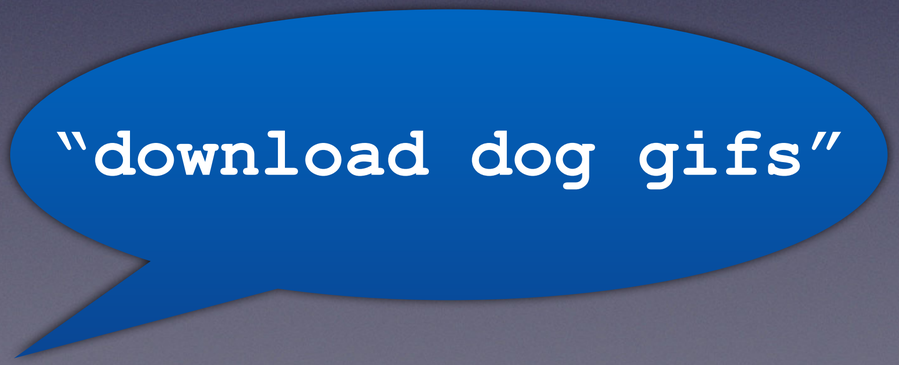


“Since we must work with [the mechanisms of human thought and communication] for the next million years, it will save time if we make our computer models compatible with the mind, rather than the other way around.”

API Design

What we want
to do

What the machine
is good at




```
mov eax, [ebx]
```

Objects:

A computer language should support the concept of “object” and provide a uniform means for referring to the objects in its universe.

Uniform Metaphor:

A language should be designed around a powerful metaphor that can be uniformly applied in all areas.

Uniform Metaphor

- Lisp: built on the model of linked structures
- APL: built on the model of arrays
- Smalltalk: built on the model of communicating objects
- Are there other languages that exhibit this design principle?

Modularity:

No component in a complex system should depend on the internal details of any other component.

Polymorphism:

A program should specify only the behavior of objects, not their representation.

Factoring:

Each independent component in a system would appear in only one place.

Leverage:

When a system is well factored, great leverage is available to users and implementers alike.


IN SMALLTALK THE USER CAN UNDERSTAND AND CHANGE SYSTEM CODE. HERE, LET ME MODIFY ARRAY AT:
SMALL TALK
WOP
MMM...

Smalltalk's incremental compilation and accessibility of kernel code encourages you to make the change while the system is running, a bit like performing an appendectomy on yourself.

Smalltalk-80: Bits of History, Words of Advice

Virtual Machine:

A virtual machine specification establishes a framework for the application of technology.

Natural Selection:

Languages and systems that are of sound design will persist, to be supplanted only by better ones.

Natural Selection:

Languages and systems Ideas that are of sound design will persist, to be supplanted only by better ones.

Even as the clock ticks, better and better computer support for the creative spirit is evolving. Help is on the way.■

- You can find the paper in the PWL GitHub repo:
- https://github.com/papers-we-love/papers-we-love/blob/master/smalltalk/Design-Principles-Behind-Smalltalk.pdf
- Would love to hear your thoughts on the paper
- I’m @jcarp on Twitter or find me in person

Thanks!!!