

Desiderata For The Programming Language Rosebud

Joseph F. Miklojcik III, <jfm3@nectarine-city.com>

2008

Abstract

We present desiderata and inspirational references for “Rosebud”, an impossibly perfect programming language.

1 Introduction

This paper should be read as more of an art manifesto than an engineer’s preliminary design document. Because Rosebud is in one sense a quixotic “impossible dream,” it seems most appropriate to speak of it in ways which suggest it already exists, and this paper does so. There is certainly a lot more to say about everything in this paper. It is purposefully only a survey. References to books and articles are given mostly to display the the author’s ignorance, in the hope that smarter readers will return to him with “oh you might like to read this” type suggestions.

The name “Rosebud” is also permissible as an adjective, either similar in meaning to “brilliant” or “ideal”.

2 Rosebud Might Never Exist

Among other meanings, the name “Rosebud” is meant to suggest unreachable perfection, similar to “Shangri-La,” “the White Whale,” or “the Holy Grail”. Implementations and growths of “Rosebud” will probably always fall short of the ideal. Despite the goal being unreachable, the pursuit of it is still important.

3 Rosebud Is Grown

Guy Steele’s OOPSLA ’98 keynote speech, “Growing a Language”, transcribed in [Steele Jr.(1999)], is a mind-blowing, Rosebud hour where he presents his answer to the meta-question of how to design the design of a programming language. He answers in part by designing the language in which the keynote speech itself is given, as it is being given, in its own terms. Nothing else has had more impact on Rosebud thinking. Rosebud is designed for growth.

Growing a language means blending the usual boundary between the designer of a language and the programmer using the language, and extending the role of the designer into meta-design. Here, “meta-design” means the design of patterns for growth of the design of the programming language. Rosebud has an informally specified meta-design, mostly because formalisms seem either impossible or inappropriate at that level. The meta-design encourages language growth in the directions the meta-designers think are important or good. This document is part of that meta-design, for example.

The process through which meta-design is realized is the most important pattern of the Rosebud meta-design. Imagine a set of features to start with. As the language grows, some of those features fall into disuse, some remain essentially the same, and some grow into strange specialized complexes. When the amount of disused features and specialized complexes passes a certain threshold, the language is regenerated. Regeneration refactors the vital features into more general and regular forms. The strangeness of the specialized complexes are normalized through decomposition and generalization, redistributing the strangeness evenly among some number of new language features. Disused features are made to disappear into these stronger ones. Backward compatibility is provided by formalizing means for code written in prior generations of the language to interact in well defined ways with code written in later generations.

This plan for growth is in direct opposition to Brooks’ “Second System Syndrome” from [Brooks(1975)].

4 Approachable

As the concept of a “computer” dissolves more and more into the human experience, the separation between “programmer” and “user” will continue to make less and less sense. Instead, there will be a continuum of complexity at which programmer-users will change a language substrate to make the computer conform to their needs. Contemporary notions of “user interface” and “programming language” will blur. Rosebud is something that can be approached without already being an expert, yet which doesn’t restrict experts.

When languages grow, they become large. This is okay for a community of people that have grown with the language. But what about newcomers who arrive after the language has been growing for a some time?

Rosebud allows users of a very wide variety of needs and broad levels of literacy graceful ways to express themselves. It does this by being cultivated to grow along rows of complexity from serving as a basic configuration language, requiring a low level of literacy, over to expressing very difficult algorithms that require a great deal of literacy. Programmers work between two neighboring rows at a time, smoothing the transition to more complex rows by never leaving a person in more than half unfamiliar territory. This also has the pleasant side effect of making Rosebud very useful for teaching programming literacy.

Perhaps the most inspirational writing regarding the idea of approachability Alan Kay’s recounting of the design of Smalltalk [Kay(1996)], where he grappled

with the idea of programming literacy and teaching programming to children.

5 Open Implementation

The Meta-Object Protocol (MOP) was initially motivated by the need to provide backward compatibility from the Common Lisp Object System (CLOS) to earlier object oriented extensions to Common Lisp. Over and above backward compatibility, however, the MOP was found to provide CLOS with what its authors call an “Open Implementation.” This had far more advantages than just backward compatibility [Kiczales et al.(1991)Kiczales, Rivieres, and Bobrow], and its principles could be advantageously applied to any object system, not just CLOS [Kiczales and Paepcke(1996)]. In short, open implementation means exposing to the programmer *just enough* of the implementation of the programming language and its run-time environment, allowing them to adjust the language itself to more closely suit their purposes.

Rosebud embraces the idea of open implementation, incorporating it throughout the entire language. Open implementation, approachability, and growth are very tightly related goals. The notion of opening *just enough* seems to also be related to secure implementation (below).

6 Secure Implementation

The run-time environments of some modern programming languages, for example MzScheme [Flatt(2007)] and Java [Yellin(1996)], take on some of the privilege separating responsibility traditionally given to the underlying operating system. This results in more robust software, as well as safer means to incorporate the programs of strangers into one’s own programs. This is called “Secure Implementation” to underscore its orthogonality with the idea of Open Implementation.

A large class of otherwise commonplace run-time errors, such as array bounds violations, are eliminated in a Secure Implementation. Secure implementations allow programmer-users to tightly couple each other’s programs under robust security related agreements.

7 Measured Implementation

The run-time environment provided with any particular implementation of a programming language often provides functionality that can not be used to solve problems that have time constraints. Further, it is often difficult to discern which language features are timely and which are not. It will become increasingly important for programs to manipulate data that flows in real-time, as computers are more used for communication than they are computation. Of lesser interest but similar problems arise in run-time environments where use of particular features can use unpredictable amounts of storage (space). Rosebud

provides programmers with clear, accurate accounting for how much time and space different things take up, and is encouraged to grow in ways which allow programmers maximal control. This is called “Measured Implementation” to underscore its orthogonality with the ideas of Open Implementation and Secure Implementation. Good examples of dynamic programming languages which run with real-time constraints can be found among those used for real-time signal synthesis, such as SuperCollider [McCartney(1996)] or PureData [Puckette(2007)]. Note that those systems do not offload the real-time component of their work to a separate subsystem, as does GNU Radio [Blossom(2004)], for example.

8 Conforms To The Programmer

Like all programming languages, Rosebud is for expressing algorithms. Rosebud embraces the idea that it is not only for expressing algorithms by a programmer to a computer, but also by a programmer to other programmers, including him or her self.

Linguists reason that the capacity for humans to use any language at all is innate – that it is an “organ of the mind” [Pinker(1994)]. A good chair conforms to not only the shape of human backs, but to how human backs actually work. Similarly, Rosebud conforms to the mental organ of language.

Unfortunately, this language organ is far from completely understood. A good summary of what is known so far is given in [Cook and Newson(2007)], although it focuses on Chomsky and is therefore light on semantics and the syntax/semantics/pragmatics interfaces. The choices (“parameters”) a human mind makes (“sets”) as it designs (“acquires”) a natural language are summarized well in [Baker(2001)].

Some general ideas related to the study of Linguistics have been incorporated in the design of the programming language Perl [Wall()]. It is important to underscore the distinction between making a programming language like a natural language, and making a programming language which takes into account what Linguists have discovered about the human mind’s innate capacity for language. Wall does not always make this distinction.

9 Conforms To The Computer

To extend the analogy of the previous section, a good chair will not only have a shape designed with respect to human backs, but will also have a well designed interface with the floor. With programming languages, the analog of the floor is a very dynamic thing, but also relatively well understood.

At the time of this writing, for example, computing hardware is changing at a rudimentary level. Only algorithms modeled for an abstract machine in which more than one computation can take place at the same time will fully utilize the capabilities of this new hardware. There are many difficulties in expressing

such parallel algorithms using existing programming systems.

The inevitable language changes needed to properly conform to new abstract hardware models will not spoil anything, as Rosebud is designed to change over time in the first place. The important thing is that the meta-design encourages good conformity with relatively contemporary computing hardware.

10 The Platform Is Mysterious

It seems desirable for Rosebud to blend with the security model of the operating system, and for Rosebud to be prevented from growing in ways which would ruin this blending. Likewise, it seems desirable that Rosebud blend with the scheduling model of the operating system, especially given that Rosebud is encouraged to grow in ways useful for writing programs with real-time constraints.

At the same time, the operating system, or perhaps what could be better termed the “Platform” on which Rosebud rests, is a relatively fluid and fast moving target, compared to hardware. The most Rosebud programming languages so far have been independent enough from their underlying operating system so that they were fully useful on a variety of such platforms. Therefore Rosebud treats platforms as members in a mysterious taxonomy. A Rosebud program can use a heterarchy of interfaces to interact with the underlying platform, making the same source code useful from embedded systems to timeshared virtual timesharing operating environment network instances (or what have you).

In short, hardware models are allowed to shape rudimentary layers of language growth. Platform models are not.

Food for thought on the subject of platform mysteriousness includes pathnames in Common Lisp [Steele Jr.(1990)] and the page count of Stevens’ and Rago’s book [Stevens and Rago(2005)].

11 Rosebud Is Beautiful

Among other meanings, the name “Rosebud” is also meant to suggest how biological processes can generate breathtaking beauty. Rosebud’s beauty does not arise from a simple crystalline pattern repeating itself over and over into a gem, but rather as a far taller tower of interdependent patterns. A small set of building blocks gives rise to an intricate system of interacting specialized cells, higher order biological systems such as photosynthesis, then colorful velvet flowerings, and ultimately powerful human symbols.

12 Interdependence

All of the desiderata in this document, explicit or not, are interdependent. For example, Open Implementation provides the plasticity required for Rosebud to grow without breaking. The organization of ideas here may be advantageously

refactored and/or reduced in various ways. If one had to condense it as far as possible, the two indispensable ideals would be growth and beauty, but only given a conception of growth and beauty that is robust enough to include the idea that a beautiful growing thing conforms and interacts well with its immediate neighbors (for example, programmer-users, algorithms, and computers) and has the poise to thrive within its ecosystems (for example, large software systems, pervasive personal computing, and operating system variety).

References

- [Baker(2001)] Mark C. Baker. *The Atoms Of Language: The Mind's Hidden Rules Of Grammar*. Basic Books, 2001.
- [Blossom(2004)] Eric Blossom. GNU Radio: Tools for Exploring the RF Spectrum. *Linux Journal*, (Issue 122), June 2004.
- [Brooks(1975)] Fred Brooks. *The Mythical Man Month*. Addison-Wesley, 1975.
- [Cook and Newson(2007)] V. J. Cook and Mark Newson. *Chomsky's Universal Grammar: An Introduction*. Blackwell Publishing, third edition, 2007.
- [Flatt(2007)] Matthew Flatt. *PLT MzScheme: Language Manual*. Matthew Flatt, December 2007. URL <http://download.plt-scheme.org/doc/372/html/mzscheme/>. Chapter 9.
- [Kay(1996)] Alan C. Kay. The Early History of Smalltalk. In *History Of Programming Languages — II*, pages 511–598, New York, NY, USA, 1996. ACM.
- [Kiczales and Paepcke(1996)] Gregor Kiczales and Andreas Paepcke. *Open Implementations and Metaobject Protocols*. Xerox Corporation, 1996. URL <http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-TUT95/for-web.pdf>. Tutorial. Contains work in progress.
- [Kiczales et al.(1991)Kiczales, Rivieres, and Bobrow] Gregor Kiczales, Jim Des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, USA, 1991. ISBN 0262111586.
- [McCartney(1996)] James McCartney. Supercollider: A new real time synthesis language. In *Proceedings of the 1996 International Computer Music Conference*, pages 257–258, 1996.
- [Pinker(1994)] Steven Pinker. *The Language Instinct: How the Mind Creates Language*. HarperCollins, New York, 1994.
- [Puckette(2007)] Miller Puckette. *The Theory and Technique of Electronic Music*. World Scientific Publishing Co. Pte. Ltd., 2007.

- [Steele Jr.(1990)] Guy L. Steele Jr. *Common Lisp the Language*. Digital Press, second edition, 1990.
- [Steele Jr.(1999)] Guy L. Steele Jr. Growing a Language. *Higher-Order and Symbolic Computation*, 12(3):221–236, 1999.
- [Stevens and Rago(2005)] W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the UNIX Environment*. Addison-Wesley, second edition, 2005.
- [Wall()] Larry Wall. Natural Language Principles in Perl. Published web-only <http://www.wall.org/~larry/natural.html>.
- [Yellin(1996)] Frank Yellin. Low Level Security in Java. *java.sun.com*, 1996.