Issue 019: Cross-Platform

Adrian Kosmaczewski

April 6th, 2020

Welcome to the nineteenth issue of *De Programmatica Ipsum*, dedicated to the subject of *Cross-Platform*. In this edition:

- Graham analyzes the economics of cross-platform systems[1], and its impact in usability, availability, and ultimately, the profit margins of vendors.
- Adrian searches old issues of Dr. Dobbs' Journal[2] to contrast the evolution of cross-platform languages, frameworks, and IDEs, in the past 25 years.
- In the Library section[3], Graham explains in detail Adele Goldberg's contributions to Smalltalk-80[4] and to the computer industry, in particular through the books *"Smalltalk-80: the Interactive Programming Environment," "Smalltalk-80: The Language and its Implementation,"* and *"Smalltalk-80: Bits of History, Words of Advice."*

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, or contribute[6] if you would like to support our work.

[1] https://deprogrammaticaipsum.com/cross-platform-is-the-platform/
[2] https://deprogrammaticaipsum.com/dr-dobbs-and-the-deathly-cross-platform-app/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/adele-goldberg/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/

Cover photo by Artur Tumasjan[7] on Unsplash[8].

[7]https://unsplash.com/@arturtumasjan?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[8]https://unsplash.com/s/photos/mind-the-gap?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Cross-Platform Is The Platform

Graham Lee

April 6th, 2020

When we talk about "cross-platform" software, we necessarily invoke the idea that there are software platforms. What are they? Do they really exist?

In a business context, a platform is a business model[1] where a company acts as a broker between suppliers and customers engaging in another transaction, taking a fee for the service. A recruitment consultancy connects suppliers (people who want jobs) to customers (employers), and is therefore a platform business. A marketplace provides a platform for vendors to sell goods to buyers.

In this sense, no computing "platform" was a platform until the arrival of the app store business model. Niche platforms like the third-party Electronic AppWrapper[2] for NeXTSTEP took early advantage of the Internet as a distribution platform for software. In a certain sense,

[1] https://www2.deloitte.com/ch/en/pages/innovation/articles/platform-business-model-explained.html

[2] https://web.archive.org/web/20150724214018/http://blog.tendigi.com/origins-of-the-appstore-the-electronic-appwrapper/

shareware libraries like Aminet[3] or Info-Mac[4] were platforms, and of course pre-empted the "free to use, pay to unlock features" model popular today, but typically these didn't act like intermediaries or brokers in the strict sense of a platform business. A developer (or somebody else) would put their software into the catalogue, which users would know they could search for interesting software, but that transaction would take place away from the platform.

Another sense for the word "platform" in computing is software that exists to make it easier to write other software. Often, this type of platform is run by a company with a multi-sided business model. Take a company like Microsoft. The "platform" is the operating system services, APIs, database and web server, and other components that they sell to developers alongside access to documentation, technical support, and the rest of the MSDN. In revenue terms, this is small fry when compared with the main business: licensing their software to end users.

In some pedantic sense, the two businesses are loosely coupled. You could pay for access to the MSDN, then make the software you write exclusively available on ReactOS[5] to people who haven't bought Microsoft licenses. Or you could buy Windows and Office, but only use third-party software developed for the web or using Cygwin[6]. In practice, of course, these situations are rare, and Microsoft maintains the software "platform" to increase the benfits to customers of their licence-selling business. Buy limited rights to use this software, and also get access to all this other software! Then the proposition to developers is "use our technologies and interfaces, and also get access to all these customers!" It starts to look like a platform again.

The true—and narrow—platform aspect of their business, Microsoft's Windows Store online marketplace, is a second-level epicycle in this hierarchy of ever-shrinking business models. In acknowledging the connective nature of the multi-sided structure described above, vendor app stores represent an attempt to enclose, control, and take a cut from the existing unstructured software distribution that was already taking place.

Thus we find that supposed "cross-platform frameworks" are actually (software, if not business) platforms in their own right. Taking, say, Qt, Delphi, or Electron, the platform is the software that makes it easier to write your own software. One of the benefits of these platforms is that you can reach customers who bought their computers and operating systems from multiple vendors. Or, to put it another way, customers of your software are not limited to a particular choice of hardware or operating system. The direction in which you take this relationship depends on whether your software is more or less valuable than the computer it runs on. Of course, this isn't binary: you could write an app using vendor tools then port those tools to other vendor software, or you could write an app using cross-vendor tools then only sell it to one vendor's customers.

The cross-vendor platforms represent a threat to the multi-sided business model of the giant companies, who realise that when computers and operating systems become commoditised, their margins will shrink. Thus is invented the "look and feel" argument, which says that the important property of, say, a project planning application running on a Mac is whether it is like other Mac software, and not whether it is a good project planner. That the particular poor choices made by this vendor (putting the controls for "close tab" and "quit application" next to each other, for example) are less bad than the particular poor choices made by

[3] https://en.wikipedia.org/wiki/Aminet
[4] https://deprogrammaticaipsum.com_wp_link_placeholder
[5] https://reactos.org/
[6] http://www.cygwin.com/

the other vendor (putting process control and UI commands on the same modifier key, for example).

The best place for a software business along the platform/vendor continuum is both highly subjective and context-dependent, but exploring the ideas behind "computing platforms" provides more options to consider.

Cover photo by Ashkan Forouzani[7] on Unsplash[8].

[7] https://unsplash.com/@ashkfor121
[8] https://unsplash.com/photos/zJsJV5CBGNE/download?force=true

Dr. Dobbs' And The Deathly Cross-Platform App

Adrian Kosmaczewski

April 6th, 2020

As I write these lines, I have three Kubernetes clusters running in the background in my computer (don't ask.) I connect to those clusters using K9s[1], a handy terminal application that provides a nice view of all those deployments and pods running beneath layers of software and other emulation facilities.

K9s is a cross-platform application written in Go[2]. The releases page[3] in GitHub shows a rather classic array (by today's standards, anyway) of supported platforms: there are binaries for macOS, Windows, and Linux. Furthermore, the binaries for these two last operating systems are provided in two CPU architecture flavors: 32 and 64 bit.

Inside those Kubernetes clusters, I run web apps; these are simply the most cross-platform of them all, created solely with the holy trinity of HTML, CSS and JavaScript. I only need a web browser to use them, and my personal choice is Firefox[4], another cross-platform application.

I keep the notes about the configuration of those clusters in a cross-platform note-taking app, called Joplin[5]. This one is an Electron[6] application, also available[7] for a wide variety of

[1] https://k9scli.io/
[2] https://golang.org/
[3] https://github.com/derailed/k9s/releases/tag/v0.19.0
[4] https://www.mozilla.org/en-US/firefox/new/
[5] https://joplinapp.org/
[6] https://www.electronjs.org/
[7] https://github.com/laurent22/joplin/releases/tag/v1.0.197

desktop systems, as well as for the quintessential mobile operating systems, iOS and Android, in this case created with React Native[8]. There is even a terminal application; all of these various clients are written in JavaScript and share a non-negligible amount of code with each other.

The notes in my Joplin application are stored in Dropbox[9], a decidedly cross-platform service that syncs files between pretty much any major operating system you can think of, desktop or mobile. The synchronization engine of Dropbox used to be written in C++[10] until recently, which allowed it to be used in both the desktop and mobile apps, featuring the corresponding user interface with the "native look and feel" expected by the end user.

Go, C++, and JavaScript are just a few of a larger array of cross-platform programming languages. We can also mention FreeBASIC[11], Java[12] (and all languages targeting most JVM implementations, like Kotlin[13], Clojure[14] or Scala[15],) PHP[16], C#[17] (plus all languages targeting .NET[18], like F#[19],) Swift[20], Python[21], Ruby[22], Rust[23], and many, many, many more.

(Go is particularly popular these days for creating cross-platform command-line tools, thanks to its support for cross-compilation off-the-box. This makes apps such as Restic[24] available in a wide array of platforms, with just one single build command.)

Take Free Pascal[25] for example:

> Free Pascal is a 32, 64 and 16 bit professional Pascal compiler. It can target many processor architectures: Intel x86 (including 8086), AMD64/x86-64, PowerPC, PowerPC64, SPARC, ARM, AArch64, MIPS and the JVM. Supported operating systems include Linux, FreeBSD, Haiku, Mac OS X/iOS/iPhoneSimulator/Darwin, DOS (16 and 32 bit), Win32, Win64, WinCE, OS/2, MorphOS, Nintendo GBA, Nintendo DS, Nintendo Wii, Android, AIX and AROS. Additionally, support for the Motorola 68k architecture is available in the development versions.
>
> Free Pascal[26] website.

And arguably, the more important among all programming languages, the most widely available, the most ported, the most hated; plain old C[27].

[8] https://reactnative.dev/
[9] https://www.dropbox.com/
[10] https://oleb.net/blog/2014/05/how-dropbox-uses-cplusplus-cross-platform-development/
[11] https://www.freebasic.net/
[12] https://www.java.com/en/
[13] https://kotlinlang.org/
[14] https://clojure.org/
[15] https://www.scala-lang.org/
[16] https://www.php.net/
[17] http://csharp.net/
[18] https://dotnet.microsoft.com/
[19] https://fsharp.org/
[20] https://swift.org/
[21] https://www.python.org/
[22] https://www.ruby-lang.org/en/
[23] https://www.rust-lang.org/
[24] https://github.com/restic/restic/releases/tag/v0.9.6
[25] https://freepascal.org/
[26] https://freepascal.org/
[27] https://en.wikipedia.org/wiki/C_(programming_language)

> A new CPU architecture or operating system can barely be considered to exist until it has a C compiler.
>
> Drew DeVault[28]

Languages are just a piece of the puzzle; another important one are component frameworks, featuring the usual bricks required to build applications: Elements[29], Xojo[30], Fyne[31], DCOM[32], OpenDoc[33], Adobe AIR[34], wxWidgets[35], Juce[36], Silverlight[37], Qt[38], POCO[39], SQLite[40]... The list goes on and on.

These days we can write, debug, and deploy code in a variety of cross-platform editors and IDEs: come to mind the usual suspects, like Vim[41], GNU Emacs[42], Visual Studio Code[43], the venerable Eclipse[44] and its close frenemy NetBeans[45], and most JetBrains[46]-branded IDEs. These are all cross-platform.

Cross-platform tools allow code to be "ported" from one system to another; that is, to compile and/or run with the least possible amount of change. This represents considerable work; hence the developers of the NetBSD operating system pride themselves (and rightfully so) about supporting 50 hardware platforms[47]... including a toaster.

Have we reached the golden age of cross-platformity? (Is that even a word?) Take a minute to look at the list of applications installed in the computer or mobile device you are using to read these lines; how many are cross-platform? The number might surprise you.

Indeed, back in the 1990s, when the author of these lines started writing code, "cross-platformity" was a hardly achievable holy grail. And most people did not care that much, to be honest; one was a happy Amiga, Mac or PC user, and usually stuck with that.

These days, the fidelity to operating systems is dwingling, and so has risen the importance of cross-platform systems. I happen to use Linux and Mac computers almost daily, and the occasional Windows virtual machine; I am glad to use the same tools in all of them, if possible. But I guess I'm not a typical user.

A search in the archives of old issues of Dr. Dobbs' Journal[48] provides an interesting overview of the complexity of making cross-platform software a quarter of a century ago. For example:

- June 1993: article about cross-platform plugins.

[28] https://drewdevault.com/2019/03/25/Rust-is-not-a-good-C-replacement.html
[29] https://www.elementscompiler.com/elements/
[30] https://www.xojo.com/
[31] https://fyne.io/
[32] https://en.wikipedia.org/wiki/Distributed_Component_Object_Model
[33] https://en.wikipedia.org/wiki/OpenDoc
[34] https://www.adobe.com/products/air.html
[35] https://wxwidgets.org/
[36] https://juce.com/
[37] https://www.microsoft.com/Silverlight/
[38] https://www.qt.io/
[39] https://pocoproject.org/
[40] https://www.sqlite.org/index.html
[41] https://www.vim.org/
[42] https://www.gnu.org/software/emacs/
[43] https://code.visualstudio.com/
[44] https://www.eclipse.org/
[45] https://netbeans.org/
[46] https://www.jetbrains.com/
[47] https://youtu.be/g7tvI6JCXD0?t=673
[48] https://archive.org/details/DDJDVD6

- December 1993: full issue about "interoperability" including an article about cross-platform compression, and an article about the Starview App Framework (for creating cross platform GUIs)
- Special Issue 1994: "The Interoperable Objects Revolution."
- March 1994: full issue dedicated to cross-platform development, featuring among others, an article about cross-Platform Development with Visual C++.
- June 1994: "Cross-Platform Database Development."
- March 1995: Issue about Cross-Platform Development.
- April 1995: article about "Serialization and MFC; Extending MFC for cross-platform portability."
- May 1995: article about "A Cross-Platform Binary Diff."
- December 1996: issue dedicated to cross-platform development and portability.
- April 1997: "QuickTime and Cross-Platform Multimedia"
- September 1997: "The Bridges of Santa Clara County" published in the "Programming Paradigms" section by Michael Swaine, about Apple introducing cross-platform tools in Mac OS X to bring developers back after almost disappearing from the face of Earth.
- February 1999 (issue dedicated to Java): comparing WFC and JFC for Visual J++, "destroying" the cross-platform promise of Java.
- February 2001: Cross-Platform DHTML.
- March 2001: Cross-Platform Coroutines in C++.
- May 2001: article about wxWINDOWS.
- January 2003: article about the CMake Build Manager.
- January 2005: Cross-Platform Builds.
- November 2007: Adobe AIR.
- June 2008: Performance Portable C++.

Clearly, the readers of Dr. Dobbs' Journal cared about cross-platform systems. A lot. They were not the only ones; a quick review of the archives of the C/C++ Users Journal brings similar gems:

- January 1994: "Handling Time-Consuming Windows Tasks."
- April 1995: Designing a Cross-Platform GUI.
- January 1997: section about cross-platform development (also "C/C++ Sources by Victor R. Volkman: Cross-Platform Resources on the Web")
- May 1999: section about cross-platform development.
- March 2000: "Cross-Platform Development Using GCC."
- August 2002: "Adaptable Dialog Boxes for Cross-Platform Programming."

Here is a quote from that January 1994 article:

> The verdict: package the non-GUI-related functions in the most portable way possible, and use the best tool you can find to generate the particular GUI that you are interested in.
>
> "Handling Time-Consuming Windows Tasks," C/C++ Users Journal, January 1994.

Timeless advice.

What are the biggest hurdles with cross-platform systems? The usual suspects appear as a common denominator across all articles surveyed above: Threading models, GUIs, binary data serialization, packaging, and distribution.

However, the most complicated problem in developing a cross-platform application is, without any doubt, the permanent desire of platform vendors to lock us in, in whichever operating system, app, tool, IDE, hardware architecture, or programming language they offer. This is a natural desire, having invested millions or billions of cash units for the development and marketing of said infrastructure piece. Natural, yes, but questionable in the long run.

As a conclusion, a telling story. Microsoft, once the champion of platform lock-in and abhorrence to cross-platform things, is more committed today than ever before to cross-platform tools and frameworks, as odd as it may sound. To the point of absurdity: even if it sells what might arguably be considered one of the hottest laptops in the market at the time of this writing, the Surface Pro X[49], running Windows on an ARM64 chip, neither Visual Studio Code[50] nor .NET Core 3.1[51] are available for that architecture.

Even weirder, .NET Core is indeed available for ARM64 systems, albeit running Linux.

Cover photo by Sarah Ehlers[52] on Unsplash[53].

[49] https://www.microsoft.com/en-us/p/surface-pro-x/8vdnrp2m6hhc?activetab=overview

[50] https://code.visualstudio.com/#alt-downloads

[51] https://dotnet.microsoft.com/download/dotnet-core/3.1

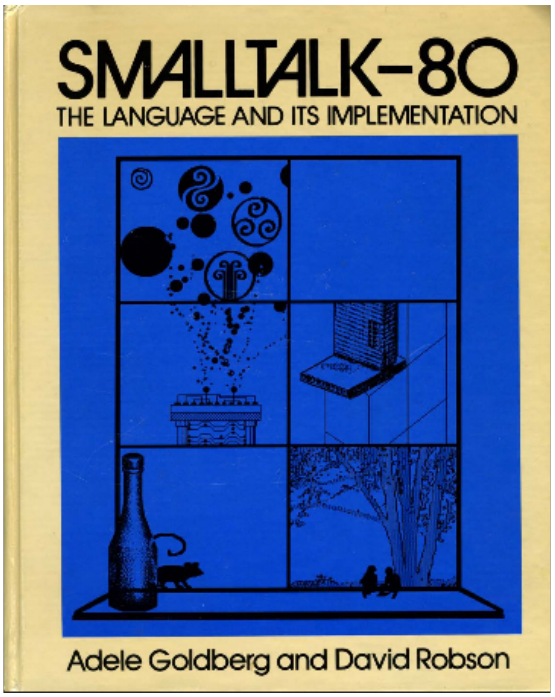
[52] https://unsplash.com/@saz86?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[53] https://unsplash.com/s/photos/cross-platform?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Adele Goldberg

Graham Lee

April 6th, 2020



These days, it's hard to appreciate that Object-Oriented Programming is so easy, it was taught to kids in junior high before it was ever taught to adults. As supposedly senior software engineers debate whether a Car truly "is a" Vehicle, and whether it wouldn't be easier to learn lambda calculus and determine the median monad blog post than to reflect the real objects in the real-world problem they're solving in their software, it seems reasonable to ask: is it really so difficult?

Were we to walk into Xerox's Palo Alto Research Center in 1972 and ask the creators of Smalltalk and Object-Oriented Programming, they would tell us that the problem is one of preconception. As Adele Goldberg recalls[1], they knew that they would have to "bootstrap" a new understanding of computing. People who already knew how to use PL/1 or SNOBOL would struggle to adapt to the Smalltalk way of doing things, but children who learned how to think about objects and the iterative, incremental approach to computing they encourage would take that understanding with them their whole lives.

Adele Goldberg was hired into the PARC team in 1972 after Alan Kay discovered her work at Stanford on teaching computing. In the past she had taught "comparative" programming: getting students to implement features from each of SNOBOL, PL/1, and BAL in the other languages to understand how each supported different concepts in different ways. The PARC team had an arrangement with a local junior high school to teach Smalltalk, and also held classes for the children of Xerox staff on Saturdays.

[1] https://www.youtube.com/watch?v=IGNiH85PLVg

Smalltalk-72 had objects and messages, but was very hard to learn. An object could process messages in multiple ways: it could consume the message, it could respond to it but leave it alone, or the object could respond to the message, modify it, and put it back on the stack. All of this made it hard to understand what behaviour would result from a system of objects collaborating with a given chain of messages.

> 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.

So, over the next few years, with Goldberg's expertise in education enriching the team, Smalltalk-72 became Smalltalk-76: goodbye, "message gobbling", but also goodbye "Myrtle the Turtle", a nod to the Logo environment from Seymour Papert[2]'s group. Hello, abstraction of objects into "classes" of related "instances". Smalltalk-76 would get shown by Goldberg and programmer Dan Ingalls to the executives and product development team at a small local startup Xerox had an interest in, but that needed an injection of fresh ideas: Apple, of course, took those ideas and ran with them.

As Smalltalk development plateaued, Goldberg realised that while their work in education was interesting and valuable to the world of education, it didn't either produce anything that Xerox management could use nor help to bring the original concept – Alan Kay's personal computer for children of all ages[3] – to fruition. They needed a community, and they needed goals for users of Smalltalk to want to try to achieve. She needed users and their trials and retroflections.

In her telling of the history, this is an issue on which she and Alan Kay diverged. He thought that if Smalltalk gained "customers" then they would have to support their needs, and lose a lot of creative flexibility. Adele Goldberg thought that having customers would improve their understanding of the experience of having and using Smalltalk and its concepts, providing useful feedback.

Whatever the internal discussions or conflicts, the community view won, and Smalltalk-80 was developed much more in the open. Collaborators from Xerox (beyond PARC), Hewlett-Packard, Tektronix, and elsewhere shared machines, source code, "bits of history, and words of advice". Smalltalk-80 was introduced to the world in Byte Magazine[4], in which Adele Goldberg contributed the introduction alongside an article co-authored with Joan Ross, "Is the Smalltalk-80 System for Children?" (from which the above quote about Smalltalk-72 was taken).

Eventually, much later than Goldberg anticipated, a series of three books about Smalltalk-80 were published: Adele wrote Smalltalk-80: the Interactive Programming Environment[5], also

[2] http://papert.org/
[3] https://www.mprove.de/visionreality/media/kay72.html
[4] https://archive.org/details/byte-magazine-1981-08/mode/2up
[5] http://sdmeta.gforge.inria.fr/FreeBooks/TheInteractiveProgrammingEnv/TheInteractiveProgrammingEnv

known as “the red book”, which is a guide to using and extending (the same thing, really) the Smalltalk-80 system. She co-authored Smalltalk-80: The Language and its Implementation[6], “the blue book” that dives into the details of how the system works.

The third book in the series, “the green book”, is Smalltalk-80: Bits of History, Words of Advice[7], a collection of essays edited by Glenn Krasner on the experiences of implementing and using Smalltalk-80. The essay provided by Adele Goldberg, the first in the book, describes the review and release process that got Smalltalk-80 from an initial tape containing the 164 kB virtual machine image to the final version incorporating the changes and suggestions of the community. Sadly, a fourth book on application design (“Smalltalk-80: Creating a User Interface and Graphical Applications”) never appeared, an absence which undoubtedly fuelled the explosive growth of the “articles interpreting Model-View-Controller” industry.

Taken together, the red and blue books act like a progressive reveal of the way Smalltalk-80 works. Here’s how you use the UI and enter text; here’s how you run that text as Smalltalk-80; here’s how you integrate that Smalltalk-80 into the rest of the system; here’s how that Smalltalk-80 gets compiled into bytecode; here’s how that bytecode gets interpreted by the virtual machine. But the amazing *tour de force* is that there’s another book in there, too. If you read *backwards* from the end of the blue book, you have a literate program describing the implementation of an object-oriented programming environment: here’s a garbage collector; here’s a language runtime that uses that garbage collector; here’s a compiler that targets that runtime; here’s a programming system leveraging that compiler, and so on.

But where a Knuth literate program[8] is an essay explaining how a particular program *works*, the blue and red books describe how Smalltalk-80 is *designed*. There are multiple listings given for the same routine, refined as new ideas are presented. Here’s how you allocate space for an object. Now that we’ve got automatic reference counting, your allocator also needs to zero the count. Now that we’ve got a mark/free collector, your allocator can compact memory if there isn’t enough space. And we didn’t show you what to do differently for strings and compiled methods.

When it became evident that Xerox management weren’t interested in commercialising PARC’s computing ideas, Goldberg and others formed ParcPlace systems, a spin-out company to capitalise on Smalltalk-80 (thus furthering her goal of finding real users with real problems to solve). Through her work in various roles in the ACM, including president, Goldberg initiated the OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) conference series and further promoted the concepts and applications of OOP.

A good idea doesn’t survive on its own, it needs to be communicated, adopted, and extended. Adele Goldberg’s background in computers in education, her community engagement, and her simple, direct writing style shaped the direction of the entire software industry for decades.

Cover image courtesy of the Squeak wiki[9]. Links to books about Smalltalk-80 provided by Stephane Ducasse at INRIA, on behalf of the authors. We gratefully acknowledge the generosity of these people for sharing their work.

.pdf

[6] http://sdmeta.gforge.inria.fr/FreeBooks/BlueBook/

[7] http://sdmeta.gforge.inria.fr/FreeBooks/BitsOfHistory/BitsOfHistory.pdf

[8] https://deprogrammaticaipsum.com/the-art-of-the-art-of-computer-programming/

[9] http://wiki.squeak.org/squeak