Better is Different

Arthur Carabott

Global Innovation Design

Royal College of Art

2016

10,298 words

Abstract

There must be a better way
We have a better way
At least we think it is better
It looks better
It smells better
How can we get people to use it?
Can we stop them from using their old way?
It would be better for them
Why won't they believe us?
Probably because it's different
But better *is* different

We need a revolution

Contents

List of Figures

Chapter 1

Introduction

The development and improvement of any 'thing' is often a struggle. What is a 'thing'? In the grand scheme of things, it does not matter. So many 'things' share the same difficulties in improving that this seems to be a *general* problem, or set of problems

- 'Thing' includes tools, like the pen
- 'Thing' includes products, like the car
- 'Thing' includes methods, like that of a teacher
- 'Thing' includes ideas, like duality
- 'Thing' includes paradigms, like peer review
- 'Thing' includes forms, like the novel
- 'Thing' includes platforms, like Mac OS X
- 'Thing' includes protocols, like semaphore
- 'Thing' includes languages, like Esperanto
- 'Thing' includes programming languages, like JavaScript
- 'Thing' includes laws, like the criminalisation of drugs
- 'Thing' includes theories, like the earth is flat
- 'Thing' includes conceptual models, like western musical theory
- 'Thing' includes interfaces, like Desktop based computing
- 'Thing' includes behaviours, like social interactions
- 'Thing' includes opinions, like 'organic food is healthier'
- 'Thing' includes standards, like IEEE 802.11 Wireless Networking –'WiFi'.

Making Better 'Things'

When an improvement to a 'thing' comes along, why does it *only sometimes* usurp its predecessor, and other times fail to take off? Are there different types of progress? Can we understand how revolutions (as opposed to evolutions) of 'things' happen? Where are the current leverage points: is it the right time to be focusing on evolutionary or revolutionary improvements? What are the practical lessons to be learnt by taking this abstract view of progress?

A model of progress

Thomas S. Kuhn's *The Structure of Scientific Revolutions* is my lens with which to examine 'things' and their progress (or lack thereof) – Kuhn's model will be outlined in Chapter 2.

From 'Things' to things

> *"Thinking, which is properly nothing and nowhere, can only lay hold of itself in the form of a thing."*
>
> (Steven Connor (2010), p.1)

'Thing' is a good general term and a personal favourite, but to transmit clear ideas and meaning I must take concrete examples of 'things', so that you, the reader, can have a clear reference point, for the ideas being discussed.

The risk with concrete examples of 'things' is that the ideas may be falsely assumed to be limited to the example 'thing', and the generality of the idea being lost. To reduce this risk, for each idea, two concrete 'things' will be given, from two contrasting domains. The hope is that this will provide enough variety to prove the idea flexible enough to be applied beyond the concrete examples. The examples are taken from domains that I am familiar with, in order to be able to best illustrate the concepts. The intention is that these domains are contrasting, yet with enough overlap so that the ideas may be shown to be flexible, while not overly bent to fit the example.

My 'things' are musical interfaces and computer programming languages, tools and environments.

My JavaScript	is your Photoshop
My EQ plugin	is your spell checker
My functional programming	is your empiricism
My MIDI	is your chef's knife

Why should you care?

> *Have you noticed that their stuff is shit, and your shit is stuff?*
>
> (George Carlin – *Stuff*)

You may not care at all about programming or musical interfaces[1], but

[1] on the other hand you probably care about the end results of these things, can you remember a day you didn't hear or at least **hum** music? Or interact with something with a computer inside it?

chances are you have your own *stuff* which has its own drunken path of progress. Some of the ideas will translate to your domain, as indeed they have translated from Kuhn's Science to mine. There are larger patterns at play that are useful regardless of field.

Focus of this dissertation

The focus of this dissertation is on the 'revolutions' not the 'evolution', on the 'new' and not the 'news'. Both are essential: but evolutionary work has a clearer path, it is 'normal science' work within a given paradigm, creating the new paradigm is a far less straightforward task. In Science, revolutionaries dominate history, in computing the inverse is true, where the 'things' most valued are consumer facing, rather than the pioneering ideas or prototypes.

By being more aware that there is more than one type of progress, we can make better decisions about where to focus our efforts: there is no point in trying to start a revolution if one has just occurred, and likewise there is little to be gained by evolving an idea that is about to have the carpet pulled out from underneath it.

Chapter 2

A Model of Progress

In *The Structure of Scientific Revolutions*, Kuhn (2012) uses case studies from the natural sciences to construct a theory of the process through which scientific revolutions occur. The stages of this structure are:

1. Pre-paradigm
2. Paradigm
3. Normal Science
4. Anomaly
5. Crisis
6. Revolution

After the first paradigm of a field is established, the remaining periods are repeated with each revolution leading to a new paradigm.

Two types of work

Kuhn makes an important distinction between types of scientific work: that which causes 'paradigm shifts'[1] and 'normal science', which is "directed to the articulation of those phenomena and theories that the [current] paradigm already supplies" (Kuhn 2012, p.24).

The stages of progress

Pre-paradigm

The pre-paradigm phase for a field can only exist once. Work is often highly speculative and theories are difficult to articulate (Kuhn 2012, p.61), leading to "frequent and deep debates over legitimate methods, problems, and standards of solution" (Kuhn 2012, p.48). The results of this period rarely lead to agreement, but more often the establishment of competing schools (Kuhn 2012, p.48, 162).

[1] forgive this now clichéd phrase, Kuhn coined it.

While many approaches may not be fruitful, a new paradigm may break through, seemingly overnight (Nisbett 2015, p.265).

Paradigm

For a paradigm to establish itself it must "seem better than its competitors", and usually solves "problems that the group of practitioners has come to recognize as acute", yet it "need not [...] explain all the facts" (Kuhn 2012, p.18, 23).

Kuhn asserts that the establishing of a new paradigm 'demands' the destruction of a prior paradigm, (Kuhn 2012, p.96). This is an area for which Kuhn has received criticism (Bird 2013). As will be established in Chapter 5, things differ in software, old paradigms and their tools can last for decades, and in the music industry fanatic reverence is found for vintage equipment.

A well established paradigm brings a wide range of benefits: criteria for choosing problems which can be assumed to have answers; a set of problems that would be almost impossible to have previously imagined; the requirement to investigate with a high degree of detail and depth; and a set of restrictions that are essential to development (Kuhn 2012, p.37, 24, 25); students can quickly acquire theory, methods, and standards, removing the requirement to start from first principles (Kuhn 2012, p.109, 20).

The examples of paradigms that Kuhn (2012) uses are Copernicus's Astronomy, Newton's Mechanics, Lavoisier's Chemistry, and Einstein's Relativity, but Kuhn notes that a revolution "need seem revolutionary only to those whose paradigms are affected by them. To outsiders they may, like the Balkan revolutions of the early twentieth century, seem normal parts of the developmental process." (Kuhn 2012, p.93).

Normal Science

'Normal science' refers to "research firmly based upon one or more past scientific achievements, achievements [...that supply] the foundation for its further practice." (Kuhn 2012, p.10). This is when science simultaneously "moves fastest and penetrates most deeply" as well as having the striking feature of very little "aim to produce major novelties, conceptual or phenomenal" (Kuhn 2012, p.76,35).

Anomaly

An anomaly signals the potential for a revolution, it is a violation of the expectations of the current paradigm (Kuhn 2012, p.52). Upon discovery, an attempt is usually made to bend the rules of normal science to see if they can account for it, often searching at random. Even with a 'severe and prolonged' anomaly, scientists can be reluctant to renounce the prevailing paradigm (Kuhn 2012, p.86-87). Only once the anomaly appears to be more than another puzzle of normal science will it transition to crisis.

Crisis

The arrival of a crisis is the time when testing of the current paradigm occurs, when its framework blurs and its rules loosen. If normal science results in

persistent failure, then the stage is set for novel theories to emerge, usually prolifically (Kuhn 2012, p.84, 144, 75). Crises too will proliferate just before (and during) a revolution, and may end in one of three ways: normal science finds a way to reconcile the problem, the crisis resists even radical approaches, or a viable candidate paradigm emerges, leading to the battle over its acceptance (Kuhn 2012, p.48, 84).

Revolution

Kuhn says that it takes a practitioner who is "little committed to the traditional rules", and either "'very young or very new to the field" to see that the rules may no longer apply. Initially there will be few supporters who may be making decisions based on faith or intuition. How the breakthrough actually occurs to the individual is "inscrutable and may be permanently so." (Kuhn 2012, pp.90,156).

The period of revolution can be tumultuous, paradigms are not easily displaced, proponents of old and new paradigms talk past each other, as their arguments are made *within their respective paradigms*, and for true communication to happen a group must be converted to the new paradigm to truly understand it! (Kuhn 2012, pp.148, 149, 94)

For the paradigm destined to prevail, the community surrounding it will improve and grow it to the point where it is able to make further converts. The criteria for credibility are not always clear: solving the problems of a crisis is 'rarely sufficient', while 'inarticulate aesthetic considerations' of a theory such as being 'neater', 'more suitable', or 'simpler' are often essential. The merits of either side of the arguments remain unclear "since no paradigm ever solves all the problems it defines and [...] no two paradigms leave all the same problems unsolved", the real question is "which problems is it more significant to have solved?"; the most important answer is "the problems that have led the old [paradigm] to a crisis" (Kuhn 2012, pp.90, 109, 147, 151, 152, 154, 157).

Inevitably the conversion is a gradual shift in opinion, at the community level, with hold-outs reassuring themselves that problems still be solved by the previous paradigm. Kuhn rather bluntly asserts that those who resist indefinitely tend to be the "older and more experienced ones", and the only path to unity for the profession being to wait for them to die (Kuhn 2012, p.150)!

The effects of a revolution can be far-reaching. Nisbett (2015, p.267) links the invention of the steam engine to cotton replacing wool as the primary material for clothing and the deregionalising of manufacturing. The revolution can also take a long time to take hold, Alan Kay (2009) notes the significant lag between the invention of the printing press (in 1440) to the truly significant effect it had on the world, through the publishing of Galileo (150 years later), Newton (250 years) and the ratification of the American Constitution (350 years) (Kay 2009, 31:05).

Using the model

There are patterns in Kuhn's model that feel applicable to fields outside science. It cannot be expected to fit in every case, the work itself received a great deal of criticism in its own field (Bird 2013, section 6).

This criticism was twofold: first the claim that in order for science to be revolutionary it must displace an incumbent paradigm[2], and second his 'incommensurability thesis', that is that theories cannot be straightforwardly compared as they do not have a shared set of requirements (Bird 2013). This second criticism is of less concern as it is permissible for competing programming languages and musical interfaces to exist simultaneously: in many cases they will have different utilities, and it is up to the user to decide which best suits their needs. The first area of criticism argues that simultaneous paradigms are able to co-exist[3]: as (Nisbett 2015, p.267) notes, revolutions can instead establish a new body of work that 'couldn't have been produced within the [previous] framework'; this could be understood as the establishing of a first (sub) paradigm.

[2] a counter example is the discovery of the structure of DNA, and the revolution in molecular biology it caused (Bird 2013, section 6.1)

[3] this is not to say they do so without conflict

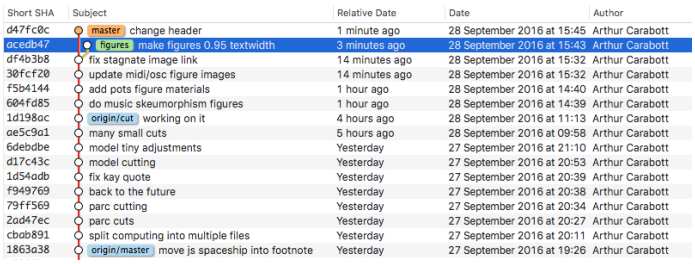
Chapter 3

Computing

Computing Definitions

'Computing' is a broad term within which I focus on programming languages and the tools and environments for programming, as these are points of significant influence on the programmer, and the resulting programs. A programming language is a compromise language that is understandable by both humans and computers; but optimised for neither. Assembly language is as close to 'the metal' [1] as a programmer will bother with these days [2]. At the other end of the spectrum are languages that favour the programmer's experience over machine considerations such as efficiency, performance, memory usage[3].

A programming *tool* is something that is used as part of the process of programming, but is not itself a language, e.g. Git is a 'version control system' used to keep track of change in code (Torvalds 2016), enabling programmers to switch between different versions of a file: a 'stable' version that is ready for production, and an work-in-progress version that has an unfinished new feature (Figure 3.1).



Short SHA	Subject	Relative Date	Date	Author
d47fc0c	master change header	1 minute ago	28 September 2016 at 15:45	Arthur Carabott
acedb47	figures make figures 0.95 textwidth	3 minutes ago	28 September 2016 at 15:43	Arthur Carabott
df4b3b8	fix stagnate image link	14 minutes ago	28 September 2016 at 15:32	Arthur Carabott
30fcf20	update midi/osc figure images	14 minutes ago	28 September 2016 at 15:32	Arthur Carabott
f5b4144	add pots figure materials	1 hour ago	28 September 2016 at 14:40	Arthur Carabott
604fd85	do music skeumorphism figures	1 hour ago	28 September 2016 at 14:39	Arthur Carabott
1d198ac	origin/cut working on it	4 hours ago	28 September 2016 at 11:13	Arthur Carabott
ae5c9a1	many small cuts	5 hours ago	28 September 2016 at 09:58	Arthur Carabott
6debdbe	model tiny adjustments	Yesterday	27 September 2016 at 21:10	Arthur Carabott
d17c43c	model cutting	Yesterday	27 September 2016 at 20:53	Arthur Carabott
1d54adb	fix kay quote	Yesterday	27 September 2016 at 20:39	Arthur Carabott
f949769	back to the future	Yesterday	27 September 2016 at 20:38	Arthur Carabott
79ff569	parc cutting	Yesterday	27 September 2016 at 20:34	Arthur Carabott
2ad47ec	parc cuts	Yesterday	27 September 2016 at 20:27	Arthur Carabott
cbab891	split computing into multiple files	Yesterday	27 September 2016 at 20:11	Arthur Carabott
1863a38	origin/master move js spaceship into footnote	Yesterday	27 September 2016 at 19:26	Arthur Carabott

Figure 3.1: A GUI for the Git version control system, with history for this dissertation.

A programming *environment* is the piece of software within which code is

[1] a term that refers to how closely matched the instructions are to the computer hardware instructions

[2] originally programming was done in pure binary, which *is* the language a computer understands, but makes no compromises for the human

[3] this was one of the design philosophies of the Ruby language (Venners 2003, p.4)

written. This ranges from a simple text editor (Figure 3.2), to an Integrated Development Environment (IDE) to whole operating systems in which everything can be coded. An IDE such as Apple's Xcode (Figure 3.3) (Apple Inc. 2016b) provides tools not only for writing code, but for 'building' [4], testing, debugging (finding errors), making large scale changes, and designing user interfaces.

Smalltalk, and its contemporary offspring (Squeak Project 2016) take the environment further; consumption and creation happen in the same environment, windows for editing code are indistinct from the programs and graphics they are creating, all of which can be modified on the fly, including the cursor! (Figure 3.4).

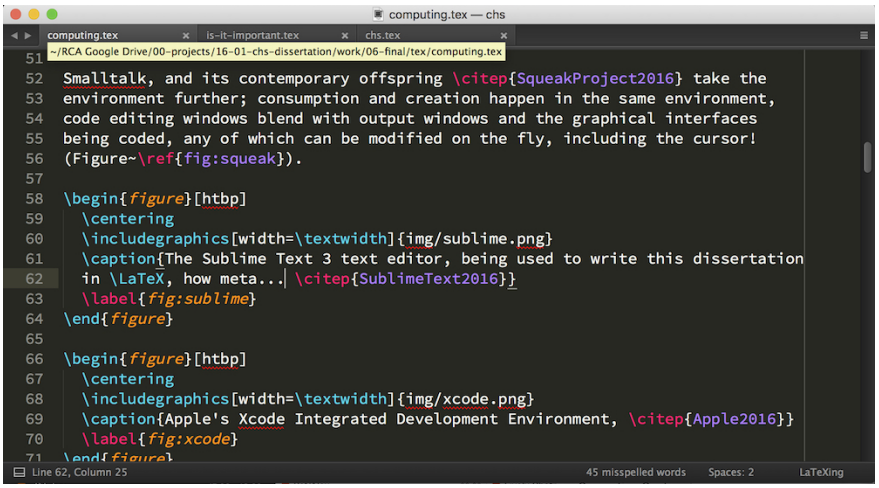


Figure 3.2: The Sublime Text 3 text editor, being used to write this dissertation in LaTeX (Sublime Text 2016).

State of computing today

Programming languages and environments are currently very poor. This may be a surprising statement to a non-programmer (and even many programmers) given that these tools are used to build all of the magical technology that we use everyday, including the most significant invention in recent history: the internet.

While surprising and perhaps controversial, this statement is hardly original. Proclamations to this effect have been made by some of today's computing significant figures: Alan Kay (Kay et al. 2007; Kay 2011), Gerry Sussman (Sussman 2011), Bret Victor (Victor 2013). The steady stream of new programming languages annually (Sureau 2015) underlines the need for better tools.

What *is* surprising is that this is true despite the huge efforts directed at these tools by programmers and the vast sums of money spent and generated by these technologies[5].

Victor (Victor 2013) argues that this is due to a collective forgetting of the pioneering ideas (Ivan Sutherland (1964), Douglas Engelbart (1962), Smalltalk

[4] the process of turning code into an application than can run on the machine

[5] Silicon Valley being the main case in point

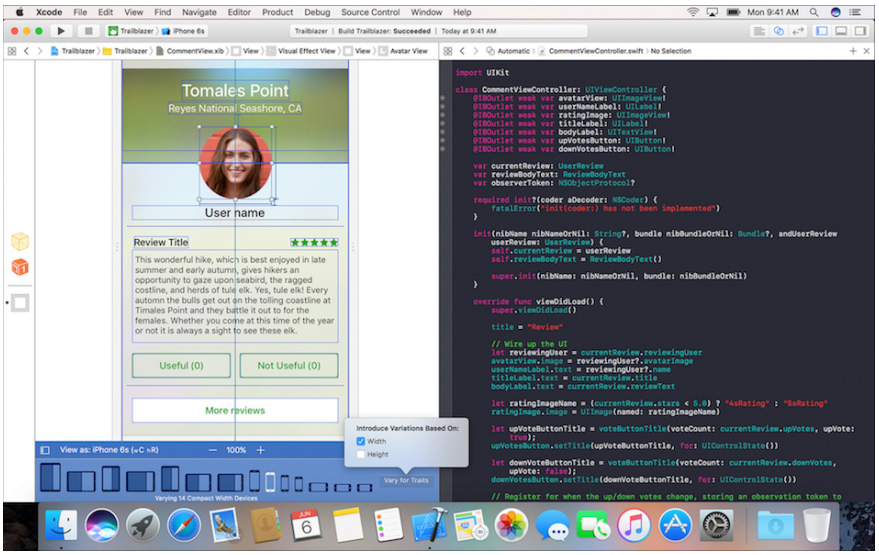


Figure 3.3: Apple's Xcode Integrated Development Environment, (Apple Inc. 2016b).

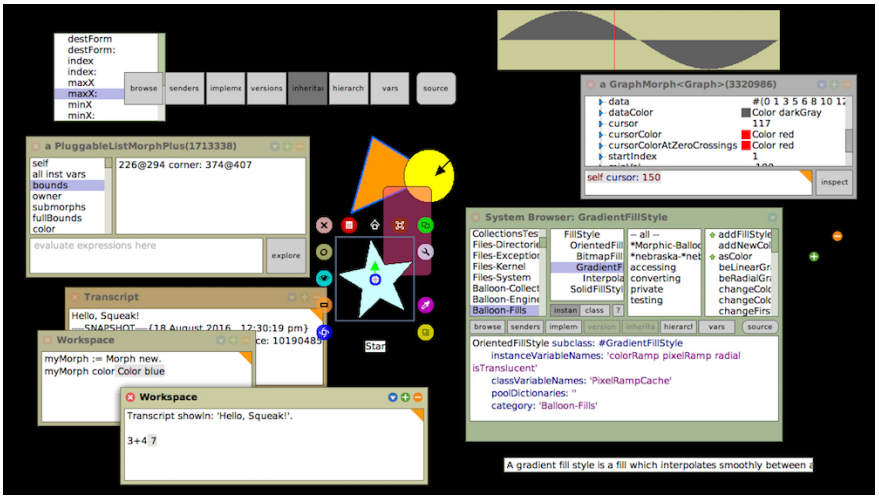


Figure 3.4: The Squeak/Smalltalk environment, with no distinction between the elements *for coding* and those *being coded* (Squeak Project 2016).

(Ingalls 1978) etc.) that emerged in the 60s and 70s, with today's programmers simply unaware that there are alternative models to what they have learned. Victor categorises the 1960s and 70s as being Kuhn's 'pre-paradigm' phase for computing. As paradigms are now been established Victor describes us as currently being in the 'normal science phase, or as Sussman (2011) more bluntly puts it we are "diddling with our details".

While it may seem that it is ideas and not money that is the issue [6], the funding needs toe be focused on visionary long term research as with Xerox PARC[7] (Hiltzik 2009, p.25) and the U.S. Defence Department's Advanced Research Projects Agency (ARPA)[8] under the guidance of J.C.R. Licklider in the 1960s (Hiltzik 2009; Victor 2013, p.45). Funding in industry focuses more on creating products realisable in the near future, even in so called 'moonshot' research labs[9].

Not just the inmates

While these concerns may only seem relevant to programmers, the effects are felt by everyone affected by computers today, which is to say, everyone. This is because systems today *still* favouring the computer and not the programmer, resulting in software that favours the computer and not the user. The effect may be as mundane as unhelpful user interfaces (Figure 3.5), or lack of interoperability that forces manual copying data from one program to another. At the other end of the spectrum grave effects abound: one in five genetics papers have errors in their gene lists because spreadsheets automatically convert gene names to dates (Ziemann et al. 2016); a coding error in 'the most influential economic analysis of recent years' (Krugman 2013) that excluded Australia, Austria, Belgium, Canada and Denmark from an economic model, resulting in a significant (and false) argument for the austerity policies that have ascended in Europe and the United States since 2010 (Herndon, Pollin, and Ash 2013, p.4).

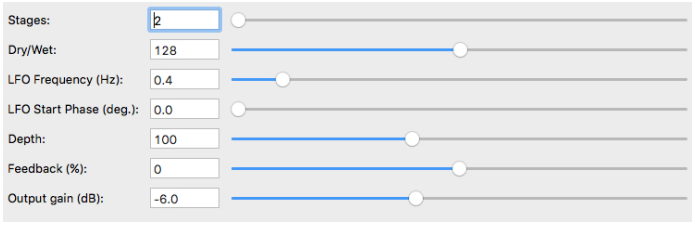


Figure 3.5: A mundane UI from the Audacity audio editor (Audacity Team 2016) with no regard for number ranges, magnitudes or growth rates. Also features a uniform interaction method that does not take into account the resulting effect, and a lack of feedback.

These effects of *computer orientated computing* are the focus of *The Inmates are Running the Asylum* in which Cooper (1999) argues that errors like those

[6] due to the abundance of it in the technology startup world the video game industry, to name just two worlds that are almost entirely dependent on programming

[7] discussed later in Chapter 5

[8] now DARPA - Defense Advanced Research Projects Agency

[9] recently demonstrated by Google putting their 'Boston Dynamics' robotics company –one of the leading robotics companies– up for sale (Clark 2016)

above are mislabelled 'human errors' when in fact the responsibility should lie with the software for treating the users as other computers, and not as humans (Cooper 1999, p.3).

Computing Paradigm Shifts

The development of programming languages with higher levels of abstraction, 'high-level languages', as opposed to those that are 'low-level' or 'close to the metal' closely matches Kuhn's model. A low-level language has a more direct mapping with hardware machine code (the lowest level of which being binary); a high-level language uses layers of abstraction so that the programmer can focus on *what* they want the machine to do, not *how* the machine should do it. The first level of abstraction was Assembly language, which used symbols to *represent* patterns of 1s and 0s, instead of the binary values themselves. The story of this early development, and an example of why the field of programming languages tends to develop slowly is well recounted by Hamming (1997, p.25).

> *"In the beginning we programmed in absolute binary... Finally, a Symbolic Assembly Program was devised [...]. At the time [the assembler] first appeared I would guess about 1% of the older programmers were interested in it – using [assembly] was "sissy stuff", and a real programmer would not stoop to wasting machine capacity to do the assembly.*[10]

This cycle continued, with higher level languages appearing, being dismissed, finding their true believers, and eventually becoming the old guard. When the C language (Ritchie and Kernighan 1978) first appeared, it was regarded as high-level; today it is considered low-level, while higher level 'scripting' languages (also initially dismissed for doing *real* work) may be the only languages used day to day (or ever) by a professional programmer. See Figures 3.6 to 3.8 for examples of the same program written in languages at different levels of abstraction.

The narrative continues today with arguments over whether visual, 'block based' programming is 'real enough' to teach children programming (Guzdial 2016) (Figure 3.9).

While this form of development fits with Kuhn's model, there is another form of paradigm within computing that differs from it; today when people refer to 'programming paradigms' they are referring to the different *families* of languages that live alongside each other simultaneously, and the modes of thinking they encourage. Nisbett 2015, p. 267 discusses how this occurs, with the result of a paradigm shift being the emergence of a new field that can exist alongside its predecessor.

---

[10] *Yes! Programmers wanted no part of it, though when pressed they had to admit their old methods used more machine time in locating and fixing up errors than the [assembler] ever used.*

[...]

FORTRAN was proposed by Backus and friends, and again was opposed by almost all programmers. First, it was said it could not be done. Second, if it could be done, it would be too wasteful of machine time and capacity. Third, even if it did work, no respectable programmer would use it – it was only for sissies!"

```
1  REVERSE  CSECT
2           USING  REVERSE,R13        base register
3           B      72(R15)           skip savearea
4           DC     17F'0'            savearea
5           STM    R14,R12,12(R13)   prolog
6           ST     R13,4(R15)        "
7           ST     R15,8(R13)        "
8           LR     R13,R15           "
9           MVC    TMP(L'C),C        tmp=c
10          LA     R8,C              @c[1]
11          LA     R9,TMP+L'C-1      @tmp[n-1]
12          LA     R6,1              i=1
13          LA     R7,L'C            n=length(c)
14 LOOPI    CR     R6,R7             do i=1 to n
15          BH     ELOOPI            leave i
16          MVC    0(1,R8),0(R9)       substr(c,i,1)=substr(tmp,n-i+1,1)
17          LA     R8,1(R8)            @c=@c+1
18          BCTR   R9,0                @tmp=@tmp-1
19          LA     R6,1(R6)            i=i+1
20          B      LOOPI             next i
21 ELOOPI   XPRNT  C,L'C             print c
22          L      R13,4(0,R13)      epilog
23          LM     R14,R12,12(R13)   "
24          XR     R15,R15           "
25          BR     R14               exit
26 C        DC     CL12'Paradigm'
27 TMP      DS     CL12
28          YREGS
29          END    REVERSE
```

Figure 3.6: Code to reverse and print the string 'Paradigm' in the low-level 360 Assembly language (Rosetta Code 2016).

```c
#include <stdio.h>
int main(){
    const int length = 9;
    char input[length] = "Paradigm";

    // copy the characters starting at the end
    int i = length - 2;
    int j = 0;
    char reversed[length];
    while(i >= 0) {
     reversed[j] = input[i];
     j++;
     i--;
    }

    // add null terminal
    reversed[length - 1] = '\0';

    printf("%s\n", reversed);
    return 0;
}
```

Figure 3.7: Code to reverse and print the string 'Paradigm' in the once high-level, now low-level language C (by author).

```
Transcript show: 'Paradigm' reversed
```

Figure 3.8: Code to reverse and print the string 'Paradigm' in the high-level language Smalltalk (by author).

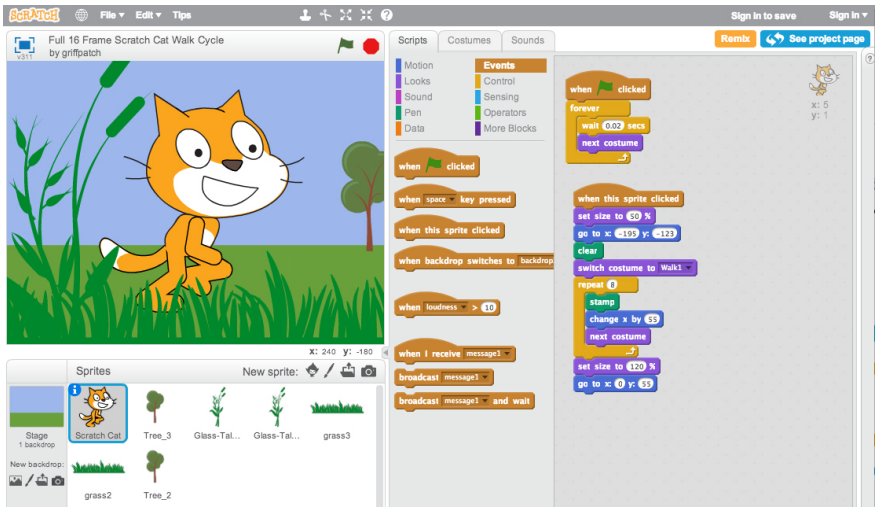


Figure 3.9: The 'block based' Scratch programming environment aimed at young children (Lifelong Kindergarten Group at the MIT Media Lab 2016).

Certain paradigms can happily co-exist as they may be more or less suited to different fields: 'Functional' languages are well suited to mathematics because the computational model is akin to mathematical functions (Figure 3.10), while 'Object Orientated' languages are well suited to video games, as the concept of modelling a world with 'objects' is a fitting mental model (Figure 3.11); players, enemies and items are all objects, which *contain* the data and functionality that they need to exist.

There is another larger, Kuhn-ian narrative of paradigm shifts within computing, that of the **dominant** paradigm (dominant in general, or to a specific field) can be overthrown by a successor. Object Orientated Programming became dominant over the Procedural paradigm since the concept was fully developed by Alan Kay with the Smalltalk language in 1971 (Kay 1993).

Is progress important in computing?

I am using as my definition of progress as 'having a way to do something, that is better than the way you were doing it before', where a 'way' may be embodied (a tool) or not (an idea), and 'better' is a value judgement that can be taken up with Robert Pirsig (1974; 1992).

Rather than a discussion about whether or not progress is *in itself* important, this section is a discussion of the effects of computing artefacts on the person using them.

> *"If computing is important –for daily life, learning, business, national defence, jobs and more– then qualitatively advancing computing is extremely important."*
>
> Ohshima et al.
> *STEPS Toward The Reinvention of Programming* (2012)

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```

Figure 3.10: An algorithm to produce the Fibonacci sequence, written in the functional language Scheme, a dialect of Lisp (Abelson and Sussman 1985, 1.2.2).

```
class MonsterBattle():
  def __init__(self, monster1, monster2):
    self.monster1 = monster1
    self.monster2 = monster2
    self.turn = 0

  def do_turn(self, attacker, target):
    attacker.attack(target)

  def start(self):
    while not self.monster1.fainted and not self.monster2.fainted:
      if self.turn % 2 == 0:
        attacker = monster1
        target = monster2
      else:
        attacker = monster2
        target = monster1

      self.do_turn(attacker, target)
      self.turn += 1
```

Figure 3.11: Code for a monster battle game, written in Python using the Object Orientated paradigm (by author).

Abstraction

As previously discussed, one of the major forms of progress in programming languages has been the 'level of abstraction' from the underlying hardware at which they operate, allowing the user to focus more on the 'what' than the 'how'.

This allows the programmer to create more complex systems, and has similar gains to a good notation in mathematics "*relieving the brain of all unnecessary work, [which] sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.*" (Whitehead 1911, ch.5).

It is not the case however that a higher level language is intrinsically better than a low-level language, there are good and bad languages at any level, as well languages that are more or less suited to particular domains; JavaScript is the only language that runs in the web browser, but its poor design[11].

Programming Experience (PX)

If we consider programming as a medium for exploring new ideas[12] then improving the *experience* of programming is important in helping realise those new ideas which may be critical to solving the problems of the day. This is an area that is generally undernourished, with the programmer being required to mentally work out what the computer is doing internally. Improving this experience by providing better environments is an active area of research (Warth et al. 2016; Victor 2012b) see Figures 3.12 and 3.13 for examples.

Side effects of languages

> *"A powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about processes."*
>
> (Abelson and Sussman 1985, p.6)
>
> *"The limits of the language mean the limits of my world"*
>
> (Wittgenstein – *Tractatus Logico-philosophicus*, p.151)

The more interesting quality of programming languages is the effect they have on the programmer. Opinion and evidence abounds that the programming languages learnt, especially the first, has a lasting effect on the programmer's thinking style. Kay (2009, 43:14) employs a poetic metaphor to describe human memory and the process of becoming fluent in a language: rainfall forms gulleys, which "self- optimize", becoming "more efficient at getting water through them, so they erode faster", within which we become comfortable or even trapped. Dijkstra is characteristically more damning:

[11] infamously rushed in 10 days (Severance 2012, 7-8), see also the Appendices "Bad Parts" and "Awful Parts" of *JavaScript: The Good Parts*, Crockford 2008) means you would not want to use it on a space ship

[12] as opposed to a tool for implementing existing ideas

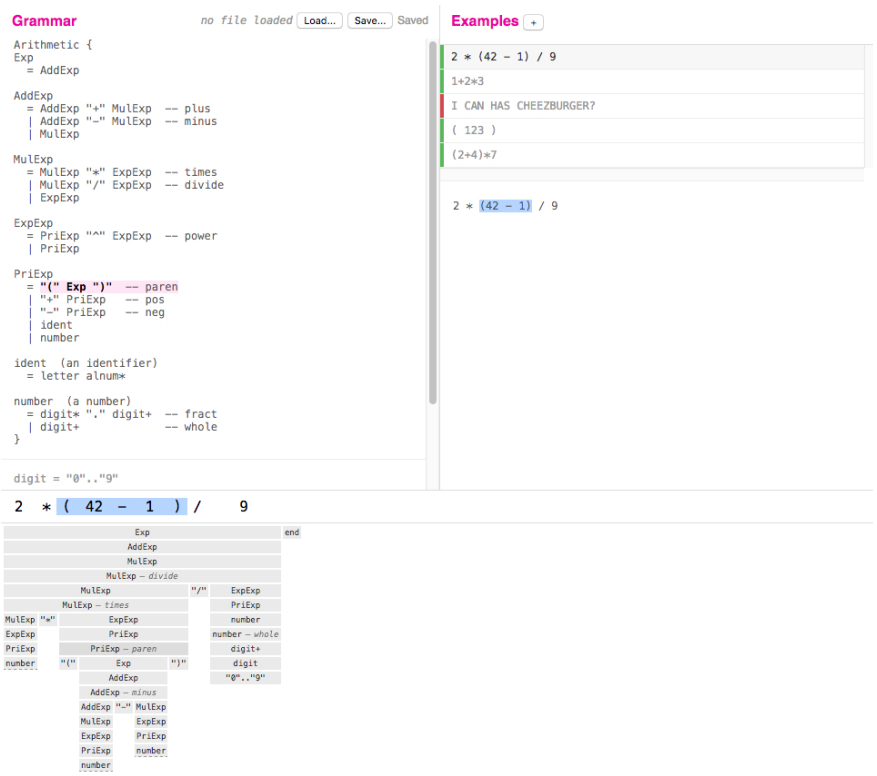


Figure 3.12: The Ohm editor, a tool for designing new programming languages. Note the spatial layout, and visual feedback. On the left is the grammar currently being written, on the right is a list of examples with feedback as to whether they parse correctly or not, and below is how the computer is parsing each component of the current example (Warth et al. 2016).

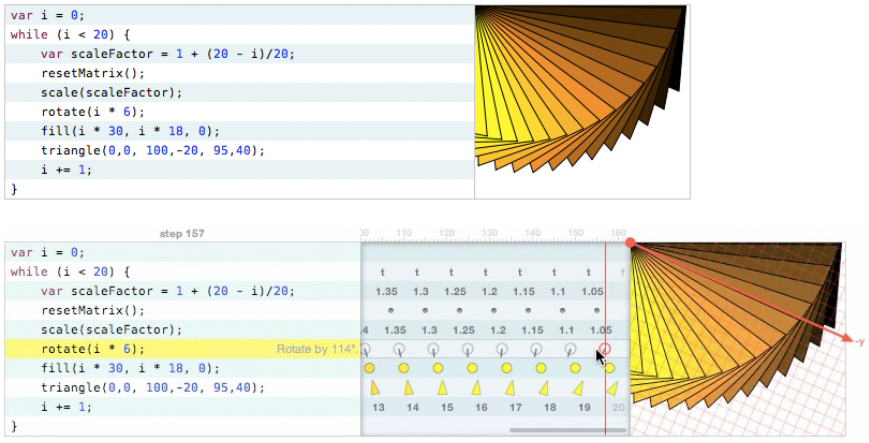


Figure 3.13: An example from Bret Victor's *Learnable Programming* of how graphics programming looks today (top) and how the PX could be improved (bottom), notice how the code is visualised for each line and over time.

> *"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."*
>
> (Dijkstra, pp.129-130)

These anecdotal accounts are backed up by a study that found that procedural paradigm programmers found object-orientated programming (OOP) difficult to learn, while OOP may have been cognitively easier, suggesting that the paradigm previously 'installed' in their brains may be interfering (White and Sivitanides 2005). Another study found that language design affected the number of (syntax *and* logic) errors made by students (Mciver 2000). While these types of studies are often limited (both of these studies were carried out in university classrooms with 25-35 students, over a short period of term; no more than a semester) they help confirm the concern in educational circles (Vujoševic-Janicic et al. 2008; Wexelblat 1980).

Further empirical comparisons of languages have found differences in the level of productivity afforded (Collier and Meyer 2000) and reliability of software (Gannon 1977).

Chapter 4

Musical Interfaces

What is a musical interface?

For the purposes of this dissertation I am defining a musical interface as a *means of creating music*, leaving aside the debate about the definition of music itself. This definition can be considered as a bell curve: at the centre would be artefacts that generate **audible** music, such as the instruments of an orchestra, or Digital Audio Workstations like Logic Pro X (Apple Inc. 2016a) and Bitwig Studio (Bitwig Studio 2016). A little further out we might find musical notation, guitar effects pedals and signal processors like a limiter[1].

Musical instruments are often described as an 'extension of the human body' (Nijs, Lesaffre, and Leman 2009), that may vastly expand the capabilities of the performer. The tools and interfaces used by musicians can be seen as *epistemic*, working "as props for symbolic offloading in our cognitive processes" (Magnusson 2009, p.2).

What is progress for a musical interface?

From my own experience I consider progress to be when a change of interface creates a more direct connection between the 'musical voice' in my head, and the sound produced by an instrument. To draw on the clichéd, but useful analogy of music and spoken language (Wooten 2008, p.41) analogy, the experience is very similar to trying to express an idea in a foreign language. While the idea is formed in the mind, the lack of vocabulary prevents the idea from being conveyed properly. This analogy can be stretched further by comparing languages: how many times have we heard the complaint that there isn't a word for something in [insert language here]? Nabakov summarised this with "No single word in English renders all the shades of *toska*." (Pushkin and Nabokov 1990, p.141) the same inexpressiveness can be true of interfaces.

More objective examples of progress in musical interfaces can be found in the shift from hardware to software; Digital Audio Workstations provide a virtual recording studio full of equipment that would cost (as hardware) tens, if not hundreds of thousands of pounds only a few decades ago. This has afforded

[1] a piece of equipment that prevents an audio signal from being louder than a given threshold

bedroom musicians the means to create music that would have otherwise been beyond their means. It also makes the studio portable, and virtual: musicians are able to work while on the road, and the 'studio' can exist in two locations simultaneously as collaborators send projects back and forth.

This type of tool can be traced back to multi-track recording (which enabled the expansive arrangements of The Beach Boys and the Beatles), and way back to to musical notation which did everything from giving composers the ability to write for larger ensembles to enabling composers like Xenakis to experiment with graphical notation as a compositional device Figure 4.1.

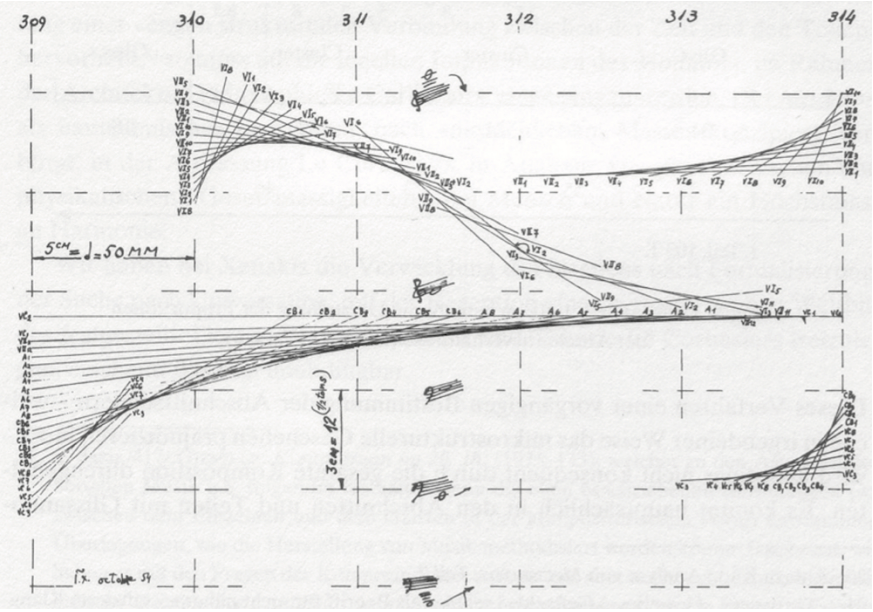


Figure 4.1: Bars 309-314 of Xenakis' graphical score for *Metastasis* (Baltensperger 1996, p125 in Sterken 2007, 40).

Is progress important in musical interfaces?

> *"Musical ideas are prisoners, more than one might believe, of musical devices"*
>
> (Pierre Schaeffer (1977), p.16 in Roads 2001, p.44)

While an interface may act as an extension of the human body, this does not come for free. Knowingly or not, by using a particular interface as part of the music making process, decisions are made about the relative importance of aspects of music: drawing on Clark, Chalmers, and Chalmers ' (1998)'s concept of the 'extended mind', Magnusson (2009, p.171) claims that

> *"...the piano keyboard 'tells us' that microtonality is of little importance; the drum-sequencer that 4/4 rhythms and semiquavers are more natural than other types; and the digital audio workstation, through its affordances of copying, pasting and looping, assures us that it is perfectly normal to repeat the same short performance over and over in the same track."*

This is not an academic concern, the musician Squarepusher has stated repeatedly that this is one of the prime reasons for custom developing music software.

> *"The main point of [developing custom software] was to make available musical possibilities that I didn't see being available through off-the-shelf software or conventional musical hardware. [...] I can determine my creative path, because I'm not being set into a predetermined creative structure that's governed by engineers, not musicians."*
>
> (Mettler 2015)

Some musicians have such a strong relationship with their instruments that they become synonymous with it (e.g. Jimi Hendrix with the electric guitar) and in Indian classical music the greatest exponents of an instrument can be given their instruments as a title (e.g. South Indian classical musician Mandolin Uppalapu Srinivas). The composer's choice of instrument(s) to write for is not arbitrary, but based on the affordances of the instrument, which may be a particular sound, or the musical possibilities afforded by the physical construction: want simultaneous chords and melody on a single instrument? The flute is out, better make it a piano. Want a melody to sound strained or 'very painful'? follow Stravinsky's example in *The Rite of Spring* and put it in the extreme upper range of a wind instrument[2].

Interface Influence

> *"Many of the most important influences on our perceptions and behaviour are hidden from us."*
>
> (Richard Nisbett, *Mindware: Tools for Smart Thinking*, p.15)

With so much power in interfaces, both to extend and to constrain their (potentially unaware) wielder, progress in terms of interfaces is critical. If an interface (for music making, programming, or other) becomes a source of distraction, drawing away attention from the task it is mediating, the effect can be devastating. This phenomenon features heavily in the work of Mihaly Csikszentmihalyi (2009) who discusses how the smallest lapse in concentration can be a disaster for anyone involved in a skilled activity, destroying their state of 'flow' (Csikszentmihalyi 2009, p.212).

[2]There are a number of accounts of Stravinsky's intentions with this solo another being to imitate the *Dudka* folk instrument (Grymes 1998)

"We shape our tools and thereafter they shape us"

(Culkin – "A schoolman's guide to Marshall McLuhan", p.70 1967[a])

[a]This quote is commonly misattributed to McLuhan rather than Culkin, a friend of McLuhan's. Clarification via Kuskis 2013

Chapter 5

Revolution, Education, Stagnation, Domination

This chapter will explore examples of revolutions (successful and failed) in computing and musical interfaces, as well as why sometimes things get stuck in a phase of normal science or stagnate entirely.

Xerox PARC: Dealers of Lightning, Catalyst of Crisis

Xerox PARC[1] opened in July 1970 as Xerox's 'blue sky' research lab, to investigate the future of digital computing, solid- state- physics and material science. Founded by Jack Goldman, who hailed from the J.C.R. Licklider[2] lineage of funding and research management (Hiltzik 2009, p.80, 28, 44).

The pop-history account is that PARC invented personal computing as we know it[3] then failed to capitalise on it, while letting Apple have the crown jewels for their Macintosh (Hiltzik 2009, p.42). While true that few *products* reached the market[4] (Hiltzik 2009, p.15) the impact it had on computing is huge.

PARC's success

So what made PARC successful? Can it *alone* be said to have had a successful revolution? Hiltzik cites four big contributors to PARC's 'explosive creativity': money, people, timing and management (Hiltzik 2009, p.26).

At the time, Xerox had a near-monopoly on the office copier market, giving it a 'seemingly limitless cascade of cash' (Hiltzik 2009, p.26). This allowed

[1] Palo Alto Research Centre

[2] then head of the Command & Control Research project at the US Defense Department's Advanced Research Projects Agency's (ARPA, now DARPA) (Hiltzik 2009, p.42)

[3] Graphical User Interfaces (GUIs), WYSIWYG (What You See Is What You Get) document editing, mixed fonts in a single document, Undo, Copy and Paste, the Desktop, Folder and Window metaphors...

[4] the laser printer being a notable exception that earned Xerox billions of dollars, a many times over return on investment for PARC (Hiltzik 2009, p.27)

researchers to take advantage of Moore's law[5] (Figure 5.1) (Moore 1998) in a practical sense; a computer (the Alto) with memory that cost $10,000 in 1973, but would only be $30 in 1983 (Hiltzik 2009, p.20). The Alto cost $22,000 per machine[6], not economically viable as a product but gave them (in 1973) "a 1989 Mac [...which meant] you could invent all the stuff the 1989 Mac was gonna run. Then 1989 would have to wind up being like what you did because nobody else would have the time to invent [it]" (Kay 2009, 50:04).

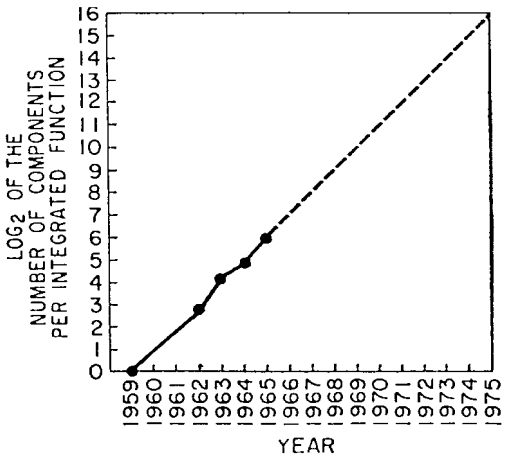


Figure 5.1: Visualisation of Moore's law, originally published 1965 (Moore 1998).

The money would have meant nothing without the right people. The political and economical climate[7] created a buyer's market for research talent, and "–outside of a handful of top universities– [PARC] was the only game in town" (Hiltzik 2009, p.107). The PARC team included three future Turing Award[8] winners[9], co-inventors of Ethernet Bob Metcalfe and David Boggs, Charles Simonyi[10], Larry Tesler[11], Adele Goldberg[12], and Dan Ingalls among others (Hiltzik 2009, p.6-11).

Undoubtedly timing was also important, it was a historic moment in which new computer architectures were becoming viable[13], and semiconductor memory chips were offering huge value in terms of speed and cost (Hiltzik 2009, p.25). The PARC engineers knew that personal computing was not yet economically viable for consumers, but recognised that they *were* at the point where they could start designing for the moment they would become so.

[5] that the number of components (thus the computing power) per inch would continue to double each year

[6] $121,900 in 2016

[7] The Vietnam war and a recession

[8] the 'Nobel Prize of Computing' (Association for Computing Machinery 2016)

[9] Alan Kay, Butler W. Lampson and Charles P. "Chuck" Thacker

[10] who would go on to mastermind Microsoft Office

[11] later Apple's Chief Scientist

[12] who became President of the Association for Computing Machinery (ACM) and the only person who refused to give a demonstration to Steve Jobs, knowing what would happen (Hiltzik 2009, p.766)

[13] a computer per person, instead of a time-shared mainframe

PARC's management style and research philosophy were also critical to its success: find the best researchers, hire them and then leave them (mostly) without directives, instructions or deadlines. (Hiltzik 2009, p.26).

These four factors lead to the invention of personal computing as we know it, Ethernet [14], Object Orientated Programming [15], the laser printer, as well as early hardware for calculating 3D geometry[16] (Hiltzik 2009, p.22, 254, 368).

PARC, Kuhn, Crisis, Apple

From a Kuhnian perspective, PARC's 'crisis' was more one of foresight than necessity: they had a vision of how much better it could be (in 1972 Alan Kay was essentially proposing the *iPad* (Figure 5.2)) and set about trying to realise it. The revolution they achieved was both the establishing of a new field (personal computing) and changing the *dominant* computing paradigm, as non-personal computing would continue, e.g. as infrastructure like web servers.

Another way to view PARC is as *part* of the revolutionary process, rather than the revolution itself. While PARC had created a new paradigm, Xerox were failing to bring it to market, if a revolution happens and no one is around to see it, did it make an impact? The revolution that reached the people came about as this technology was put into products that people could get their hands on, namely Apple's Macintosh. From this perspective, PARC *were the crisis*, certainly for Steve Jobs, who during a demo was "waving his arms around saying 'Why hasn't this company brought this to market? What's going on here? I don't get it!' " (Hiltzik 2009, p.416). PARC were an anomaly, something outside the language, their system wasn't fast enough or cheap enough for consumers, but they changed ideas about what was possible. Bill Atkinson one of the Apple engineers present at the demo said that "seeing the overlapping windows on the Alto screen was for him more a confidence-builder than a solution [...] knowing it could be done empowered me to invent a way it could be done" (Hiltzik 2009, p.417)[17].

Smalltalkin' 'bout a revolution

The GUI was not PARC's only paradigm shift that found success through influence. Smalltalk, the powerful and flexible programming language/environment that powered the Alto (Kay 1993; Ingalls 1978) and was the first full embodiment of the Object Orientated paradigm, never became a dominant language[18]. It *has* been a phenomenal influence on other languages, of the top 20 languages in an index of the most popular languages 14 are either Object Orientated or support it (TIOBE - The Software Quality Company 2016). Languages such as Apple's Objective-C[19] and Ruby[20] draw *heavily* on Smalltalk.

[14] still the standard in networking today

[15] now the predominant paradigm

[16] which enabled rendering 3D images

[17] I call this the '900°' moment: at the 1999 X-Games Tony Hawk performed the first mid-air 900° spin on a skateboard (World Sport 2015), it changed everyone's ideas about what was possible; in 2012, aged 12, Tom Schaar landed a 1080° spin (Red Bull 2012)

[18] Being popular is not the only measure of success, it is only a measure of popularity, which an interesting phenomena in itself

[19] until recently the primary language used to write for Apple devices.

[20] heavily used in web development, Twitter was originally written in it.

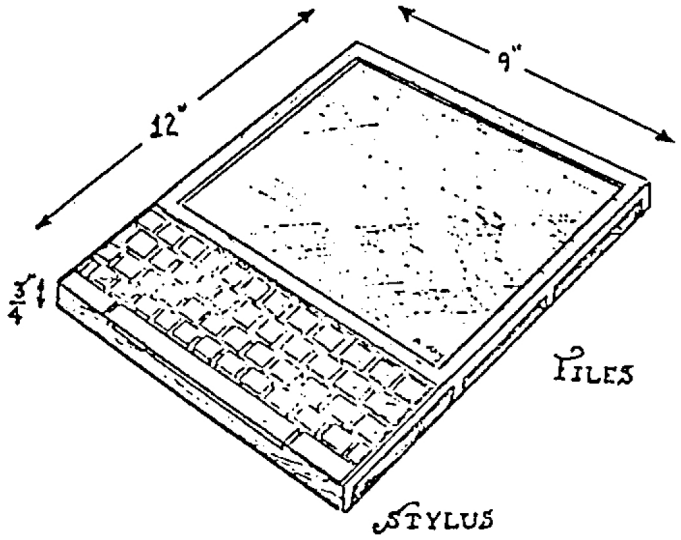


Figure 5.2: Alan Kay's *Dynabook*, (Kay 1972, p.6).

Many of Smalltalk's concepts still haven't made it into the most popular languages of today: Smalltalk combined the notions of a language and an environment, operating in symbiosis: the language was written within the environment, and the environment was programmed in the language. The benefit is that users can not only *use* tools to create (documents, media, simulations) but can *modify* them. Smalltalk also has a notion of *images* where the current state of a system can be saved as a file, which can be transferred to other systems (Kay 1993, p.8).

Instead of saving your thesis *document* you would also save the word processor, your bibliography manager and your PDF reader, which could be reopened in exactly the same state on another computer, regardless of differences in hardware[21].

Smalltalk began life at PARC as part of a Learning Research Group[22] project to design a computer system simple enough to be used by children, but powerful enough for them to learn through it (Hiltzik 2009, p.312). After a number of iterations, Dan Ingalls, the lead implementer focused on transforming it into a 'full-service' programming language: Smalltalk-76, much to Kay's dismay as "no kid ever wrote any code for Smalltalk-76" (Hiltzik 2009, p.437). According to Bret Victor this was Smalltalk's own paradigm 'freeze', the constant flow of new ideas that came with each previous version of Smalltalk ended as the normal science phase began (Victor 2016b).

So Smalltalk's method (whether it was intended or not) to revolution was to become a *great influencer*, and while Kay (Kay 2011, 19:00) describes some of the ways Smalltalk's ideas were adapted to more traditional forms[23] as 'terrible', the method of *influence* is a powerful one.

[21] I recently had my first experience of Smalltalk, see Appendix A for a personal account of a going back to the future.

[22] Alan Kay's group

[23] C++ and Java applied Object Orientated ideas to the C language form

How to shift paradigms and influence people

As an alternative approach to designing new *things* that embody a proposed new paradigm is to design things that express the *ideas* of that new paradigm. The distinction here is that while something that is intended to be used will have limitations on how well it can perform, something that only aims to express ideas is free from the details of implementation. This is one of the strengths of the musical score, it *demands* interpretation, whereas a recording provides an exact performance which encourages copying verbatim.

This approach of 'design for influence' seems like an effective one when proposing new paradigms, as at first the new paradigm is unlikely to be able to solve all of the problems of a field Kuhn, p.147, which would only be highlighted by a concrete implementation. It is the underdog approach in design, which tends to focus on "fiddling with the world out there rather than the ideas and attitudes inside our heads that shape the world out there." (Dunne and Raby 2013, p.2).

This is the approach that Bret Victor[24] has taken in talks and demos. Victor presents concepts as prototypes that function *enough* to be believable, but may involve some smoke and mirrors in the implementation[25]. In the talk *Inventing on Principle* (Victor 2012a, 37:00) Victor likens this work to activism, i.e. Dunne and Raby's 'idea fiddling'. Victor justifies the smoke and mirrors approach because all of the important work, the ideas, are there on the surface and not in the implementation (Victor 2016b). The approach has proven effective, Victor is highly praised, and just two examples of real world products inspired by Victor's work are Apple's Playgrounds (Lattner 2016; Apple Inc 2016) (Figure 5.3) and the code editor Light Table (Kodowa Inc 2016; Granger 2012).

The downside of this approach is that the work is open to misinterpretation, if a prototype looks *enough* like a real product, the audience can have false expectations about what is currently possible, and how ready for production use it might be. This is something Bret has experienced, with people becoming upset when the prototypes they mistook for products were not released (Victor 2016b).

I like to think of this approach as a *person* being a Kuhnian anomaly, by creating new ideas that are outside of the common language (and publicising them) they can induce a crisis in the minds of others.

Peter Norvig[26] described influence as one of the Lisp language's successes: in response to the question 'Why didn't Common Lisp fix the world' Norvig responded by listing several common features that were uncommon before Lisp and saying "the ideas won, but the Common Lisp implementations didn't" (Norvig 2016). Lisp is also the grandparent of JavaScript, the seventh most popular programming language in 2016 (TIOBE - The Software Quality Company 2016) and the only language that runs in the web browser.

Making judgements as to whether an idea has been a 'success' in changing

[24] a researcher and one time 'Human- Interface Inventor' at Apple, whose work focuses on new tools for thinking and understanding, and makes the case for paradigm shifts e.g. projects like *Kill Math* (Victor 2011, 2016a)

[25] Victor's previous work *was* more product focused, having worked on synthesisers for Alesis, and Al Gore's interactive climate change app

[26] Director of Research at Google and author of books on Lisp

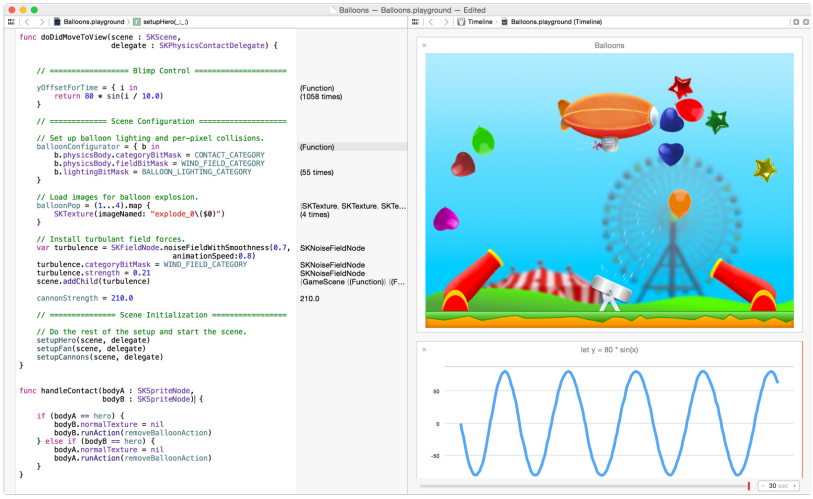


Figure 5.3: Apple's *Swift Playgrounds*, which was "heavily influenced" by Bret Victor's work (Lattner 2016).

minds and paradigms is difficult because it can take a long time for both the fall of the old and the appearance of the new (Kuhn 2012, p.86). Xerox PARC worked on the Alto for more than a decade before the ideas were brought to the public with Apple's Macintosh (Hiltzik 2009, p.16). On the other hand, sometimes an invention instantly defines a paradigm, the Fender Stratocaster has been the archetype electric guitar since 1954 (Minhinnett and Young 2006, p.28) and forms of the Lisp language have been in use since being introduced in 1960 (McCarthy 1960).

Start a cult

A tried and true method for influencing people is to start a cult, usually with a charismatic leader. In reading for this dissertation I found a number of characters who were referred to in cult-like or religious terms, there wasn't time or space to explore this further, but see Appendix B for some quotes.

Education

"Students accept theories on the authority of teacher and text, not because of evidence. What alternatives have they, or what competence?" (Kuhn 2012, p.80). This gives the educator a position of huge influence, able to guide students towards new emerging ideas, or to reinforce the well-trodden road. In computing, education is where students are likely to first become fluent in a language[27], which also makes it a prime leverage point for a language to gain popularity.

Processing (Fry and Reas Casey 2016) and openFrameworks (Community 2016), two 'creative coding' frameworks were both initially developed and used

[27] the effects of which were discussed in Chapter 3

in educational settings[28], where a class curriculum might *require* their use. This instantly built their communities, helping them become two of the most popular tools for artists and designers working with code.

General purpose programming languages have a symbiotic relationship with education, where the most popular languages in university introductory courses overlap heavily with the most popular languages in industry[29] creating a cycle of popularity (and potential for stagnation!).

Max (Figure 5.4) is a 'visual programming language for media' (Cycling '74 2016a), popular with musicians. It has specifically targeted educational (Cycling '74 2016b) institutes such as universities and IRCAM[30] which runs Max educational courses for professional composers.

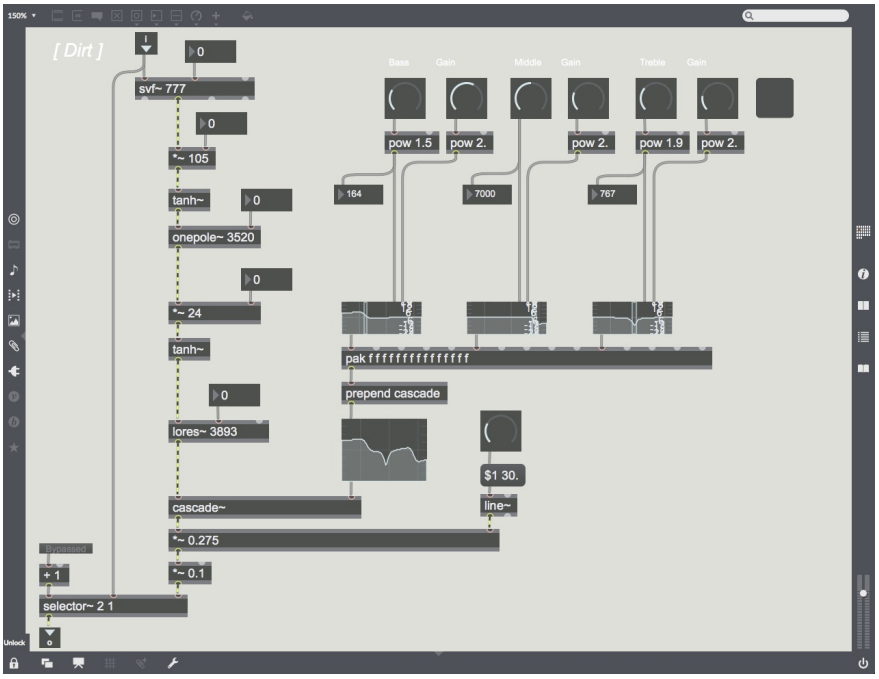


Figure 5.4: A 'patch' made in Cycling '74's Max (Cycling '74 2016a).

While targeting education can have benefits for a tool[31] there are potential negative effects for the student[32].

[28] MIT/Interaction Design Institute Ivrea/UCLA and Parsons/NYU respectively

[29] four out of five of the most popular teaching languages in 2014 (Guo 2014) are also in the top five languages in industry (TIOBE - The Software Quality Company 2016), the other is number 15.

[30] the Institute for Research and Coordination in Acoustics/Music, a public research centre 'dedicated to both musical expression and scientific research' (IRCAM 2016)

[31] further support for their paradigms, or from a more cynical perspective, more revenue for commercial tools like MATLAB or Max

[32] educational discounts are in a sense buying life long customers who have trouble leaving the now familiar paradigm if the crisis (price) is not big enough

...or lack thereof

The establishing of a paradigm brings with it the development of textbooks[33], which while proving clarity, precision and a systematic form (Kuhn 2012, p.164), also limit the historical sense of the reader to the most recent revolutions in the field (Kuhn 2012, p.136). There is also the tendency to rewrite history 'partly because the results of scientific research show no obvious dependence upon the historical context of the inquiry, and partly because, except during crisis and revolution, the scientist's contemporary position seems so secure' (Kuhn 2012, p.137).

This can be disastrous, as many of the good but lesser known ideas of the past are easily forgotten, this is common with programming languages where an idea or feature may be forgotten because the *language* falls out of fashion. In my own computer science education, there was no course dedicated to the history of the field; curious as to whether this was the norm I performed brief survey[34] of the top ten Computer Science university departments in the world (QS Quacquarelli Symonds Limited 2016), and found only Carnegie Mellon and Berkeley currently offered a history class, while Princeton and Stanford previously ran one.

Technology is people

> *"The important thing for activists to realize is that everything comes down to community. It's always about people."*
>
> (Srdja Popovic[35] - *Blueprint for Revolution*, p.545)

As much as we might want to imagine that 'hard', 'rational' subjects like Science and Computing are on a rational march of progress, or that the beautiful fields of Music, Science and Mathematics have an ethereal existence beyond our mortal hands, the reality is that they too are subject to our flawed humanity; our ideas and symbols for these things are *entirely* our creation. Kuhn discusses in depth the challenges a nascent paradigm faces: as well as having to explain existing phenomena, the scientific community must be persuaded, who will have an array of irrational biases, such as the personality, reputation and nationality of the innovator (Kuhn 2012, p.147-157). Hamming notes this in the more general process of progress:

> *"Often it is not physical limitations which control but rather it is human made laws, habits, and organizational rules, regulations, personal egos, and inertia, which dominate the evolution to the future"*
>
> ((Hamming 1997, p.5))

A large scale study of programming language adoption (Meyerovich and Rabkin 2013) found that the second most important factor[36] were the social aspects: especially team and personal experience. Surprisingly, objective factors

[33] literal or methodological

[34] results published at `https://github.com/acarabott/cs-history-education-review`

[35] One of the leaders of the Otpor! movement that helped topple Serbian president Slobodan Milošević (Popovic and Miller 2015, p.1)

[36] the most important was the availability of open source libraries: ready to use chunks of code for a particular domain

such as performance, reliability, simplicity, and security were weak influences on language choice (Meyerovich and Rabkin 2013, p.9).

Community matters

The Processing creative coding framework (Fry and Reas Casey 2016) and Arduino hardware prototyping platform (Arduino S.R.L 2016) were both developed and taught in educational settings. They provide extensive tutorials, encourage sharing of code, exhibit users' work prominently and nurture their communities.

These factors have made them admirably popular, despite their technical shortcomings[37].

Stagnation

Sometimes a lack of progress is caused by resistance to change. The 'that's not real programming' cycle has already been discussed[38], where it can take a new generation to embrace a new paradigm. Software also has a tendency to stick around, re-tooling is expensive (Kuhn 2012, p.76) and building a half-life into software is difficult (Kay 2009, 36:55).

Since the success of Leo Fender's Stratocaster in the 1950s (Bacon 2010, loc.458), the electric guitar has hardly changed in ways that are not cosmetic, in the words of Gibson Guitar's CEO Henry Juszkiewicz "The industry hasn't changed in 50 years. That's a lifetime!" (McIntyre 2015).

Musical interfaces have a tendency for skeumorphism: software tools go to great lengths to emulate the high quality sound of their analogue ancestors; unfortunately the philosophy carries through to their interfaces. The immediate familiarity will have helped convert sceptics to the new digital world, but are terrible interfaces considering their new medium.

Figure 5.5: The UAudio 1176 Limiter. One of these is a photograph of the hardware interface, the other is the software emulation.

At the basic level of interaction, a virtual rotating potentiometer is a ridiculous thing on a screen. These types of controls work in hardware because of their physical affordances: you can actually hold them! They are gripped and

[37] Processing has been described as 'poorly-designed' and ignorant of 'decades of learning about learning' (Victor 2012b)

[38] Chapter 3 – Computing Paradigm Shifts

turned with the thumb, not with a single pointing digit. On a screen, trying to rotate them is cumbersome, so they tend to use the vertical dragging movement that has no relation to the on screen control! (Figure 5.6)

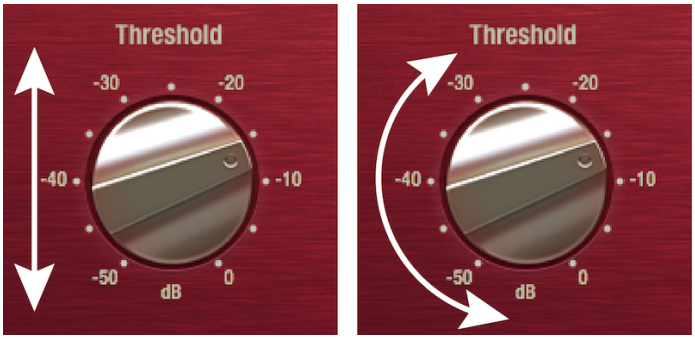


Figure 5.6: Two software dials from Logic Pro X (Apple Inc. 2016a) Neither of these two common interaction movements makes sense for the on screen control.

An even less excusable example of ignoring the computer is the fact that these interfaces are only designed for a real-time context, while the computer provides parallel, non-real-time access to data; interfaces usually provide feedback for the audio signal passing through them *now* despite the fact that they will be affecting many minutes of audio (Figure 5.7). Again, this makes sense *in the hardware world* where the device can not know the signal it is going to receive in the future. In the computer world however, we *know exactly* what the audio will look like in the future (and looked like in the past), it's right there, recorded to the hard disk or in memory![39]. So these interfaces provide feedback for a tiny sliver of time, while affecting the entirety of the signal, see Figure 5.8 for a threshold control that can be understood in the context of the whole recording. How to deal with stagnation? Unfortunately the smell may be something else...

> *"a new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die"*
>
> *(Thomas S. Kuhn (2012), p.150)*

Crisis

Kuhn talks of crisis as a part of the process of paradigm shifts (Kuhn 2012, p.66), common sense gives an understanding for this: what good is a revolution without a problem to solve? 'If [the paradigm] ain't broke, don't fix it'. Even if an alternative has a good claim, it needs to be enough of an improvement to demand 're-tooling', an expensive activity (Kuhn 2012, p.76). Crises are valuable because they *create value* for the *people* involved.

> *"you had to suffer shipwreck through your own efforts before you were ready to seize the lifebelt"*
>
> *(E Herrigel* - Zen in the art of archery*, p.35)*

[39] yes we could record to tape before computers, but there was no way to read the whole tape simultaneously

Figure 5.7: A threshold control from Logic Pro X (Apple Inc. 2016a) with disconnected, real-time only feedback.

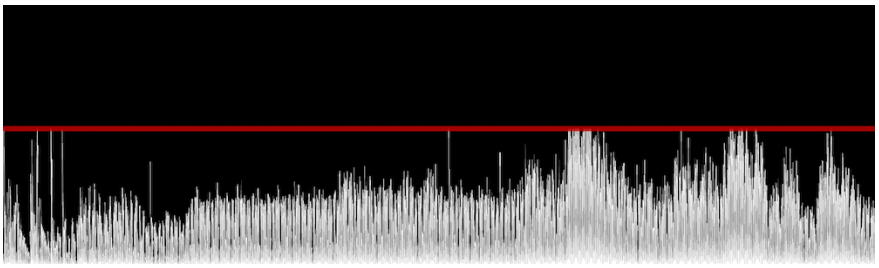


Figure 5.8: An improved threshold control by the author that places the threshold control in the context of the *whole* recording.

Python 2/3 - a lack of crisis (the curse of good enough)

The Python language has suffered a lack of crisis in its attempt at revolution. Version 3 was released in 2008, and was intentionally backwards *incompatible* with code written in version 2 (Rossum 2008). Despite settings 2015 as the 'End of Life' for version 2, in April 2014 this was extended to 2020 (Rossum 2014), as for many industries it is still the standard. One of the most incompatible changes was to how text data is handled (defaulting to Unicode instead of ASCII), which while important to web development[40] is less critical to say, Hollywood special effects automation; there is little incentive to invest millions of dollars to convert decades of reliable software.

MIDI and OSC: Crisis, what crisis?

MIDI: Crisis!

Before MIDI (Musical instrument Digital Interface) was established in 1982, spearheaded by Dave Smith of Sequential Circuits and Ikutaru Kakehashi of Roland, the burgeoning synthesiser industry was beginning to face a true crisis: compatibility; musicians wanted to control multiple synths using a single keyboard (to combine their sounds) but the manufacturers were using different mappings between voltage and parameters such as pitch (Billias, ch5-6).

After publishing a proposal at the Audio Engineering Society in 1981 (Smith and Wood 1981), Dave Smith organised a meeting at the 1982 January NAMM (National Association of Music Merchants) show with the big industry players. This had all the turmoil Kuhn describes as typical of the proposal face of a new paradigm (Kuhn 2012, pp.148, 149, 94) with companies making refusals and no resolution found. After the meeting, a small group of representatives from Japanese companies approached Smith, and together they forged ahead anyway, releasing the first MIDI equipped products in time to make the first MIDI connection between two companies' devices at the 1983 NAMM show (Billias; Smith 1997). With products from other conspirators following soon after, it was enough to make MIDI a de-facto standard, because "if Roland, Korg, Yamaha and Sequential did something, nobody would have a choice, they'd have to do it, they'd be left in the cold if they didn't abide by it." (Smith 1997, 2:24).

OSC: What crisis?

Since being announced in 1997 (Wright and Freed 1997) the OSC (Open Sound Control) protocol has offered itself as a competitor to MIDI as the standard for communication for 'sound synthesizers, computers and other multimedia devices' i.e. digital musical interfaces. The protocol was said to enable 'lower costs, increased reliability, greater user convenience, and more reactive musical control' and to avoid MIDI's flaws (Wright and Freed 1997). One of MIDI's biggest weaknesses is that data is 7-bit, so only whole numbers between 0 and 127 are possible, this is not ideal if you want to achieve a smooth curve over a long period of time as the jumps between values will be noticeable (see Figure 5.9). Today, OSC has a growing popularity in software communication, with support appearing in major applications, but has made little impact in the world of music hardware, where MIDI still reigns supreme; hardware makes up 18% of the

[40] ¡ éšṗéçîâłẏ îf̌ ẏøü çåré åb̲øüt̲ ṗéøṗłé ẅhø ḍø ñøt̲ ẅrît̲é îñ Éñg̲łîšh 凄すごいね!

OSC website's list of implementations (Open Sound Control Organisation 2016) (the rest are software), all of which for control (of other instruments sound, not sound generating devices themselves), all but one are amateur projects and one of which is to convert OSC signals into MIDI.

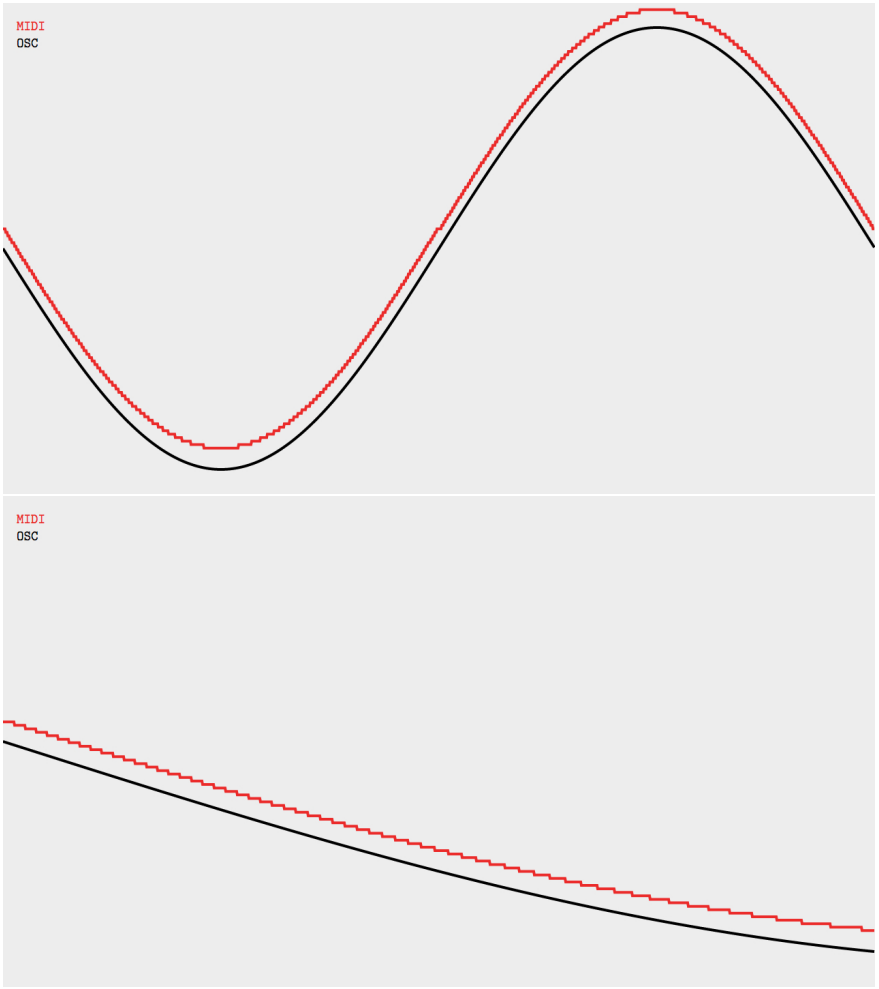


Figure 5.9: A sine curve (at two levels of zoom) drawn at MIDI and OSC resolutions, note the harsh stepping of MIDI compared with the smoother curve of OSC.

Despite MIDI's flaws, it has proven sufficient[41] for most musicians, as well as simple and cheap enough to be included on millions of products. Unfortunately for OSC, there wasn't enough of a crisis to warrant it becoming the new standard[42]

[41] or *not insufficient enough* to warrant uproar

[42] I have a personal sense of conflict bashing OSC, I have used it on many software projects, and it is really useful! I feel it should only be regarded as a failure on its own terms!

MIDI: the sequel(s)

True to Kuhn's model, MIDI established itself as the paradigm: becoming frozen by professionalism, hostile to change (after ironing out some early issues (Smith 1997)) and leading to an "immense restriction of the scientist's vision" (Kuhn 2012, p.64) (swap scientist for musician or audio engineer). MIDI has been accused of 'sapping momentum' from synthesis algorithm research by composers (Smith III 1991).

Fortunately for MIDI, it has a growing crisis on its hands: 'multidimensional polyphonic expression'. MIDI held up well with discrete notes (like those on a keyboard) combined with a pitch bend control that could smoothly bend the current notes, unfortunately (for MIDI) there has been a growth of controllers that permit bending *individual* notes, *independently of each other*, allowing for nuances like per key vibrato and changes in timbre, some of which are becoming commercially successful! (Figure 5.10)

Currently these have to use overly complex and inefficient set-ups to work properly, in some cases with a separate synthesiser per musical voice[43]. Fortunately for MIDI this means it has it's very own crisis on it's hands, hooray, time for a revolution!

While it remains to be seen how this will play out, there are currently *two* proposals for advancing MIDI, with differing approaches. The first is the "HD Protocol", for which there is no public specification, or even feature list, despite being under discussion since 2005 by The MIDI Manufacturers association (MMA), 'a volunteer-run organization that operates on industry consensus' (The MIDI Association 2015). Such troubling 'beginnings'[44], design-by-committee, and a weak battle cry: it will not replace MIDI, this would be "an unrealistic expectation [...] MIDI is extremely cost effective [...] while HD is not; and MIDI already handles most applications that people have today" (Synthtopia 2015), do not bode well. This could have been the fate of MIDI 1.0 if the band of rebel's had not said "let's just do it" (Smith 1997).

This was precisely the approach of the second proposal. Multidimensional Polyphonic Expression (MPE) proposes to *augment* the existing MIDI protocol with a set of rules that work on top of the original specification to support per-note parameter control. The group behind it was a collective known as the MPE / Expressive MIDI consortium, made up of representatives from a small number of companies[45] who released a *public* draft specification in March 2015 (Gaynes et al. 2015). The rapid development has meant that by December 2015 the group (now called the MPE Working Group) has been recognised by the MMA, who are now providing support and guidance (Adam et al. 2015).

Unfortunately for anyone holding out for a full blown revolution, this appears to be some 'normal science' work, bending the paradigm to sweep away an anomaly. Given the problems with OSC and the HD Protocol, this is perhaps a rational and practical approach, which has the potential for a revolution in controllers with polyphonic continuous control, successfully reducing one of the big gaps between acoustic and digital instruments[46]. Unfortunately this crisis

[43] this is like having six guitars, one for each string

[44] 11 years and counting

[45] Apple, Roger Linn Design, Eigenlabs, Moog Music, Uwyn, Haken Audio, Moog Music, Bitwig, Madrona Labs, ROLI, Roger Linn Design, Keith McMillen Instruments

[46] the violin has had this feature for hundreds of years

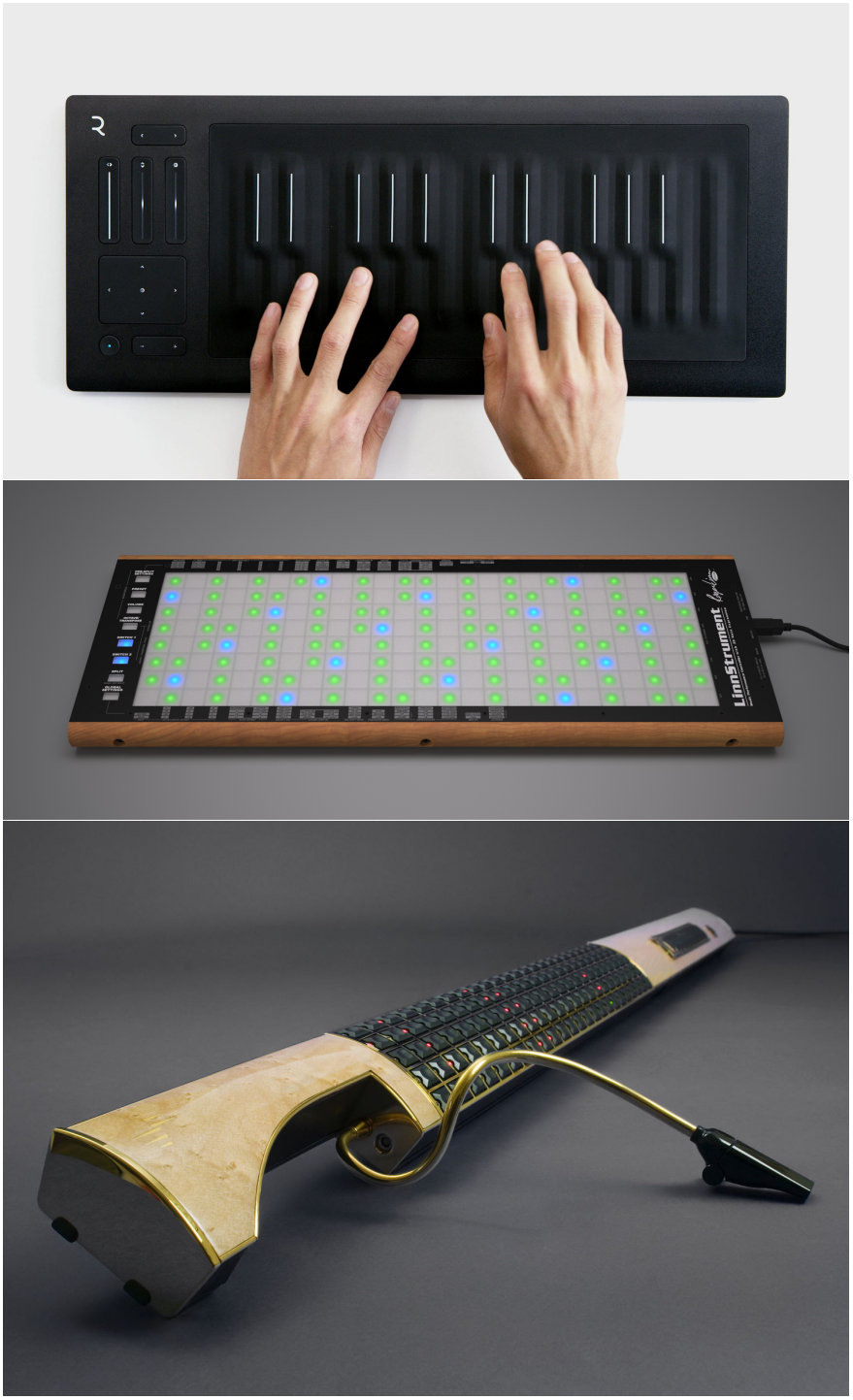


Figure 5.10: Three continuous MIDI controllers: ROLI's *Seaboard Rise* (ROLI 2015), Roger Linn Design's *LinnStrument* (Roger Linn Design 2014) and Eigenlabs' *Eigenharp* (Eigenlabs 2010).

diffusion will mean that it will be a while before a real improvement can become a standard.

Force and Endorse

Of course a simple method of ensuring your paradigm is successful is (providing you have the leverage) simply force it upon people.

Apple is known to use its considerable influence to kill incumbent technologies: the floppy disk with the iMac, Flash in the browser (while endorsing the new paradigm, modern HTML/CSS/JavaScript) with the original iPhone, and more recently the 3.5mm "minijack" headphone connection.

Apple are also able to promote their own programming languages (Swift and Objective-C) by making them the only way to write software that takes full advantage of their platform[47].

The shining example of success through force (or forced choice) is JavaScript; hastily designed in 10 days (Severance 2012, p.7-8), but the *only* language that can run within all web browsers.

[47] this is true of other vendors, like Microsoft with their .NET platform

Chapter 6

How to start a revolution

> *History suggests that the road to a firm research consensus is extraordinarily arduous.*
>
> (Thomas S. Kuhn - *The Structure of Scientific Revolutions*, p.15)

No royal road to revolution

There is no royal road to revolution[1], but we have looked at a number of roads (somewhat) travelled, from these we can take notes and look for similarities to our own situations in order to find clues as to what the most effective next step should be.

Kuhn's model provides some guidance: look for anomalies and question them, as they can be seen 'from another viewpoint as a counterinstance and thus as a source of crisis' (Kuhn 2012, p.80). A crisis provides a need for revolution and evidence to others that one is necessary; the case of MIDI and OSC[2] showed this to be true, and the case of Python[3] was an example of an attempt at revolution without *enough* crisis. More obviously a proposed paradigm must 'seem better than its competitors', but fortunately need not 'explain all the facts' (Kuhn 2012, p.18). Kuhn also claimed that (in science) a new paradigm demands the destruction of a prior paradigm (Kuhn 2012, p.96) but as we have already seen this is not true for all cases[4]. One anxiety inducing observation Kuhn makes is that those 'achieve these fundamental inventions of a new paradigm have been either very young or very new to the field whose paradigm they change' (Kuhn 2012, p.15). This was certainly true of the PARC researchers, who were often freshly graduated PhD's (Hiltzik 2009, p.178). This makes sense if we consider Kuhn's model on the *P-revolutionary*, personal level. If a practitioner has become too embedded in the current paradigm, their skill will become the 'limits of their language'[5], lacking the distance of a newcomer.

[1] to borrow a phrase from Euclid's response to King Ptolemy's request for a short cut to understanding geometry (Proclus and Morrow 1992, p.57)

[2] Chapter 5 - MIDI and OSC: Crisis, what crisis?

[3] Chapter 5 - Python 2/3 - a lack of crisis (the curse of good enough)

[4] Chapter 5 - Stagnation

[5] Chapter 3 - Is progress important in computing?, Chapter 4 - Is progress important in musical interfaces?

On the other hand, *enough* familiarity with the status quo is required to spot anomalies, as one needs to know 'with precision' what to expect (Kuhn 2012, p.65).

PARC[6] had a number of variables which were likely contributors to its success: a 'blue sky' research methodology, talented researchers, good timing and a near endless cash flow. Relative timing should also be considered, if a revolution has recently taken place, it is unlikely that the field will embrace *another* revolution soon afterwards.

Rather than trying to start a revolution alone, it may be more effective to create a convincing argument for one, focusing on changing people's minds, especially if the idea is not yet fully formed; make your ideas so compelling and beyond other's language that *your work is their crisis*. If pursuing this approach, it is key to know where it is and is not worth expending effort: on presenting the *ideas* in a way that is believable, not on having a fully functioning implementation[7].

Ideas stay alive because they are in the heads of living people, if those people die, so do the ideas. Building a community of 'believers' is critical, they will progress the idea into reality[8]

Education can be a key leverage point for influencing ideas and building a community[9], but be conscious of the fact that your impact can be significant and possibly permanent[10].

Maintaining progress will require an awareness of the fact that you are in the midst of a stagnant paradigm[11] just recognising the fact may be enough of a cause for concern that anomalies gain new importance, and start being prodded, in the hope of finding a real source of crisis.

With enough leverage, you may have the ability to turn tides with, -at most- self appointed crises or -at least- whimsy[12].

If you really want to start a revolution, shift or create a paradigm, or change the world, it may be best to follow Dave Smith's example and just go and do it[13] because as Serbian revolutionary Srdja Popovic points out, the person who will start your revolution 'has to be you' (Popovic and Miller 2015, p.308).

Patterns in Paradigms

I found Kuhn's model of progress in Science compelling both because it was a formalisation of what feels like an emergent process, one determined by chance ('who knows when the next big thing will happen? It could be right around the corner!') and also because I felt a similarity with progress in the fields I engage with.

By the model's own reasoning it should not be taken as infallible, it is as much a paradigm as any it describes. It had to fight detractors and win over followers when it first sought to establish itself, and (being a piece of writing)

[6]Chapter 5 - Xerox PARC: Dealers of Lightning, Catalyst of Crisis
[7]Chapter 5 - How to shift paradigms and influence people
[8]Chapter 5 - Technology is people
[9]Chapter 5 - Education
[10]Chapter 3 - Is progress important in computing?
[11]Chapter 5 - Stagnation
[12]Chapter 5 - Force and Endorse
[13]Chapter 5 - MIDI: Crisis!

it stiffened due to the necessity of publication, and eventually froze with the death of its author. We are now left with 'normal science' work in the form of introductory essays in anniversary editions (Kuhn 2012), and explorations of the ideas such as this dissertation. These may provide useful or illuminating, but in the bigger picture can only serve to bend the paradigm to account for anomalies until the **anomaly** → **crisis** → **revolution** → **paradigm** pattern can occur, which in the new paradigm will be accounted for entirely differently!

Writing and thinking about the model applied to these different fields lead me to a realisation as to why I found this model so appealing and applicable. It became a framework that I felt applied to more than just particular *fields*, but to personal endeavours. Studying an instrument, developing technique and proficiency require a great deal of work, but the biggest jumps in progress seemed to come from realisations, mental breakthroughs, changes in perspective. In my teens I attended a guitar camp in the woods near Gothenberg run by one of my teenage guitar heroes, Mattias "IA" Eklundh (Eklundh 2016) who explained this experience as "like running, and wondering why you are finding it so difficult until you look down and realise your shoelaces are tied together".

Once these realisations occur, it takes a *lot* of 'normal science' (i.e. practice) to realise their potential. At this point they become ingrained habits, letting you tackle material with ease: e.g. being able to quickly determine fingerings or picking patterns. The downside is that your technique determines the music you *can* make, and thus the music that you *will* make.

Habits are difficult to break, especially if they are serving you well. Changing them is a process of unlearning and relearning. As you explore the new technique, at first you are unable to play everything you could before, just as new scientific paradigms cannot account for all the phenomena that their predecessors could, the experience is humiliating. It is also time consuming, it is a 're-tooling', which is expensive (Kuhn 2012, p.76). Martin Taylor is one of the most highly regarded guitarists today, who with phenomenal technique plays bass lines, chords and melodies simultaneously. Yet Taylor advocates that students **do not** copy the practise of resting the little finger of the picking hand on the guitar, which Taylor regards as a bad habit (Taylor and Mead 2003, p.40). Doing this robs the player of a useful finger, but for a player with technique as formidable as Taylor's, it is evidently not enough of a hindrance to induce a Kuhnian crisis.

For these reasons, it is a process that is avoided by many musicians[14]. Indeed, it can take a crisis to force a musician to make such changes. A memorable personal experience of crisis occurred when attending a 'Gypsy Jazz' or 'Jazz Manouche'[15] workshop, which concluded in a large group jam session. The genre is primarily acoustic, and we were all playing acoustic guitars. When my turn to solo came around, my electric guitar technique[16] let me down: it did not project anywhere close to those who played exclusively on acoustic instruments. I was shocked, and began work on a new picking technique that would produce

[14]ever wondered *why* musician's later work tends to be a re-hashing of their early material? Although credit has to be given to the audiences, who establish their own expectations of artists...

[15]the genre established by guitarist Django Reinhardt in the 1930s

[16]with an amplified instrument there is no need to play as loudly as possible, which requires bigger movements. There *is* the requirement to mute the strings you are not playing more than on an acoustic instrument (as these will be amplified), which means the hand has to stay close to the guitar, using smaller movements

more volume on an acoustic guitar.

A similar story was be found with programmers, who despite continuing to learn new languages regardless of age (Meyerovich and Rabkin 2013), have a tendency to program in the style of their first (Section 3).

The story is repeated in many aspects of life, how many people fail to address an aspect of their health -their posture, weight, sedentariness, exercise- that is easily identifiable as *not so great*, until it creates a real crisis. How many people still 'hunt and peck' at the keyboard instead of learning to touch type? If and when it is dealt with, it can often require whole lifestyle changes, followed by proclamations of the process being 'life changing', and now being 'a whole new person'.

In *The Creative Mind: Myths and Mechanisms* Boden coins the terms *P-creative* and *H-creative* to account for two different types of creative events: *personal* and *historical*. P-creativity occurs when a person "has an idea which she could not have had before", it is creative "no matter how many people may have had the same idea already". H-creativity occurs when "no one has had that thought before" (Boden 2004, p.43). Boden references Kuhn's work (Boden 2004, p.75), and unsurprisingly the big scientific names overlap in their work[17]. There is a pattern at play here, the process of revolution is not limited to Science, Computing, or Musical Interfaces, it is a human pattern that permeates our efforts at progress; it can be *P-revolutionary*, *H-revolutionary* or *R-revolutionary*, *Relatively revolutionary* - "revolutionary only to only those whose paradigms are affected by them" (Kuhn 2012, p.93).

[17]Einstein, Kepler, Newton, Darwin...

"Learning a new idea, something that your brain isn't well set up for, could require almost as much creativity as inventing it in the first place, as you have to invent it inside your head. And so you can say that 'normal' is the greatest enemy with regard to creating the 'new'. The way of getting around this is you have to understand normal not as reality, but just a construct. The way to do that, for example, is just travel to a lot of different countries, and you'll find a thousand different ways of thinking the world is real."

(Alan Kay - Normal Considered Harmful *2009, 39:08)*

Appendices

Appendix A

Squeak/Smalltalk session with Yoshiki Ohshima

Notes from a Smalltalk / Squeak (Squeak Project 2016) pair programming session with Yoshiki Ohshima, a researcher who has collaborated with Alan Kay[1] and Dan Ingalls[2] since 2000 (Viewpoints Research Institute 2016).

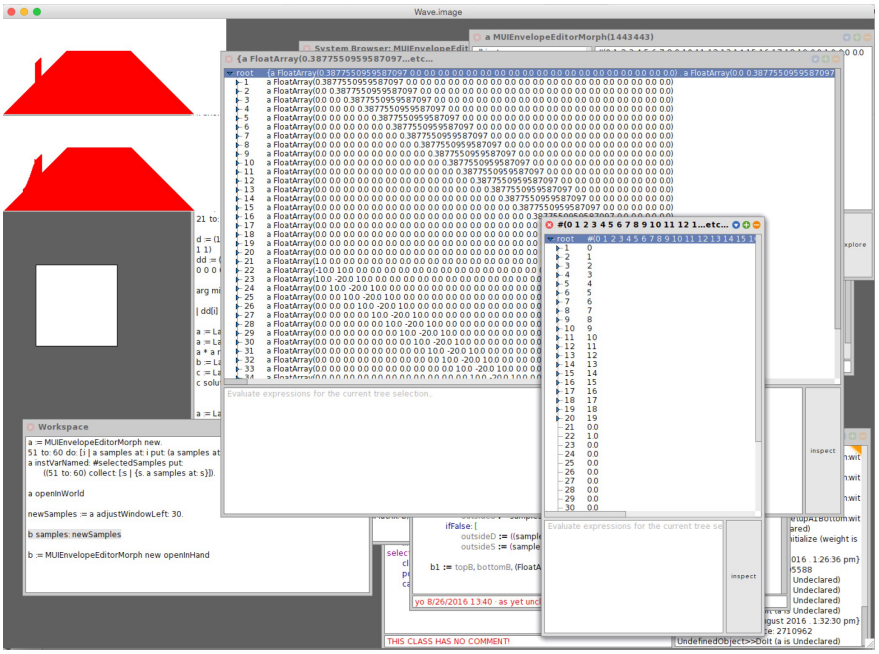


Figure A.1: Working on an envelope GUI widget in Squeak / Smalltalk.

[1] Smalltalk's inventor
[2] Smalltalk's lead implementer and greatest champion

25/8/16 CHS

Yo
SQUEAK PAIR PROG.

"FIND ~~ATTENTION~~ self feeling "this isn't the work factor, the hard bit is the thinking"

What is this? feeling

Combo of
- 'don't understand'
- unfamiliar
- Why so clicking
- this is really nice
- the gap is big
MAYA Lovers

- How to help this win?

Almost
No reload/ compiling/ waiting
even if slow progress, don't wait for computer ever

Lots of clicking

"We have liftoff"

Another metaphor messy desk

Transcription

- "Find self feeling this isn't that much faster, the hard bit is the thinking"
- "Almost no reload[ing]! / compiling, even if slow progress, [we] don't wait for the the computer ever" (we were having difficulties with the underlying *problem* more than the *programming*)
- "What is this feeling? Combo of
  - '[I] don't understand'
  - unfamiliarity
  - 'why so [much] clicking [with the mouse]?'
  - 'this is really nice'
  - 'the gap is big - MAYA' (Loewy 1951)
  - 'How to help this win?',
- "Where did it go?" (The large number of overlapping windows turned the desktop metaphor brought with it the problems of a *messy* desktop).

Appendix B

Cult

In reading for this dissertation a theme of cult or religious like behaviour appeared[1]. In computing discussions, many topics are referred to as being 'religious', especially in arguments over operating systems, browsers, programming languages, and the ultimate religious war: text editors[2].

I have come to the conclusion that starting a form of cult is probably one of the most effective (if morally questionable) methods of guiding a lot of human power towards progress (in the direction of the cult leader's choosing). Fully exploring this would be for another piece of writing, instead here are some quotes from the cutting room floor.

From Thomas Kuhn

> *"The man who embraces a new paradigm at an early stage must often do so in defiance of the evidence provided by problem-solving. He must, that is, have faith that the new paradigm will succeed with the many large problems that confront it, knowing only that the older paradigm has failed with a few. A decision of that kind can only be made on faith."*
>
> *(Kuhn 2012, p.156)*

> *"But crisis alone is not enough. There must also be a basis, though it need be neither rational nor ultimately correct, for faith in the particular candidate chosen. [...] Men have been converted by them at times when most of the articulable technical arguments pointed the other way." (Kuhn 2012, p.157)*

Alan Kay

Head of PARC's Learning Research Group. I wrote the majority of this dissertation in the mornings, weekends and evenings while interning at HARC, a

[1] Although this may just reflect the authors' writing...

[2] Richard Stallman the cult leader-ish free software activist and creator of GNU Emacs (one of the two editors at the centre of this religious war, the other being Vi) has jokingly created The Church of Emacs and the character of St IGNU-cius

research group initiated and supported by Alan Kay; some of the researchers have been working with Alan for decades (Dan Ingalls since the PARC days). Alan's presence was entrancing, and he was regularly spoken of in reverential terms[3] and quoted regularly; this created the feeling of a slight cult, or at least of a cult leader, and I felt myself drink the Kool-Aid readily...

> *"Among the computer scientists familiar with his ideas, half thought he was a crackpot and the other half a visionary. His name was Alan Kay."*
>
> *(Hiltzik 2009, p.132)*

> *"Having spent decades as an intellectual lone wolf, Kay redirected his gift for communicating enthusiasm toward the goal of attracting followers"*
>
> *(Hiltzik 2009, p.307)*

> *"Kay was known for wielding the most beguiling paintbrush in the building. No one evangelized more convincingly on behalf of ideas he found compelling, whether they were his own or belonged to others. Kay proselytized out of necessity. The experience of emerging from grad school with a four- hundred-page thesis describing a machine that could not be physically realized had sent him into a psychological tailspin. An old tendency toward depression, spurred by his inability to execute, reasserted itself. "Right about that first year at PARC, under psychotherapy, I discovered I was confusing my talent with my temperament," he said. "I didn't have the temperament of a programmer. I realized I needed a group." This epiphany resembled that of a poet suddenly finding his voice."*
>
> *(Hiltzik 2009, p.307)*

> *"Drawn toward a new vision of Ideaspace, he brought the entire group to Pajaro Dunes for a three-day offsite in January 1976 to chart the new journey. Reinfused with enthusiasm, he even gave the retreat a theme–"Let's burn our disk packs," an allusion to the big yellow Alto storage disks on which they kept Smalltalk's master code. Then he discovered that they were no longer willing to follow him blindly."*
>
> *(Hiltzik 2009, p.436)*

Douglas Engelbart

Computer visionary who aimed to 'augment human intellect', inventing hypertext, video conferencing, the desktop, and the mouse in the process (Hiltzik 2009, pp.112-114), usually reduced in obituaries to 'inventor of the mouse'.

> *"young engineers and scientists who felt their lives altered by their first meetings with the charismatic Doug Engelbart and who regarded his vision with an almost religious awe. "He not only made sense," recalled Bill Duvall, one of the early disciples."*
>
> *(Hiltzik 2009, p.112)*

[3] not undeservedly, he's the kind of person who built his own harpsichord while inventing a powerful new programming paradigm, reads about 300 books a year and is good friends with world renowned scientists, music producers, CEOs...

"As a team they infused Engelbart's principles into PARC like apostles spreading religion."
(Hiltzik 2009, p.116)

"One admirer called him "a prophet of biblical dimensions," a role he fit down to his physical appearance. Tall and craggy, with deepset eyes and a hawklike nose, he might have been carved from a slab of antediluvian granite. Soft-spoken but intransigent, his years of battling unbelievers had convinced him that he was fated to remain the solitary leader of a devoted cadre."
(Hiltzik 2009, p.115)

"One other individual entranced by Engelbart's work was Bob Taylor."
(Hiltzik 2009, p.112)

Bob Taylor

Associate manager of PARC's Computer Science Laboratory.

"I saw Taylor's relationship to his lab members as analogous to Jim Jones and the Jonestown cult"
(Hiltzik 2009, p.518)

" "It was almost a cult-like thing" remembered Lynn Conway"
(Hiltzik 2009, p.222)

"Taylor's a very powerful personality. Here he was in the background with these gunslingers out front and the groupies in back."
(Hiltzik 2009, p.222)

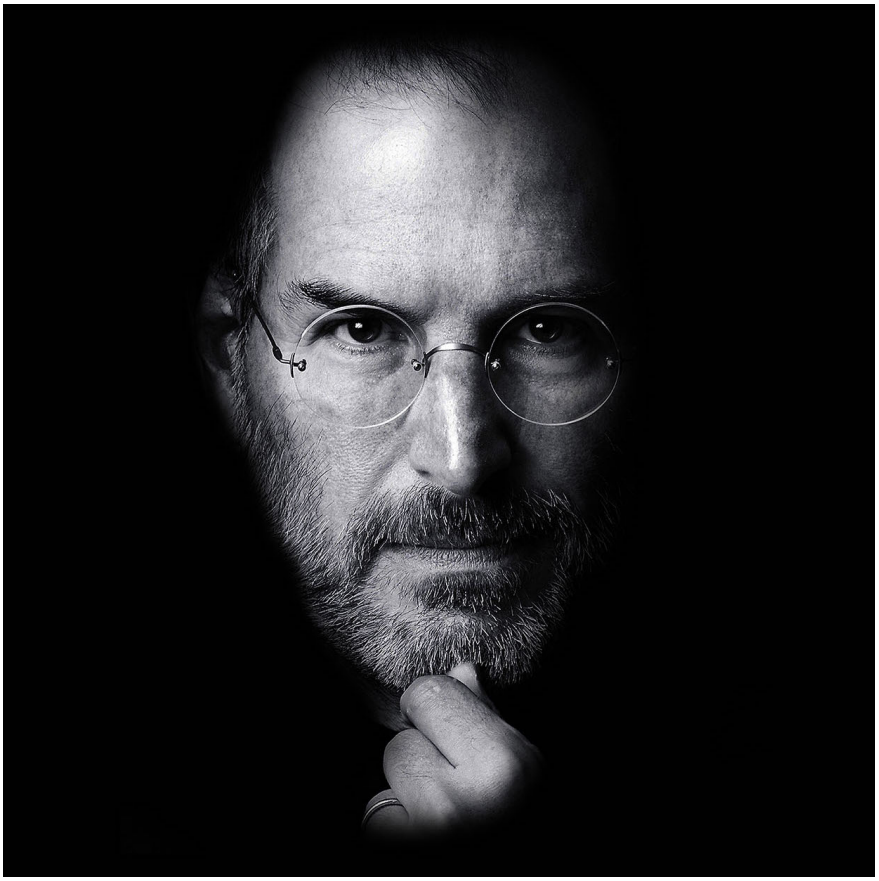


Figure B.1: Cult leader?

Appendix C

Interview with Bret Victor

On the 18th of August 2016 I interviewed Bret Victor at the HARC Los Angeles office. This is included in the bibliography and cited as (Victor 2016b). My notes are below, text in boxes/partial boxes are my summaries of the points discussed.

BRET VICTOR INTERVIEW

18/8/10 @HARC-LA ~18-19:00

- D&R
- COMP - CRISIS - APPLY TO
- KUHN MODEL APPLY TO COMP, MUS, etc

BRET VICTOR INTERVIEW 18/8/16 @HARC-LA 1/2

- loose fit / } [Kuhn
- binding
- paper - [maybe criticism
- Kuhn conference - panels

[look for Kuhn critics

[talking past each other

[goal of work

Real enough, not fantasy. fantasy

[Believable, inspiring

- Everything is there

ODK

- Microsoft videos
- AMS vision vids
- Akay 'invent the future + live in it"
- Not enough to create vision
- Prototyping [right word
  - being used
  - bootstrapping? [explain

Building (things) to build on top of

MISREPRESENTATION

~~Misrepresentation~~

"most people don't know what a prototype is"

mad when not released - [people mad when no code

'Gilded Cage' [rat stuff up and don't build on it]

"don't want to pollute the world" - [code not good enough. to build everything on is visible: UI

pollute

very different, version to version → flow of : ideas

- Smalltalk ~~diverging~~ - became a paradigm

escaped lab ~~escaping~~ @v.80

'frozen smalltalk, end of ideas

backwards compatible

- LUA + JS

- LUA embed. not ~~backwards compat~~, just change version to version, game to game

- ~~Just swap~~ - Just swap

- JS big ~~userbase~~ userbase @ day 1 → paradigm → Frozen

DBX TALK considered ~~not considered~~ ~~bad~~ worth'

part of 'my work' was a joke

C++ → Java - Python

can be construed as paradigm shifts

apps not used anymore

BRET VICTOR INTERVIEW 18/8/16 @HARC-LA

disappointed that a something SMALLTALK

- Alan ~~disappointed~~ a better than ST basic, some along. hasn't come along
- " in favour obsoleting squeak —— kill squeak!
- small group of people comfortable in it. - HOSTILE to OUTSIDERS/NEW
  no attempt to expand ~~user~~ userbase

- Lively not sure what it wants to be
  - prototype?
  - idea maker?
  - Researcher tool

"horific"

Deploy or Die

- This is what
  ↓ ~~Processing~~ Processing is
  - Not a product

- GITHUB CULTURE
  - NOT thought-out
  - just put out
  - crap piles ..

~~Before~~ ~~10 years~~
Research takes 10 years

Common ~~Common~~ LISP
- ~~Standardistion~~ standardisation

~~HASKELL~~

- A Standardisation
- Haskell similar for lazy, typed langs

Bibliography

Abelson, Hal, and Gerald Jay Sussman. 1985. *The Structure and Interpretation of Computer Programs*. 9:81. 3.

Adam, Chris, et al. 2015. *MPE : Proposed specification - Revision 1.25a*. Tech. rep. MIDI Manufacturers Association - MPE Working Group. https://docs.google.com/document/d/1vpjxoPHw82X3xyNvE6%7B%5C_%7DhsDeL86vloNQZC83NHD8edow.

Apple Inc. 2016a. *Logic Pro X*. Visited on 09/23/2016. http://www.apple.com/logic-pro/.

Apple Inc. 2016. *Swift Playgrounds*. Visited on 09/15/2016. http://www.apple.com/swift/playgrounds/.

Apple Inc. 2016b. *Xcode*. Visited on 08/29/2016. https://developer.apple.com/xcode/.

Arduino S.R.L. 2016. *Arduino*. Visited on 09/17/2016. http://www.arduino.org/.

Association for Computing Machinery. 2016. *A.M. Turing Award*. Visited on 09/09/2016. http://amturing.acm.org/.

Audacity Team. 2016. *Audacity*. Visited on 09/23/2016. http://www.audacityteam.org/.

Bacon, Tony. 2010. *The Stratocaster Guitar Book: A Complete History of Fender Stratocaster Guitars*. Backbeat Books.

Baltensperger, A. 1996. *Iannis Xenakis und die stochastische Musik*. Publikationen der Schweizerischen Musikforschenden Gesellschaft / 2 v. 1. Haupt.

Billias, Athan. *MIDI History*. https://www.midi.org/articles/the-history-of-midi.

Bird, Alexander. 2013. "Thomas Kuhn". In *The Stanford Encyclopedia of Philosophy*, Fall 2013, ed. by Edward N Zalta. http://plato.stanford.edu/archives/fall2013/entries/thomas-kuhn/.

Bitwig Studio. 2016. *Bitwig*. Visited on 09/23/2016. https://www.bitwig.com/.

Boden, Margaret A. 2004. *The Creative Mind: Myths and Mechanisms*. Second. London: Routledge.

Carlin, George. 1986. *Stuff*. Los Angeles: HBO. https://www.youtube.com/watch?v=MvgN5gCuLac.

Clark, Andy, David Chalmers, and David Chalmers '. 1998. "The extended mind". *Source: Analysis* 58187254 (1): 7–19.

Clark, Jack. 2016. *Why Google Wants to Sell Its Robots: Reality Is Hard*. Visited on 08/30/2016. http://www.bloomberg.com/news/articles/2016-03-18/why-google-wants-to-sell-its-robots-reality-is-hard.

Collier, David a., and Susan M. Meyer. 2000. "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl". *International Journal of Operations {&} Production Management* 20 (6): 705–729.

Community, openFrameworks. 2016. *openFrameworks Homepage*. Visited on 09/17/2016. http://openframeworks.cc/.

Connor, Steven. 2010. "Thinking things". *Textual Practice* 24 (1): 1–20.

Cooper, A. 1999. *The Inmates are Running the Asylum*. Sams.

Crockford, D. 2008. *JavaScript: The Good Parts*. O'Reilly Media.

Csikszentmihalyi, Mihaly. 2009. *Flow*. Harper Perennial Modern Classics. Harper-Collins.

Culkin, John M. 1967. "A schoolman's guide to Marshall McLuhan". *Saturday Review* 50 (11): 51–53, 70–72.

Cycling '74. 2016a. *Max*. Visited on 09/17/2016. https://cycling74.com/products/max/.

— . 2016b. *Max in Education*. Visited on 09/17/2016. https://cycling74.com/max-in-education/.

Dijkstra, E W. 2012. *Selected Writings on Computing: A personal Perspective*. Monographs in Computer Science. Springer New York.

Dunne, A, and F Raby. 2013. *Speculative Everything: Design, Fiction, and Social Dreaming*. MIT Press.

Eigenlabs. 2010. *Eigenharp*. Visited on 09/05/2016. http://www.eigenlabs.com/.

Eklundh, Mattias "IA". 2016. *Mattias IA Eklundh's Freak Guitar - The Official Homepage*. Visited on 09/08/2016. http://freakguitar.com/.

Engelbart, Douglas. 1962. "Augmenting Human Intellect: A Conceptual Framework". *Contract* 49 (3578): 80.

Fry, Ben, and Reas Casey. 2016. *Processing*. Visited on 09/17/2016. https://processing.org/.

Gannon, J. D. 1977. "An Experimental Evaluation of Data Type Conventions". *Commun. ACM* 20, no. 8 (): 584–595.

Gaynes, Amos, et al. 2015. *MIDI Specifications for Multidimensional Polyphonic Expression (MPE) - Revision 1.10*. Tech. rep. MPE / Expressive MIDI

Consortium. https://docs.google.com/document/d/1-26r0pVtVBrZHM6VGA05hpF-ij5xT6aaXY9BfDzyTx8/edit.

Granger, Chris. 2012. *Light Table by Chris Granger - Kickstarter*. Visited on 09/15/2016. https://www.kickstarter.com/projects/ibdknox/light-table/description.

Grymes, James A. 1998. "Dispelling the myths: The opening bassoon solo to the Rite of Spring". *The Journal of the International Double Reed Society* 26.

Guo, Philip. 2014. *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*. Tech. rep. Communications of the ACM. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext.

Guzdial, Mark. 2016. *Five Principles for Programming Languages for Learners*. Visited on 08/31/2016. http://cacm.acm.org/blogs/blog-cacm/203554-five-principles-for-programming-languages-for-learners/fulltext.

Hamming, Richard. 1997. *The Art of Doing Science and Engineering : Learning to Learn*. 218. Gordon / Breach.

Herndon, Thomas, Robert Pollin, and Michael Ash. 2013. "Does High Public Debt Consistently Stifle Economic Growth ? A Critique of Reinhart and Rogoff". *Political* 38 (2): 257–279.

Herrigel, E. 1953. *Zen in the art of archery*. Vintage spiritual classics. Pantheon Books.

Hiltzik, M A. 2009. *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*. HarperCollins.

Ingalls, Daniel H H. 1978. "The Smalltalk-76 programming system design and implementation". In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 9–16. ACM.

IRCAM. 2016. *Ircam: Formations professionnelles*. Visited on 09/17/2016. http://www.ircam.fr/formations.html.

Kay, Alan. 1972. *A Personal Computer for Children of All Ages*. Tech. rep. Palo Alto.

— . 2009. *Normal Considered Harmful*. UIUC. https://youtu.be/FvmTSpJU-Xc.

— . 2011. "Programming and Scaling". In *HPI-Colloquium (ST 2011)*. Potsdam: Hasso-Plattner-Institut. http://www.tele-task.de/archive/video/flash/14029/.

— . 1993. "The early history of Smalltalk". In *The second ACM SIGPLAN conference on History of programming languages - HOPL-II*, 28:69–95. 3. New York, New York, USA: ACM Press.

Kay, Alan, et al. 2007. "STEPS Toward The Reinvention of Programming". *VPRI Technical Report*, no. TR-2007-008.

Kodowa Inc. 2016. *Light Table*. Visited on 09/15/2016. http://lighttable.com/.

Krugman, Paul. 2013. *The Excel Depression*. New York, New York, USA. http://www.nytimes.com/2013/04/19/opinion/krugman-the-excel-depression.html.

Kuhn, Thomas S. 2012. *The Structure of Scientific Revolutions*. 50th Anniv. University of Chicago Press.

Kuskis, Alex. 2013. *We shape our tools and thereafter our tools shape us*. Visited on 09/28/2016. https://mcluhangalaxy.wordpress.com/2013/04/01/we-shape-our-tools-and-thereafter-our-tools-shape-us/.

Lattner, Chris. 2016. *Chris Lattner's Homepage*. Visited on 09/15/2016. http://www.nondot.org/sabre/.

Lifelong Kindergarten Group at the MIT Media Lab. 2016. *Scratch Homepage*. Visited on 08/31/2016. https://scratch.mit.edu/.

Loewy, Raymond. 1951. "The MAYA Stage". In *The Industrial Design Reader*, ed. by Carma Gorman, 155–158. Allworth Press.

Magnusson, Thor. 2009. "Of Epistemic Tools: musical instruments as cognitive extensions". *Organised Sound* 14 (02): 168–176.

McCarthy, John. 1960. "Recursive functions of symbolic expressions and their computation by machine, Part I". *Communications of the ACM* 3 (4): 184–195.

McIntyre, Hugh. 2015. "Gibson Guitars ' $ 40 Million , 11 Year Tech Gamble". *Forbes*. http://www.forbes.com/sites/hughmcintyre/2015/06/16/gibson-guitars-40-million-11-year-tech-gamble/print/.

Mciver, Linda. 2000. "The Effect of Programming Language on Error Rates of Novice Programmers".

Meyerovich, Leo a., and Ariel S. Rabkin. 2013. "Empirical analysis of programming language adoption". *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications - OOPSLA '13*, no. Section 7: 1–18.

Minhinnett, R, and B Young. 2006. *The Story of the Fender Stratocaster*. Carlton.

Moore, Gordon E. 1998. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff." *Proceedings of the IEEE* 86 (1): 82–85.

Nijs, L, M Lesaffre, and M Leman. 2009. "The musical instrument as a natural extension of the musician". *the 5th Conference of Interdisciplinary*.

Nisbett, Richard. 2015. *Mindware: Tools for Smart Thinking*. Penguin Books Limited.

Norvig, Peter. 2016. *Why didn't Common Lisp fix the world?* Visited on 09/15/2016. https://www.quora.com/Where-did-we-go-wrong-Why-didnt-Common-Lisp-fix-the-world.

Ohshima, Yoshiki, et al. 2012. *STEPS Toward The Reinvention of Programming, 2012 Final Report Submitted to the National Science Foundation (NSF) October 2012*. 2012:52. 0639876.

Open Sound Control Organisation. 2016. *Open Sound Control - Implementations*. http://opensoundcontrol.org/implementations.

Pirsig, Robert. 1992. *Lila: An Inquiry Into Morals*. A Bantam Book. Bantam Books.

— . 1974. *Zen and the art of motorcycle maintenance*. A Bantam book. Bantam.

Popovic, Srdja, and Matthew Miller. 2015. *Blueprint for Revolution: How to Use Rice Pudding, Lego Men, and Other Nonviolent Techniques to Galvanize Communities, Overthrow Dictators, or Simply Change the World*. Random House Publishing Group.

Proclus and G R Morrow. 1992. *A Commentary on the First Book of Euclid's Elements*. Classics/History of mathematics. Princeton University Press.

Pushkin, A S, and V V Nabokov. 1990. *Eugene Onegin: Commentary and index*. Bollingen Series (General) Series. Princeton University Press.

QS Quacquarelli Symonds Limited. 2016. *Computer Science & Information Systems Rankings*. http://www.topuniversities.com/university-rankings/university-subject-rankings/2016/computer-science-information-systems%7B%5C#%7Dsorting=rank+region=+country=+faculty=+stars=false+search=.

Red Bull. 2012. *Tom Schaar stomps first-ever skateboarding 1080*. Visited on 09/16/2016. https://www.youtube.com/watch?v=tbjzZHuGTng.

Ritchie, DM, and B Kernighan. 1978. *The C programming language*.

Roads, Curtis. 2001. *Microsound*. 409. MIT Press.

Roger Linn Design. 2014. *LinnStrument*. Visited on 09/05/2016. http://www.rogerlinndesign.com/linnstrument.html.

ROLI. 2015. *Seaboard RISE*. Visited on 09/05/2016. https://roli.com/products/seaboard-rise.

Rosetta Code. 2016. *Reverse a string*. Visited on 08/28/2016. http://rosettacode.org/wiki/Reverse%7B%5C_%7Da%7B%5C_%7Dstring%7B%5C#%7D360%7B%5C_%7DAssembly.

Rossum, Guido van. 2014. *Python PEP 373 - Extend Python 2.7 life till 2020*. https://hg.python.org/peps/rev/76d43e52d978.

— . 2008. *What's New In Python 3.0*. Visited on 09/03/2016. https://docs.python.org/3/whatsnew/3.0.html.

Schaeffer, Pierre. 1977. *Traité des objets musicaux essai interdisciplines*. Second. Éditions du Seuil.

Severance, Charles. 2012. "JavaScript: Designing a Language in 10 Days". *Computer* 45, no. 2 (): 7–8. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6155645.

Smith III, Julius O. 1991. “Viewpoints on the History of Digital Synthesis”. In *Proceedings of the International Computer Music Conference*, 1–10. Montreal.

Smith, Dave, and Chet Wood. 1981. “The ’USI’, or Universal Synthesizer Interface”. In *Audio Engineering Society Convention 70*. Audio Engineering Society.

Squeak Project. 2016. *Squeak/Smalltalk Homepage*. Visited on 08/29/2016. `http://squeak.org/`.

Sterken, Sven. 2007. *Music as an Art of Space: Interactions between Music and Architecture in the Work of Iannis Xenakis*. Resonance (Ames, Iowa). Culicidae Architectural Press.

Sublime Text. 2016. *Sublime Text Homepage*. Visited on 08/29/2016. `https://www.sublimetext.com/`.

Sureau, Denis. 2015. *The list of programming languages by dates*. Visited on 08/30/2016. `http://www.scriptol.com/programming/chronology.php`.

We Really Don’t Know How To Compute! 2011. Strange Loop. `https://www.infoq.com/presentations/We-Really-Dont-Know-How-To-Compute`.

Sutherland, Ivan. 1964. “Sketch pad a man-machine graphical communication system”. In *Proceedings of the SHARE design automation workshop*, 6–329. ACM.

Taylor, M, and D Mead. 2003. *Mel Bay Presents the Martin Taylor Guitar Method*. Mel Bay.

The MIDI Association. 2015. *MIDI Manufacturers Investigate HD Protocol*. Visited on 09/04/2016. `https://www.midi.org/articles/midi-manufacturers-investigate-hd-protocol`.

TIOBE - The Software Quality Company. 2016. *TIOBE Index*. Visited on 09/10/2016. `http://www.tiobe.com/tiobe-index/`.

Torvalds, Linus. 2016. *Git Homepage*. Visited on 08/29/2016. `https://git-scm.com/`.

Venners, Bill. 2003. *The Philosophy of Ruby - A Conversation with Yukihiro Matsumoto*. Visited on 08/29/2016. `http://www.artima.com/intv/ruby4.html`.

Victor, Bret. 2016a. *Bret Victor Website*. Visited on 09/15/2016. `http://worrydream.com/`.

— . 2012a. “Inventing on Principle”. In *Canadian University Software Engineering Conference*. Montreal, Quebec. `https://vimeo.com/36579366`.

— . 2011. *Kill Math*. Visited on 09/15/2016. `http://worrydream.com/KillMath/`.

— . 2012b. *Learnable Programming*. Visited on 08/27/2016. `http://worrydream.com/LearnableProgramming/`.

— . 2013. *The Future of Programming*. `http://worrydream.com/dbx/`.

Viewpoints Research Institute. 2016. *Viewpoints Research Institute - Yoshiki Ohshima*. Visited on 09/14/2016. http://vpri.org/html/team%7B%5C_%7Dbios/yoshiki.htm.

Vujoševic-Janicic, Milena, et al. 2008. "The role of programming paradigms in the first programming courses". *The Teaching of Mathematics, XI* 2:63–83.

Warth, Alex, et al. 2016. "Language Hacking in a Live Programming Environment". In *30th European Conference on Object-Oriented Programming (ECOOP 2016)*, Workshop. Rome. https://ohmlang.github.io/pubs/live2016/.

Wexelblat, Richard L. 1980. "The Consequences of One's First Programming Language". In *Proceedings of the 3rd ACM SIGSMALL Symposium and the First SIGPC Symposium on Small Systems*, 11:52–55. SIGSMALL '80 7. New York, NY, USA: ACM. http://doi.acm.org/10.1145/800088.802823%20http://dx.doi.org/10.1002/spe.4380110709.

White, Garry, and Marcos Sivitanides. 2005. "Cognitive Differences Between Procedural Programming and Object Oriented Programming". *Information Technology and Management* 6 (4): 333–350.

Whitehead, A N. 1911. *An Introduction to Mathematics*. Home university library of modern knowledge. H. Holt.

Wittgenstein, L. 1922. *Tractatus Logico-philosophicus*. International library of psychology, philosophy, and scientific method. Harcourt, Brace, Incorporated.

Wooten, V. 2008. *The Music Lesson: A Spiritual Search for Growth Through Music*. New York, New York, USA: Berkley Books.

World Sport. 2015. *Tony Hawk on performing the first 900*. Visited on 09/16/2016. https://www.youtube.com/watch?v=x3uJNssoaQg.

Wright, Matthew, Adrian Freed, et al. 1997. "Open sound control: A new protocol for communicating with sound synthesizers". In *Proceedings of the 1997 International Computer Music Conference*, 2013:10. 8.

Ziemann, Mark, et al. 2016. "Gene name errors are widespread in the scientific literature". *Genome Biology* 17, no. 1 (): 177.

Interviews

Kay, Alan, and Arthur Carabott. 2016. *Interview / Conversation with Alan Kay*. Los Angeles.

Mettler, Mike. 2015. *Squarepusher Interview*. http://www.digitaltrends.com/music/squarepusher-interview-the-software-behind-damogen-furies/%7B%5C#%7D/2.

Ohshima, Yoshiki, and Arthur Carabott. 2016. *Squeak/Smalltalk Pair Programming Session*. Los Angeles.

Smith, Dave. 1997. *Interview: Dave Smith on MIDI*. Ed. by Eric Chasalow and Cassidy Barbara. https://www.youtube.com/watch?v=Jq6%7B%5C_%7Dvy4Pcwk.

Synthtopia. 2015. *MIDI Manufacturers Association Says New HD Protocol 'Has Reached A Milestone' - Interview with MMA President Tom White*. Visited on 09/04/2016. http://www.synthtopia.com/content/2015/01/16/new-midi-hd-protocol-has-reached-a-milestone/.

Victor, Bret. 2016b. *Interview with Bret Victor*. Ed. by Arthur Carabott. Los Angeles.

Exhibitions / Studio Visits

San Francisco Museum of Modern Art. 2016. *Typeface to Interface*. San Francisco.

Visit to Bret Victor's Research Group. 2016. Oakland.

Visit to Dan Ingalls' Research Group. 2016. San Francisco.