The Techno-Galactic Guide to Software Observation
Methods from the Techno-Galactic Software Observatory
(Brussels, June 2017)

Constant, Association for Art and Media

The Techno-Galactic Guide to Software Observation

> *I am less interested in the critical practice of reflection, of showing once-again that the emperor has no clothes, than in finding a way to diffract critical inquiry in order to make difference patterns in a more worldly way.*[1]

A spectre is haunting the galaxy. It is not revolutionary[2], it is not just another global scale vulnerability either[3]. No matter where you happen to be, with your phone or watched over by a passing satellite, software daemons are running somewhere in the background. Look! There they are, staring right back at you. Software's entanglement in the everyday appears complete, but is it really? Let's look more closely.

We find ourselves in a universe built on the 1990s revival of a 1960s dream: software as a service (SaaS)[4], a framework in which the *use* of software has increasingly been knitted into the *production* of software. While the rhetoric, rights, and procedures that proliferate in this universe suggest that software production and consumption constitute separate realms, this knitting together radically alters the operative role of software in society. The corresponding shifts ripple across galaxies, through social structures, working conditions, and trans-universal logistics. It results in a profusion of apparatuses that aspire to be seamless while optimizing and monetizing individual and collective flows of information in line with the interests of a handful of actors. The diffusion of software

---

1 Donna Haraway: *Modest Witness: Feminist Diffractions in Science Studies*. In: *The Disunity of Science: Boundaries, Contexts, and Power*. Ed. by Peter Galison and David J. Stump. 1996, pp. 428–442

2 Karl Marx and Friedrich Engels: *The communist manifesto*. Penguin, 2002

3 Jann et al. Horn: *Meltdown and Spectre Attack*. 2018

4 Thomas Haigh: *Software in the 1960s as Concept, Service, and Product*. In: *IEEE Annals of the History of Computing* 24.1 (2002), pp. 5–13

services affects individuals and communities in the form of intensified identity shaping and self-management. It also transforms the public by capturing institutions and common use infrastructures in supercharged start-up visions or the inertia of tech giants. As more and more software centralizes data flows in *cloud services*, effectively collapsing all societal spheres into the same processing logic, their service oriented architectures come to blur the last traces of the thin line that separates bio- from necro-politics. [5]

And so software spins its web around us. As it twists we turn, trying to take some distance, only to realize that we are too well tangled in its gooey net. Caught upside down and dangling [6] we ask: How can we interact, respond to, and think with software? What approaches can we use to recognize the agency of different actors, their ways of working, and their politics? What methods of observation are conducive to critical inquiry and affirmative discord? How can we resurface software and find sites where its infrastructures are reconfiguring the everyday? How can we take stock of the ways software is always at work, especially where it is designed to disappear?

Overloaded by all these questions, vexed by constant pop-ups and push notifications, a fantasy takes shape: Let's smash our devices with a sledge hammer, throw them out, retreat off grid and out of sight, and live happy software-free lives. But this fantasy cannot hold, nor can it hold us. While 'disconnection' seems to be the latest luxury-item for tech-billionaires – the same gang that profits from the extension of connectivity – for most human and non-human inhabitants of this galaxy unplugging is not an option.

---

[5] Tung-Hui Hu: *A Prehistory of the Cloud*. MIT Press, 2015

[6] `https://www.youtube.com/watch?v=NTZhrwR7CoE`

Even those that experienced the joy that comes from catapulting a smart device across the street know that the liberating feeling that follows does not last. We need a better option: one that lets us poke and push back at the daemons that insist on saturating our time and surrounding our environment.

In June 2017 we gathered with a group of Software Curious Persons (➡ P.116) in Brussels for six days at the Techno-Galactic Software Observatory[7]. There we engaged in the observation of the multiple scales of software, the industries and communities that produce it and their devastatingly relational political economies. While clutching at our computers and keyboards, we picked up the word *observation*, turning it over and around. We brought it right up to our noses and then held it outstretched to get a better look at the term's colonial and positivist legacy. Observation, and the enabling of it through intensive data-centric feedback mechanisms, is part of the cybernetic spells that underpin the present day software production-consumption complex. Holding the pain and the promise of our limitations, of our situated point of view, we synced our breath to the rhythm of our software and, with an agile sun salutation, began to explore the possibilities of engagement with software's implications. "Excuse me, do you know where to find the Techno-Galactic Walk-in Clinic?"

The Techno-Galactic Guide to Software Observation was collectively produced as an outcome of this temporary Observatory. The methods tried and tested there were collected and compiled in this guide. Although our modest acts of witnessing are incomplete and ongoing, through this kind of collective ghost-hunting we found some ways for being in, around, through, and with software.

---

7 The Techno-Galactic Software was a worksession organised by Constant. `http://constantvzw.org/site/-The-Technogalactic-Software-Observatory-.html`

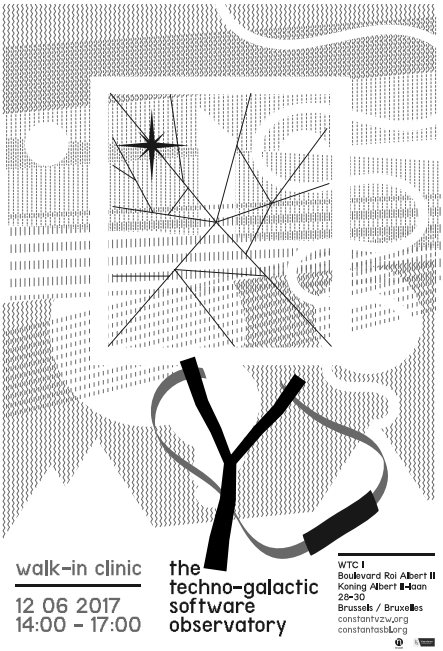
This guide invites you to experiment with ways to stay with the trouble of software. In it you will find a set of practical and impractical tools for the tactical (mis)use of software, empowering and enabling users to resist embedded paradigms and assumptions. It is a collection of methods for approaching software, experiencing its myths and realities, its risks and benefits. Drawing on theories of software and computation developed in the wilderness of academia and through rigorously undisciplined research, we grounded our methods in hands-on exercises and experiments that you can now try at home. These methods were developed in and inspired by the context of software production, hacker culture, software studies, computer science research, Free Software communities, privacy activism, and artistic practice. Exploring and cross-contaminating ways of looking from these different milieus, we realised that reading is not enough. Software is not just text, formats are always already in execution, code produces its own normative perspective and depth of field.

Observation is one potential beginning – a way to turn engagement into tickling. Come join us at the edge of the universe, on the metro platform, at your kitchen table or at your next biometric identification, as we mark and measure critical distances from and entanglements with the seemingly endless software systems that surround us.

Invitation to The Walk-in Clinic, June 2017:

Do you suffer from the disappearance of your software into the cloud, feel oppressed by unequal user privilege, or experience the torment of software-ransom of any sort? Bring your devices and interfaces to the World Trade Center! With the help of a clear and in-depth session, at the Techno-Galactic Walk-In Clinic we guarantee immediate results. The Walk-In Clinic provides free hands-on

observations to software curious people of all kinds. A wide range of professional and amateur practitioners will provide you with Software-as-a-Critique-as-a-Service on the spot. Available services range from immediate interface critique, collaborative code inspection, data dowsing, various forms of network analyses, unusability testing, identification of unknown viruses, risk assessment, opening of black-boxes and more. Free software observations provided. Last intake at 16:45.



Announcing the Walk-in Clinic. Poster design: Harrisson

● **Retrospective / Take Out**

This service verifies whether your expectations are being met. If so, we provide you with your official record and you are ready to leave the clinic.

Approximate duration: **10 minutes**

● **WTC-time**

● **Flow of the Chart**
Chart of the flow on demand!

SSoGY (SOFTWARE SKETCHING OBSERVATION YUPPIES) is here to provide you with one and only personalized flow chart & chat for your human-machine situation!

Approximate duration: **15 minutes**

● **Refreshment**

The fountain brings a soothing corporate noise to refresh any unsettlingly quiet or unhappily frenetic spaces with water.

Approximate duration: **ongoing**

● **Continuous integration**

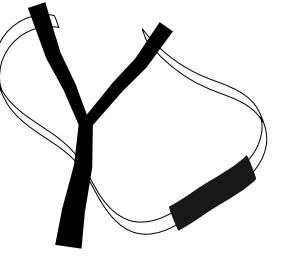
We monitor the seamless integration of all services at the software observatory clinic, including the circulation of bodies and bits.

Approximate duration: **ongoing**

● **Process invocation**

Process invocation is a service helping the user to discover and visualize processes normally kept concealed from the user.

Approximate duration: **15 minutes**



the techno-galactic software observatory

walk-in clinic

12 06 2017
14:00 - 17:00

● **Something In The Middle Maybe (SITMM)**

SomethingInTheMiddleMaybe observes the network interactions that your chosen software makes.
We ask patients what they want to observe (e.g. a daily interaction with my smartphone, or a specific application on my computer)

Patients get a log of the network connections performed by their chosen interaction.

Approximate duration: **5 to 10 minutes**

● **RSOC - Relational Software Observatory Consultancy Service**

By paying a visit to our ethnomethodology interview practice you'll be mapping perspectives to know how to observe software from different angles / perspectives. Our practioners have at heart to make possible the "what is the relation to software" discussions into a service

Approximate duration: **12 minutes**

● **Techno-galactic software walk-in clinic intake**

● **Future Blobservation Booth**

The hand which holds your mouse everyday hides many secrets. Visit the Future Blobservation Booth to have your fortunes read and derive life insight from the wisdom of software.

Approximate duration: **The approximate duration of our service will be on average 2 minutes, never less than 1 and no more than 2 minutes and 30 seconds.**

● **Interface Détournement**

Baseline of the critical service:
Tired of business jargon and unnavigable services ?
Come to do an Interface Détournement !

Motivation of the critical service:
- capitalism as sorcery (Isabelle Stengers) :
  we are under the black magic of capitalism interface making
- we have the feeling that some words are imposed to us,
  conseauently we are reclaiming a power of saying/naming

Goal/Description of a situation at hand
Methods of Observation:
-Unpacking an interface, fixed our browsing speed
-Live redesign of an interface using the browser developer tools.
-Study the layout by removing all the content
= Play with the interface content, add some humour, irony to it.

Approximate duration: **unknown**

● **File therapy**

Do you have software concerns that regard a specific file? Through this service you can experiment with various therapeutic forms. Turn your software anguish into a takeaway file.

Do you encounter software problems or do you have concerns about particular files? Through this service you can experiment with various therapeutic forms. Our therapeutic approach is inspired by the unix file system paradigm in which every component of a computer can be represented by a file, be it your hard drive, memory or sound card. Going together through the affordances and limitations of such a paradigm, we hope to provide a more intimate access to your software.
Whether it is about specific problematic situations with your computer, or to address a general curiosity about filesystems, we will take you by the hand through an extensive intake.

* First we will discuss your interest or your problem, to try and pick a file that speaks of your concern. For example, if you experience problems with connecting to the network, we can take a networkconfiguration file. Or maybe you have already a file in mind that you find concerning.
* Then you transfer the file to our dedicated usb stick and we will inspect it on our computer.
* During the inspection you will get an explanation of the various translations which happen to the file.
* On the basis of (initially) a visual inspection, we will discuss which measures should be taken. Can we intervene in the file, provide some "pixel surgery" and provide you with a restored version? Or do we need to move forward to more meditative treatments such as sound therapy?
* We hope that our service will shift your perspective on your situation by experiencing how files look like in a different environment.

Approximate duration: **15 minutes**

● **Agile Sun Salutation**

Welcome to Agile Sun Salutation, an introduction to Agile Yoga (Why Organise ? Go Agile !), a new methodology to improve your professional and personal life.

Approximate duration: **10 minutes**

● **"What is it like to be an elevator?"**

In this service you can experience and analyse a verbal live demo of a reverse engineered elevator software. There is also the possibility to re-enact the elevator software as a internalized elevator, after SCPs finished the flow chart station.

Approximate duration: **To experience some minutes! To reenact around 10 to 15 minutes!**

TECHNOGALACTIC SOFTWARE OBSERVATORY WALK-IN CLINIC

Name of the Software Curious Person .....

Situation .....

Description of situation at hand

.....

.....

.....

Directed to Services

☐ Intake

☐ Retrospection

Total time available ☐

Estimated Time for service ☐

☐ Something In The Middle Maybe (SiTMM)

☐ Relational Software observatory Consultancy Service

☐ "What is it like to be an elevator?"

☐ Interface Détournement

☐ Flow of the chart - chart of the flow on demand!

☐ Agile Sun Salutation

☐ Continuous integration

☐ Future Blobservation Booth

☐ WTC-time

☐ File therapy

What would be a outcome

.....

.....

.....

E X I T S T A T U S

NOT DONE ☐ **DONE** ☐

Can this file be published **YES** ☐ **NO** ☐

.....

SIGNATURE/DATE

STORE FOR DELETION ☐

Intake form for Software Curious People (SCP)

WARNING

The survival techniques described in the following guide are to be used at your own risk in cases of emergency as well as in cases of nonurgent software curiosities.

our methods for observation,
like mapping, come with their luggage.

The publisher will not accept any responsibility in case of damages caused by misuse, misunderstanding of instruction, or lack of curiosity. By trying the methods described in the guide, you accept responsibility for losing or loosing data and for altering hardware, including hard disks, usb key, cloud storage, and screens perhaps by throwing them on the floor, or by falling on the floor yourself upon tangling your feet in an entanglement of cables such that your laptop goes flying.

No harm has been done to humans, animals, computers or plants while creating the guide. No firearms or other kinds of weapons are needed in order to survive software. Just a little bit of patience.

Physical fitness plays a great part of software observation. Be fit or CTRL-Quit.

Remember, software observation can be stressful. You might experience:

Anxiety
Sleep deprivation
Forgetting about eating
Loss of time tracking

Can you cope with software? You just have to.

Introduction **3** Warning **11** Table of Contents **12**

14 Close encounters – Meetings between people, software and hardware :: Encounter several historical collections of hardware back-to-back **17** Interview people about their histories with software **30** `Interview with Jean Huens` **35** Ask several people from different fields and age groups the same question: "What is software?" **49** FMEM and /DEV/MEM **52** Pan/Monopsychism **57** Setup a Relational Software Observatory Consultancy (RSOC) **121** Hand reading **146**

60 Temporality – Constellations of time and locality :: Space-Noise Rearrangements **64** `I hope you are comfortable, [insert name]` **69** "Nannyware": Software that observes and addresses the user **71** Useless scroll against productivity **79** The ends of time **85** `How humans and machines negotiate the experience of time` **89**

100 Languaging – Observing software as/through writing :: Quine **103** Glossaries as an exercise **106** Adding qualifiers **112** Searching "software" through software **114** Persist in calling everyone a Software Curious Person **116**

118 Healing and embodiment – Feeling software :: Setup a Relational Software Observatory Consultancy (RSOC) **121** Agile Sun Salutation **135** `Agile Sun Salutation` **141** Hand reading **146** Bug reporting for sharing observations **150** `When dirty.db get's dirty` **153** Interface Détournement **156** Comportments of software (softwear) **162**

164 Flows – Flow-regulation, logistics and seamlessness :: Continuous Integration **167** Space-Noise Rearrangements **64** make make do **171** Flowcharts (Flow of the chart –> chart of the flow: on demand!) **175** What is it like to be AN ELEVATOR*? **196**

188 Invasive observations – Being on the side, in the middle or behind :: Something in the Middle Maybe (SitMM) **191** What is it like to be AN ELEVATOR*? **196** Side Channel Analysis **202**

206 Collections – Compiling observations :: Compiling a bestiary of software logos **208** Glossaries as an exercise **106** Encounter several historical collections of hardware back-to-back **17** Testing the testbed: testing software with observatory ambitions (SWOA) **220** Prepare a Reader to think theory with software **224** "Nannyware": Software that observes and addresses the user **71**

Keyword Index **236** Bibliography **238** Colophon **242**
Free Art License **248**

CLOSE ENCOUNTERS

Meetings between
people, software and hardware

:: Descending into the depths of any hardware reveals insights into the computer.[8]

:: Admittedly, facilitating a usability study is not a natural way to interact with other human beings. So it is totally understandable why most of us have trouble facilitating, making classic **mistakes** such as:[. . .] **Treat the test session as a conversation rather than an observation**. Talking too much, at inappropriate times, or leading the user can affect what he does and says, which can invalidate part or all of the research findings. Interviewing methods are different from observational methods.[9]

:: [T]he ENIAC's "master programmer" was not a person, but a machine component, responsible for executing loops and linking sequences together. That is, the master programmer handled the "program control" signal that each unit produced after it successfully executed a function.[10]

---

[8] David A. Patterson and John L. Hennessy: *Computer Organization and Design MIPS Edition, Fifth Edition: The Hardware/Software Interface*. English. 5 edition. Amsterdam ; Boston: Morgan Kaufmann, Oct. 2013. ISBN: 978-0-12-407726-3, pg.19

[9] Kara Pernice: *Talking with Users in a Usability Test*. en. 2014. Visited on Jan. 31, 2018

[10] Wendy Hui Kyong Chun: *Programmability*. In: *Software Studies*. Ed. by Matthew Fuller. 2008, pp. 225–228. ISBN: 978-0-262-06274-9. Visited on Jan. 31, 2018, pg.225

> I participated in the hackathon much in the spirit of a critique that had run out of steam. I concerned myself to coconstruct knowledge — to make a thing. Although the hackathon did draw people together in a Latourian spirit around a matter of concern, in a very Latourian spirit, this actually existing site of design practice revealed that its politics were in its forms and norms — in its manufactured urgency, in the distance between the studio and the world, and the media ecologies that made it possible to promise to cross that distance without walking it.[11]

---

[11] Lilly Irani: *Hackathons and the Making of Entrepreneurial Citizenship*. en. In: *Science, Technology, & Human Values* 40.5 (Sept. 2015), pp. 799–824. ISSN: 0162-2439, 1552-8251. DOI: 10.1177/0162243915578486. Visited on Jan. 31, 2018, pg.20

METHOD::

Encounter several historical collections of hardware back-to-back

HOW:: This can be done by identifying one or more computer museums and visiting them with little time in-between. Visiting a friend with a large basement and lots of left-over computer equipment can have a similar effect. Seeing and possibly touching hardware from different contexts (state-administration, business, research, . . .), periods of time, cultural contexts (California, Germany, French-speaking Belgium) and price ranges allows you to sense the interactions between hardware and software development.

NOTE:: This method offers a perfect opportunity to hear people speak about objects in their contexts, how they worked and did not work and how particular pieces of hardware and software are linked one with another. It also shows the economic and cultural aspects of softwares.

WARNING:: **DO NOT FOLD, SPINDLE OR MUTILATE**

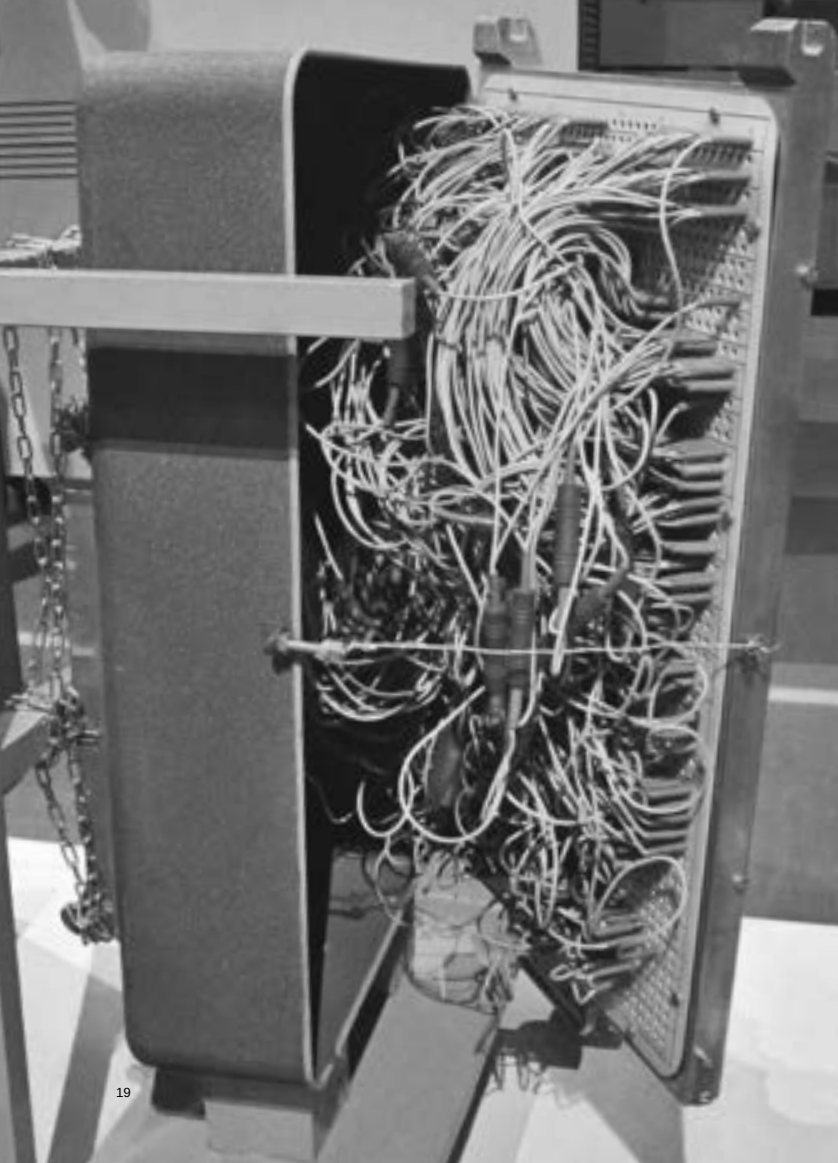
EXAMPLE:: Spaghetti Suitcase

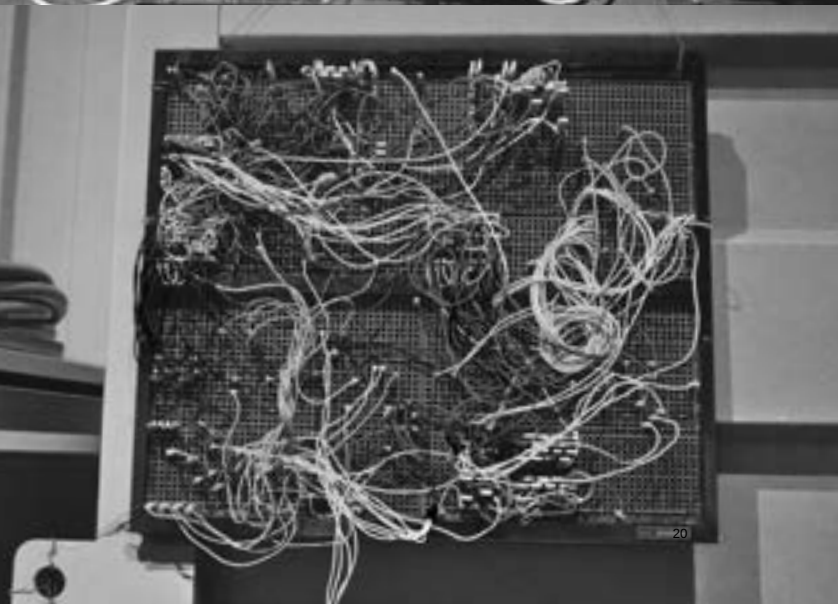
At one point during the demonstration of a Bull computer at the Computer Museum NAM-IP, our museum guide revealed the system's "software" – a suitcase sized module with dozens of patch cords. She made the comment that the term "spaghetti code" (a derogatory expression for early code that used many "GOTO" statments) had its origin in this physical arrangement of code as patchings.

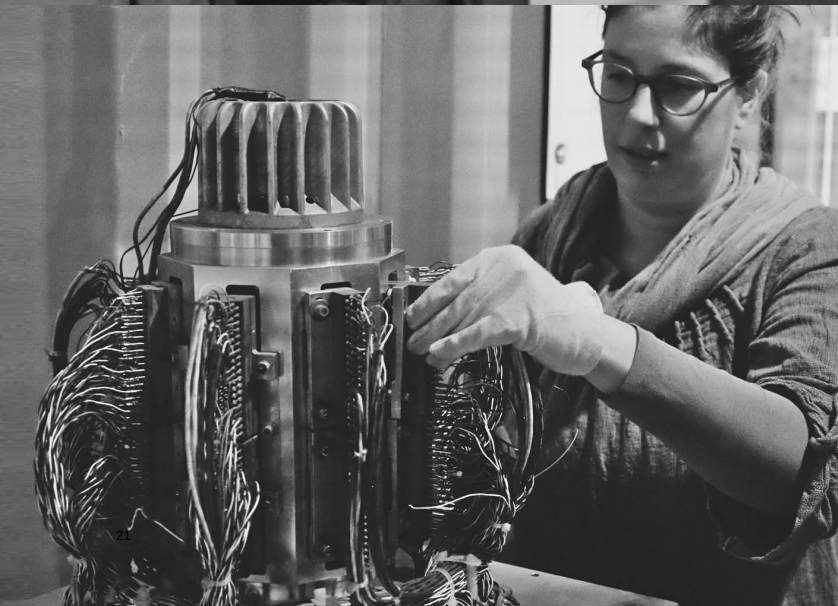
Preserving old hardware is a means of observing the physical manifestation of software. Operating older computers can result in actually touching software.

EXAMPLE:: Play with binary. Create punch cards.

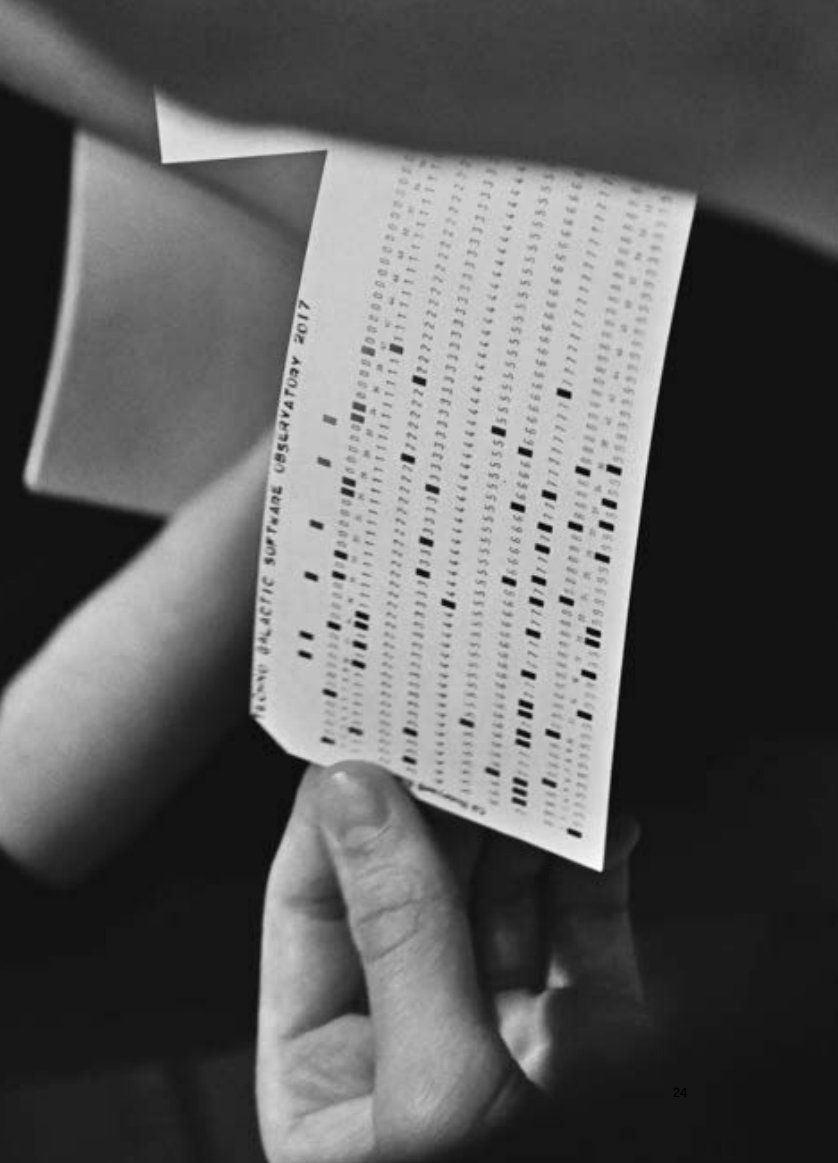
"The highlight of the collection is a recreation of a real punch card workshop of the 1960s."







La table des
manipulations


GALACTIC SOFTWARE OBSERVATORY 2017

ORDINATEUR GAMMA 30 – 1963
Bull

Code alphanumérique BULL
(ANT)
11 0 1 2 3 4 5 6
7 8 9 Plus
7 A B C D E F G H
8 I J K L M N P Q R
9 S T U V W X Y Z
. 0 1 2 3 4 5 6

EXAMPLE:: Collection de la Maison des Écritures d'Informatique & Bible, Maredsous

The Collection de la Maison des Écritures d'Informatique & Bible was donated to NAM-IP by the *Centre Informatique et Bible* that was founded in 1979 at the Benedictine abbey of Maredsous. The center was located in the "House of Scripture", a pavilion near the entrance to the abbey, where biblical scholars, linguists and computer scientists applied computer processing to bible-texts and it's references.

The idea of introducing informatics as a method for working with and on the Bible dates back to 1971 when it was done via punch cards and then transferred to the memory of magnetic tape. Then came the step of analyzing texts using computers while at the same time introducing computers to the delights of reading the Bible with the help of Optical Character Recognition. The particularity of this collection lies in the fact that it conserves and presents multiple moments in the life of a text rendered into and analyzed through software, starting from its initial stages of computerization and going up to contemporary forms of software.

SEE ALSO::
Interview people about their histories with software
➡ P.30

Maredsous 89.
15 ans d'Informatique et Bible
Au commencement était le Verbe...

LEXIQUE FREQ GREC Λ-Ρ
LEXIQUE FREQ GREC Σ-Ω
INTERFACE
1980 – 2005
25 ans
d'Informatique & Bible
Informatique & Bible
Mandaloue

SOURCE::

Machines à cartes perforées de marque Bull ➡ http://www.histoireinform.com/Histoire/+Infos/jmclcadr.htm

IMAGES:: Spaghetti Codes **P.19** Museum guide at NAM-IP touching software. **P.21** “À la table des manipulations”, manually creating punch-cards with the EMPUNCH model 80. **P.22** TGSO Punchcards **P.24** Atelier mécanographique Bull **P.25** ‘Maredsous 89. 15 ans d’Informatique et Bible’ (‘Maredsous 89. 15 years of informatics and bible’) **P.27** Continuous form paper with lexical research and a copy of the *Interface Bulletin*, published by the Centre Informatique et Bible: http://www.cibmaredsous.be/cib3000.htm **P.28**

METHOD::

Interview people about their histories with software

WHAT:: Collect personal narratives around software history. Retrace the path of an individual's relation to software, how it changed during the years, and what human access memories surround it. Look at software through personal relations and emotions.

HOW:: Interviews are a good way to do it, but informal conversations work, too.

WHEN:: Whenever (talking to people who are retired is a plus).

WHO:: Anyone with ten years or more of any kind of experience with software.

URGENCY:: High.

WARNING:: Oral histories will be lost if they are not recorded.

EXAMPLE::

Jean Heuns has been collecting servers, calculators, softwares, magnetic tapes and hard disks for thirty years. He came to an agreement with the Department of Computer Sciences (KU Leuven) for them to be displayed in the department hallways.

At the time computers were mainframes and you did feed them with programs by punch cards. Programming was writing down the program, then you punched it, and the next day you get results, mostly error. That's the way it worked at that time. To teach us programming, they used the language of that time, Fortran, and some kind of invented assembly langage, it didn't really exist, it was simplified.

As an anecdocte, I remember punchcards were a block of paper, we were jealous of the people who came into the computer center with huge stacks of cards. They were really small programs, not complicated ones. In fact you had to learn to use the computer from scratch, everything was new, you couldn't rely on previous experiences. And to start with small programs was difficult enough.

I remember the first program I had to work on. You got three numbers, you had to make the program decide if you can construct a triangle with the numbers being the sides of the triangle.

Dysan

How did you see the rise of personal computer and the software that goes with it?

I have to admit that we missed it. We didn't see that what was new was powerful. We were not impressed by the software. I remember Windows 95, the news on TV, people sleeping on the street to get it. I remember when I got my first Windows version, with a box of 22 floppies or so. Then I wanted to do something, to compile a program, but there was no compiler.

For me software is the magic that makes computers usable.

SOURCE!!

Quotes taken from an interview with Jan Huens, see following pages.

Interview with Jean Huens

The following is a transcript of an interview with Jean Huens conducted by Peggy Pierrot on December 21, 2017 in an office of Departement Computerwetenschappen of the KU Leuven.

You have been using computers since a long time, as you explained to us during the tour we had the joy of attending during the TGSO. I heard that until recently, let's say the late 1990s, that to learn computer programming you had to start by writing down the program on paper?

In 1973, I did study computer sciences, it was the second year of the Computer Science course, here in Leuven. At the time computers were mainframes and you did feed them with programs by punch cards. Programming was writing down the program, then you punched it and the next day you get results, mostly error. That's the way it worked at that time. To teach us programming, they used the language of that time, Fortran and some kind of invented assembly language, it didn't really exist, it was simplified.

Most of the time you wrote down your program, the teacher and the assistant would have to tell you what is wrong or good, you didn't have much practical experience.

What was it like, tons of pages?

No. As an anecdocte, I remember punchcards came in huge stacks, we were jealous of the people who came into the computer center with piles of cards. They were really small programs, not complicated one. In fact you had to learn to use the computer from scratch, everything was new, you couldn't rely on previous experiences. And to start with small programs was difficult enough.

That was also the time when Edsger W. Dijkstra, who fought against the *goto* and other common but bad constructs.[12] I still remember him, in an audience, giving us a lecture on good techniques in programming at the International computing symposium 1977 in Liège.[13] It was a rather hard course because Dijkstra expected you to quickly understand and implement his ideas.

And then from ’75 on, computers became cheaper and could be bought by departments and smaller entities. And then came the first interactive computer. They were still beasts, about two cubic meters, costing two millions belgian francs. But it was cheap enough to allow people to type in directly commands. I started working here in ’75 as a researcher, it has evolved continuously and I saw computers getting cheaper and getting easier to use.The way to use it was simplified, you could start to go to the beast and try it out.

I remember more or less: you wrote down the program in Fortran for instance. We always thought in the beginning that it would work and it didn’t, so after a turn around you were giving it back to the operator and you have to wait another half a day or a night to get the results.

What kind of program were you working on?

I remember the first program I had to write was: you got three numbers and you had to make the program decide if you could construct a triangle with the numbers being the sides of the triangle.

---

[12] Edsger W. Dijkstra: *A Case against the GOTO Statement*. 1972
[13] Edsger W. Dijkstra: *Programming: From craft to scientific discipline*. 1977

That took an afternoon to be processed?

Well, yeah. You had to imagine how to do it which was also one problem, then you prepare the cards with the punching machine and you give the package of cards to the operator who was feeding the computer with the cards. Operators were people who had the task of feeding the computers by hand, with paper cards, and then the cards got written on disks and later processed by the mainframe from disk. Finally results got printed on paper. And after a number of hours or a night you got a listing on a piece of paper with the results or program compilation errors.

When where you able to record the program onto the machine?

I think that happened between ’75 and ’80. After the punching cards, which were already a way to record the program, we had typewriters and then terminals. We could record the programs by typing it on the terminal and the computer would store them on magnetic discs. IBM had number for everything, I remember the 3300 [manufactured from 1970 on] was a magnetic disk. We didn’t touch the machines, operators would change the disks, we could only access the typewriter/terminal. That was our most direct connection to the computer. The times were different. The mainframes were such huge costly beasts that it had operators day and night to keep it busy. Even at night there were two or three guys mounting tapes, disc mounting tapes, that kind of stuff (➡ P.38).

When did you stop working with operators?

Somewhere between ’85 and ’90. There were still operators at the main computer of the university, but there were also small computers, so researchers and student could have direct access when needed.

3D CG concept of an IBM 3330 Direct Access Storage Facility by Oliver Obi. https://commons.wikimedia.org/wiki/File:IBM_magnetic_disk_drives_3330%2B3333.png

When did you start to work with software sold by others or given away in order to do your research?

You still needed an operating system for the mainframe that was delivered by IBM or the constructor delivered the mainframe. When you bought mini-computers, which were the smaller versions of mainframes, in most cases you bought an operating systems, compilers and all the stuff you needed to be able to program or work those things. In most cases, the computer sciences department bought the mini-computer from, for instance *Digital Equipment Corporation*, a manufacturer at that time, but they didn't buy DEC's software. The DEC software was not open source or free, and as a university we could get Unix for free and could run it on those DEC computers. That is what the computer department of the University of Louvain la Neuve *l'unité d'informatique*, did and so did we.

When did you start working with Unix as an OS?

I took a trip to Amsterdam, it must have been in 1979 or 1980. Well, the story starts in our French sister university (LLN) where they had a PDP-11 with Unix on it. You could get Unix, at the time, when you were a university, from Bell Labs (part of AT&T), the American telephone company. AT&T was a mega organisation at the time before the dismantlement of AT&T/Bell system[14]. Due to monopoly, the US authorities split it up into different Baby Bells. One of the restrictions was that the research Baby Bell, could not sell anything. There were computer scientists there who developed some interesting things (e.g. Unix). Universities got the source code for free, but NO support, nothing not even how to

[14] `https://en.wikipedia.org/wiki/Breakup_of_the_Bell_System`

install it. In Unité d'informatique, where they had a PDP-11/70[15] they were lucky: they got the same computer that Unix was developed on. They got the disk by snail mail as a post-office package. They mounted it and it did spin up and they had Unix. Here, we had a cheaper PDP11, the PDP 11/60[16]. We got the Unix disk distribution from Bell Labs but we could not run it, because we got another type of disk (RK07). We could look at it but that was all, because our disc format was different than the one at Bell Labs. Somebody here (P. Verbaeten) wrote a driver for that disk in l'unité d'informatique of LLN. It compiled it but we couldn't try it because we did not have a RKO7. The nearest place with Unix and a RK07 was in Vrije Universiteit Amsterdam. Andrew Tenenbaum, who wrote a number of books on operating systems, was working there. He was learning to speak Dutch. We understood his English better but his assistant explained that he needed to practice speaking Dutch because he would have to pass an exam. There we compiled Unix with an RK07 driver and stored it on a bootable RK07 disk. With that disk back in Leuven we could start Unix on our PDP11/60. An error in the process meant a trip Leuven-Amsterdam and back. In my opinion the start of Open Source was Unix because we had to maintain it by ourselves and we learned a lot from the sources. In that time for software distribution you had to store it on magnetic tape and send the tape to different places with associated problems. For example, the agents at the customs at the airport or post office didn't know what to do with the tapes or discs in terms of custom clearance. They used magnetic or x-rays as inspection tools and we had to write on the package that it should not be x-rayed. And sometimes we still got unreadable tapes/discs.

---

[15] `https://i0.wp.com/www.utterpower.com/wp-content/uploads/2011/10/pdp11-70-panel.jpg`

[16] `https://dave.cheney.net/tag/pdp-11`

So it took weeks between the transportation and the customs?

Yes, the custom had to clear the packages. I remember I've been in the customs in Zaventem. I had received a letter that my package from Bell Labs had arrived. I went there thinking I would have to pay something and go. But no. It had to be cleared so there was some organisation that had to do administrative stuff, so I only got it a few days later. The response time was huge.

After that period of time, how did you switch from that kind of collaboration on how to make software and make software work, to another kind of collaboration on networks and between universities and researchers?

In the Unix systems at that time there was a kind of networking protocol which worked on serial lines (UUCP). To connect two Unix systems together you only need a serial line. We had a line between LLN and Leuven, it's only 25 kms, but we had to ask Belgacom, then La Régie des Télégraphes et Téléphones, to get a modem to connect through their network and then over that modem you could reach an amazing speed of 300 characters per second. And that was the first communication. It happened somewhere around 1985. And then you had connection with another computer by writing something here and doing the transfer to the other computer and executing it there.

What was the first thing you sent through a network like this?

I didn't do it myself, I remember it was a kind of chat program. On the LLN side was Professor Elie Milgrom. He is retired now. On this side, it was Professor Yves Willems. And I remember that they typed something and they got a response from the other side and that was it. But I remember that Professor Willems sent "Ok,

we stop now.” And that Elie Milgrom answered “See you later, Alligator” [a reference to Bill Haley and the Comets]. At the time we still had these kind of songs in mind. (lol). Those Unix to Unix connexions were first. And there was a worldwide Unix to Unix based network, Usenet (1979). It still exists somewhere.You can download loads of illegal movies and that stuff. That was not what we meant at the time. But that is what is left. And Usenet became gradually replaced by the Internet in USA. In Europe, the Internet came a lot later, but we had limited access to the Internet by mail and even FTP; We could send mail to the Internet and we could receive mail from them because some friendly American university (and later companies) translated internet mail to UCP mail. Moreover they translated specially formatted mails into FTP request and sent the result back to us by mail. That was before the 1990’s.

Your relation to software had changed at the time because you did not have to do all this manipulations with the software your received. Was it already floppy discs?

No, magnetic tapes, for the mainframe at least. Even most of the mini-computers used some (cheaper) magnetic tapes. Between 1990 and 1995 the main machines used for Unix software development was a VAX from DEC. Once a year the software came from Berkeley. It was still Unix but had been modified by Berkeley for virtual memory and all the stuff which is common now. We received a tape, with lot of goodies, I still remember we didn’t know what was on the tape but we scrutinized it to find out what was on there, what we could use… In Europe, companies like Siemens were trying to set up software centers to make software avaiable. They were even trying to set up European networks. But researchers loved the Internet. It worked. It was useful for their research. So companies in Europe were trying to set up their own networks, I remember for example the X/OPEN transport protocol,

which was a set of standards about what software layers should do. But it didn't succeed because around 1992 IBM decided to quit their own private network and to go for TCP/IP which was the switching point. Everything done for X/OPEN were suddently forgotten. For the mainframes, and certainly in Belgium computing was mainly mainframes, I'm talking about universities here, so when IBM decided to go for TCP/IP the universities followed. But the adoption of TCP/IP was really slow. For example, Belgacom was not interested in running the Domain Name System so we did it here, for some time, starting in 1993 with Pierre Verbaten[17]. Belgacom were focused on X25 (DCS - `https://en.wikipedia.org/wiki/X.25`) so they missed it.

It was like the french Minitel?

Well, at that time we were really jealous about Minitel. Could you imagine that? Terminals? At home? You could work home with a computer! You could send emails! All in one, a PC computer with a modem!

How did the research about software evolve in the department?

I am mainly talking about network research but here. There were people working on numerical analysis, computing, artificial intelligence. I was just not involved in that research so I don't know much about it

---

[17] `https://www.dnsbelgium.be/en/history`

You were focused on software and networks?

Yes, we produced some software. We made software to generate compilers. There was some A.I. software in Prolog, numerical analysis libraries, etc. Some were sold or shared, it depends. We sold some software from the beginning I think when it was not even a department but a professor and a couple of students.

Is that when you joined?

No it was already a department. I studied industrial engeneering in electronics first. At the time in electronics you heard something about computers, I was interested in it. I wanted to know more. I heard KU Leuven had started a degree in computer engineering. It was the first year where industrial engineers could join. The only thing we knew about computers were from movies. I wanted to know how they worked. I'm not sure if I know now it now, but I have some ideas.

How many languages do you know?

Assembly, FORTRAN, PL-1 the language invented by IBM to be the universal one, C, Cobol. Perl... I don't know... some more. I'm not fluent in any computer language, but I can read any language especially well written, I know I can understand it and, if needed, learn it... I know JAVA also. As a system administrator, the thing I use most are the command line of Unix and Perl. But that's because i'm more familiar with those two so it goes faster.

How did you see the rise of personal computer and the software that goes with?

I have to admit that we missed it. We didn't see the new thing was powerful because it was available for everybody. We were not impressed by the software. I remember Windows 95, the news on TV, people sleeping on the street to get it. I remember when I got my first Windows version, with a box of 22 floppies or so. Then I wanted to do something, to compile a program, but there was no compiler. We were used to Unix, which was a basic system but haa lot of utilities in it. Software was for us free, and we were used to having it be free, and on a PC it was not free anymore.

Does that mean you didn't have a personal computer until late?

Yes. I think it was round 2000. I didn't need it really. I got one when the Belgian UUCP network became commercial. Of course I did not get Windows on it. I think we got Linux here in 1994. There was a guy involved in the kernel studying here. So we did used it very fast.

So, what is software for you?

For me, software is the magic that makes computer usable. I can only give examples of what I mean by it. A PC without software – most people, even computer scientists, can't do anything with it. You need basic software. Some software is already integrated in the computer hardware. It is the same for phones, tablets. The main characteristic, for me, is that it should be user friendly. That is very difficult I know, because you have to imagine what the people using your software are going to do with it and how they want to use it. That's the magic you need to solve.

Is command line user friendly?

Well, somehow you have to learn the Unix command language; it is in fact a computer language where you can make all the constructs that other languages offers. You can create loops, you can test, whatever. It's not so different from computer languages like C or JAVA or whatever. It is certainly programmer friendly.

But it's already software, no? User friendly depends on what you put behind these words, no?

Yes it's already software. I don't have a strong opinion about command lines. It depends on people. I can imagine that someone that has never heard about the command line expects that you click somewhere and it happens. But if you know the command language it's sometimes perhaps easier or productive to type commands and let it run. Sometimes, I find it easier if I have a screen and a number of buttons and I don't have to imagine what happens behind that screen. And sometimes I want to do something with the command line, because I'm used to it. That's why I don't trust myself about user friendliness because I have a long experience in computer interfaces. I'm so used to some of them perhaps I can't even imagine that somebody else would like to do it otherwise or doesn't like what I'm used to.

Apart from maintaining the museum, what do you work on? You are still active in the department even though you are retired?

I work on sensors, in the Internet of Things world in one of the university spin-offs. What I do for them is: wireless sensors need power, most of them are on batteries. They have designed a complete system which should make the battery last for two, three, four years. They need a system to measure that. So I have been

working on a mechanism to measure that without waiting four years. That's programming instrumentation computers to measure e.g. power or whatever. Most of it on Unix systems. There are enough new or unknown things and maybe because I'm older, I have another way of looking at things, so we have a lot of discussions. They don't like my Perl, they use Python, JAVA and those things... Perl, it has its disadvantages, but you can do a lot of things. Every character has a meaning, that makes Perl programs often unreadable. With Python, you need all those libraries... It's easy when you know the libraries, what you have to call. When you know the arguments of the librairies. So it's also complicated. And if the library changes for some reasons, there you go. In my humble opinion it's the same with all the languages. It starts easy from scratch but if you have to interface them with existing things from the Internet world or whatever it gets complicated.

Do you have a last word?

I was surprised by your group reaction to the visit. I thought all of this was known. Maybe I'm getting old. Currently I don't do much about the museum department, but sometimes I like to talk about the little stories about it. I remember a lot of little stories, a lot of techniques that have been used and forgotten, and some techniques that have been kept alive. I thought when I was young that the best techniques would succeed. That you would have visionary people who would see the way. But I realized that a lot of things happen by accident.

METHOD::

Ask several people from different fields and age groups the same question: "***What is software?***"

WHAT:: By paying close attention to the answers, and possibly logging them, observations about the ambiguous place and nature of software can be made.

R E M E M B E R

The answer to this question will vary depending on who is asking it to whom.

EXAMPLE::

"*It is difficult to answer the question 'what is software', but I know what is good software*" [18]

"*Software is a list of sequential instructions! Hardware for me is made of silicon, software a sequence of bits in a file. But naturally I am biased: I'm a hardware designer so I like to consider it as unique and special*". [19]

"*This, you have to ask the specialists.*" [20]

```
*what is software?
──the unix filesystem says: it's a file
────what is a file?
────in the filesystem, if you ask xxd:
────── it's a set of hexadecimal bytes
───────what are hexadecimal bytes?
────── –b it's a set of binary 01s
────if you ask objdump
───────it's a set of instructions
──side channel researching also says:
────it's a set of instructions
──the computer glossary says:
────it's a computer's program, plus the procedure for their use
   http://etherbox.local/home/pi/video/A_Computer_Glossary.webm
  #t=02:26
────── a computer's programs is a set of instructions for
  performing computer operations
```

---

[18] Jean Huens (system administrator at the department of Computer Science, KULeuven)

[19] Thomas Cnudde (hardware designer at ESAT - COSIC, Computer Security and Industrial Cryptography, KULeuven)

[20] Amal Mahious (Director of NAM-IP, Namur)

SOURCE::

Notes on fileflowchart.raw.html ➡ `http://observatory.constantvzw.org/etherdump/fileflowchart.raw.html`

SOURCE::

Notes on Multiple Software Axes ➡ `http://observatory.constantvzw.org/etherdump/multiple-software-axes.md.raw.html`

SEE ALSO::

Persist in calling everyone a Software Curious Person

➡ P.116

METHOD::

FMEM and /DEV/MEM

WHAT:: Different ways of exploring your system memory (RAM). As in unix-based systems everything can be approached as a file, you can access your memory as if it were a file.

URGENCY:: Observing the operational level of software, getting closer to its workings. Examining the instruction-being of an executable/executing file, the way it is when it is loaded into memory rather than when it sits in the harddisk.

R E M E M B E R

In Unix-like operating systems, a device file or special file is an interface for a device driver that appears in a file system as if it were an ordinary file. In the early days you could fully access your memory via the memory device (`/dev/mem`) but over time the access was more and more restricted in order to avoid malicious processes from directly accessing the kernel memory. The kernel option CONFIG_STRICT_DEVMEM was introduced in kernel version 2.6 and upper (2.6.36–2.6.39, 3.0–3.8, 3.8+HEAD). So you'll need to use the Linux kernel module fmem: this module creates `/dev/fmem` device, that can be used for accessing physical memory without the limits of /dev/mem (1MB/1GB, depending on the distribution).

`/dev/fmem` tools to explore processes stored in the memory

```
ps ax | grep process
cd /proc/numberoftheprocess
cat maps
```

–> check what it is using

The proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at `/proc`. Most of it is read-only, but some files allow kernel variables to be changed.

dump to a file–>change something in the file–>dump new to a file–>diff oldfile newfile

"where am i?"

to find read/write memory addresses of a certain process

```
awk -F "-| " '$3 ~ /rw/ { print $1 " " $2}' /proc/PID/maps
```

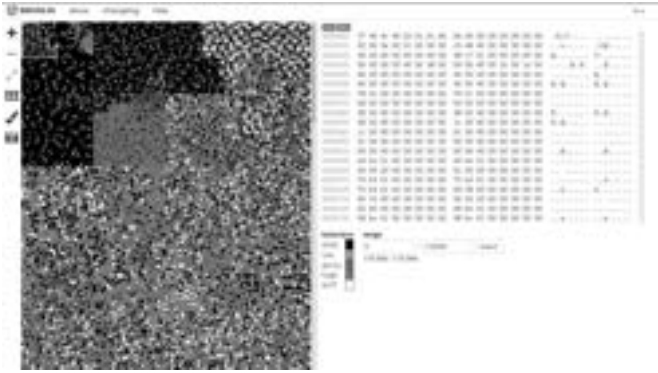
take the range and drop it to hexdump

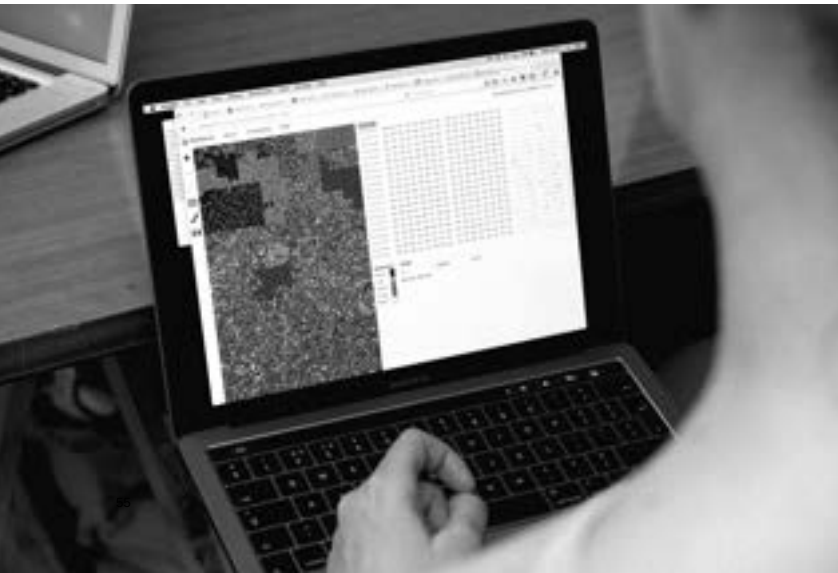
```
sudo dd if=/dev/fmem bs=1 skip=$(( 16#b7526000 - 1 )) \
        count=$(( 16#b7528000 - 16#7b7526000 + 1)) | hexdump -C
```

SOURCE::

http://observatory.constantvzw.org/etherdump/files.md 511-535

Besides opening the memory dump with a hex editor you can also try to explore it with other tools or devices. You can open it as a raw image, you can play it as a sound or perhaps send it directly to your frame-buffer device (`/dev/fb0`).





WARNING:: Although your memory may look like/sound like/read like gibberish, it may contain sensitive information about you and your computer!

Forensic and debugging tools can be used to explore and problematize the layers of abstraction of computing.

SEE ALSO::
Pan/Monopsychism
➡ P.57

SOURCE::

Notes on how to observe files ➡ `http://observatory.constantvzw.org/etherdump/files.html`

METHOD::

Pan/Monopsychism

WHAT:: Reading and writing sectors of memory from/to different computers.

URGENCY:: Memory, even when it is volatile, is a trace of the processes happening in your computer in the form of saved information, and is therefore more similar to a file than to a process. Challenging the file/process divide, sharing memory with others will allow a more intimate relation with your and other's computers.

ABOUT:: Monopsychism is the philosophical/theological doctrine according to which there exists but one intellect/soul, shared by all beings.

SEE ALSO::
FMEM and /DEV/MEM
➡ P.52

NOTE::The parallel allocation and observation of the same memory sector in two different computers is in a sense the opposite process of machine virtualization, where the localization of multiple virtual machines in one physical computers can only happen by rigidly separating the memory sectors dedicated to the different virtual machines.

WARNING:: THIS METHOD HAS NOT BEEN TESTED. IT CAN PROBABLY DAMAGE YOUR RAM MEMORY AND/OR COMPUTER.

HOW::

First start the fmem kernel module in both computers:

```
sudo sh fmem/run.sh
```

Then load part of your computer memory into the other computer via dd and ssh:

```
dd if=/dev/fmem bs=1 skip=1000000 count=1000 | \
ssh user@othercomputer dd of=/dev/fmem
```

Or viceversa, load part of another computer's memory into yours:

```
ssh user@othercomputer dd if=/dev/fmem bs=1 skip=1000000 count=1000 | \
                       dd of=/dev/fmem
```

Or even, exchange memory between two other computers:

```
ssh user@firstcomputer dd if=/dev/fmem bs=1 skip=1000000 count=1000 | \
ssh user@secondcomputer dd of=/dev/fmem
```

pan/monopsychism:
(aquinas famously opposed averroes..**who**'s philosophy can be interpreted **as** monopsychist)

shared memory

copying the same memory to different computers

https://en.wikipedia.org/wiki/Reflection_%28computer_programming%29

it could **cut** through the memory like a worm

or it could go through the memory of different computers one after the other and take and leave something there

SOURCE:

Note-snippets and code speculations during the TGSO-worksession: http://observatory.constantvzw.org/etherdump/files.md.diff.html

TEMPORALITY

Constellations of time and locality

> [T]he railroad, which worked a revolutionary change in civilization. It increased the tempo of business activities [. . .] The foundation on which our business is built is the saving of time for all people and all industries throughout the world, to give more time in which to do given tasks, and to make available more time for still further advancement and progress. The railroads and [IBM] [. . .] have a common mission—both function to increase the profi ts and accelerate the progress of business by conserving the most precious of all commodities — T I M E."[21]

> The concept of time sharing was developed in the late 1950s, mainly motivated by the aim to make efficient use of expensive mainframe computers by avoiding idle times. Time sharing refers to the (seemingly) simultaneous access of multiple users that are connected via terminals to a central computer, technically based on the flexible allocation of CPU time to concurrent user processes. The first experimental implementation, the Compatible Time Sharing System (CTSS), was deployed at the MIT in 1961 on an IBM 709 computer,[22]

---

[21] John Harwood: *The Interface: IBM and the Transformation of Corporate Design, 1945–1976*. English. 1 edition. Minneapolis, MN: Univ Of Minnesota Press, Nov. 2011. ISBN: 978-0-8166-7039-0, pg.102

[22] Christoph Neubert: *The Tail on the Hardware Dog*. English. In: *There is no Software, there are just Services*. Ed. by Irina Kaldrack and Martina Leeker. Lüneburg, 2015, pp. 21 –37. ISBN: 978-3-95796-055-9, pg.25

:: [M]achines have their natural cycle: the vibrating pulses of its internal clock drives the cycles according to which the processor works. The time of the computer is linked to this clock cycle, as it consists in counting cyclical ticks. This produces a cyclical time rhythm in the hardware, on which time experience in the software is based. However, time in a computer is no unique or unified experience. Several hardware components and a diverse collection of software organised in layers and processes create a whole ecology of interdependent time experiences.[23]

:: *Real time* is defined as time measured from some fixed point, either from a standard point in the past (see the description of the Epoch and calendar time below), or from some point (e.g., the start) in the life of a process (elapsed time). *Process time* is defined as the amount of CPU time used by a process. This is sometimes divided into user and system components. *User CPU time* is the time spent executing code in user mode. *System CPU time* is the time spent by the kernel executing in system mode on behalf of the process (e.g., executing system calls). The **time**[24] command

---

[23] Hans Lammerant: *How Humans and Machines negotiate the Experience of Time*. 2017

[24] `https://linux.die.net/man/1/time`

can be used to determine the amount of CPU time consumed during the execution of a program. A program can determine the amount of CPU time it has consumed using **times**[25], getrusage[26], or clock[27].[28]

[25] https://linux.die.net/man/2/times

[26] https://linux.die.net/man/2/getrusage

[27] https://linux.die.net/man/3/clock

[28] time was written by David MacKenzie. The man page was added by DirkEddelbuettel: *TIME(1) General Commands Manual*

METHOD::

Space-Noise Rearrangements

WHAT:: Interventions in your working-environment.

HOW:: Different strategies can be applied to temporarily redefine the workspace and its perceptual structures.

URGENCY:: Acknowledging space and its correlated noise, as conditioning observations (the World Trade Center vs. Museum vs. University vs. Startup Office vs. Shifting Walls that became Water Fountains).

NOTE:: EU-OSHA (European Agency for Safety and Health at Work) Directive 2003/10/EC describes the minimum health and safety requirements regarding the exposure of workers to the risks arising from physical agents (noise). However no current European guidelines exist on the potential benefitial uses of tactially designed additive noise systems.

EXAMPLE:: Fountain refreshment: augmenting a piece of standardized office equipment designed to dispense water to perform a decorative and soothing function.

Actual silence is not at the moment considered comfortable. One of the visible symptoms of our desire to take the edge off the silence can be observed through the appearance of fountains in public space. The fountain's purpose is to give off neutral sound, like white noise without the negative connotations.

NOTE:: Gaining access to standardized water dispensing equipment turned out to be more difficult than expected as such equipment is typically licensed/rented rather than purchased outright. Acquiring a unit that could be modified required access to secondary markets of second hand office equiment in order to purchase a disused model.

EXAMPLE:: One-way mirrors can be used to partition your working environment in a 4-dimensional way.

As the foil reacts to light, it appears transparent to someone standing in the dark, while leaving the side with the most light with an opaque surface. Using this foil as room dividers in a room with a changing light, what is hidden or visible will vary throughout the day. So will the need for comfortable silence.

SOURCE::

These two examples of *Space-Noise Rearrangements* were developed by Mia Melvaer ➡ `http://www.miamelvaer.com/technogalactic` .

I HOPE YOU
ARE COMFORTABLE,
...SARAH....

I hope you are comfortable, [insert name]

For the past 100 years the western ideal of a corporate landscape has been moving like a pendulum, oscillating between grids of cubicles and organic, open landscapes, in a nearly perfect 25-year rhythm. These days the changes in office organization is supplemented by sound design in corporate settings mostly to create comfortable silence. Increase the sound and the space becomes more intimate, the person at the table next to you can not immediately hear what you are saying. It seems that actual silence in public and corporate spaces has not been sought after since the start of the 20th century. Actual silence is not at the moment considered comfortable. One of the visible symptoms of our desire to take the edge off the silence is to be observed through the appearance of fountains in public space. The fountain's purpose is to give off neutral sound, like white noise without the negative connotations. However as a sound engineer's definition of noise is unwanted sound that all depends on one's personal relation to the sound of dripping water.

This means that there needs to be a consistent inoffensiveness to create comfortable silence.

In corporate architecture the arrival of glass buildings were originally seen as a symbol of transparency, especially loved by governmental buildings. Yet the reflectiveness of this shiny surface once combined with strong light – known as the treason of the glass – was only completely embraced at the invention of one-way-mirror foil. And it was the corporate business-world that would come to be known for their reflective glass skyscrapers. As the foil reacts to light, it appears transparent to someone standing in the dark, while leaving the side with the most light with an opaque surface. Using this foil as room dividers in a room with a changing light, what is hidden or visible will vary throughout the day. So will the need for comfortable silence.

Disclaimer :
Similar to the last 100 years of western office organisation,
this fountain only has two modes:
on or off

If it is on it also offers two options:
cold water and hot water

This fountain has been tampered with and has not in any way been approved by a professional fountain cleaner. I do urge you to consider this before you take the decision to drink from the fountain.

Should you chose to drink from the fountain, then I urge you to write your name on your cup, in the designated area, for a customised experience of my care for you.

I do want you to be comfortable.

Mia Melvaer, June 2017

METHOD::

“Nannyware”: Software that observes and addresses the user

WHAT::

Nannyware is software meant to protect users while limiting their space of activity. It is software that passive-aggressively suggests or enforces some kind of discipline. In other words, create a form of parental control extended to adults by means of user experience/user interfaces.

Nannyware is a form of Content-control software: software designed to restrict or control the content a reader is authorised to access, especially when utilized to restrict material delivered over the Internet via the Web, e-mail, or other means. Content-control software determines what content will be available or be blocked.

[. . . Restrictions] can be applied at various levels: a government can attempt to apply them nationwide (see Internet censorship), or they can, for example, be applied by an ISP to its clients, by an employer to its personnel, by a school to its students, by a library to its visitors, by a parent to a child's computer, or by an individual user to his or her own computer. [29]

Unlike filtering, accountability software simply reports on Internet usage. No blocking occurs. In setting it up, you decide who will receive the detailed report of the computer's usage. Web sites that are deemed inappropriate, based on the options you've chosen, will be red-flagged. Because monitoring software is of value only "after the fact", we do not recommend this as a solution for families with children. However, it can be an effective aid in personal accountability for adults. There are several available products out there. [30]

---

[29] Wikipedia contributors: *Content-control software — Wikipedia, The Free Encyclopedia*. 2018

[30] TechMission UrbanMinistry.org: *SafeFamilies.org | Accountability Software: Encyclopedia of Urban Ministry*. 2018

URGENCY:

As with all new lifestyle technologies that come along, in the beginning there is also some chaos until their impact can be assessed and rules put in place to bring order and respect to their implementation and use in society. When the automobile first came into being there was much confusion regarding who had the right of way, the horse or the car. There were no paved roads, speed limits, stop signs, or any other traffic rules. Many lives were lost and much property was destroyed as a result. Over time, government and society developed written and unwritten rules as to the proper use of the car. [31]

Disadvantages of explicit proxy deployment include a user's ability to alter an individual client configuration and bypass the proxy. To counter this, you can configure the firewall to allow client traffic to proceed only through the proxy. Note that this type of firewall blocking may result in some applications not working properly. [32]

EXAMPLE:

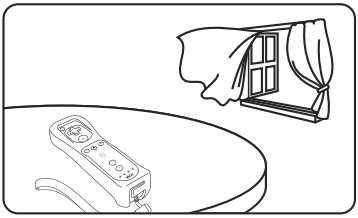
The main problem here is that the settings that are required are different from person to person. For example, I use Workrave with a 25 second micropause every two and a half minute, and a 10 minute restbreak every 20 minutes. I need these frequent breaks, because I'm recovering from RSI. And as I recover, I change the settings to fewer breaks. If you have never had any problem at all (using the computer, that is), then you may want much fewer breaks, say 10 seconds micropause every 10 minutes, and a 5 minute restbreak every hour. It is very hard to give proper guidelines here. My best advice is to play around and see what works for you. Which settings "feel right". Basically, that's how Workrave's defaults evolve. [33]

---

[31] Content Watch Holdings, Inc.: *Protecting Your Family*. 2018

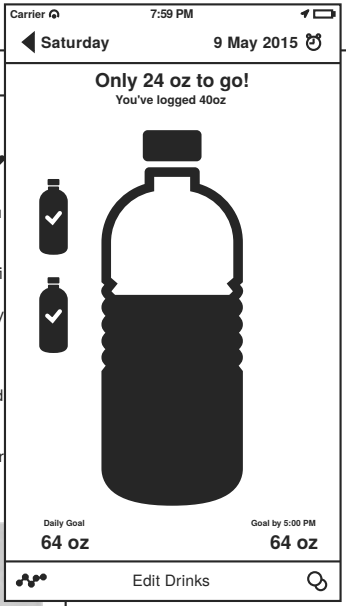
[32] websense.com: *Explicit and transparent proxy deployments*. 2012

[33] workrave.org: *Frequently Asked Questions*. 2018



Why not take a break?
You can pause the game
by pressing ⊕

WiiSports



Siempo
Gesponsord ·

Siempo is a new phone that is designed to help you reclaim control over your time and attention.

It features all the essentials like calling, texting, wi-fi hotspot and maps, while limiting access to the services that tend to interrupt us throughout the day

The Mindful Morning feature helps you create a distraction-free morning routine and the dedicated Pause button helps you hit mute on the digital world for a set period of time.

We're launching shortly — sign up below to be the fir in line for Siempo

Today

/Users/noms/Code/tiny-terminal-care

Always get plenty of sleep, if you can

– Your friends at Slack

SiteCoach content filter

provisio.com
kiosks.org
public.ch
browser.de
site net
software.
terminal.info
digital.be
publicinternet.fi

Access Denied

SiteCoach thinks this website contains content harmful to a young public. The page was blocked!

Reason:
Forbidden Keyword pedophillia!

Please provide us a brief comment, if you believe that this webpage has been blocked wrongly

Hi, this is my site and I'm a syndicated columnist in papers across the U.S. and Canada. The

OK Send

Week

/Users/nom
6889476 –
93d0426 –
2b440fa –
eac7fdf –
d75de24 –
8f98ce4 –
c5a7d0e –
7d9258b –
98ab797 –
68c07c7 –
b2e9820 –
eea1452 –
0693a6e –
caa2bae – (24 ho
ago)
f7c1786 – twitter helper returns what account the tweet is from (24 hours
585af3b – make standup helper care about days (24 hours ago)

tiny care terminal — node care.js — 164×42

Commits

In San Francisco, CA it's 16C and mostl
right now. Today, it will be mostly clo
the forecasted high of 18 and a low of

Not sure how to meditate? Maybe downloa
mindfulness app for your phone!

Listening at a high volume for a long time may damage your hearing. The volume will be increased above safe levels.

Cancel

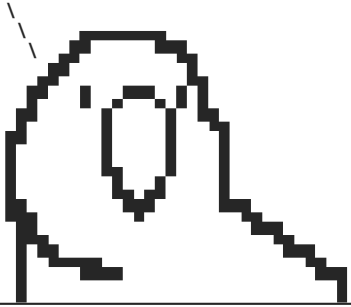
OK

hears of a beehive locat
evotes her life to findin

4 hours

ours ago)

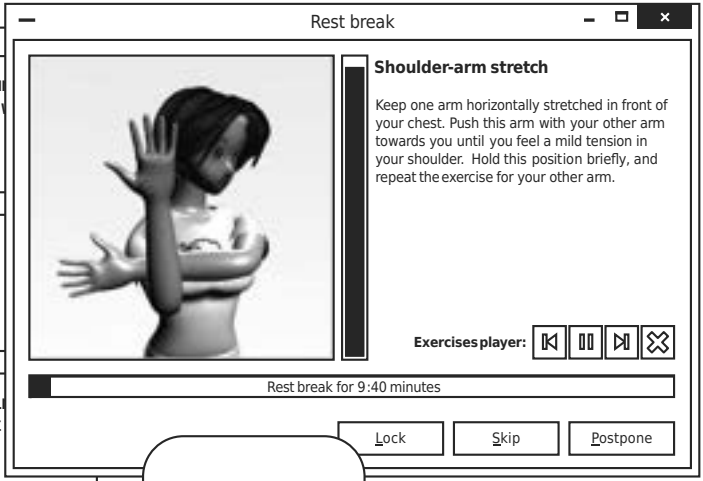
```
+-------------------------------------+
| please remember to breathe deeply |
+-------------------------------------+
```

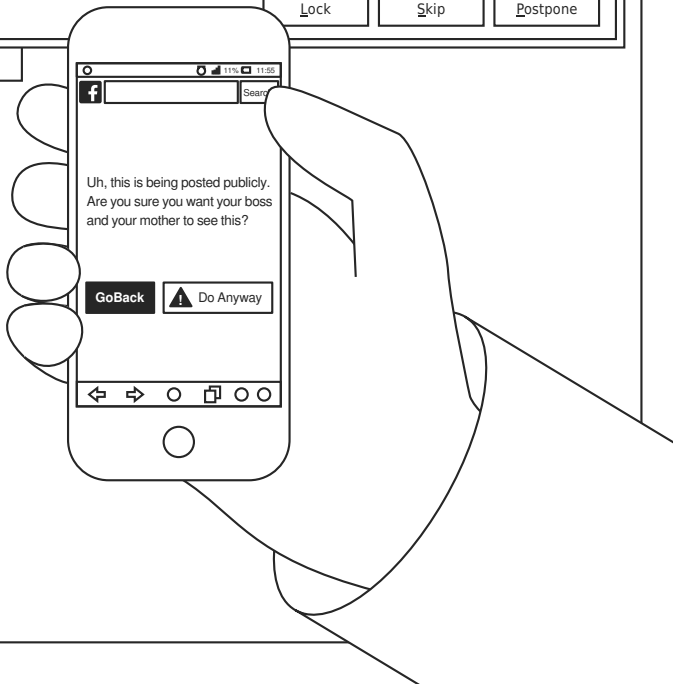


mostly su
y cloudy
w of 11.

wnload a

located i
inding it





SOURCE::

This method was developed by Silvio Lorusso, and presented during the worksession. Excerpts taken from his presentation (and related websites): A Constellation of -Wares, Some Thoughts on Mandatory Entrepreneurialism, Work Ethic Dystopia, Psycho-Cybernetics, Nannyware, and the Rhetoric of Software. The full source can be found at: `https://cryptpad.fr/slide/#/1/view/p8EiphzdeHeVf1yI3NJGEQ/hWWCDdvOB+ulWbxgpk9Y9Q2ixCSShE8TQRxjRoM80aA/`

METHOD::

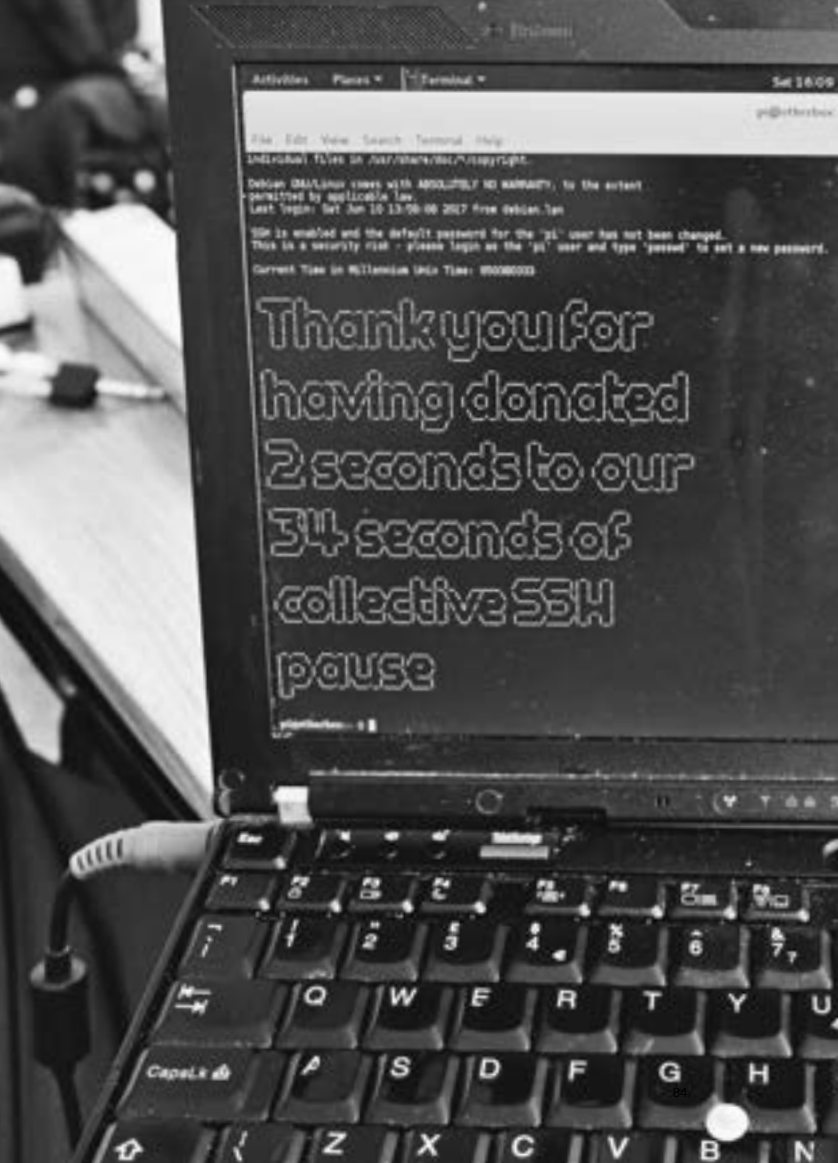
Useless scroll against productivity

<

useless scroll against productivity

SOURCE::

This method was discovered at the end of an etherpad that contained collective notes from the Technogalatic Observation sessions taking place on Friday 8 June, 2017. ➡ `http://observatory.constantvzw.org/etherdump/friday.md.diff.html`


Thank you for having donated 2 seconds to our 34 seconds of collective SSH pause

METHOD::

The ends of time

WHAT:: Command line scripts to investigate computer clocks, system cycles, and software temporalities.

EXAMPLE:: Sundial Time Protocol Group tweaks

```
printf 'Current Time in Millennium Unix Time: '
printf $((2147483647 - `date +%s`))
echo
sleep 2
echo $((`cat ends-of-times/idletime` + 2)) \
     > ends-of-times/idletime
idletime=`cat ends-of-times/idletime`
echo
figlet "Thank you for having donated 2 seconds to \
       our ${idletime} seconds of collective SSH  pause "
echo
echo
```

EXAMPLE:: The Year 2038 problem[34]

Exact moment of the epoch: 03:14:07 UTC on 19 January 2038

local UNIX time of this machine

```
date +%s
```

UNIX time + 1

```
echo $((`date +%s` +1 ))
```

EXAMPLE:: Goodbye unix time

```
while :
do
  sleep 1
  figlet $((2147483647 - `date +%s`))
done
```

SEE ALSO::
How humans and machines negotiate the experience of time
➡ P.89

SEE ALSO::
Useless scroll against productivity
➡ P.79

---

[34] https://en.wikipedia.org/wiki/Year_2038_problem

SOURCE::

Method extracted from notes on the ends of time. ➡ `http://observatory.constantvzw.org/etherdump/ends-of-time.html` On the following pages a reflection by Hans Lammerant, *How humans and machines negotiate the experience of time*.

How humans and machines negotiate the experience of time

The experience of time is an essential element of any form of experience or cognition. Emotions depend to a large extent on expectations, or the potential coming of a future event. Any observing or experience of difference, of presence related to an earlier or later absence, is linked with an experience of time. However, how the actual experience of time is shaped is strongly influenced by all sort of design decisions and implementations, both for humans and machines. Also, the experience of time is a conglomerate of different experiences: time as a common moment, the duration of a certain time, time as cyclic events, historical time, and so on. Researching how humans and machines experience time and negotiate their time experiences is therefore an interesting avenue to explore

Humans have developed a time experience which was linked to natural life cycles, but it has been influenced by both technology and social conditions. The first time cycle is the day-night time cycle. Humans do their stuff during the day, but need to sleep. The night is generally the preferred time for sleep, but the availability of artificial light and the need for long-distance coordination has influenced how humans deal with the day cycle and the place for sleep in it. Part of this experience is measuring time. Early time measurement was linked to observation of natural conditions like sunrise and sunset. Measurement of such sun cycles (which is in fact an earth cycle) through sundials allowed for more precise time referencing, but it is very place and season dependent. More light during summer than winter implied that hours were longer in summer than in winter. Similarly such changes were greater at higher latitudes, while near the equator such changes are limited. In other words, time measured by sundials provided a local common reference of time, but not one common time over longer distances.

This seasonal time experience when the human world was flat reflects the unknown spherical geometry of the earth cycle projected on the flat earth. When humans became aware of earth as a sphere, they responded by flattening and linearising time. Mechanical clocks allowed for the unification of time lengths and thereby also standardized time. Physical observation and all sorts of economic processes needed such standardized time measurement. Early versions of such time measurement, like hourglasses, existed but remained disconnected from day time measurement. Mechanical clocks also allowed for the unification of different time cycles and scales and for the standardization of time over longer distances, departing from the local sundial time to time zones, which were less strictly linked to the seasonal rhythm of the sun and more to geographical zones. Long distance trade, industrialization and later long distance communication by electromagnetic signals (comparable to the speed of light) demanded more geographical coordination. In the nineteenth century, coordination developed through clock networks, with a master clock driving the slave clocks. Nowadays we work with atomic clocks and a global Coordinated Universal Time or UTC as a reference to which geographical time zones are connected. Humans organize their activities accordingly, if necessary by de-linking from the solar rhythm. The borders of time zones diverge very often from the longitudinal lines for economic and political reasons. A similar socialization and globalization of time occurred for computers. Where in the early computer age time was set and counted locally by each machine, it is now common to continuously synchronize time over the Internet through time servers and the Network Time Protocol. Humans get woken, cron jobs get triggered, precision bombs get guided, and trains count their delays in measure with the drill signal of the Master Clock of the US Naval Observatory, only differing by tiny variations in latency times.

Humans have also developed ways to relate to time on longer scales, which were originally linked to natural rhythms: years, life cycles. Calendar systems are used to determine seasonal agricultural needs (when to plant, when to harvest), while they also make it possible to keep track of life cycles and historical time. Again, such calendar systems have been very diverse and local, but have been slowly fused to a couple of dominant models. Generational or birth, life, and death rhythms, originating from the human experience, have been projected on all sorts of phenomena. Religions tried to explain the origin of everything, while also often predicting the end in apocalyptic visions. The demise of religion at the hand of science did not let such generational visions disappear. They got new expressions in scientific theories of the beginning (big bang, evolution) and end (the heath death of the universe, the end of the earth at the final burnout phase of the sun) of everything. Similar apocalyptic visions are now embedded through technological design decisions (Y2K, the end of Unix-time in 2038). In all versions the end is often linked to the specific design of time (e.g. the end of the Maya calendar, millenarian movements). But just like religious visions can extend their apocalypse in a new versions (e.g. the always near but always delayed apocalypse of the Saints of the last days), machinic accounts of time can always be extended by enlarging the bit size of the time range (cfr the extension of Unix time till AD 292277026596, or safely after the end of the observable universe according to contemporary physics).

The time(s) of the machine

Machines also have their natural cycle: the vibrating pulses of the internal crystal clock drives the cycles according to which the processor works. All PCs have such a Real Time Clock (RTC), independent of the processor. The time of the computer is linked to this clock cycle, as it consists of counting cyclical ticks. This

produces a single cyclical, and completely linear, time rhythm in the hardware. In contrast to the earthly or natural cycles it is without any seasonal difference or interference between several cycles. This makes a computer into a monadic time capsule, disconnected from the outside rhythms. The time experience in the software is in principle based on this local time cycle. However, once computers and software gets linked and networked, they have to negotiate and synchronize their times. To start, time in a computer is no unique or unified experience. Several hardware components and a diverse collection of software organised in layers and processes create a whole ecology of interdependent time experiences. The operating system experiences other software components, and users through them, as a bunch of processes screaming for attention. One of the most pushy interrupts is the timer forcing the processor to count another click and update the system time. An internal kernel process performs the negotiation of time through which the clock count is linked with a system time. This starts already during the booting process, when the operating system is determining the clock signal frequency. PCs have another time-measuring device, the Programmable Interval Timer (PIT), which can be set by the processor at a defined frequency. During a short time indicated by this PIT, the clock cycles of the RTC get counted and the frequency of the RTC gets calculated. Several other timing devices are present in the hardware, synchronized by kernel processes into a range of times available for the operating system and the processes. As said, one of these times is avail able as system time and gets communicated to all other processes when demanded. With the advent of the Internet, computers are not monads anymore, but are socialized in the common rhythm broadcasted with the Network Time Protocol. The local system time serves as back up, but gets continuously adapted to the network time. On a Ubuntu/Debian machine these times can be checked with *timedatectl status*. The *timedatectl* command

enables to change these synchronisation processes. The actual synchronisation is done by the *timesyncd* process and the time servers to be used can be set in the /etc/systemd/timesyncd.conf file. Other Linux flavours work with the older ntp process which can be configured in /etc/ntp.conf.

The actual process time is completely different from this system time. Most of the time is in the process of making time to do it. Most of the time processes are put on hold and when the scheduler gives them time they can proceed till the next on hold is forced to make time for another process. The scheduler is the big organizer of time in the internal ecology of processes. System time is externally counted and therefore an external global variable to these processes. They can all send a demand to the kernel to get the system time. This gets communicated through the software stack with a range of system calls of the kernel and through the specific time modules of the programming language the software is programmed in. The difference between the actual process time and system time does not appear in how time is perceived by the process. The process only perceives time as the difference between two demands of system time and is oblivious of its time being put on hold. The processor on the other hand spent most of its processing time as idle time: a processor is doing time waiting for slow system components like memory and even slower hard drives and network connections to respond and switches between checking for responses of these laggards. L'enfer du temps perdu, ce sont les autres.

Connecting computers into a network demands new negotiations of time. We have already mentioned the networked and globalised UTC timekeeping. But temporal negotiations happen on all levels of the system. Network protocols have timed choreographies to make connections and proceed with communications, with time out fail-safes to break off when something goes wrong. The timing

of an action is an essential component making the difference between a meaningful signal and noise. Using a browser over http to connect to a website, or more accurately the server providing the website on your request, is a process built on a discontinuous time practice. In a REST architecture the server just deals with a queue of requests and does not see continuity over time between certain requests from a single user. New temporal practices have been designed and technically implemented. Through cookies, the server is able to recognise users and their state in time. The border between your computer and servers on the internet becomes fuzzy when external software is dropped on your computer and runs in your web browser. Javascript modules get dropped on your computer and can start performing tasks on demand of an external server, be it rendering a graph or mining digital money. Similarly, the browser becomes a border where time gets negotiated through synchronization processes. What time exactly is depends on the task at hand. Each tool contains its specific temporal practices, dependent on the speed and subject matter of the synchronization process needed. For example, for collaborative document editing tools like Google Docs or Etherpad it is a sequence of versions of the document. A range of strategies have been developed to synchronize the changes made by different users with the least amount of conflict. The Easysync-protocol used by Etherpad constructs a document from changesets. A changeset details the edits made by a user or the difference with the former local version of the text. In other words, the flow of time is represented through a series of changesets which can be summed to the present state. The server keeps a sequence of acknowledged changesets. These acknowledged changesets represent the current common state and get communicated to all clients. Locally each client builds its present state by summing up the acknowledged changesets received from the server, and with the local changesets which are not yet acknowledged. Each client communicates its local changesets, every 500 ms after the

latest acknowledgment. As this changeset can be relative to an earlier version the server first recalculates a changeset relative to the current version. Then it sends an acknowledgment to the sender and the changeset to the other clients. Different collaborative document editing tools use different methods to deal with conflicts, which also are dependent on the purpose or content of the document. Conflicts between numeric edits in a collaborative spreadsheet are dealt with differently than conflicts between text edits. But what is common to both is that the present gets continuously constructed both on the server and on the clients and negotiated through the specific protocol. The user can always access its local copy of the text and create its local present, but this local present then gets negotiated with the server to create a common present. And every tool rings also its own disasters and apocalyptic experiences through the collapse of the negotiation process and thereby of the common state.

This negotiation of time is in the first place a negotiation of the present. But also what is past and future gets constructed in this present. The ordered sequence connected to the system time(s) already imply such a past and future. But the construction of a past consists also of a negotiation of what remains and what gets forgotten. Data gets stored in memory or written to storage devices, while the operating systems assigns time stamps to these traces of the present becoming past and writes other traces to a range of log files. As writers trying to capture their stream of consciousness already experienced, it is impossible to completely store the present for future use as past. There will always be a remainder that escapes and can not be written down. Similarly, an operating systems can not keep a log of all its operations without getting stuck in an infinite regress in which it tries to log its logging operations. What will be the past for a future present needs to be constructed and selected. Everything else gets forgotten. Linux

stores its logs in /var/log, while all sorts of programs can create their own logs. All these logs are specific histories, for example of what programs got installed. To get an idea of your past on a Linux system in the file system, you can collect the timestamps of all files with:

```
os.system("ls -R -l --time-style=long-iso \
           > modlistsystem-long-iso.txt")
os.system("ls -R -lu --time-style=long-iso \
           > accesslistsystem-long-iso.txt")
```

Renegotiating time with your computer: sundial network time.

Technology has been a tool through which humans create distance from natural cycles and design their own time experiences. This means we can critically intervene in the functioning of the technology and develop alternative time practices and experiences. As an example we put forward as a not-yet-existing-proof of concept the possibility to let the computer work according to sundial time. Is it possible to reconfigure the time of the computer and rewire the connections with human and natural rhythms? Would we be able to let the computer function in an older human time experience, e.g. the time of the sundial? How can such sundial experience be built into the system and what would be the impact on the user? Sundial time is a geographically and seasonally localised time. It also includes a distinction between day and night. The night is ‘out of time’. This distinction has been extradited from linear time practices, where the amount of light is just a variable external to steady beat of time. Linear time turns the night into economically productive time. Running your computer on sundial time re-introduces the night in your system.

An intervention to introduce sundial time in your computer is possible in the whole range of places we discussed where time gets negotiated. The most fundamental but also most difficult route would be to exchange the clock with something reflecting a more natural rhythm. However, as we have seen that this clock has become only a back up tool and that the actual system time is always renegotiated with other systems connected through the network, another renegotiation process can be introduced. Therefore, the most easy way to introduce sundial time would be to reconfigure the *timesyncdor ntp* processes and make them listen to an alternative sundial time server. Such a sundial time server can be built in different ways. A hardware version can be made of a sundial combined with light sensors, measuring the location of the shadow through the difference in light and deriving the sundial time from this location. This would link the time server to the actual earthly day and night cycle. A simulated version could be a local piece of software which looks up the timing of sunrise and sunset at the specific location and day. Based on this information it recalculates the local sundial time, which it provides through the NTP protocol. The local computer receives this sundial time from the nearest sundial time server and continuously adapts its system time, as it does now already based on UTC. Further, it can be programmed to go into sleep modus during the night and to wake up only when a sunrise signal is received from the time server.

Letting a computer run on sundial time is a conscious effort to disconnect it from human-made linear time and to reconnect it with the old earth-driven cycles and the ancient time experience by humans. The unified time of UTC and the time zones gets broken up into extremely localized time servers, and the difference between day and night gets introduced into the functioning of the computer. This re-enactment of such cycles through the computer will remind human users that the experience of time

is a socially negotiated and technically implemented experience. Further, a computer running on sundial time would be a great piece of nannyware nudging the user to live in harmony with his natural environment and a synchronization tool to re-establish the link between the users bio-rhythm and the earthly day and night cycle. It would also be a great reminder why humans tried in the first place to escape from these earthly rhythms through technology and started hacking time.

Hans Lammerant, June 2017 - January 2018

NOTE:: This text was written in and in resistance/conflict to WTC time, which is exemplary for the bureaucratic office time experience. The WTC building boots up at 9AM and is turned off in several steps (at 5PM the air conditioning is turned off, later entrances close) into a wake state. The concierge can let you in and out through the night side entry, but main life support systems remain turned off outside office hours. This wake state continues during the weekend, turning the building into a glass house with limited air in storage. On Monday morning this bureaucratic time capsule revives from its slumber. A suffocating experience, but actually the building is a great piece of nannyware nudging towards a healthy office rhythm without too much extra hours outside the herd work ethic.

Such office time can also be implemented and negotiated on your local computer. A small Proof of Concept was the adapted bashrc on the etherbox simulating a system getting bored and stealing time. This can be further developed by including a differentiated response during versus outside office hours, aging (slower when timestamp of original install is more remote in time), etc.

LANGUAGING

Observing software as/through writing

LANGUAGING

Observing software as/through writing

:: Motivation:
- capitalism as sorcery (Isabelle Stengers): we are under the black magic of capitalist wor(l)d making
- reclaim a power of saying: we have the feeling that some words are imposed on us[35]

:: She is in search of “a different alphabet, a different language,” a means of communication which would be "constantly in the process of weaving itself, at the same time ceaselessly embracing words and yet casting them off to avoid becoming fixed, immobilized.[36]

:: It is surprising how rarely language appears in the list of relevant programming metaphors, despite periodic attempts to envisage program code as a form of literary expression. It is as if we have become so accustomed to think of programming languages as languages - that we forget that this analogy has its own history.[37]

---

[35] Techno-Galactic Software Observatory: *Notes from the Observatory on glossaries and vocabularies*. 2017

[36] Sadie Plant: *Zeroes and Ones: Digital Women and the New Technoculture*. English. 1st edition. New York: Doubleday, Sept. 1997. ISBN: 978-0-385-48260-8, pg.140

[37] David Nofre, Mark Priestley, and Gerard Alberts: *When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960*. en. In: *Technology and Culture* 55.1 (Mar. 2014), pp. 40–75. ISSN: 1097-3729. DOI: 10.1353/tech.2014.0031. Visited on Jan. 31, 2018, pg.43

We write on paper, but we write to a magnetic disk (or tape). Part of what the preposition contributes here is a sense of interiority; because we cannot see anything on its surface, the disk is semantically refigured as a volumetric receptacle, a black box with a closed lid. If we were writing on the disk we would be able to see the text, like a label. Instead, the preposition of choice, “to,” becomes a marker for our intuition that the verb “write” is not altogether appropriate, a rough fit at best.[38]

Just as freedom of speech is a convenient myth under which something else entirely can safely be left to occur, the ideal of a word processor is that it creates an enunciative framework that remains the same whether what is being written is a love letter or a tax return. What kind of language is the language of Word?[39]

---

[38] Matthew Kirschenbaum: *Extreme Inscription: Towards a Grammatology of the Hard Drive*. In: *TEXT Technology* 2 (2004). Visited on Jan. 31, 2018, pg.101

[39] Matthew Fuller: *Behind the Blip: Essays on the Culture of Software*. English. Brooklyn, NY: Autonomedia, Mar. 2003. ISBN: 978-1-57027-139-7, pg.146

METHOD::

Quine

WHAT:: A program whose function consists of displaying its own code. Also known as “self-replicating program.”

WHY:: Quines show the tension between “software as language” and “software as operation.”

HOW:: By running a quine you will get your code back. You may choose to go a step further and wonder about functionality and aesthetics, uselessness and performativity, data and code.

EXAMPLE:: A quine (Python). When executed it outputs the same text as the source:

```
s =  's = %r\nprint(s%%s)'
print(s%s)
```

EXAMPLE:: A oneline unibash/etherpad quine, created during relearn 2017:

```
wget -qO- "http://192.168.73.188:9001/p/quine/export/txt" | \
 curl -F  "file=@-;type=text/plain" \
          "http://192.168.73.188:9001/p/quine/import"
```

The encounter with quines may deeply affect you. You may want to write one and get lost in trying to make an ever shorter and more elegant one. You may also take quines as point of departure or limit-ideas for exploring software dualisms.

“A quine is without why. It prints because it prints. It pays no attention to itself, nor does it asks whether anyone sees it.”
“Aquine is aquine is aquine.”
Aquine is not a quine.
This is not aquine.

R E M E M B E R

Although seemingly absolutely useless, quines can be used as exploits.

Exploring boundaries/tensions

databases treat their content as data (database punctualization)
some exploits manage to include operations in a database

SEE ALSO:
Pan/Monopsychism
➡ P.57

SOURCE::

This method is part of *Aquine*, a discussion of and research into dualism in software. Notes: `http://observatory.constantvzw.org/etherdump/auqinas.diff.html`

METHOD::

Glossaries as an exercise

WHAT:: Using the technique of psychoanalytic listening to compile (gather, collect, bring together) a list of keywords for understanding software.

HOW:: Create a shared document that participants can add words to as their importance emerges. To do pyschoanalytic listening, let your attention float freely, hovering evenly, over a conversation or a text until something catches its ear. Write down what your ear/eye catches. When working in a collective context invite others to participate in this project and describe the practice to them. Each individual may move in and out of this mode of listening according to their interest and desire and may add as many words to the list as they want. Use this list to create an index of software observation.

URGENCY:: Not creating and troubling categories on a regular basis risks path determinacy.

NOTE:: Do not remove someone else's word from the glossary during the accumulation phase. During the editing phase (which comes after the conclusion of the accumulation phase and is ideally conducted through collective consensus), you can mark them for attention. If possible, keep traces of those terms nominated for removal or merging.

NOTE:: There was no consensus regarding the preceding note.

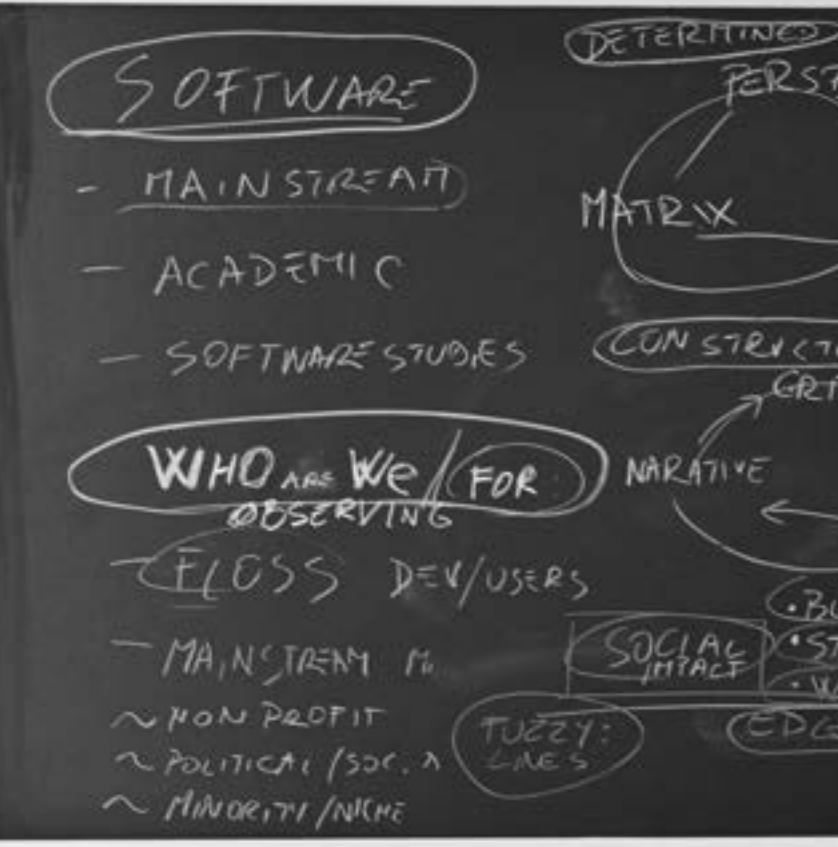
WARNING:: This method is not exclusive to and was not developed for software observation. It may lead to awareness of unconscious processes and to shifts in structures of feeling and relation.

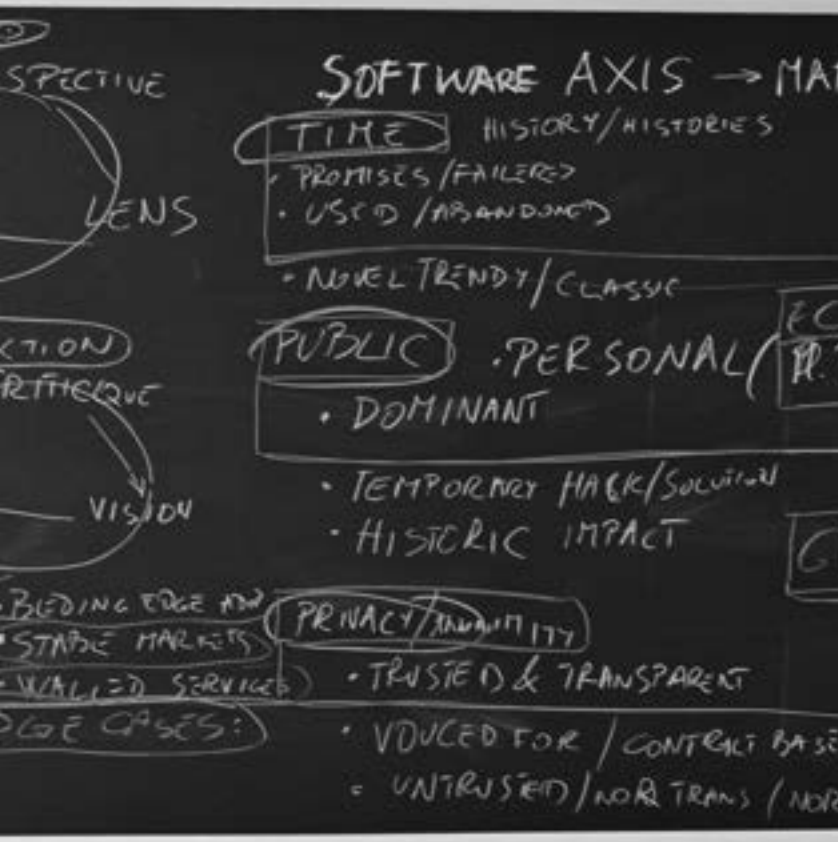
SOURCE::

Notes on vocabulary ➡ `http://observatory.constantvzw.org/etherdump/vocabulary.md.diff.html`

Agile
Aquine
Authority
Attack
Bash
Battery
Beast
Bestiary
Bounce
Bug
Clouds
Code
Colonial
Comfortable
Command Line
Communication
Compile
Comportment
Connectivity
Contract
Corporate
Crash
Curious
Daemon
Dirty
Emotional
Flow
Fountain
Galaxies
Gooey
Green
Guide
Hand
Icon
Intake
Intimate
Imperial
Issues
Kernel
Libraries
Machine
Magic
Mantra
Memory

Museum
Naming
Noise
Observation
Passive-aggressive
Parental
Pause
Perception
Power
ProductionPower
Programmers
Progress
Promiscuous
Public
Punch
Quine
Quit
Relational
Red
Scripting
Scroll
Scrum
Silence
Spin
Spindle
Software
Softwear
Sundial
Survival
Technology
Test
Thank you
Time
Trailing
Urgency
Useless
Volatile
Warning
WhiteBoard
Write
Yoga


SOFTWARE
- MAINSTREAM
- ACADEMIC
- SOFTWARE STUDIES
WHO ARE WE / FOR
OBSERVING
- FLOSS DEV/USERS
- MAINSTREAM
~ NON PROFIT
~ POLITICAL / SOC.
~ MINORITY / NICHE
DETERMINED
MATRIX
NARATIVE
SOCIAL IMPACT
FUZZY: LINES


SOFTWARE AXIS → MA
TIME
HISTORY/HISTORIES
PROMISES/FAILERS
USED/ABANDONED
NOVEL TRENDY/CLASSIC
PUBLIC
PERSONAL
DOMINANT
TEMPORARY HACK/SOLUTION
HISTORIC IMPACT
PRIVACY/ANONYMITY
TRUSTED & TRANSPARENT
VOUCED FOR / CONTRACT BASE
UNTRUSTED/NOR TRANS
SPECTIVE
LENS
VISION
BLEDING EDGE AND
STABLE MARKETS
WALLED SERVICES
DGE CASES:

METHOD::

Adding qualifiers

WHAT:: Applying a moral, ethical, or otherwise evaluative/adjectival/validating lens.

R E M E M B E R

[V]alues are properties of things and states of affairs that we care about and strive to attain. . . Values expressed in technical systems are a function of their uses as well as their features and designs.[40]

HOW:: Adjectives create subcategories. They narrow the focus by naming more specifically the imagined object at hand and by implicitly excluding all objects that do not meet the criteria of the qualifier. The more adjectives that are added, the easier it becomes to answer the question “what is software?”. Or so it seems. Consider what happens if you add the words good, bad, bourgeois, queer, stable, expensive to software. Now make a list of adjectives and try it for yourself. Level two of this exercise consists of observing a software application and deducing from this the values of the individuals, companies, and societies that produce, distribute, and use it.

NOTE:: A qualifier may narrow definitions to undesirable degrees.

---

[40] Mary Flanagan and Helen Nissenbaum: *Values at Play in Digital Games*. 2014

WARNING:: This exercise may be more effective at identifying normative and ideological assumptions at play in the making, distributing, using, and maintaining of software than at producing a concise definition.

EXAMPLE:: “When asked, Jean Heuns had difficulty answering the question”what is software“, but he said that he could answer the question”what is **good** software". What is **good** software?

SOURCE::

Notes on Multiple Software Axes ➡ `http://observatory.constantvzw.org/etherdump/multiple-software-axes.html`

METHOD::

Searching “software” through software

WHAT:: A quick way to sense the ambiguity of the term “software” is to go through the manual files on your hard drive and observe the cases in which the term is used.

HOW:: Command-line oneliner

WHY:: Software is a polysemous term that takes on different meanings depending on where, when and by who it is summoned. It comes with different assumptions for the different agents involved in its production, and for those whom otherwise use, encounter, or are subjected to it in any way or form. From the situated point of view of the software present on your machine, when and why does software call itself by that name.

EXAMPLE::

So software exists only outside your computer? Only in general terms? Checking for the word software in all manual pages:

```
grep -nr software /usr/local/man
!!!!
```

Software appears only in terms of license:

```
This program is free software
This software is copyright (c)
```

We don't run software. We still run programs.
Nevertheless software is everywhere.

SOURCE::

See notes line 574-589 Day1 ➡ `http://observatory.constantvzw.org/etherdump/files.md.diff.html`

SEE ALSO:: Ask several people from different fields age groups the same question: "What is software?"

➡ P.49

METHOD::

Persist in calling everyone a Software Curious Person

WHAT:: Naming can be a method for changing a person's relationship to software. For example, by (sometimes forcibly) calling everyone a Software Curious Person it might be possible to help people realizing their actual knowledge and practices and encourage them to engage more in understanding what software is, in order to reclaim their power over tools.

HOW:: Insisting on curiosity as a relation, rather than for example fear or admiration might help cut down the barriers between different types of expertise and allow multiple stakeholders to feel entitled to ask questions, to engage, to investigate and to observe.

WHEN:: Persistently

URGENCY:: Software is too important to not be curious about. Observations could benefit from recognising different forms of knowledge. It seems important to engage with software through multiple from multiple perspectives and positions, not only by means of technical expertise.

EXAMPLE:: This method was used to address each of the visitors at the Technogalactic Walk-in Clinic.

HEALING AND EMBODIMENT

Feeling software

:: Giving some meaning / emotion to the time of “nothing happening”, the moment of pause when logging in to a service, a return to the experience of the body.[41]

:: The programmer, who needs clarity, who must talk all day to a machine that demands declarations, hunkers down into a low-grade annoyance. It is here that the stereotype of the programmer, sitting in a dim room, growling from behind Coke cans, has its origins. The dis-order of the desk, the floor; the yellow Post-it notes everywhere; the whiteboards covered with scrawl: all this is the outward manifestation of the messiness of human thought. The messiness cannot go into the program; it piles up around the programmer. Soon the programmer has no choice but to retreat into some private interior space, closer to the machine, where things can be accomplished.[42]

:: The list of things machines are good at continually expands, and assertions about the things humans are said to be good at generally consider only whether a human can physically or cognitively accomplish a task, rather than

---

[41] Techno-Galactic Software Observatory: *Notes from the Observatory on When and Where is Software*. 2017

[42] Ellen Ullman: *Close to the Machine: Technophilia and Its Discontents*. English. Reprint edition. New York: Picador, Feb. 2012. ISBN: 978-1-250-00248-8, pg.23

whether the task is morally and ethically defensible or desirable. [. . .] Assuming a timeless, natural division of labor in which we divvy up the work for humans and the work for machines each according to their abilities, deflects attention from the specific conditions under which humans labor, and the changing systems of compensation and reward in which they contribute value to the projects of others.[43]

:: Our therapeutic approach is inspired by the unix file system paradigm in which every component of a computer can be represented by a file, be it your hard drive, memory or sound card. Going together through the affordances and limitations of such a paradigm, we hope to provide a more intimate access to your software. Wheter it is about specific problematic situations with your computer, or to address a general curiosity about filesystems, we will take you by the hand through an extensive intake.[44]

---

[43] Hamid Ekbia and Bonnie Nardi: *Heteromation and its (dis)contents: The invisible division of labor between humans and machines*. en. In: *First Monday* 19.6 (May 2014). ISSN: 13960466. Visited on Jan. 31, 2018

[44] Techno-Galactic Software Observatory: *Introduction to file therapy*. 2017

METHOD::

Setup a Relational Software Observatory Consultancy (RSOC)

WHAT:: Ethnomethodological interviews.

HOW:: Read the signs. Considering the ever changing nature of software development and use, and its vast impact on globalized societies, it is necessary to recognize that the problems arising from software are often either passively-perceived or actively-observed without an articulation of the relations. Reading the signs of the relational aspect of software observance will give you another view on software that will shape your ability to survive any kind of software disaster.

URGENCY::

- Collectivise research around hacking to save time.
- Self-articulate software needs as your own Operating (system) perspective.
- Change the lens by looking at software with a time-based perspective.

WHO:: A practitioner who can facilitate the “what is our relation to software” discussion and administer the RSOC interview as a service.

EXAMPLE: What follows is an example of a possible diagnostic questionnaire.

What to expect

Through administration of this questionnaire, you will obtain a cartographic view of software users profiles. It will help you to shape your own relation to software. You will be able to construct your own taxonomy and classification of software users which is necessary in order to find a means of rescue in case of a software catastrophe.

User Habits

- What kind of user would you say that you are?
- What is your most frequently used type of software?
- How often do you install/experiment/learn new software?

History

- What is your first recollection of software use?
- How often do/when did you last purchase software or pay for a software service?

Ethics

- What is the software feature you care about the most?
- Do you use any Free Software?
  If yes than
  - do you remember your first attempt at using this software service? Do you still use it? If not why?
- Do you pay for media distribution/streaming services?

- Do you remember your first attempt at using Free Software and how did that make you feel?
- Have you used any of these software services: Facebook, dating apps (Grindr, Tinder, etc.), Twitter, Instagram or equivalent.
- Can you talk about your favorite apps or webtools that you use regularly?
- What is the most popular software your friends use?

Skill

- Would you say that you are a specilised user?
- Have you ever used the command line?
- Do you know about scripting?
- Have you ever edited an HTML page? A CSS file? A PHP file? A configuration file?
- Can you talk about your most technical encounter with your computer / telephone?

Economy

- How do you pay for your software use?
- Please elaborate (for example, do you buy the software? / contribute in kind / deliver services or support)
- What is the last software that you paid for using?
- What online services are you currently paying for?
- Is someone paying for your use of service?

Personal

- What stories do you have concerning contracts and administration in relation to your software, Internet or computer?
- How does software help you shape your relations with other people?

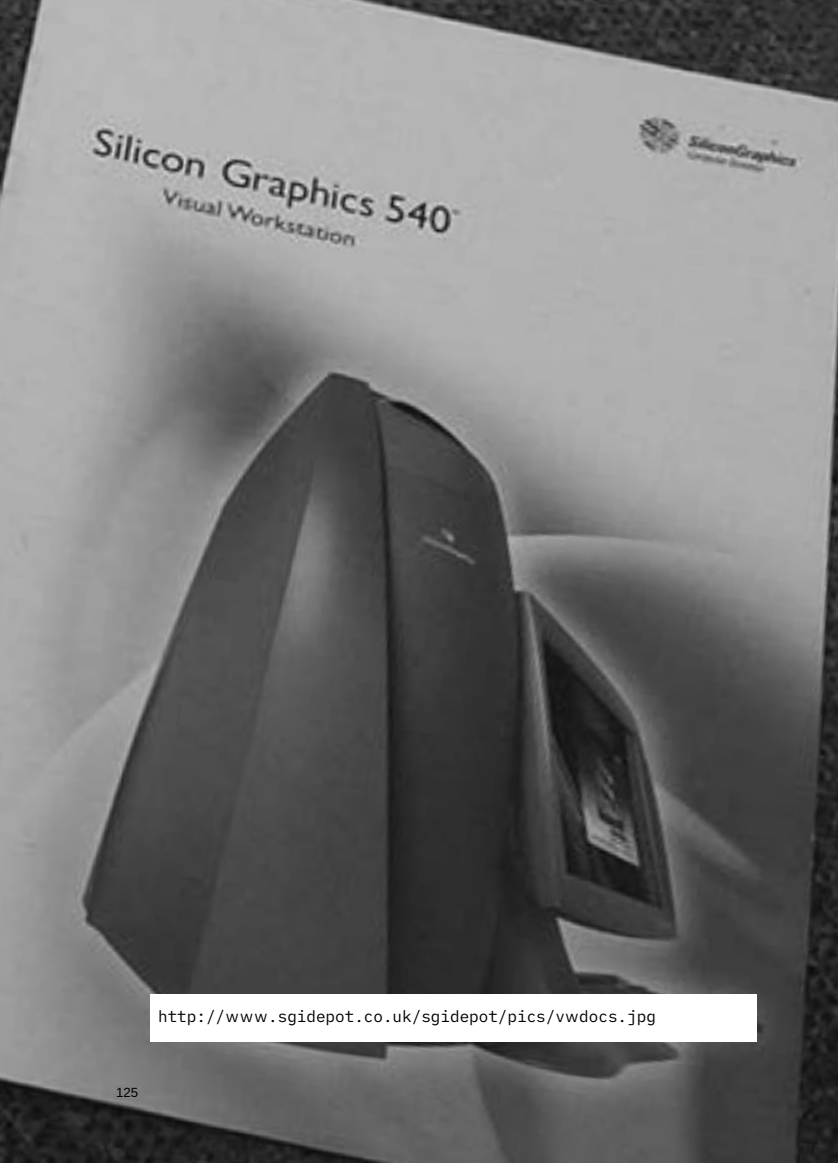
- From which countries does your softwares come from / reside? How do you feel about that?
- Have you ever read a terms of use for a software service, what about one that is not targeting the American market?

Possible/anticipated user profiles

...meAsHardwareOwnerSoftwareUSER: I did not own a computer personally until very very late as I did not enjoy gaming as a kid and had no interest in spending much time behind a PC beyond work (and work computer). My first experience was hence I think in 2005 and it was a SGI workstation that was the computer of the year 2000 (cost 10.000USD) and I got it for around 300USD. Proprietary drivers for unified graphics+RAM were never released, so it remained a software dead-end in gorgeous blue curved chassis (➡ P.125).

...meAsSoftwareCONSUMER: I payed/purchased software only twice in my life (totalling less then 25eur), as I could access most commercial software as widely pirated in Balkans and later had more passion for FLOSS anyway, this made me relate to software as material to exchange and work with, rather than commodity goods I could or could not afford.

...meAsSoftwareINVESTOR: I did it as both of those apps were niche products in early beta (one was Jeeper Elvis, real-time-non-linear-video-editor for BeOS) that failed to reach market, but I think I would likely do it again and only in that mode (supporting the bleeding edge and off-stream work), but maybe with more than 25eur.



http://www.sgidepot.co.uk/sgidepot/pics/vwdocs.jpg

...meAsSoftwareUserOfOS: I would spend most of 80s ignoring computers, 90s figuring out software from high-end to low-end, starting with OSF/DecAlpha and SunOS, than IRIX and MacOS, finally Win 95/98 SE, that permanently pushed me into niches (of montly Linux distro install fests, or even QNX/Solaris experiments and finally BeOS use).

...meAsSoftwareWEBSURFER: I got used to websurfing in more than 15 windows on Unix systems and never got used to less than that ever since, furthermore with addition of more browser options this number only multiplied (always wondered if my first system was Windows 3.11 - would I be a more focused person and how would that form my relations to browser windows>tabs).

...meAsSoftwareUserOfProprietarySoftware: I signed one NDA contract in person on the paper and with ink on a rainy day while stopping of at train station in northern Germany for the software that was later to be pulled out of market due to problematic licensing agreement (intuitively I knew it was wrong) - it had too much unprofessional pixeleted edges in its graphics.

...meAsSoftwareUserOfDatingWebsites: I got one feature request implemented by a prominent dating website (to search profiles by language they speak), however I was never publicly acknowledged (though I tried to make use of it few times), that made our relations feel a bit exploitative and underappreciated.

...meAsSoftwareUserTryingToGoPRO: My only two attempts to get into a software company failed as they insisted on full time commitments. Later I found out one was intimidated in the interview and the other gave it to a person that negotiated to work part time with a friend! My relation to professionalism is likely equally complex and perverted as my one to the software.

SOURCE::

This method was developed by The RSOC Group.

Intake

Case study : W. W.

...ww.AsExperiencedAdventurousUSER: *Experiments with software every two days as she uses FLOSS and GNU/Linux, cares the most for malleability of the software - as a result she has big expectations of flexibility even in software category which is quite conventional and stability focused like file-hosting.*

...ww.AsAnInvestorInSoftware: *Paid for a compiled version of FLOSS audio software 5 years ago as she is supportive of economy and work around production, maintenance and support, but she also used closed hardware/software where she had to agree on licences she finds unfair, but then she was hacking it in order to use it as an expert - when she had time.*

...ww.AsCommunicationSoftwareUSER: *She is not using commercial social networks, so she is very conscious of information transfers and time relations, but has no strong media/format/design focus.*

Q: What is your first recollection of software use?
A: MS-DOS in 1990 at school – I was 15 or 16. Oh no 12. Basic in 1986.

Q: What are the emotions related to this use?
A: Fun. I'm good at this. Empowering.

Q: How often do/when did you last purchase software or pay for a software service?
A: I paid for ardour five years ago. I paid the developper directly. For the compiled version. I paid for the service. I pay for my website and email service at Domaine Public.

Q: What kind of user would you say you are?
A: An experienced user drawing outside the lines. I don't behave.

Q: Is there a link between this and your issue?
A: Even if it's been F/LOSS there is a lot of decision power in my package.

Q: What is your most frequently used type of software?
A: Web browser. Email. Firefox & Thunderbird.

Q: How often do you install/experiment/learn new software?
A: Every two days. I reinstall all the time. My old lts system died. Stop being supported last april. It was Linux Mint something.

Q: Do you know about scripting?
A: I do automating scripts for any operation I have to doi several times like format conversion.

Q: Can you talk about your most technical encounter with your computer/telephone?
A: I've tried to root it. But i didn't succeed.

Q: How much time do you wish to spend on such activities like hacking, rooting your device?
A: Hours. You should take your time.

Q: Did you ever sign license agreement you were not agree with? How does that affect you?
A: This is the first thing your when you have a phone. It's obey or die.

Q: What is the software feature you care for the most?
A: Malleability. Different ways to approach a problem, a challenge, an issue.

Q: Do you use any free software?
A: Yes. There maybe are some proprietary drivers.

Q: Do you remember your first attempt at using free software and how did that make you feel?
A: Yes I installed my dual boot in... 10 years ago. Scared and powerful.

Q: Facebook, dating apps (Grindr or the sort), Twitter, Instagram or equivalent?
A: Google, Gmail. That's it.

Q: Can you talk about your favorite apps or webtools that you use regularly?
A: Music player. Vanilla music and f-droid. browser. I pay attention to clearing my history, no cookies. I also have iceweasel. Https by default. Even though I have nothing to hide.

Q: What stories around contracts and administration in relation to your software internet or computer?
A: Nothing comes to my mind. I'm not allowed to do, to install on a phone. When it's an old phone, there is nothing left that is working you have to do it.

Q: How does software help you shape your relations with other people?
A: It's a hard question. If it's communication software of course it's its nature to be related to other people. There is an expectency of immediate reply, of information transfer... It's troubling your relation with people in certain situations.

Q: From which countries does your softwares live/is coming from? How do you feel about that?
A: I think I chose the Netherlands as a mirror. You are hoping to reflect well in this mirror.

Q: Have you ever read a terms of software service; one that is not targeting the American market?
A: I have read them. No.

Chest, Knee to Floor Pose). With a deep
lower your body down till your forehead, chest, knees, hands
touching the mat, your butt tilted up. Working software is the
measure of progress.
(Cobra Pose). With a deep inhalation, you slowly snake
your head to up, your back arched concave, as much as possible.
sustainable development. You are all maintaining a
sponsors, developers, and users together.
Adho Svanasana (Downward Facing Dog Pose).
technical excellence and good design enhances agility.
(Sun Sight Pose). Simplicity—the art of
work not done—is essential.
(Hand to Foot Pose). The best architectures,
requirements, and designs emerge from self-organizing teams.
Chandrasana (Half Moon Pose). At regular intervals, you
how to become more effective, then tune and adjust
accordingly.
with a salute to honor your agile yoga practices.
a productive scrum meeting, now i invite you to open your
around a bit, from the feet up to the head and back
again.

and let's do a scrum together if you're ok being touched
by someone else. if not, you can do it on your own. so put your
of the SCP around you. now we're joined together, let's
together as we inhale and exhale, syncing our body together
with our own internal software, modulating our oxygen levels
availability of our service facilities.

a couple of exercice to protect and strengt
as entrepreneurs, they are a ve
in order to be able to type, to swipe, to
them in good health.
towards each other in a prayer pose, aroun
but i'm using my extreme programming
press the palms together firmly, press th
that while breathing in and out twice.
our arms towards us, in the air, [redacted] finge
make your shoulders round. let's breath while
apple mantra : Individuals and interactions ov

the arms next to the body and

METHOD::

Agile Sun Salutation

R E M E M B E R

Agile software development describes a set of values and principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. These principles support the definition and continuing evolution of many software development methods.[45]

WHAT:: You will be observing yourself

[45] Wikipedia contributors: *Agile software development — Wikipedia, The Free Encyclopedia*. 2018

HOW::

Scrum is a framework for managing software development. It is designed for teams of three to nine developers who break their work into actions that can be completed within fixed duration cycles (called “sprints”), track progress and re-plan in daily 15-minute stand-up meetings, and collaborate to deliver workable software every sprint. Approaches to coordinating the work of multiple scrum teams in larger organizations include Large-Scale Scrum, Scaled Agile Framework (SAFe) and Scrum of Scrums, among others. [46]

WHEN:: Anywhere where it’s possible to lie on the floor

Self-organization and motivation are important, as are interactions like co-location and pair programming. It is better to have a good team of developers who communicate and collaborate well, rather than a team of experts each operating in isolation. Communication is a fundamental concept. [47]

---

[46] Wikipedia contributors: *Scrum (software development) — Wikipedia, The Free Encyclopedia*. 2018

[47] Wikipedia contributors: *The Manifesto for Agile Software Development*. 2018

Because of the broad aims of this chapter, we have relied on a combination of methodologies. This includes over 20 in-person and telephone interviews with relevant industry experts, including software developers, devops, product managers and developers, data engineers, a/b testers, AI experts, and privacy officers. During these conversations, we inquired how the production of software and services is organized, as well as how relevant transformations have come to affect the conditions for privacy governance. In addition to the interviews, we have relied on industry white papers, legal, policy and technical documents, as well as relevant scientific literature, in particular from the fields of computer science and engineering, industrial management, software studies, regulation and law. We build on Yoo and Blanchette's volume on the regulation of the cloud and the infrastructural moment of computing (Yoo and Blanchette 2015) as well as Kaldrack and Leeker's edited volume on the dissolution of software into services (Kaldrack and Leeker 2015).

In the coming sections, we first describe the three shifts that constitute what we call the agile turn. For each of the shifts, we touch on their historical roots and sketch some of its current motions. Next, we introduce the three perspectives through which we explore the implications of the agile turn to privacy governance, namely modularity, temporality and capture. These perspectives also allow us to question some of the underlying assumptions of privacy research and policy when it comes to the production of software and digital functionality more generally.

2. The Agile Turn

Over the last decade and a half, the production of (non-critical) software has been fundamentally transformed as the result of three parallel developments. First, increasingly software producers have moved from the use of heavyweight and planned development models for information systems such as the so-called waterfall model, to lightweight and lean methods[5]. These latter models are categorized under the umbrella term 'agile' software development and involve an emphasis on user-centricity, short development cycles, continuous testing and greater simplicity of design (Douglass 2015).

Second, pervasive connectivity and advances in flexible client-server models have made possible a shift from "shrink wrapped software" products to software as services as the model for architecting and offering digital functionality. In this so-called service-oriented architecture (SOA) model, software no longer runs only on the client side, but is redesigned to run on a *thin client* that connects to a *server* which carries out most of the necessary computation. In addition, the core functional components of a

[5] A 2015 survey conducted by HP as part of their report titled "State of Performance Engineering" with 601 IT developers in 400 US companies indicated that two thirds of these companies are either using "purely agile methods" or "leaning towards agile". "Is agile the new norm?" http://techbeacon.com/survey-agile-new-norm

realized that its internal solutions for the production and management of virtual machines could be the basis of an external offering as well (Black 2009). To phrase it differently, Amazon's cloud offerings emerged from internally oriented engineering innovations related to the efficient production of their services in a new production paradigm. Amazon's cloud services are leading in the industry (Knorr 2016).

More recently, a similar move can be observed in the proliferation of the container model for the production and management of service components in a cloud environment (Metz 2014). This container model involves a further advancement in the use of the cloud for production of digital functionality. It involves an abstraction away from the virtual machine and a focus on making the service component the dominant building block, both for development as well as for operations. In the words of the Cloud Native Computing Foundation (CNCF), that is spearheading the container model: "Cloud native applications are container-packaged, dynamically scheduled and microservices-oriented" (Fay 2015). The foundation includes the likes of Cisco, Google, Huawei, IBM, Red Hat, Intel, Docker and the Linux Foundation. Google's contribution involves the donation of open sourced container manager 'Kubernetes',[11] an open sourced solution derived from its internal solution called Borg (Metz 2015).

The agile turn has accelerated software production while transforming business operations. Clearly, this has great implications for different aspects of privacy governance. Many of the elements of the agile turn have been addressed by privacy researchers and policymakers in some way, but an integrated perspective on the implications of the agile turn for privacy governance has so far been missing. In the next sections, we develop three perspectives that allow us to look at the privacy implications of the agile turn and to start reflecting upon the ability of existing privacy governance frameworks to address some of the related challenges.

3. Modularity

The agile turn comes with an increase in modularity in the software as a service environment. The term modularity is used to describe the degree to which a given (complex) system can be broken apart into subunits (modules), which can be coupled in various ways (Baldwin 2015). As a design or architectural principle modularity refers to the "building of a complex product or process from smaller subsystems that can be designed independently yet function together as a whole" (Baldwin and Clark 1997). The concept of modularity and its application have been the subject of research in different engineering disciplines and industrial management (Dörbecker and Böhmann 2013). It is generally used to manage complexity of systems and to allow for independent implementation and reuse of system components (Clark et al. 2005) and is an important design and policy principle for the Internet (Van Schewick 2010; Yoo 2016). Modular design involves the mantra that the independence of system components is

[11] Kubernetes is derived from κυβερνήτης and is Greek for "helmsman" or "pilot".

URGENCY:: Use Agile Software Development Methods to develop a new path into your professional and personal life towards creativity, focus and health.

The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and also therefore more effective.[48]

SOURCE::

Agile Sun Salutation was developed by Anne Laforet and performed by Allegra. See following pages for the full script.

[48] Philippe Kruchten: *Agile's Teenage Crisis?* 2011

Agile Sun Salutation

Hello and welcome to the presentation of the agile yoga methodology. I am Allegra, and today I'm going to be your personal guide to YOGA, an acronym for "whY Organize? Go Agile!". I'll be part of your team today and we'll do a few exercises together as an introduction to a new path into your professional and personal life towards creativity, focus and health.

A few months ago, I was stressed, overwhelmed with my work, feeling alone, inadequate, but since I started practicing agile yoga, I feel more productive. I have many clients as an agile yoga coach, and I've seen new creative business opportunities coming to me as a software developer.

For this first experience with the agile yoga method and before we do physical exercises together, I would like to invite you to close your eyes. Make yourself comfortable, lying on the floor, or sitting with your back on the wall. Close your eyes, relax. Get comfortable. Feel the weight of your body on the floor or on the wall. Relax.

Leave your troubles at the door. Right now, you are not procrastinating, you are having a meeting at the <SAY THE NAME OF YOUR LOCATION HERE>, a professional building dedicated to business, you are meeting yourself, you are your own business partner, you are one. You are building your future.

You are in a room standing with your team, a group of lean programmers. You are watching a white board together. You are starting your day, a very productive day as you are preparing to run a sprint together. Now you turn towards each other, making a scrum with your team, you breathe together, slowly, inhaling and exhaling together, slowly, feeling the air in and out of your body. Now you all turn towards the sun to prepare to do your

ASSanas, the Agile Sun Salutations or ASS with the team dedicated ASS Master. She's guiding you. You start with Namaskar, the Salute. your palms joined together, in prayer pose. You all reflect on the first principle of the agile manifesto. Your highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Next pose, is Ardha Chandrasana or (Half Moon Pose). With a deep inhalation, you raise both arms above your head and tilt slightly backward arching your back. You welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. Then you all do Padangusthasana (Hand to Foot Pose). With a deep exhalation, you bend forward and touch the mat, both palms in line with your feet, forehead touching your knees. You deliver working software frequently.

Surya Darshan (Sun Sight Pose). With a deep inhalation, you take your right leg away from your body, in a big backward step. Both your hands are firmly planted on your mat, your left foot between your hands. You work daily throughout the project, business people and developers together. Now, you're flowing into Purvottanasana (Inclined Plane) with a deep inhalation by taking your right leg away from your body, in a big backward step. Both your hands are firmly planted on your mat, your left foot between your hands. You build projects around motivated individuals. You give them the environment and support they need, and you trust them to get the job done.

You're in Adho Mukha Svanasana (Downward Facing Dog Pose). With a deep exhalation, you shove your hips and butt up towards the ceiling, forming an upward arch. Your arms are straight and aligned with your head. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Then, Sashtang Dandawat (Forehead, Chest, Knee to Floor Pose). With a deep exhalation, you lower your body down till your forehead, chest, knees, hands and feet are touching the mat, your butt tilted up. Working software is the primary measure of progress.

Next is Bhujangasana (Cobra Pose). With a deep inhalation, you slowly snake forward till your head is up, your back arched concave, as much as possible. Agile processes promote sustainable development. You are all maintaining a constant pace indefinitely, sponsors, developers, and users together.

Now back into Adho Mukha Svanasana (Downward Facing Dog Pose). Continuous attention to technical excellence and good design enhances agility.

And then again to Surya Darshan (Sun Sight Pose). Simplicity–the art of maximizing the amount of work not done–is essential. Then to Padangusthasana (Hand to Foot Pose). The best architectures, requirements, and designs emerge from self-organizing teams.

You all do Ardha Chandrasana (Half Moon Pose) once again. At regular intervals, you as the team reflect on how to become more effective, then tune and adjust your behavior accordingly. You end our ASSanas session with a salute to honor your agile yoga practices. You have just had a productive scrum meeting. Now i invite you to open your eyes, move your body around a bit, from the feet up to the head and back again.

Stand up on your feet and let's do a scrum together if you're okay being touched on the arms by someone else. If not, you can do it on your own. So put your hands on the shoulder of the SCP around you. Now we're joined together, let's look at the screen together as we inhale and exhale, syncing our bodies together to the rythms of our own internal software, modulating our oxygen level intake requirements to the oxygen availability of our service facilities.

Now, let’s do together a couple of exercises to protect and strengthen our wrists. As programmers, as internauts, as entrepreneurs, our wrists are a very crucial parts of the body to protect. In order to be able to type, to swipe, to shake hands vigourously, we need them in good health. So bring to hands towards each other in a prayer pose, around a book, a brick. You can do it without an object but I’m using my extreme programming book - embrace change - for that. So press the palms together firmly, press the pad of your fingers together. Do that while breathing in and out twice.

Now let’s extend our arms out in the air, palms and fingers facing down, like we’re typing. Make your shoulders round. Let’s breath while visualizing in our heads the first agile mantra: Individuals and interactions over processes and tools.

Now let’s bring back the arms next to the body and raise them again. And let’s move our hands towards the ceiling this time, strenghtening our back. In our head, the second mantra: Working software over comprehensive documentation.

Now let’s bring the hands back out again in the standing position. Once again the first movement while visualizing the third mantra: Customer collaboration over contract negotiation.

And then the second movement once more while thinking about the fourth and last mantra: Responding to change over following a plan and of course we continue breathing.

Now to finish this session, let’s do a sprint together in the corridor!

METHOD::

Hand reading

WHAT:: Have your fortunes read and derive insights into life from the wisdom of software.

HOW:: Put your hand in the nearest Future Blobservation Booth, and get your command lines read.

WHY:: The hand which holds your mouse everyday hides many secrets.

EXAMPLE::

sample reading timeline:

- 15:00 a test user, all tests clear and systems are online a user who said goodbye to us another user a user who thought it'd be silly to say thank you to the machine but thank you very much another kind user who said thank you yet another kind user another user, no feeback a nice user who found the reading process relieving yet another kind user a scared user! took the hand out but ended up trusting the system. "so cool thanks guys" another user a young user! this is a funny computer
- 15:35 another nice user
- 15:40 another nice user
- 15:47 happy user (laughing)

- 15:51 user complaining about her fortune, saying it's not true. Found the reading process creepy but eased up quickly
- 15:59 another nice user: `http://etherbox.local:9001/p/SCP.sedyst.md`
- 16:06 a polite user
- 16:08 a friendly playful user (stephanie)
- 16:12 a very giggly user (wendy)
- 16:14 a playful user - found the reading process erotic DEFRAGMENTING? NO! Thanks Blobservation `http://etherbox.local:9001/p/SCP.loup.md`
- 16:19 a curious user
- 16:27 a friendly user but oh no, we had a glitch and computer crashed. But we still delivered the fortune. We got a thank you anyway
- 16:40 a nice user, the printer jammed but it was sorted out quickly
- 16:42 another nice user
- 16:50 nice user (joak)
- 16:52 yet another nice user (jogi)
- 16:55 happy user! (peter w)
- 16:57 more happy user (pierre h)
- 16:58 another happy user
- 17:00 super happy user (peggy)
- 17:02 more happy user

EXAMPLE::

Software time is not the same as human time. Computers will run for AS LONG AS THEY WILL BE ABLE TO, provided sufficient power is available. You, as a human, don't have the luxury of being always connected to the power grid and thus have to rely on your INTERNAL BATTERY. Be aware of your power cycles and set yourself to POWER-SAVING MODE whenever possible.

OUT OF ORDER
Please come back
later

METHOD::

Bug reporting for sharing observations

WHAT:: Sharing the experience of trying to solve a hard-boiled software noir.

WHEN:: It is difficult to take notes while working on critical infrastructure, but the sooner notes are compiled, the more vivid the report.

WHO:: Bug reports are often presented in a dry format intended for insiders only. But they do not have to be dry. On the contrary, they can have a rather convivial format.

URGENCY:: Embracing moments of breakdown as opportunities to demystify the workings of software and the practice of software debugging.

EXAMPLE::

On monday morning, with the Walk-In Clinic about to open, Etherpad had stopped working but it was unclear why. Where did the etherpad *live*? What could be done to bring it back to normal operation? A detailed bug report was filed, starting by looking around the pi's filesystem by reading `/var/log/syslog` in `/opt/etherpad` and in a subdirectory named `var/` there was `dirty.db`, and dirty it was.

After some getting used to the various commands to navigate in hexedit the unwanted zeroes were gone in an instant.

Martino asked about the trailing '`.`' character and I checked a different copy of the file. No '`.`' there, so that had to go too. My biggest mistake in a long time! The '`.`' we were seeing in Martino's copy of the file was in fact a '`\n`' (`0a`)!

We still don't know why exactly etherpad stopped working sometime Sunday evening or how the zeroes got into the file dirty.db

SOURCE::

The example was extracted from an e-mail sent by J. Hofmüller to the Technogalactic Software Observatory mailinglist. The full text is on the following pages.

ÉGALITÉE!

```
from jogi@mur.at to [Observatory]
When dirty.db get's dirty
```

Dear all,
as promised yesterday, here my little report regarding the broken etherpad.

When dirty.db get's dirty

When I got to WTC on Monday morning the etherpad on etherbox.local was disfunct. Later someone said that in fact etherpad had stopped working the evening before, but it was unclear why. So I started looking around the pi's filesystem to find out what was wrong. Took me a while to find the relevant lines in `/var/log/syslog` but it became clear that there was a problem with the database. Which database? Where does etherpad 'live'? I found it in `/opt/etherpad` and in a subdirectory named `var/` there it was: dirty.db, and dirty it was.

A first look at the file revealed no apparent problem. The last lines looked like this:


```
{"key":"sessionstorage:Ddy0gw7okwbkv5BzkR1DuSLCV_IA5_jQ","val":{"cookie
":{"path":"/","_expires":null,"originalMaxAge":null,"httpOnly":true,
"secure":false}}}

{"key":"sessionstorage:AU1cffgcTf_q6BV9aIdAvES2YyXM7Gm1","val":{"cookie
":{"path":"/","_expires":null,"originalMaxAge":null,"httpOnly":true,
"secure":false}}}

{"key":"sessionstorage:_H5SdUlDvQ3XCuPaZEXQ5lx0K6aAEJ9m","val":{"cookie
":{"path":"/","_expires":null,"originalMaxAge":null,"httpOnly":true,
"secure":false}}}
```

What I did not see at the time was that there were some (AFAIR something around 150) binary zeroes at the end of the file. I used tail for the first look and that tool silently ignored the zeroes at the end of the file. It was Martino who suggested using different tools (`xxd` in that case) and that showed the cause of the problem. The file looked something like this:

```
00013730: 6f6b 6965 223a 7b22 7061 7468 223a 222f  okie":{"path":"/
00013740: 222c 225f 6578 7069 7265 7322 3a6e 756c  ","_expires":nul
00013750: 6c2c 226f 7269 6769 6e61 6c4d 6178 4167  l,"originalMaxAg
00013760: 6522 3a6e 756c 6c2c 2268 7474 704f 6e6c  e":null,"httpOnl
00013770: 7922 3a74 7275 652c 2273 6563 7572 6522  y":true,"secure"
00013780: 3a66 616c 7365 7d7d 7d0a 0000 0000 0000  :false}}}.......
00013790: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

So Anita, Martino and I stuck our heads together to come up with a solution. Our first attempt to fix the problem went something like this:

```
dd if=dirty.db of=dirty.db.clean bs=1 count=793080162
```

which means: write the first 793080162 blocks of size 1 byte to a new file. After half an hour or so I checked on the size of the new file and saw that some 10% of the copying had been done. No way this would get done in time for the walk-in-clinic. Back to the drawing board.

Using a text editor was no real option btw since even vim has a hard time with binary zeroes and the file was really big. But there was hexedit! Martino installed it and copied dirty.db onto his computer. After some getting used to the various commands to navigate in hexedit the unwanted zeroes were gone in an instant. The end of the file looked like this now:

```
00013730: 6f6b 6965 223a 7b22 7061 7468 223a 222f  okie":{"path":"/
00013740: 222c 225f 6578 7069 7265 7322 3a6e 756c  ","_expires":nul
00013750: 6c2c 226f 7269 6769 6e61 6c4d 6178 4167  l,"originalMaxAg
00013760: 6522 3a6e 756c 6c2c 2268 7474 704f 6e6c  e":null,"httpOnl
00013770: 7922 3a74 7275 652c 2273 6563 7572 6522  y":true,"secure"
00013780: 3a66 616c 7365 7d7d 7d0a                 :false}}}.
```

Martino asked about the trailing '`.`' character and I checked a different copy of the file. No '`.`' there, so that had to go too. My biggest mistake in a long time! The '`.`' we were seeing in Martino's copy of the file was in fact a '`\n`' (`0a`)! We did not realize that, copied the file back to etherbox.local and waited for etherpad to resume it's work. But no luck there, for obvious reasons.

We ended up making backups of dirty.db in various stages of deformation and Martino started a brandnew pad so we could use pads for the walk- in-clinic. The processing tool chain has been disabled btw. We did not want to mess up any of the already generated .pdf, .html and .md files.

We still don't know why exactly etherpad stopped working sometime Sunday evening or how the zeroes got into the file dirty.db. Anita thought that she caused the error when she adjusted time on etherbox.local, but the logfile does not reflect that. The last clean entry in `/var/log/syslog` regarding nodejs/etherpad is recorded with a timestamp of something along the line of 'Jun 10 10:17'. Some minutes later, around 'Jun 10 10:27' the first error appears. These timestamps reflect the etherbox's understanding of time btw, not 'real time'.

It might be that the file just got too big for etherpad to handle it. The size of the repaired dirty.db file was already 757MB. That could btw explain why etherpad was working somewhat slugishly after some days. There is still a chance that the time adjustment had an unwanted side effect, but so far there is no obvious reason for what had happened.

– J.Hofmüller
`http://thesix.mur.at/`

METHOD::

Interface Détournement

WARNING:: We are under the black magic of capitalist digital interface making! Critical radars will detect some conceptual gibberish over here!

WHAT:: Satirical détournement of your favorite bullshit website interface, through parody and poetry. Heal your eyes and brain by scratching and diffracting the surface of everyday browsing.

URGENCY:: If you have the feeling that some words are imposed to you and want to reclaim a power of saying/naming.

WHEN:: When you reach your limit and can no longer tolerate the jargon of the digital economy and the rhetorics of Silicon Valley.

R E M E M B E R

Be careful in planning your interface hoax. You might get caught and accused of fake news design and circulation.

SOURCE:

The *Interface Détournement*-method was developed by Loup Cellard.

EXAMPLE:

Example of an interface template study by Henrik Jan Grievink (➡ P.158)

1. Choose a particular website where you want to intervene. You can pick an interface in relation to:

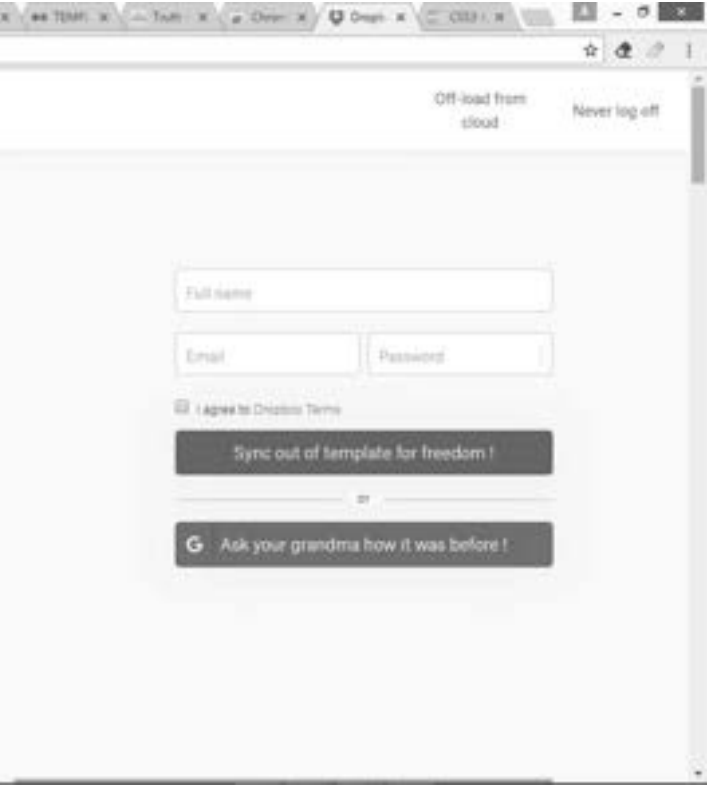
- contestable content: bullshit slogans and jargon, problematic rhetoric, unnecessary content that is interesting to reframe, etc.
- problematic interface issues: unreadabillity, problems of orientation/disorientation, interaction design issues, etc.

2. Open the “developer tool” of your browser. With this tool you will be able to visualize and modify locally the content of your website (the html rendering) and the way it is styled (the css file).
3. Then you have to adopt a strategy of intervention guiding your détournement. Here is a list of proposed strategies:

- The classical détournement: play with the interface content, replace textual elements, add others with a good dose of humour and irony.
- The layout and template observation: study the interface layout by removing all the textual content. The layout will appear, after having done this exercise on several interfaces you will be able to compare the regularities between websites. Then feel free to move elements and reconstruct the interface the way you want.

4. After having done your détournement, you can share the outcome of your work as a hoax: present the new interface you have done just as if it was a mundane and “real” one. Observe the reactions of the tested credulous and refine your détournement with their insights. You can also take a screenshot of your work and send it to an audience.

Example of an interface detournement of the dropbox website


Off-load from cloud
Never log off
Full name
Email
Password
I agree to Dropbox Terms
Sync out of template for freedom !
or
Ask your grandma how it was before !

METHOD::

Comportments of software (softwear)

R E M E M B E R

The analysis of common sense, as opposed to the exercise of it, must then begin by redrawing this erased distinction between the mere matter-of-fact apprehension of reality–or whatever it is you want to call what we apprehend merely and matter-of-factly–and down-to-earth, colloquial wisdom, judgements, and assessments of it.[49]

WHAT:: Observe and catalog the common gestures, common comportments, and common sense(s) surrounding software.

NOTE:: The common senses and comportments of software are informed and conditioned by those of hardware and so perhaps this is more accurately a method for articulating comportments of computing.

---

[49] Clifford Geertz: *Common Sense as a Cultural System*. In: *The Antioch Review* 33.1 (1975), pp. 5–26. ISSN: 00035769

WARNING:: Software wears on both individual and collective bodies and selves. Software may harm your physical and emotional health and that of your society both by design and by accident.

HOW:: This can be done through observation of yourself or others. Separate the apprehended and matter of fact from the meanings, actions, reactions, judgements, and assessments that the apprehension occasions.

1. Begin by assembling a list of questions such as: When you see a software application icon what are you most likely to do? When a software application you are using presents you with a user agreement what are you most likely to do? When a software applciation does something that frustrates you what are you most likely to do? When a software application you are using crashes what are you most likely to do?
2. Write down your responses and the responses of any subjects you are observing.
3. For each question, think up three other possible responses. Write these down.
4. (This step is only for the very curious) Try these other possible responses out the next time you encounter each of the given scenarios.

SEE ALSO::
Agile Sun Salutation
➡ P.135

FLOWS

Flow-regulation, logistics and seamlessness

:: Contentious meetings. Users trying to articulate needs that don’t fit neatly into all the flowcharts and drawings. Compromises. Promises of “future development” to take unaddressed needs into account. “Then it moves to programming,” said the vice president.[50]

:: To get to the demo in five days, the people coming together had to be sufficiently similar, sufficiently flexible, and sufficiently few. The participants all spoke English fluently, had obtained at least college undergraduate degrees, and had trained as engineers — with the exception of the anthropologist, Prem.[51]

:: The “Agile Manifesto” and related commentaries read as a peculiar combination of working methods with moral values, yielding a work ethic tuned towards efficiency, productivity, and customer satisfaction. While emphasizing categories such as “individuality,” “freedom,” and “respect,” many of the recommended principles and methods are in fact reminiscent of the theory of “egoless programming” (. . .).[52]

---

[50] Ellen Ullman: *Close to the Machine: Technophilia and Its Discontents*. English. Reprint edition. New York: Picador, Feb. 2012. ISBN: 978-1-250-00248-8, pg.48

[51] Lilly Irani: *Hackathons and the Making of Entrepreneurial Citizenship*. en. In: *Science, Technology, & Human Values* 40.5 (Sept. 2015), pp. 799–824. ISSN: 0162-2439, 1552-8251. DOI: 10.1177/0162243915578486. Visited on Jan. 31, 2018, pg.13

[52] Christoph Neubert: *The Tail on the Hardware Dog*. English. In: *There is no Software, there are just Services*. Ed. by Irina Kaldrack and Martina Leeker. Lüneburg, 2015, pp. 21 –37. ISBN: 978-3-95796-055-9, pg.33

:: Adrift in the doped lattices of a silicon crystal, a hole is a positive particle before it is the absence of a negatively charged electron, and the movement of electrons toward the positive terminal is also a flow of holes streaming the other way.[53]

[53] Sadie Plant: *Zeroes and Ones: Digital Women and the New Technoculture*. English. 1st edition. New York: Doubleday, Sept. 1997. ISBN: 978-0-385-48260-8, pg.57

METHOD::

Continuous Integration

WHAT:: A sophisticated form of responsibility management: it is the fascia of software services. Continous Integration picks up after all other services and identifies what needs to happen so that they can work in concert. It is a way of observing the evolution of (micro)services through cybernetic (micro)management.

HOW:: Keeping track of changes to all services and allowing everyone to observe if they still can work together after all the moving parts are fitted together.

WHEN:: In a world of distributed systems where there are many parts being organized simultaneously, continuous integration is a form of observation that helps (micro)services maintain a false sense of independence and decentralization while constantly subjecting them to centralized feedback.

WHO:: All services and their operators will submit themselves to the feedback loops of continuous integration. This can be a democratic process or not.

URGENCY:: Continuous Integration is the reconfiguration of the divisions of labor in the shadows of automation. How can we surface and question its doings and undoings?

WARNING:: When each service does one thing well, the service makers tend to assume everybody else is doing the things that they do not want to do.

At the Walk-in Clinic, Continuous Integration was introduced as a service to respond to the integration hell that was produced when Software Curious Persons attempted to engage in more than one of the TGSO services on offer. Due to demand, the Continuous Integration service was extended to do “service discovery” and “load balancing” once the walk-in clinic was in operation.

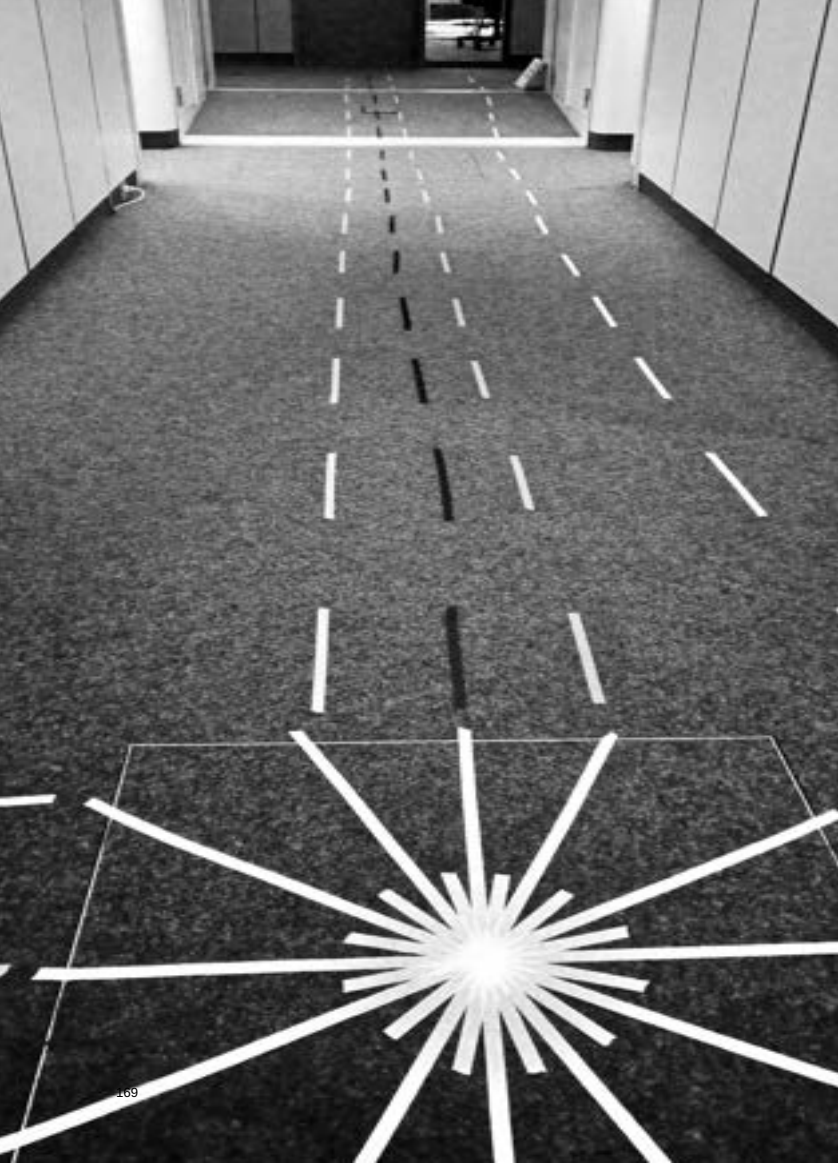
Continuous Integration worked by visiting the different services of the walk-in clinic to check for updates, test the functionality, and think through implications of integration with other services. If the pieces didn’t fit, the service delivered error messages and solution options.

When we noticed that Software Curious Persons visiting the walk-in clinic were having troubles finding the different services, and that some services were overloaded with Software Curious Persons, Continuous Integration was extended. We automated service registration using colored tape and provided a lookup registry for Software Curious Persons.

Load balancing meant that Software Curious Persons were forwarded to services that had capacity. If all other services were full, the load balancer defaulted to sending the Software Curious Person to the Agile Sun Salutation[54] service.

---

[54] `http://pad.constantvzw.org/p/observatory.guide.agile.yoga`



WARNING:: At TGSO, the bundling of different functionalities into the Continuous Integration service broke the “do one thing well” principle, but saved the day. (We register this as technical debt for the next iteration of the Walk-In Clinic.)

SOURCE::

While Continous Integration held the day together, we are sorry to report that the work is only documented in images and in the choreography of the day but not in any of our writings.

R E M E M B E R

Continous Integration may be the string that holds your current software galaxy together. When operated successfully, it is easily overlooked like the air we breathe.

More technically, I am interested in how things bounce around in computer systems. I am not sure if these two things are related, but I hope continuous integration will help me.[55]

[55] Software Curious Person expressing her hopes for Continuous Integration

METHOD::

make make do

WHAT:: Makefile as a method for a promiscuous publication.

It possessed many heads, the exact number of which varies according to the source. [56]

[56] Wikipedia contributors: *Lernaean Hydra — Wikipedia, The Free Encyclopedia*. 2018

NOTE:: May the traditional software workflow of a Makefile[57] serve as a platform for a hyper flexible constellation of multi-style mini-programs spread out over files and folders, media and metadata?

A promiscuous publication is NOT like parallel, hybrid, or cross-media publishing. It is a multi-headed and polycentric form of making public. It enjoys its various forms, and embraces their idiosyncracies. As a multi-headed publication it is NOT obsessed with providing uniform outcomes across diverse media. Having many centers, it can produce new focal points and obfuscate its origins. It explores playfully a spectral diversity with sibling outcomes and bastard children that operate out of sync or tune.

We are curious about creating pipelines that stir and create currents and behaviours spiraling outside of the continuity of predictable flows, where their modality switches as their inputs solidify, evaporate and gelatinize into diverse forms in reactions to conditions, while influencing and changing the others.

How can the making-public itself be an engaging experience, interactive but with erratic options? How could the process of iterating between known sources and diverse and unfamilair pipelines be made tangible?

```
# find all .md files in the directory
md=$(shell ls etherdump/*.md)
# map *.mp => *.html for mdsrc
md2html=$(md:%.md=%.html)
md2pdf=$(md:%.md=%.pdf)
# inputs
# .mdsrcs: A listing of (local) markdown files
```

[57] http://www.gnu.org/software/make/manual/make.html

```
mdsrcs=$(shell ls etherdump/*.mdsrcs)
2col=$(mdsrcs:%.mdsrcs=%.2col.pdf)
pdocpdf=$(mdsrcs:%.mdsrcs=%.pdoc.pdf)
pdochtml=$(mdsrcs:%.mdsrcs=%.html)
# .pdfsrc: A listing of local PDF URLS,
# optionally with #page= fragment
pdfsrcs=$(shell ls etherdump/*.pdfsrcs)
scrp=$(pdfsrcs:%.pdfsrcs=%.scrp.pdf)
all: $(md2html) $(md2pdf) $(pdochtml) $(pdocpdf) $(2col) $(scrp)
dump:
        cd etherdump && etherdump pull --all \
        --pub /home/pi/etherdump \
        --css lib/styles.css \
        --script lib/versions.js --no-raw-ext
        cd etherdump && etherdump index *.meta.json \
        --templatepath /home/pi/etherdump/lib \
        --template index.template.html > _index.html
.PHONY: fixnames
fixnames:
        rename "s/ /_/g" *
        rename "s/[\(\)\?\']//g" *
        rename "s/^(\d)([^\d])/0\1\2/g" *
today:
        touch `date +"%Y-%m-%d.md"`
now_folder:
        mkdir `date +"%Y-%m-%d-%H%M%S"`
# .html <== .md using pandoc
%.html: %.md
        pandoc --from markdown \
                --to html \
                --standalone \
                $< \
                -o $@
# .pdf <== .md using pandoc/latex
```

```
%.pdf: %.md
        pandoc --from markdown \
                --table-of-contents \
                --standalone \
                $< \
                -o $@
##################################################################
# Recipes for lists of MARKDOWN SOURCES
# pandoc/latex pdf assembled from markdown sources
%.pdoc.pdf: %.mdsrcs
        cat $< | python scripts/urls2paths.py | \
        xargs cat | pandoc --from markdown -o $@
# html from markdown sources
%.pdoc.html: %.mdsrcs
        cat $< | python scripts/urls2paths.py | xargs cat | \
        pandoc --from markdown --to html --standalone -o $@
# 2 column bare bones PDF using Report Lab from markdown sources
%.2col.pdf: %.mdsrcs
        cat $< | \
        python scripts/urls2paths.py | \
        xargs cat | \
        pandoc --from markdown --to html --standalone | \
        python scripts/rl2cols.py --output $@
##################################################################
# Recipes for lists of PDF SOURCES
# (with possibly #page=start,end fragments)
%.scrp.pdf: %.pdfsrcs
        python scripts/pdfsrcs.py $< $@ | bash
# special rule for debugging variables
print-%:
        @echo '$*=$($*)'
```

METHOD::

Flowcharts (Flow of the chart –> chart of the flow: on demand!)

WHAT:: A flowchart is a type of diagram that represents an algorithm, workflow, or process by showing the steps as boxes of various kinds and depicting their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting, or managing a process or program in various fields. [58]

WHY:: Through flowcharts one will be able to analyze and reflect on one's own software situation.

URGENCY:: Using flowchart observation methods towards analysis will result in on-demand revelation of flows, relations, and connections previously obscured.

[58] https://en.wikipedia.org/wiki/Flowchart

NOTE:: The term *software situation* is suggested as a good replacement for *software problem*.

HOW::

Sample Flowchart Questionnaire

```
1. Validate your situation?
2. Is it human or machine situation?
human             machine                  I don't know
            |                   |                         |
3. Choose a flowchart element that best describes your situation?
```

Operation ➡ Represented as trapez ◢

Process ➡ Represented as rectangle ▬

Decision ➡ Represented as rhombus ◆

https://unicode-table.com/en/

According to your answer go to question 4.

```
 4.1 Identify operation(s) that could
     help you out through the situation?
 4.2 Identify process(es)  that could
     help you out through the situation?
 4.3 Identify decision(s)  that could
     help you out through the situation?
```

```
5. Was that actually helpful?
         Yes                                      No
          |                                       |
        Done                        o. How does that make you feel?
(end of this questionnaire)              (jump to the beginning)
```

SOURCE::

The SOFTWARE SKETCHING OBSERVATION YUPPIES (SSoGY).

EXAMPLE::

`http://observatory.constantvzw.org/etherdump/clinic.newflow.diff.html`

Service Log

1. Microsoft word copy/paste situation
2. Machine or human situation: it's uncertain, but looks like machine situation as client tried to uninstall and reinstall, 6 months later the situation repeated
3. & 4). Process: back up files, obtain a copy of OS of choice and then reinstall Word. o) How she feels? Feeling angry

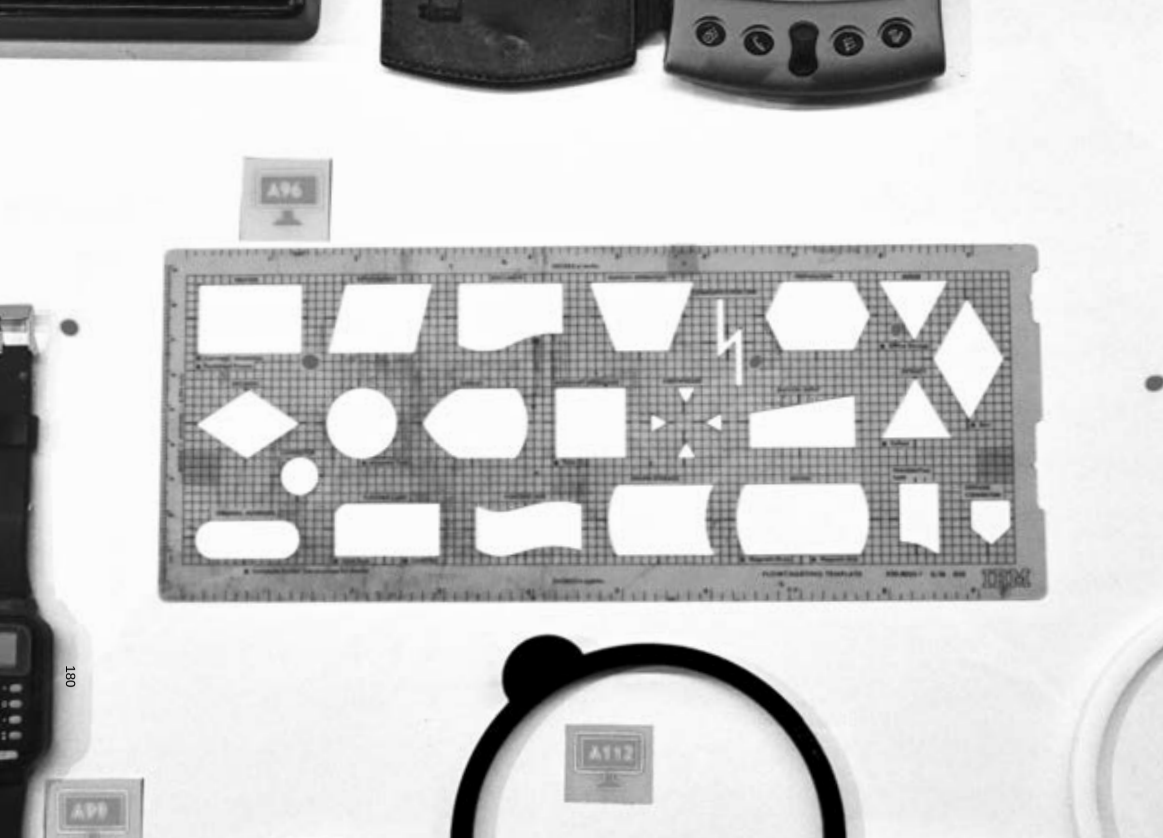
More info ➡ see Rafaella flowchart pic

Appendix A:

- **IBM Ruler:**
  - IBM Flowcharting Template found at NAM-IP (➡ P.180)
  - `http://observatory.constantvzw.org/images/wednesday/P1040728.JPG`
  - left to right
  - process - rectange
  - input / output - skew square
  - document - rectangle, but buttom line is a wave
  - manual operation -
  - communication link -
  - preperation -
  - merge - down-pointing triangle
  - decision – rhombus
  - connector - small circle
  - magnetic tape - circle
  - display -
  - auxilary operation - square
  - arrow head - four triangle
  - manual input -
  - Punched Card - rectangle with left cut edge
  - Punched tape - like a flag

- **Burroughs Ruler:**
  - Burroughs Flowcharting Template found at NAM-IP (➡ P.181)
  - `http://observatory.constantvzw.org/images/wednesday/P1040730.JPG`
  - Left to right, top to bottom:
  - input/output = parallelogram
  - Auxiliary operation = square / gyrational square
  - Preparation - hexagon
  - Drum or disk - oval
  - Process - rectangle
  - Off-line storage - triangle

- On-line storage - like a quarter moon
- Display - one side oval and one side a little oval
- Manual input - rectangle, but top side askew / irregular rectangle
- Document - rectangle, but buttom line is a wave
- Manual Operation - tilted rectangle
- Magnetic Tape - circle but with a square on the buttom right
- Connector - circle
- Communication Link - a z shaped thingy
- Decision - rhombus
- . . . unknown x 2
- Punched Card
- Terminal
- Punched Tape
- Flow Direction
- Card Deck


A96
A112
A99

Burroughs

Appendix B

Friday:

15:30 - 16:15

We started our undertaking by analysing the rulers by IBM and Burroughs. It seems that every symbol on the two rulers has a specific shape and has a specific meaning. There are similarities between the shapes on the two rulers. On the Burroughs we found some symbols without descriptions. We tried to figure out their meaning. We counted up the symbols on the IBM and Burroughs rulers. The specific literature explained to us that engineers have a kind of standard for these symbols. These symbols are vocabulary.

16:15 - 16:40

We started with analyzing an educational flow chart as our case example[59]: We were able to figure out what the arrows on the ruler meant. They are used in combination with the decision symbol. In shape studies, triangles can be used to direct movement based on the direction they point. In the flowchart, the options are represented by rectangles and squares. In shape studies rectangles and squares are stable. They're familiar and trusted shapes and suggest honesty.

16:45 - 17:00

We analyzed a flow chart of a multiplication algorithm. Its content illustrated an instruction to shift the number left or right . The decision box instructed you to repeat it 32 times for the whole number! The rectangles were again representing options, while the flexible factors were illustrated by the rhombus, the symbol for Decision. The final symbol with the title "Done" was illustrated by something inbetween a circle and a rectangle.

---

[59] `https://en.wikipedia.org/wiki/Flowchart#/media/File:LampFlowchart.svg`

17:00 - 17:15
We realized in our undertaking that we needed to jump into an internal perspective and get more familiar with the task of flow chart drawing. For this flow chart we decided to chart the elevator to the first floor from the perspective of the elevator. To achieve this task we agreed to do field research and go to the first floor by elevator for real. Some of us used this opportunity to smoke a cigarette and to elaborate on the experience of the elevator. The final goal is the create a flow chart based on our observations, but we are going to do this tomorrow. [60]

17:15 - 17:40
To round off the day we ended with some general reflections. It seems that the different interpretations of symbols have been integrated into flowcharts from the early ones of the 1920s until now. [61] To put this knowledge into practice we started to sketch out the flowchart on the wall using Post-its.

Current research goals:

- Flowchart or intervention: what is it like to be a elevator
- Zenit browser extension
- Iconographic analyses of flow chart symbols
- Call-flow && zenit modeling template for the drop-in clinic interview

---

[60] Further elevator ruminations were made through reference to Marc Isaacs. Lift. UK, 2001. `https://www.youtube.com/watch?v=FJNAvyLCTik`

[61] Justin Lau and Xing: *Understanding Flowcharts*. 2009

Saturday:

11:00 - 11:40
We updated our research followers and fellow researchers about our undertakings.

14:45-18:00
Zenit(h) modeling/analysis
A method derived from a removal of textual informating from flowcharts – related to previous work with Zenit: International Review of Arts and Culture [62] created by Ljubomir Micic, founder of Zenitism (Zenitizam) an early 20th Century movement, and zenith – an imaginary point directly "above" a particular location, on the imaginary celestial sphere [63]

Connection inspiration: Procedure Flow Chart [64]

Some historical experimentation examples: `http://www.e-w-n-s.net/minis-html/futuremillionaire/transition/v.gif` `http://www.e-w-n-s.net/minis-html/futuremillionaire/transition/arrows.gif` `http://www.e-w-n-s.net/minis-html/_navigation.htm`

Material: Flow chart from Wikimedia Commons, the free media repository [65]

Afternoon
Joseph working on all the ideas at the same time
Continue with flow chart elevator
Lara working on Zenit(h) modeling/analysis leading to call-flow template, and zenit browser plugin
Michaela helping out the flow chart of the elevator as a case study

---

[62] `https://monoskop.org/Zenit`
[63] `https://en.wikipedia.org/wiki/Zenith`
[64] `https://helios.gsfc.nasa.gov/flowchart.html`
[65] `https://commons.wikimedia.org/wiki/Flow_chart`

Sources:

Software design[66], Flowchart[67], Software design description[68], HTML Unicode UTF-8[69], Penguin Dictionary of Symbols[70], Flowchart humour[71], Power Point as Knowledge Communication[72]

Blank diagrams:

See Nothing Volume Three[73], Blank Diagrams #4: NSA Slides of PRISM Program[74], Blank Diagrams #1 – Claude Shannon, schematic diagram of a general communication system[75]

The Grammar of Shapes: `http://vanseodesign.com/web-design/visual-grammar-shapes`

---

[66] https://en.wikipedia.org/wiki/Software_design
[67] https://en.wikipedia.org/wiki/Flowchart
[68] https://en.wikipedia.org/wiki/Software_design_description
[69] https://www.w3schools.com/charsets/ref_utf_geometric.asp
[70] http://www.iausdj.ac.ir/ostad/DocLib71/J._C._Cirlot_Dictionary_of_Symbols__1990.pdf
[71] http://observatory.constantvzw.org/video/A_Computer_Glossary.webm#t=02:26
[72] http://computationalculture.net/article/one-damn-slide-after-another-powerpoint-at-every-occasion-for-speech
[73] http://silviolorusso.com/see-nothing-volume-three
[74] http://silviolorusso.com/blank-diagrams-4-nsa-slides-of-prism-program
[75] http://silviolorusso.com/blank-diagrams-1-claude-shannon-schematic-diagram-of-a-general-communication-system

How to create browser add on /or browser extension:

1. Make a new dir at your local machine. Lets called it adds-on & and a folder called icons
2. Make an icon for your browser extension
3. Open text editor and copy paste this jason code
4. You will need to modify the last two lines to point to the ether-box.local & css to point to the extension.css
5. Make an css script place your css there

On the browser:

1. Go to extension

2. Load debugger from `about:debugging`
3. Load your extension

```
manifest.json:
{
  "manifest_version": 2,
  "name": "extension",
  "version": "1.0",
  "description": "this is the browser extensions",
  "icons": {
    "48": "icons/icon-name-here.png"
  },
  "content_scripts": [
    {
      "matches": ["*://etherbox.local/*"],
      "css": ["extension.css"]
    }
  ]
}
```

Version 3384 Saved June 10, 2017. Authors: m, lara, carlin + 5 unnamed authors

INVASIVE OBSERVATIONS

Being on the side,
in the middle or behind

INVASIVE OBSERVATIONS

Being on the side,
in the middle or behind

:: looking away from the ‘center’, towards peripheries; observe the ‘negative’ space of software. whats around it[76]

:: Infrastructure-as-a-service (IaaS) cloud-computing services provide virtualized system resources to end users, supporting each tenant in a separate virtual machine (VM). Fundamental to the economy of clouds is high resource utilization achieved by sharing: providers co-host multiple VMs on a single hardware platform, relying on the underlying virtual-machine monitor (VMM) to isolate VMs and schedule system resources. While virtualization creates the illusion of strict isolation and exclusive resource access, in reality the virtual resources map to shared physical resources, creating the potential of interference between co-hosted VMs. A malicious VM may learn information on data processed by a victim VM and even conduct side-channel attacks on cryptographic implementations.[77]

---

[76] Techno-Galactic Software Observatory: *Notes from the Observatory on glossaries and vocabularies*. 2017

[77] F. Liu et al.: “Last-Level Cache Side-Channel Attacks are Practical”. In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 605–622. DOI: 10.1109/SP.2015.43, pg.605

:: Unlike proprietary software, Service as a Software Substitute (SaaSS) does not require covert code to obtain the user's data. Instead, users must send their data to the server in order to use it. This has the same effect as spyware: the server operator gets the data—with no special effort, by the nature of SaaSS [. . .] With SaaSS, the server operator can change the software in use on the server. He ought to be able to do this, since it's his computer; but the result is the same as using a proprietary application program with a universal back door: someone has the power to silently impose changes in how the user's computing gets done.[78]

[78] Richard Stallman: *What Does That Server Really Serve?* en. June 2012. Visited on Jan. 31, 2018

METHOD::

Something in the Middle Maybe (SitMM)

R E M E M B E R

In the sexist and militarized language of computer security a “man in the middle” attack refers to a kind of surveillance where an attacker relays and possibly alters the communications of two peers that believe they are communicating directly to each other. These peers may or may not be human.

WHAT:: SitMM allows the Software Curious Person to observe the network connections that your software makes to the outside world. Software running in an isolated device might be powerful, but when the device is networked it becomes a peer in a wilderness of millions of agencies, some benign, some less so. It is in the network that machines touch, fluctuate, and penetrate each other in a promiscuous non-stop bath of data packets, some real, some spoofed.

HOW:: SitMM takes a closer look at the network traffic coming from/going to a software curious person's device. The Software Curious Person using SitMM may start the sniffer functionality with a single click of a button, perform the interaction with the device that they wish to observe and SitMM will issue a report at the end of that interaction.

The Software Curious Person gets to observe their own traffic. Ideally, observing ones own network traffic should be available to anyone, but using such software may be deemed illegal in some jurisdictions.

For example, in the US, Wiretap Law limits packet-sniffing to parties that own the network that is being sniffed. Alternatively the party that sniffs the network must have consent from communicating parties. Section 18 U.S. Code § 2511 (2) (a) (i) says:

> *It shall not be unlawful . . . to intercept . . . while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service*

See here for a paper[79] on the topic.

It is no surprise that Google went on a big legal spree to defend their right to capture unencrypted wireless traffic with Google StreetView cars. The courts were concerned about wiretapping and infringements on the privacy of users, and not with the leveraging of private and public WiFi infrastructure for the gain of a for-profit company. The case raises hard questions about state, corporate, and individual claims on the use of information, and the

---

[79] `http://spot.colorado.edu/ sicker/publications/issues.pdf`

material reality of WiFi signals. So, while WiFi sniffing is common and the tools like SitMM are widely available, it is not always possible for Software Curious Persons to use them legally or to neatly filter out the network “traffic” of one specific individual from that of “others”, as networks often act as carriers to the interactions of many people.

WHEN:: SitMM can be used any time a Software Curious Person might suspect that their software is connecting to external parties, perhaps by “calling home”, the name for when a piece of software contacts its manufacturer to gather usage metrics or device details that they might be collecting illegitimately. For example untill 2014, when a user first signed up to WhatsApp the entire list of contacts in the user’s phone was sent to WhatsApp servers, which allowed WhatsApp the company to build the largest (but invisible) social network in the world. At the time, it’s network was bigger even than Facebook’s.

WHY:: SitMM is intended to be a tool that gives artists, designers, and educators an easy to use custom WiFi router to work with networks and explore the aspects of our daily communications that are exposed when we use WiFi. The goal is to use the output to encourage open discussions about how we use our devices online.

URGENCY:: “Something in the Middle Maybe” wants to be a *sousveillance* software with various goals. Perhaps the most important goal is to demilitarize and emasculate the language of computer security. By introducing gender-neutral terminology and ambiguity, SitMM brings poetry where before there only was room for engineered surveillance. SitMM aims to be usable and accessible to non-experts and is meant as a working tool for artists and designers alike.

NOTE:: SitMM builds on a tool called scapy[80] to implement what is called a network packet sniffer.

Snippets of a Something In The Middle, Maybe - Report

```
UDP 192.168.42.32:53649 -> 8.8.8.8:53
TCP 192.168.42.32:49250 -> 17.253.53.208:80
TCP 192.168.42.32:49250 -> 17.253.53.208:80
TCP/HTTP 17.253.53.208:80 GET http://captive.apple.com/mDQArB9orEi
TCP 192.168.42.32:49250 -> 17.253.53.208:80
TCP 192.168.42.32:49250 -> 17.253.53.208:80
TCP 192.168.42.32:49250 -> 17.253.53.208:80
UDP 192.168.42.32:63872 -> 8.8.8.8:53
UDP 192.168.42.32:61346 -> 8.8.8.8:53
...
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443
TCP 192.168.42.32:49260 -> 17.134.127.97:443


##################################################
Destination Address: 17.253.53.208
Destination Name:    nlams2-vip-bx-008.aaplimg.com


Port: Connection Count
  80:    6


##################################################
Destination Address: 17.134.127.79
Destination Name:    unknown


Port: Connection Count
 443:    2
##################################################
Destination Address: 17.248.145.76
```

[80] http://www.secdev.org/projects/scapy/

```
Destination Name:    unknown

Port: Connection Count
 443:   16
```

SOURCE::

https://github.com/AlternativeLearningTank/SomethingInTheMiddle/

SOURCE::

SitMM emerges from the itinerating practice of Luis Rodil-Fernandez, crossed with those of Jogi Hofmueller, BalkonTactics and the Alternative Learning Tank. *SitMM* is deeply indebted to projects that have served as inpiration such as Dowse ➡ http://dowse.equipment/ , alt.exit ➡ http://alternativelearningtank.net/ and the NetAidKit ➡ https://netaidkit.net/ . http://observatory.constantvzw.org/SomethingInTheMiddle/

METHOD::

What is it like to be AN ELEVATOR*?

NOTE::(*) Where this method refers to AN ELEVATOR, the name of any comparable software system may be substituted.

WHAT:: Understanding software systems by becoming them.

HOW:: Creating a flowchart to incarnate a software system you use everyday.

WARNING:: Uninformed members of the public may panic when confronted with a software performance in a closed space.

EXAMPLE:: What is it like to be an elevator?

what
is
it
like
to be
an
elevator?

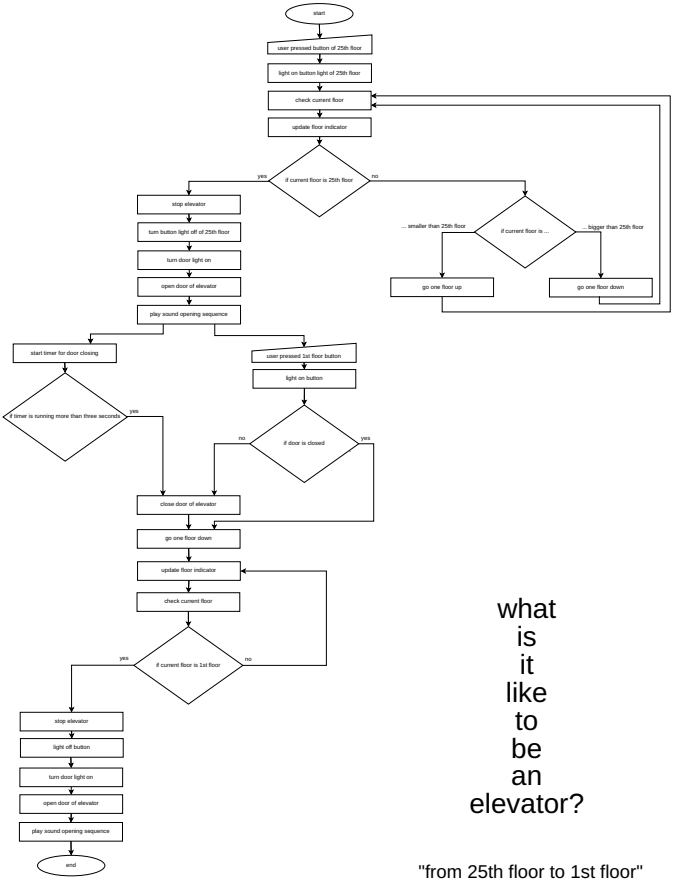
"from 25th floor to 1st floor"

light on button light of 25th floor
check current floor
if current floor is 25th floor
no
if current floor is ...
go one floor up
... smaller than 25th floor
go one floor down
... bigger than 25th floor
stop elevator
turn button light off of 25th floor
turn door light on
open door of elevator
play sound opening sequence
yes
start
user pressed button of 25th floor
close door of elevator
if door is closed
user pressed 1st floor button

```
start timer for door closing
if timer is running more than three seconds
yes
yes
light on button
go one floor down
no
if current floor is 1st floor
update floor indicator
check current floor
stop elevator
no
yes
light off button
turn door light on
open door of elevator
play sound opening sequence
end
update floor indicator
```

SOURCE::

What is it like to be ? was developed by Joseph Knierzinger, Michaela Lakova + other members of the SSOGY group.



what
is
it
like
to
be
an
elevator?

"from 25th floor to 1st floor"

"What is it
like to be an
elevator?"

METHOD::

Side Channel Analysis

WHAT:: A side channel attack is conducted by taking advantage of “leakage from boundaries”. They are made possible by disregarding the abstraction of software into pure logic and focusing on the physical effects of the running of software which become backdoors to observe its functioning, hence re-affirming the materiality of software.

HOW:: As software runs on hardware, it emanates radio-magnetic waves, and you can, for example, build an antenna to capture these waves and then by analyzing them, reconstruct what the software is doing.

R E M E M B E R

“Hardware lives in the real world and real world properties lead to side channels.”

WHO:: bad guys/good guys

WARNING:: **engineers are good guys!**

SOURCE::

This method is inspired by a talk by Thomas De Cnudde from the COSIC research group at the department of Electrical Engineering, KULeuven. Notes from his presentation: `http://observatory.constantvzw.org/etherdump/side-channel-analysis.diff.html`

IMAGES:: Thomas De Cnudde from the COSIC research group at the department of Electrical Engineering, KULeuven **P.204**

COLLECTIONS

Compiling observations

COLLECTIONS

Compiling observations

> Each group will thus contain all the records pertaining to or embracing a particular statistical item, and by counting the cards the numerical value of that item can be readily ascertained. Having thus separated the record-cards into general divisions, (and, if desired, subdivided each group on the same plan.) any additional series of statistical items can be compiled either by the further division or subdivision of the record-cards or by passing all the cards or certain groups only through the electrical apparatus.[81]

[81] Herman Hollerith: “Art of compiling statistics”. US395781 A. Jan. 1889. Visited on Jan. 31, 2018, pg.4

METHOD::

Compiling a bestiary of software logos

WHAT:: The visual culture of software has relied heavily on animal representations since the early days of GNU/Linux. This tendency was cemented into a tradition through the line-drawings used on the covers of the ubiquitous O'Reilly publications [82]. What actors are populating the realm of software observation, and to what effect?

HOW:: Compile a collection of logos and note the metaphors for observation and their surrounding vocabularies. How are different relations between observers and objects of observation established through a combination of vocabularies and images?

EXAMPLE:: This Bestiary was initiated during *Testing the testbed*, a two-day workshop intended to critically evaluate the *Internet of Things (IoT) Inspector*, a testsuite for embedded devices proposed by the Princeton University's Center for Information Technology and Policy (CITP). Participants from Constant, Dyne:BXL, COSIC Leuven and others gathered to test the testbed with the help of their cameras, smartphones, and other "things". As we compared different hard- and software set-ups for observing dataflows in IoT environments, we noticed that many of them were represented by animals and other agents with human traits that would perform the job of looking, analysing, inspecting and investigating. Here, we have expanded the initial collection to address network observation in general. What does it mean to "look" at the activity in a computer network? Who or what is looking and in what way?

---

[82] `http://animals.oreilly.com/browse`

Autopsy®

`https://www.sleuthkit.org/autopsy`

tags: investigation, dissection, forensics

[legacy] Hound dog with face mask and surgeon hat.

Doberman dog with black scarf clenching a magnifying glass between his teeth.

“Autopsy® is a digital forensics platform and graphical interface to The Sleuth Kit® and other digital forensics tools. It is used by law enforcement, military, and corporate examiners to investigate what happened on a computer. You can even use it to recover photos from your camera’s memory card.”

Dowse

`http://dowse.eu`
tags: divination, transubstantiation, clairvoyance

Instrument touching or stirring liquid which is held in a round receptacle.

“Dowse is a transparent proxy facilitating the awareness of ingoing and outgoing connections, from, to, and within a local area network. Dowse provides a central point of soft control for all local traffic: from ARP traffic (layer 2) to TCP/IP (layers 3 and 4) as well as application space, by chaining a firewall setup to a transparent proxy setup. A core feature for Dowse is that of hiding all the complexity of such a setup.”

Ghostery

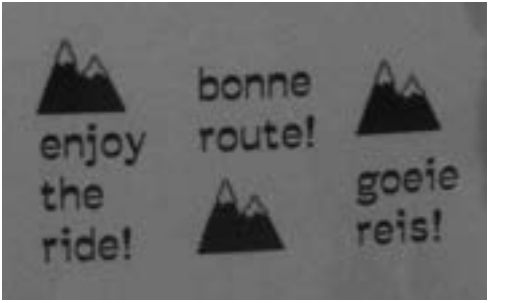
`https://www.ghostery.com`
tags: ghostbusting, time travel, immaterialization

Friendly bright blue ghost with see-through eyes.

“Faster, safer, and smarter browsing. Ghostery helps you browse smarter by giving you control over ads and tracking technologies to speed up page loads, eliminate clutter, and protect your data.”

Hovelbot

`http://www.constantvzw.org/site/On-Journey-with-Hovelbot,2661.html`
tags: exploration, sight-seeing, exquisite corpse



Two pixelated snowy mountains floating between a message 'enjoy the ride' (in three languages).

"Hovelbot is a computer program that, just like Frankenstein's monster, quietly observes how humans live, in order to learn and be able to share its stories with them. The visitor is asked to connect her phone to a local network. By doing so, the visitor can observe how the hidden activity happening on her device is giving form to Hovelbot. In turn, Hovelbot takes its unintentional "teachers" on a journey that, even though it might remind us of the 19th century romantic pursues, will rather be a confrontation between our networked self and the artificial beings that make this network."

Iot inspector

`http://www.iot-inspector.com`
tags: rangefinding, voyeurism, scouting

Wireless signal looking through binoculars.

“Detect vulnerabilities in the firmware of IoT devices, no source code required, instant results, comprehensive reporting and alerting, covers a broad range of IoT devices, including IP Cameras, Routers, Printers and many more, ISP specific solution for CPE devices available. Inspection as a service.”

Privacy Badger

`https://www.eff.org/privacybadger`
tags: outsmarting, watching you watching me, shielding

Grinning badger looking cheeky.

“Privacy Badger is a browser add-on that stops advertisers and other third-party trackers from secretly tracking where you go and what pages you look at on the web. If an advertiser seems to be tracking you across multiple websites without your permission, Privacy Badger automatically blocks that advertiser from loading any more content in your browser. To the advertiser, it’s like you suddenly disappeared.”

The Sleuth Kit®

`https://www.sleuthkit.org`

tags: inspection, protection, hard-boiled

[legacy] Hound dog with a fedora hat looking clueless at a laptop.

Hound dog looking fierce and defensive.

"The Sleuth Kit® (TSK) is a library and collection of command line tools that allow you to investigate disk images. The core functionality of TSK allows you to analyze volume and file system data. The plug-in framework allows you to incorporate additional modules to analyze file contents and build automated systems. The library can be incorporated into larger digital forensics tools and the command line tools can be directly used to find evidence."

Little Snitch

`https://www.obdev.at/products/littlesnitch`
tags: tattletaling, whistleblowing

[legacy] A small humanoid figure (it could also be a duck?) wearing glasses, a red-blue striped propeller hat and blowing a whistle.

Orange striped propeller hat, so-called 'geek-attire'.

"As soon as you're connected to the Internet, applications can potentially send whatever they want to wherever they want. Most often they do this to your benefit. But sometimes, like in case of tracking software, trojans or other malware, they don't. But you don't notice anything, because all of this happens invisibly under the hood. Little Snitch makes these Internet connections visible and puts you back in control!"

Netflix Stethoscope

`http://techblog.netflix.com/2017/02/introducing-netflix-stethoscope.html`
tags: auscultation, stretching out, diagnostic

Vacantly smiling giraffe carrying a stethoscope.

"Netflix is pleased to announce the open source release of Stethoscope, our first project following a User Focused Security approach. The notion of *User Focused Security* acknowledges that attacks against corporate users (e.g., phishing, malware) are the primary mechanism leading to security incidents and data breaches, and it's one of the core principles driving our approach to corporate information security. It's also reflective of our philosophy that tools are only effective when they consider the true context of people's work. Stethoscope is a web application that

collects information for a given user's devices and gives them clear and specific recommendations for securing their systems. If we provide employees with focused, actionable information and low-friction tools, we believe they can get their devices into a more secure state without heavy-handed policy enforcement."

The Transparency Grenade

`https://transparencygrenade.com`
tags: explosion

Translucent replica of a Soviet F1 Hand Grenade.

"The lack of Corporate and Governmental transparency has been a topic of much controversy in recent years, yet our only tool for encouraging greater openness is the slow, tedious process of policy reform. Presented in the form of a Soviet F1 Hand Grenade, the Transparency Grenade is an iconic cure for these frustrations, making the process of leaking information from closed meetings as easy as pulling a pin."

Wireshark

`https://www.wireshark.org/`
tags: knifing through, smelling blood, sinking teeth

Shark fin cutting through ocean waves.

"Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998."

SEE ALSO:
Glossaries as an exercise
➡ P.106

SEE ALSO:
Testing the testbed: testing software with observatory ambitions (SWOA)
➡ P.220

METHOD::

Testing the testbed: testing software with observatory ambitions (SWOA)

WHAT:: Observing Software With Observatory Ambitions (SWOA).

HOW:: The interwebs hosts many projects that aim to produce software for observing software, or simply Software with Observatory Ambitions. A comparative methodology can be produced by testing different SWOA to observe software of interest. Comparing SWOA reveals what is considered as worthy of observation (e.g., what protocols, what space, which devices), the granularity of the observation (e.g., how is the observation captured, in what detail), the logo and conceptual framework of choice that underlies the SWOA, as well as its architecture (e.g., gradware, SWOA as a service). Observing SWOAs puts their observatory ambitions to the test. It enables an analysis of what is made transparent, what is made invisible, and how, as a result, SWOAs can reconfigure power.

WHEN:: Ideally, SWOA can be comparatively observed whenever you feel the urge.

WARNING:: Institutions, laws, and administrators like to limit the use of SWOA to people who are running these networks. Hence, we are presented with the situation that the use of SWOA is condoned when it is done by researchers and pen testers (i.e., they were hired) and shunned when done by others (often subject to name-calling as hackers or attackers). This may hamper your ability to observe SWOA at work.

WHO:: If you can run multiple SWOAs, you can do it.

WARNING:: We find that observation can surface power asymmetries and lead to defensiveness or desires to escape the observation in the case of the observed, or an instinct to try to conceal that observation is taking place. Will people like it if you turn your gaze on their SWOA?

NOTE:: Good SWOA uses an animal as a logo.

WARNING:: Many of the SWOA projects we looked at are promises more than running software or available code. Much of it is obsolete gradware, making observation difficult.

R E M E M B E R

Most software has a recursive observatory ambition (it wants to be observed in its execution, output etc.). Debuggers, logs, dashboards are all instances of software with observatory ambitions. Continuous integration is the act of folding the whole software development process into one big feedback loop. So, what separates SWOA from software itself? Is it the intention of observing software with a critical, agonistic or adversarial perspective vs. one focused on productivity and efficiency?

SOURCE::

The “original testbed” was proposed by collaborators at Princeton University. Testing this particular testbed happened at a workshop in Brussels organized by Constant[^]{http://constantvzw.org/site/Testing-the-testbed,2739.html}.

EXAMPLE:: To elucidate this method further, one can take a look at the [Something in the Middle Maybe], which is an instance of a SWOA. To complete a comparative analysis use different sniffing software to observe wireless networks, e.g., wireshark vs tcpdump vs SitMM.

SEE ALSO::
Compiling a bestiary of software logos
➡ P.208

SEE ALSO::
Something in the Middle Maybe (SitMM)
➡ P.191

METHOD::

Prepare a Reader to think theory with software

WHAT:: Compile a collection of texts about software.

HOW:: Choose texts from different areas. Software observations are mostly done in the realm of the technological and the pragmatic. The ecology of texts around software includes first and foremost manuals, technical documentation, and academic papers by software engineers which all *live* in different realms of expertise. More recently, the field of software studies opened up additional perspectives fuelled by cultural studies and sometimes philosophy. A Reader allows all of these different kinds of text to intersect and intermingle. It helps to understand the many types of vocabularies that exist around software, and to see what types of observation each of them invites.

NOTE:: Selected quotes from the reader are used to introduce each of the chapter headers in this guide.

EXAMPLE:: Chapters and index from the *Techno Galactic Software Observatory reader*:

I. WHAT IS SOFTWARE

Viewing software in the long-term context of historical ‘numerical artefacts’ is an occasion to reflect on the conditions of its appearance, and allows us to take on current-day questions from a genealogical perspective. What is software? How did it appear as a concept, in what industrial and governmental circumstances? The selected texts explore the materiality of software, its relation to hardware, language, discourse and abstraction with each their own way of questioning and proposing agendas and assumptions.

- Herman Hollerith. Art of compiling statistics. U.S. Patent 395,781 filed June 18, 1887, and issued January 8, 1889.
- Jean-François Blanchette. “A material history of bits.” JASIST 62, 1042-1057. 2011.
- David A. Patterson and John L. Hennessy. Computer Organization and Design, Fifth Edition: The Hardware/Software Interface (5th ed.). Morgan Kaufmann Publishers Inc, 2013
- Friedrich Kittler, “There Is No Software,” Ctheory (October 18, 1995)
- Thomas Haigh, Mark Priestley, Crispin Rope, Reconsidering the Stored-Program Concept, IEEE Annals of the History of Computing Volume 36, Number 1, January-March 2014
- Wendy Hui Kyong Chun. “Programmability.” In Software Studies: A Lexicon, edited by Matthew Fuller, 224–229. MIT Press, 2008
- Sadie Plant, Zeros + Ones: Digital Women + The New Technoculture. Fourth Estate, 1997
- David Nofre, Mark Priestley, Gerard Alberts, When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-1960. in Technology and Culture, 55(1), 40-75. 2014
- Graham White. Hardware, Software, Humans: Truth, Fiction and Abstraction. HISTORY AND PHILOSOPHY OF LOGIC vol. 36, (3) 278-301. 2015.

II. WHEN AND WHERE IS SOFTWARE

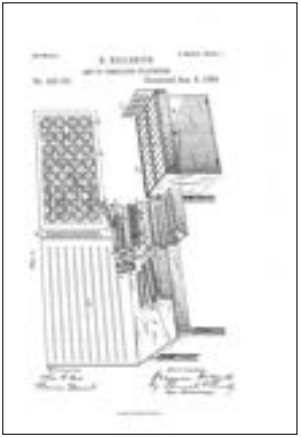
How do layers of abstraction have an effect on the way software is produced and vice versa? What is the space-time dimension of IT development or where and when is software made today? The way computer programs and operating systems are manufactured changed tremendously through time, so its production times and places changed too. From military labs via the mega-corporation cubicles to the open-space freelancer utopia, the texts in this chapter trace the ruptures and continuities in software production. From time-sharing to user-space partitions and containerization, this chapter looks at the separations at work. What happens to the material conditions of software production (factory labor, hardware but also minerals) when it evaporates into a cloud?

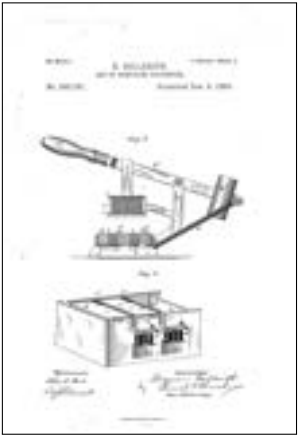
- John Harwood, The Interface: IBM and the Transformation of Corporate Design, 1945-1976. University of Minnesota Press, 2011
- Nathan Ensmenger, The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise, 2010
- Ellen Ullman, Close to the Machine: Technophilia and Its Discontents. City Lights Books, 1997
- Femke Snelting. Dividing and sharing. 2009
- Hamid Ekbia and Bonnie Nardi, Heteromation and its (dis)contents: The invisible division of labor between humans and machines. First Monday 19(6) · June 2014
- Richard Stallman, Who does that server really serve?. 2016
- Critisticuffs. Free Property - On Social Criticism in the Form of a Software Licence. 2013
- Seda Gurses and Joris van Hoboken. “Privacy after the Agile Turn.” Open Science Framework, 2016.
- Christoph Neubert, “The Tail on the Hard-ware Dog”: Historical Articulations of Computing Machinery, Software, and Services in Irina Kaldrak and Martina Leeker, There is not software, there are just services. 2015

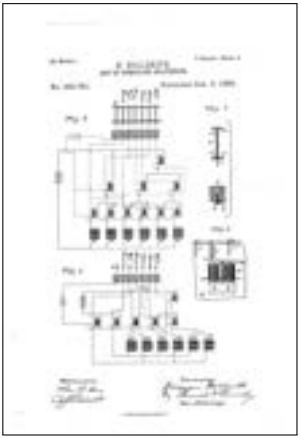
III. OBSERVATION AND ITS CONSEQUENCES

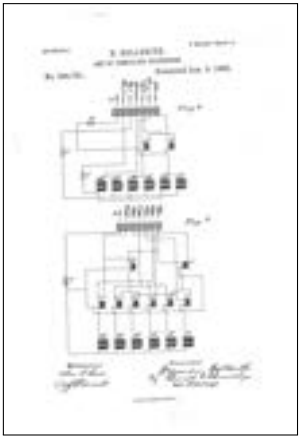
The development of software encompasses a series of practices whose evocative names are increasingly familiar: feedback, report, probe, audit, inspect, scan, diagnose, explore . . . What are the systems of knowledge and power within which these activities take place, and what other types of observation are possible? The material in this section is a compendium of probes such as learning by doing; exploring software through the analysis of its language and grammar; critical ethnography and self-testing as a user. In addition, we have included some conventional methods and tools for increasing the performance and security of software. Appropriating them for Techno-galactic software observation first of all turns the gaze onto the process of observation itself, and eventually opens up possibilities to actively interfere with the functioning of software.

- Lilly Irani, Hackathons and the Making of Entrepreneurial Citizenship, 2015
- Kara Pernice (Nielsen Norman Group), Talking with Participants During a Usability Test, January 26, 2014,
- Matthew G. Kirschenbaum, Extreme Inscription: Towards a Grammatology of the Hard Drive. 2004
- Alexander R. Galloway, The Poverty of Philosophy: Realism and Post-Fordism, Critical Inquiry. 2013
- Edward Alcosser, James P. Phillips, Allen M. Wolk, How to Build a Working Digital Computer. Hayden Book Company, 1968
- Matthew Fuller, “It looks like you’re writing a letter: Microsoft Word”, Nettime, 5 Sep 2000
- Barbara P. Aichinger, DDR Memory Errors Caused by Row Hammer. 2015
- Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, Ruby B. Lee. Last-Level Cache Side-Channel Attacks are Practical. 2015

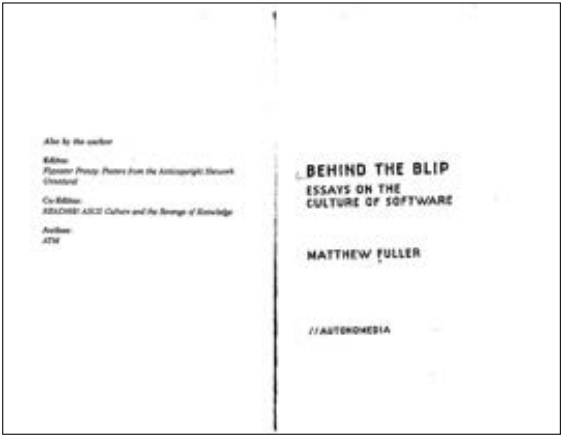








Patent: H. Hollerith. Art of compiling statistics



Cyberpunk meets software studies: Matthew Fuller, “It looks like you’re writing a letter: Microsoft Word”

Original Article

Science, Technology, & Human Values
1–26
© The Author(s) 2015
Reprints and permissions:
sagepub.com/journalsPermissions.nav
DOI: 10.1177/0162243915578486
sth.sagepub.com
SAGE

Hackathons and the Making of Entrepreneurial Citizenship

Lilly Irani[1]

Abstract

Today the halls of Technology, Entertainment, and Design (TED) and Davos reverberate with optimism that hacking, brainstorming, and crowdsourcing can transform citizenship, development, and education alike. This article examines these claims ethnographically and historically with an eye toward the kinds of social orders such practices produce. This article focuses on a hackathon, one emblematic site of social practice where techniques from information technology (IT) production become ways of remaking culture. Hackathons sometimes produce technologies, and they always, however, produce subjects. This article argues that the hackathon rehearses an entrepreneurial citizenship celebrated in transnational cultures that orient toward Silicon Valley for models of social change. Such optimistic, high-velocity practice aligns, in India, with middle-class politics that favor quick and forceful action with socially similar collaborators over the contestations of mass democracy or the slow construction of coalition across difference.

[1]Department of Communication and Science Studies Program, University of California, San Diego, La Jolla, CA, USA

Corresponding Author:
Lilly Irani, Department of Communication and Science Studies Program, University of California, San Diego, 9500 Gilman Dr #0503, La Jolla, CA 92093, USA.
Email: lirani@ucsd.edu

Downloaded from sth.sagepub.com at RUTGERS UNIV on June 8, 2015

Keywords
development, futures, entrepreneurialism, citizenship, technology

Introduction

In the last decade, work practices associated with software production have come to signify collaboration, voluntarism, optimism, and wealth, tested in software practice and ready to enter new domains of public life. Ex-Wired editor Chris Anderson (2012) wrote:

> The past ten years have been about discovering new ways to create, invent, and work together on the Web. The next ten years will be about applying those lessons to the real world.

Anderson's prediction—one which he pursues through his own enterprises in innovation evangelism—voices a broader enthusiasm for bringing Silicon Valley's practices of hacking, designing, and crowdsourcing to the practice of public life. This celebration of scientific and engineering ethos in everyday life in the United States and Europe lags behind formerly colonized countries like India, where modernizing nationalists have long held up scientists and engineers as model citizens (S. Roy 2007; Abraham 2006). Scientific and technological practices do not only make knowledge and things, people also draw on the legitimacy of technoscience to remake culture.

This article focuses on a hackathon, one emblematic site of social practice where techniques from the Web make their way into "the real world."[1] Hackathons bring software programmers and designers together for multiday, voluntary software production sprints. Although hackathons ostensibly produce "demos" (software prototypes), this article argues that hackathons more powerfully produce entrepreneurial subjects. They manufacture urgency and an optimism that bursts of doing and making can change the world. Participants in hackathons imagine themselves as agents of social progress through software, and these middle-class efforts to remake culture draw legitimacy from the global prestige of technology industry work practices. The article uses ethnographic and historical methods to make the case.

Hackathons *sometimes* produce technologies, and they always, however, produce subjects. Science and technology studies (STS) has long examined the subjects and social orders reproduced and valorized in practices of

Downloaded from sth.sagepub.com at RUTGERS UNIV on June 8, 2015

Science studies ethnography: Lilly Irani, Hackathons and the Making of Entrepreneurial Citizenship

Extreme Inscription: Towards a Grammatology of the Hard Drive[1]

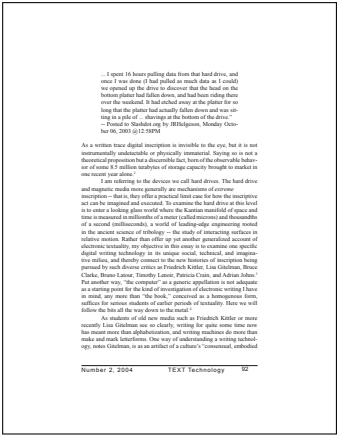
Matthew G. Kirschenbaum
University of Maryland
mgk@umd.edu

Abstract

"Extreme Inscription" attempts to articulate the grammatological primitives of the hard drive, the inscription technology that has had the single greatest impact on computing in the latter half of the 20th century. Rather than offer up yet another generalized account of electronic textuality, my objective in this essay is to examine one specific writing machine in its unique social, technical, and imaginative milieu. Random access disk storage, I argue, is the technology that embodies the "database paradigm" a critic such as Lev Manovich sees as fundamental to new media. The history of hard drive technology is treated in the essay, as is the cultural impact of new hard drive-based technologies like iPod, TiVo, and Gmail's massive multi-gigabyte quotas. Ultimately the article seeks to establish a place for the often invisible and certainly unglamorous presence of storage technologies amid the largely visual and screen-based approaches that currently prevail in new media theory.

> "Had however this friction really existed, in the many centuries that these heavens have revolved they would have been consumed by their own immense speed of every day ... we arrive therefore at the conclusion that the friction would have rubbed away the boundaries of each heaven, and in proportion as its movement is swifter towards the center than toward the poles it would be more consumed in the center than at the poles, and then there would not be friction anymore, and the sound would cease, and the dancers would stop..."
> –Leonardo da Vinci, *The Notebooks*, F 56 V

> "One Monday morning, one of my customers had their WINNT 3.51 server hard drive crash. It was a head crash, you could hear the heads riding the platter. An awful noise



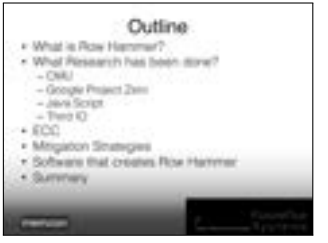
> ... I spent 16 hours pulling data from that hard drive, and once I was done (I had pulled as much data as I could) we opened up the drive to discover that the head on the bottom platter had fallen down, and had been riding there over the weekend. It had etched away at the platter for so long that the platter had actually fallen down and was sitting in a pile of ... shavings at the bottom of the drive."
> -- Posted to Slashdot.org by JRHalgason, Monday October 06, 2003 @12:58PM

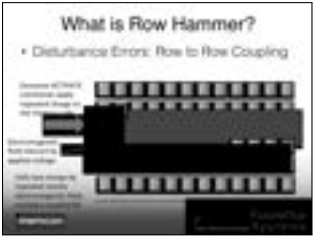
As a written trace digital inscription is invisible to the eye, but it is not instrumentally undetectable or physically immaterial. Saying so is not a theoretical proposition but a discernible fact, born of the observable behavior of some 8.5 million terabytes of storage capacity brought to market in one recent year alone.[2]

I am referring to the devices we call hard drives. The hard drive and magnetic media more generally are mechanisms of *extreme* inscription—that is, they offer a practical limit case for how the inscriptive act can be imagined and executed. To examine the hard drive at this level is to enter a looking glass world where the Kantian manifold of space and time is measured in millionths of a meter (called microns) and thousandths of a second (milliseconds), a world of leading-edge engineering rooted in the ancient science of tribology—the study of interacting surfaces in relative motion. Rather than offer up yet another generalized account of electronic textuality, my objective in this essay is to examine one specific digital writing technology in its unique social, technical, and imaginative milieu, and thereby connect to the new histories of inscription being pursued by such diverse critics as Friedrich Kittler, Lisa Gitelman, Bruce Clarke, Bruno Latour, Timothy Lenoir, Patricia Crain, and Adrian Johns.[3] Put another way, "the computer" as a generic appellation is not adequate as a starting point for the kind of investigation of electronic writing I have in mind; any more than "the book," conceived as a homogeneous form, suffices for serious students of earlier periods of textuality. Here we will follow the bits all the way down to the metal.[4]

As students of old new media such as Friedrich Kittler or more recently Lisa Gitelman see so clearly, writing for quite some time now has meant more than alphabetization, and writing machines do more than make and mark letterforms. One way of understanding a writing technology, notes Gitelman, is as an artifact of a culture's "consensual, embodied

Media studies: Matthew G. Kirschenbaum, Extreme Inscription: Towards a Grammatology of the Hard Drive.





Technical presentation slides: Barbara P. Aichinger, DDR Memory Errors Caused by Row Hammer

SEE ALSO:: Ask several people from different fields and age groups the same question: "What is software?"

➡ P.49

SOURCE::

The *Technogalactic Software Observatory reader* can be consulted here: `http://pad.constantvzw.org/p/observatory.reader`

Keyword Index

Abstraction, 56, 202, 225, 226
Abuse, 11
Accident, 47, 163
Agility, 5, 108, 135–137, 139, 141–144, 165, 168, 226
Air, 98, 141, 143, 144, 170
Algorithm, 175, 182
Ambiguity, 49, 114, 193
Assembly, 31, 35, 44
Attack, 108, 191, 202

BeOS, 124, 126
Binary, 18, 50, 154
Bits, 50, 225
Bounce, 108, 170
Bug, 108, 150, 151
Bull, 18, 29
Bullshit, 156, 159
Bytes, 50

Catastrophe, 122
Cobol, 44
Colonialism, 5, 108
Command line, 108
Compilation, 33, 37, 45, 106, 108, 208, 224, 225, 228
Control, 15, 71, 210, 211, 216
Convenience, 65, 69, 70, 102, 108, 141
Cost, 41, 61, 112, 122–124, 129, 131
Crash, 108
Critique, 3, 4, 6–9, 16, 150, 156, 222, 227

Debugging, 56, 150, 174, 186, 222
DecAlpha, 126
Decentralization, 167
Digital Equipment Corporation, 39, 42
Dirty, 108, 151, 153–155

Emotion, 30, 89, 108, 119, 129, 163
Error, 31, 35, 40, 151, 155, 168
etherbox, 50, 98, 147, 153, 155, 186
Ethics, 122
Executable, 15, 41, 52, 62

File, 37, 50, 52–54, 56, 57, 59, 93, 95, 96, 104, 114, 115, 120, 123, 129, 151, 153–155, 159, 172, 177, 182, 215
File system, 50, 53, 151, 153
Flow, 3, 4, 94, 108, 166, 172, 175, 179, 182–184
Fortran, 31, 35, 36, 44
Free Software, 124, 129, 130
Friendly, 42, 45, 46, 147, 211

Gesture, 162
Ghost, 5, 211

Glitch, 147

Habit, 122
Hard-boiled, 150, 215
Healing, 156
Hexadecimal, 50
History, 30, 31, 43, 96, 101, 122, 131, 225

IBM, 37–39, 43, 44, 61, 178, 182, 226
Idiosyncracy, 172
Incarnation, 196
Inspection as a Service, 189
Integration, 167, 168, 170, 222
Interface, 6, 29, 47, 53, 61, 156, 157, 159, 160, 209, 225, 226
Internet of Things, 208, 213
Intimacy, 57, 69, 108, 120
IRIX, 126

JAVA, 44, 46, 47

Kernel, 45, 53, 58, 62, 92, 93, 108

Linux, 45, 53, 62, 63, 93, 96, 126, 129, 130, 208

MacOS, 126
Magic, 33, 45, 101, 108, 156
Mainframe, 37, 39, 42, 61
Materiality, 71, 124, 184, 193, 202, 225–227
Measuring, 89, 90, 92, 97
Military, 191, 209, 226
Minitel, 43
Misunderstanding, 11
Musée de l'Informatique Pionnière en Belgique, 18, 26, 29, 50, 178

Noise, 64, 65, 69, 94, 109

Obfuscation, 172
Office, 35, 40, 64, 65, 69, 70, 98

Panic, 197
Parody, 156
Passive-aggressive, 109
PDP11, 40
Perl, 44, 47
PL-1, 44
Poetry, 156, 193
Problems, 40, 120, 121, 126, 159
Productivity, 3, 5, 6, 79, 83, 115, 129, 165, 222, 226
Professionalism, 6, 70, 126, 139, 141
Promiscuity, 109, 171, 172, 191

Reconfiguration, 167
Relational, 5, 30, 109, 121, 123, 126, 129, 131, 175, 208
Relaxation, 64, 94, 141
RK07, 40

Scroll, 79, 83, 109
Scrum, 109, 136, 141, 143
Separation, 57, 62, 120, 207, 226

Silence, 65, 69, 109
Software as a Critique as a Service, 8, 9
Software as a Service, 3
Software with Observatory Ambitions, 220–223
Solaris, 126
Statistics, 207, 225, 233
Stress, 103
Success, 15, 170
SunOS, 126
Surveillance, 191, 193
Survival, 11, 109
System memory, 52, 58, 124

TCP/IP, 43, 210
Terms of use, 40, 106, 115, 124, 131
Therapy, 120
Time, 4, 11, 17, 26, 31, 35, 36, 39–44, 53, 61–63, 73, 85–87, 89–98, 101, 109, 119, 121, 124, 126, 129, 130, 144, 148, 151, 154, 155, 163, 184, 193, 211, 226
Timeless, 120
Touch, 17, 18, 29, 37, 142, 143, 173, 191, 210

Unix, 39–42, 44–47, 50, 52, 53, 85, 86, 91, 120, 126
Useless, 79, 83, 103, 104, 109
UUCP, 41, 45

VAX, 42
Virtual, 42, 57, 189
Vocabulary, 50, 101, 106, 107, 182, 189
Vulnerability, 3

Windows 95, 33, 45, 126

X/OPEN, 42, 43

Read Further!

Bentley, Jon and Don Knuth: *Programming Pearls: Literate Programming*. In: *Commun. ACM* 29.5 (May 1986), pp. 384–369. ISSN: 0001-0782. DOI: 10.1145/5689.315644. http://doi.acm.org/10.1145/5689.315644.

Content Watch Holdings, Inc.: *Protecting Your Family*. accessed 7.1.2018. 2018. https://www.netnanny.com/products/netnanny/protecting-your-family.

Dijkstra, Edsger W.: *A Case against the GOTO Statement*. accessed 9.2.2018. 1972. https://www.cs.utexas.edu/users/EWD/transcriptions/EWD02xx/EWD215.html.

Dijkstra, Edsger W.: *Programming: From craft to scientific discipline*. accessed 9.2.2018. 1977. https://www.cs.utexas.edu/users/EWD/transcriptions/EWD05xx/EWD566.html.

Ekbia, Hamid and Bonnie Nardi: *Heteromation and its (dis)contents: The invisible division of labor between humans and machines*. en. In: *First Monday* 19.6 (May 2014). ISSN: 13960466. http://firstmonday.org/ojs/index.php/fm/article/view/5331 visited on Jan. 31, 2018.

Flanagan, Mary and Helen Nissenbaum: *Values at Play in Digital Games*. 2014.

Fuller, Matthew: *Behind the Blip: Essays on the Culture of Software*. English. Brooklyn, NY: Autonomedia, Mar. 2003. ISBN: 978-1-57027-139-7.

Geertz, Clifford: *Common Sense as a Cultural System*. In: *The Antioch Review* 33.1 (1975), pp. 5–26. ISSN: 00035769.

Haigh, Thomas: *Software in the 1960s as Concept, Service, and Product*. In: *IEEE Annals of the History of Computing* 24.1 (2002), pp. 5–13.

Haraway, Donna: *Modest Witness: Feminist Diffractions in Science Studies*. In: *The Disunity of Science: Boundaries, Contexts, and Power*. Ed. by Peter Galison and David J. Stump. 1996, pp. 428–442.

Harwood, John: *The Interface: IBM and the Transformation of Corporate Design, 1945–1976*. English. 1 edition. Minneapolis, MN: Univ Of Minnesota Press, Nov. 2011. ISBN: 978-0-8166-7039-0.

Hollerith, Herman: “Art of compiling statistics”. US395781 A. Cooperative Classification G06F7/42. Jan. 1889. http://www.google.com/patents/US395781 visited on Jan. 31, 2018.

Horn, Jann et al.: *Meltdown and Spectre Attack*. accessed 9.2.2018. 2018. https://meltdownattack.com.

Hu, Tung-Hui: *A Prehistory of the Cloud*. MIT Press, 2015.

Irani, Lilly: *Hackathons and the Making of Entrepreneurial Citizenship*. en. In: *Science, Technology, & Human Values* 40.5 (Sept. 2015), pp. 799–824. ISSN: 0162-2439, 1552-8251. DOI: 10.1177/0162243915578486. http://journals.sagepub.com/doi/10.1177/0162243915578486 visited on Jan. 31, 2018.

Kirschenbaum, Matthew: *Extreme Inscription: Towards a Grammatology of the Hard Drive*. In: *TEXT Technology* 2 (2004). http://texttechnology.mcmaster.ca/pdf/vol13_2_06.pdf visited on Jan. 31, 2018.

Kruchten, Philippe: *Agile's Teenage Crisis?* accessed 7.1.2018. 2011. https://www.infoq.com/articles/agile-teenage-crisis.

Kyong Chun, Wendy Hui: *Programmability*. In: *Software Studies*. Ed. by Matthew Fuller. DOI: 10.7551/mitpress/9780262062749.003.0032. 2008, pp. 225–228. ISBN: 978-0-262-06274-9. http://mitpress.universitypressscholarship.com/view/10.7551/mitpress/9780262062749.001.0001/upso-9780262062749-chapter-32 visited on Jan. 31, 2018.

Lammerant, Hans: *How Humans and Machines negotiate the Experience of Time*. 2017. http://etherdump.constantvzw.org/p/observatory.guide.experiencingtime.diff.html.

Lau, Justin and Xing: *Understanding Flowcharts*. 2009. http://odec.ca/projects/2009/xing9t2/hist.htm.

Liu, F. et al.: "Last-Level Cache Side-Channel Attacks are Practical". In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 605–622. DOI: 10.1109/SP.2015.43.

Marx, Karl and Friedrich Engels: *The communist manifesto*. Penguin, 2002.

Neubert, Christoph: *The Tail on the Hardware Dog*. English. In: *There is no Software, there are just Services*. Ed. by Irina Kaldrack and Martina Leeker. Lüneburg, 2015, pp. 21 –37. ISBN: 978-3-95796-055-9.

Nofre, David, Mark Priestley, and Gerard Alberts: *When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960*. en. In: *Technology and Culture* 55.1 (Mar. 2014), pp. 40–75. ISSN: 1097-3729. DOI: 10.1353/tech.2014.0031. https://muse.jhu.edu/article/538908 visited on Jan. 31, 2018.

Patterson, David A. and John L. Hennessy: *Computer Organization and Design MIPS Edition, Fifth Edition: The Hardware/Software Interface*. English. 5 edition. Amsterdam ; Boston: Morgan Kaufmann, Oct. 2013. ISBN: 978-0-12-407726-3.

Pernice, Kara: *Talking with Users in a Usability Test*. en. 2014. https://www.nngroup.com/articles/talking-to-users/ visited on Jan. 31, 2018.

Plant, Sadie: *Zeroes and Ones: Digital Women and the New Technoculture*. English. 1st edition. New York: Doubleday, Sept. 1997. ISBN: 978-0-385-48260-8.

Stallman, Richard: *What Does That Server Really Serve?* en. June 2012. http://bostonreview.net/richard-stallman-free-software-DRM visited on Jan. 31, 2018.

TechMission UrbanMinistry.org: *SafeFamilies.org | Accountability Software: Encyclopedia of Urban Ministry*. accessed 7.1.2018. 2018. http://www.urbanministry.org/node/4902.

Techno-Galactic Software Observatory: *Introduction to file therapy*. 2017. http://observatory.constantvzw.org/etherdump/clinic.file_therapy.md.diff.html.

Techno-Galactic Software Observatory: *Notes from the Observatory on glossaries and vocabularies*. 2017. http://observatory.constantvzw.org/etherdump/vocabulary.md.diff.html.

Techno-Galactic Software Observatory: *Notes from the Observatory on When and Where is Software*. 2017. http://observatory.constantvzw.org/etherdump/saturday.diff.html.

time was written by David MacKenzie. The man page was added by DirkEddelbuettel: *TIME(1) General Commands Manual*. http://freeze.sh/man/time.

Ullman, Ellen: *Close to the Machine: Technophilia and Its Discontents*. English. Reprint edition. New York: Picador, Feb. 2012. ISBN: 978-1-250-00248-8.

websense.com: *Explicit and transparent proxy deployments*. accessed 7.1.2018 via waybackmachine (18.4.2012). 2012. https://web.archive.org/web/20120418150020/http://www.websense.com/content/support/library/web/v75/wcg_deploy/WCG_Deploy.1.3.aspx.

Wikipedia contributors: *Agile software development — Wikipedia, The Free Encyclopedia*. accessed 7.1.2018. 2018. https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=818894701.

Wikipedia contributors: *Content-control software — Wikipedia, The Free Encyclopedia*. accessed 7.1.2018. 2018. https://en.wikipedia.org/w/index.php?title=Content-control_software&oldid=818780033.

Wikipedia contributors: *Lernaean Hydra — Wikipedia, The Free Encyclopedia*. accessed 7.2.2018. 2018. https://en.wikipedia.org/w/index.php?title=Lernaean_Hydra&oldid=812030608.

Wikipedia contributors: *Scrum (software development) — Wikipedia, The Free Encyclopedia*. accessed 7.1.2018. 2018. https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=816207177.

Wikipedia contributors: *The Manifesto for Agile Software Development*. accessed 7.1.2018. 2018. https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=818894701#The_Agile_Manifesto.

workrave.org: *Frequently Asked Questions*. accessed 7.1.2018. 2018. http://www.workrave.org/documentation/faq.

The Techno-Galactic Guide to Software Observation was compiled by Carlin Wing, Martino Morandi, Peggy Pierrot, Anita Burato, Christoph Haag, Michael Murtaugh, Femke Snelting, Seda Gürses and includes contributions from Manetta Berends, Željko Blaće, Larisa Blazic, Freyja van den Boom, Anna Carvalho, Loup Cellard, Joana Chicau, Cristina Cochior, Pieter Heremans, Joak aka Joseph Knierzinger, Jogi Hofmüller, Becky Kazansky, Anne Laforet, Ricardo Lafuente, Michaela Lakova, Hans Lammerant, Silvio Lorusso, Mia Melvaer, An Mertens, Lidia Pereira, Donatella Portoghese, Luis Rodil-Fernandez, Natacha Roussel, Andrea di Serego Alighieri, Lonneke van der Velden, Ruben van de Ven, Kym Ward, Wendy Van Wynsberghe and Peter Westenberg.

Techno-Galactic Software Observation team,
WTC Brussels, June 2017 (➡ P.243)

Copy editing: Carlin Wing
Layout, document engineering and bespoke finish: Christoph Haag
Photography: Michaela Lakova, Michael Murtaugh, Peter Westenberg, Donatella Portoghese
Printing: Online-Druck.biz, Krumbach (Schwaben)
Published by: Constant, Association for Art and Media, Brussels (2018) **ISBN**: 978-9-08114-596-1
License: Free Art License (➡ P.248)

SOURCE:

```
http://observatory.constantvzw.org
http://gitlab.constantvzw.org/ch/observatory.guide
```

Notes on Layout:

Typesetting this guide is part of an ongoing exploration of document engineering along with ideas of lightweight markup languages and infinite rubber lengths. While moving through interfaces of digital editing, transplanting paradigms, experiencing limitations and possibilities, it seems not like an end is in sight. To put it in the spirit of illiterate programming: *When was the last time you spent a pleasant evening in a comfortable chair, cuddling with a multi-headed monster.* [83]

`https://freeze.sh/_/2018/tgso`

The fonts used in this guide were prepared to be integrated into the Techno-Galactic LaTeX Toolchain and are available via the *fontain* font collection.

`https://fontain.org/arimo`
`https://fontain.org/plexmono`
`https://fontain.org/iaduospace`
`https://fontain.org/pxpcgathin`

Arimo is a sans serif typeface developed by Steve Matteson [84] and released under the Apache 2.0 License. Together with Tinos (serif) and Cousine (monospace) it provides the Chrome OS core fonts, a collection of fonts that are metrically compatible with Monotype Corporation's Arial, Times New Roman, and Courier New.

---

[83] Jon Bentley and Don Knuth: *Programming Pearls: Literate Programming*. In: *Commun. ACM* 29.5 (May 1986), pp. 384–369. ISSN: 0001-0782. DOI: 10.1145/5689.315644

[84] `http://www.monotype.com/people/steve-matteson`

Plex Mono is a monospaced typeface and part of the superfamily IBM Plex[85], which was developed to replace fifty years of Neue Helvetica as IBM's corporate typeface. It is released under the SIL Open Font License.

IA Writer Duospace is an adaptation of Plex Mono, aiming at better readability while keeping the look and feel[86] of a monospaced typeface. It is released under the SIL Open Font License.

CGA Thin is a 8x8 pixel font recreated in truetype format by VileR.[87] It is based on text mode fonts shipped on the character ROM of IBM's first video solutions. As part of the **The Ultimate Oldschool PC Font Pack** it is available according to Creative Commons Attribution-ShareAlike 4.0 International License.

Support:

Vlaanderen
verbeelding werkt

brussel

---

[85] https://github.com/IBM/plex
[86] https://ia.net/topics/in-search-of-the-perfect-writing-font
[87] http://int10h.org

Free Art License 1.3. (C) Copyleft Attitude, 2007. You can make reproductions and distribute this license verbatim (without any changes). Translation: Jonathan Clarke, Benjamin Jean, Griselda Jung, Fanny Mourguet, Antoine Pitrou. Thanks to framalang.org

PREAMBLE

The Free Art License grants the right to freely copy, distribute, and transform creative works without infringing the author's rights.

The Free Art License recognizes and protects these rights. Their implementation has been reformulated in order to allow everyone to use creations of the human mind in a creative manner, regardless of their types and ways of expression.

While the public's access to creations of the human mind usually is restricted by the implementation of copyright law, it is favoured by the Free Art License. This license intends to allow the use of a works resources; to establish new conditions for creating in order to increase creation opportunities. The Free Art License grants the right to use a work, and acknowledges the right holders and the users rights and responsibility.

The invention and development of digital technologies, Internet and Free Software have changed creation methods: creations of the human mind can obviously be distributed, exchanged, and transformed. They allow to produce common works to which everyone can contribute to the benefit of all.

The main rationale for this Free Art License is to promote and protect these creations of the human mind according to the principles of copyleft: freedom to use, copy, distribute, transform, and prohibition of exclusive appropriation.

DEFINITIONS

"*work*" either means the initial work, the subsequent works or the common work as defined hereafter:

"*common work*" means a work composed of the initial work and all subsequent contributions to it (originals and copies). The initial author is the one who, by choosing this license, defines the conditions under which contributions are made.

"*Initial work*" means the work created by the initiator of the common work (as defined above), the copies of which can be modified by whoever wants to

"*Subsequent works*" means the contributions made by authors who participate in the evolution of the common work by exercising the rights to reproduce, distribute, and modify that are granted by the license.

"*Originals*" (sources or resources of the work) means all copies of either the initial work or any subsequent work mentioning a date and used by their author(s) as references for any subsequent updates, interpretations, copies or reproductions.

"*Copy*" means any reproduction of an original as defined by this license.

OBJECT

The aim of this license is to define the conditions under which one can use this work freely.

SCOPE

This work is subject to copyright law. Through this license its author specifies the extent to which you can copy, distribute, and modify it.

FREEDOM TO COPY (OR TO MAKE REPRODUCTIONS)

You have the right to copy this work for yourself, your friends or any other person, whatever the technique used.

FREEDOM TO DISTRIBUTE, TO PERFORM IN PUBLIC

You have the right to distribute copies of this work; whether modified or not, whatever the medium and the place, with or without any charge, provided that you: attach this license without any modification to the copies of this work or indicate precisely where the license can be found, specify to the recipient the names of the author(s) of the originals, including yours if you have modified the work, specify to the recipient where to access the originals (either initial or subsequent). The authors of the originals may, if they wish to, give you the right to distribute the originals under the same conditions as the copies.

FREEDOM TO MODIFY

You have the right to modify copies of the originals (whether initial or subsequent) provided you comply with the following conditions: all conditions in article 2.2 above, if you distribute modified copies; indicate that the work has been modified and, if it is possible, what kind of modifications have been made; distribute the subsequent work under the same license or any compatible license. The author(s) of the original work may give you the right to modify it under the same conditions as the copies.

RELATED RIGHTS

Activities giving rise to authors rights and related rights shall not challenge the rights granted by this license. For example, this is the reason why performances must be subject to the same license or a compatible license. Similarly, integrating the work in a database, a compilation or an anthology shall not prevent anyone from using the work under the same conditions as those defined in this license.

INCORPORATION OF THE WORK

Incorporating this work into a larger work that is not subject to the Free Art License shall not challenge the rights granted by this license. If the work can no longer be accessed apart from the larger work in which it is incorporated, then incorporation shall only be allowed under the condition that the larger work is subject either to the Free Art License or a compatible license.

COMPATIBILITY

A license is compatible with the Free Art License provided: it gives the right to copy, distribute, and modify copies of the work including for commercial purposes and without any other restrictions than those required by the respect of the other compatibility criteria; it ensures proper attribution of the work to its authors and access to previous versions of the work when possible; it recognizes the Free Art License as compatible (reciprocity); it requires that changes made to the work be subject to the same license or to a license which also meets these compatibility criteria.

YOUR INTELLECTUAL RIGHTS

This license does not aim at denying your author's rights in your contribution or any related right. By choosing to contribute to the development of this common work, you only agree to grant others the same rights with regard to your contribution as those you were granted by this license. Conferring these rights does not mean you have to give up your intellectual rights.

YOUR RESPONSIBILITIES

The freedom to use the work as defined by the Free Art License (right to copy, distribute, modify) implies that everyone is responsible for their own actions.

DURATION OF THE LICENSE

This license takes effect as of your acceptance of its terms. The act of copying, distributing, or modifying the work constitutes a tacit agreement. This license will remain in effect for as long as the copyright which is attached to the work. If you do not respect the terms of this license, you automatically lose the rights that it confers. If the legal status or legislation to which you are subject makes it impossible for you to respect the terms of this license, you may not make use of the rights which it confers.

VARIOUS VERSIONS OF THE LICENSE

This license may undergo periodic modifications to incorporate improvements by its authors (instigators of the Copyleft Attitude movement) by way of new, numbered versions. You will always have the choice of accepting the terms contained in the version under which the copy of the work was distributed to you, or alternatively, to use the provisions of one of the subsequent versions.

SUB-LICENSING

Sub-licenses are not authorized by this license. Any person wishing to make use of the rights that it confers will be directly bound to the authors of the common work.

LEGAL FRAMEWORK

This license is written with respect to both French law and the Berne Convention for the Protection of Literary and Artistic Works.